

**CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY**



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

**INTELLIGENT MEDICAL CHATBOT USING
RAG, USER MEMORY, AND PROACTIVITY**

Student: Tran Quoc Duy

Student ID: B2111974

Class: 2021-2025 (K47)

Advisor: Dr. Pham Nguyen Khang

Can Tho, 07/2025

CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY



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

**INTELLIGENT MEDICAL CHATBOT USING
RAG, USER MEMORY, AND PROACTIVITY**

**Student: Tran Quoc Duy
Student ID: B2111974
Class: 2021-2025 (K47)
Advisor: Dr. Pham Nguyen Khang**

Can Tho, 07/2025

ACKOWLEDGEMENTS

I would like to sincerely thank all the lecturers at Can Tho University, especially those from the Faculty of Information Technology, for their dedicated teaching and guidance throughout my time at university. Thanks to their support and knowledge, I was able to build a strong foundation to complete this thesis. I am especially grateful to Mr. Pham Nguyen Khang for his kind support and clear guidance during the entire process. His advice helped me overcome difficulties and complete the topic more easily and effectively. During the process of doing this thesis and preparing the report, I know there may still be mistakes and shortcomings. I kindly ask for your understanding and would truly appreciate any comments or suggestions from the teachers to help improve my work. Once again, thank you very much for all your support!

Tran Quoc Duy

ABSTRACT

In recent years, artificial intelligence (AI) has become more popular and widely applied in many fields. However, in Vietnam, the use of AI in healthcare is still quite limited.

This thesis focuses on building a smart medical chatbot using large language models (LLMs), aiming to support users with health-related questions. The system combines several technologies, including Retrieval-Augmented Generation (RAG), OpenAI's API, user memory, and proactive interaction features. A vector database is used for document retrieval, and natural language processing (NLP) techniques help improve the chatbot's responses.

Through testing, the chatbot showed good performance in answering user queries in a natural and relevant way.

The results suggest that this chatbot model could be useful and has the potential for further development in healthcare applications.

Keywords:

Artificial Intelligence (AI), Medical Chatbot, Retrieval-Augmented Generation (RAG), Large Language Model (LLM), User Memory, Natural Language Processing (NLP), Vector Database, Proactive Interaction.

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION	9
1. Research Objective	9
2. Research Problem.....	9
3. Related Work.....	9
4. Thesis and General Approach	10
5. Criteria for Study Success.....	10
6. Outline of the Thesis.....	10
CHAPTER 2: PRESENTS THE THEORETICAL FOUNDATIONS, CONCEPTS, AND RELATED DOCUMENTS	12
1. LLMs.....	12
2. Retrieval-Augmented Generation (RAG):	14
3. Hybrid Search.....	18
4. ChatGPT – The Generation Component.....	21
5. User Memory in Conversational AI.	22
6. Summary of chapter 2.....	24
CHAPTER 3: SYSTEM DESIGN AND IMPLEMENTATION	26
1. Document Collection And Processing.....	26
2. Embedding Documents And Building The Vector Database	28
3. Hybrid Search Retrieval.....	31
4. Response Generation And Conversation Flow	33
5. Memory Integration	36
CHAPTER 4 FOCUSES ON THE IMPLEMENTATION AND EVALUATION OF THE PROPOSED MEDICAL CHATBOT MODEL	38
1. Implementation Process	38
2. Evaluation Of Chatbot Performance	40
CHAPTER 5. CONCLUSION AND FUTURE WORK	50
1. Conclusion.....	50
2. Future Work	51

LIST OF TABLES

Table 1. compare 2 model	22
Table 2. 65 disease	28
Table 3. comparison between models	29

LIST OF FIGURES

Figure 1 LLM	12
Figure 2 RAG work.....	15
Figure 3 Overview of RAG process	17
Figure 4 Hybrid Search Work.....	19
Figure 5 Chat GPT	21
Figure 6 System Architecture of Medical Chatbot	26
Figure 7: sentence-transformers/multi-qa-mpnet-base-dot-v1	30
Figure 8 local final FAISS	31
Figure 9 vector search	31
Figure 10 BM25 Search.....	32
Figure 11 Fuzzy Search	32
Figure 12 Re-ranking.....	32
Figure 13 BAAI/bge-reranker-base.....	33
Figure 14 Scores.....	33
Figure 15 send request to OpenAI	35
Figure 16 result	35
Figure 17 Process flow from Hybrid Search to answer generation	36
Figure 18 process of memory	37
Figure 19 run file prepare_vector_db.....	38
Figure 20 Technical Details	39
Figure 21 Chatbot execution.....	39
Figure 22 answer of chatbot.....	40
Figure 23 Test model gpt-4.....	41
Figure 24 number of requests	41
Figure 25 Number of token.....	42
Figure 26 cost of ada model.....	42
Figure 27 Number of token(ada).....	43
Figure 28 model local LLM.....	44
Figure 29 number of tokens input(gpt-3.5)	45
Figure 30 cost input(gpt-3.5)	45
Figure 31 Result of Integrated RAG	45
Figure 32 Chart of Accuracy(%).....	46
Figure 33 Chart of Response Time(s)	48
Figure 34 Chart of Cost (\$).....	49

LIST OF ABBREVIATIONS

Abbreviation	Full Form
AI	Artificial Intelligence
API	Application Programming Interface
BM25	Best Matching 25 (Ranking Algorithm)
CPU	Central Processing Unit
FAISS	Facebook AI Similarity Search
GGUF	GPT-Generated Unified Format
GPT	Generative Pre-trained Transformer
LLM	Large Language Model
PDF	Portable Document Format
RAG	Retrieval-Augmented Generation
UI	User Interface
VLLM	Very Large Language Model (tên gọi chung)
NLP	Natural Language Processing
QA	Question Answering
VS Code	Visual Studio Code
GPT	Generative Pre-trained Transformer

CHAPTER 1: INTRODUCTION

1. Research Objective

In today's rapidly evolving technological landscape, the use of artificial intelligence (AI) has become increasingly natural in both personal and social aspects of life. Researching and developing AI applications not only contributes to making life more convenient and efficient but also helps improve the quality of life and meet the diverse needs of people around the world. This study aims to develop an intelligent AI system to assist users in the healthcare field, thereby emphasizing the practical role and potential of AI in real-world applications.

2. Research Problem

Recently, large language models (LLMs) have become more widely used because they can understand and generate text in a way that feels natural, like human conversation. This makes them a good choice for building medical chatbots. However, using LLMs alone still has some issues. For example, the chatbot might forget what was said earlier in the conversation or give answers that are not very accurate because it doesn't access real medical information.

To solve this, combining LLMs with a retrieval system (RAG) and user memory can make the chatbot better. It can understand specific medical topics like definitions, causes, and symptoms of diseases, and respond in a way that's easier for users to understand. That's why this topic was chosen for the project.

3. Related Work

Currently, there are many research works related to large language models (LLMs) and the development of RAG-based systems for chatbots. Studies such as "*Retrieval-Augmented Generation for AI-Generated Content: A Survey*" and "*Retrieval-Augmented Generation or Long-Context LLMs? A Comprehensive Study and Hybrid Approach*" provide overviews and analyze the advantages and limitations of each approach. These papers are easy to find online, as the topic is becoming increasingly popular in the field of artificial intelligence.

In addition, many LLM models are being developed, with ChatGPT being one of the most well-known. It is a powerful AI system capable of engaging in natural and flexible conversations with users. ChatGPT is also widely used as a reference when studying how modern LLMs work.

In Vietnam, medical chatbot systems have been introduced, but they are not yet widely adopted. One of the main reasons is that these systems are not powerful enough to communicate naturally or provide accurate answers — which is crucial because the information relates directly to users' health. Therefore, a stronger solution is needed, and LLMs offer a suitable choice.

Unlike previous studies, this project proposes a flexible combination of modern techniques, including RAG, LLM, and user memory. The goal is to build a chatbot that can interact naturally, is easy to use, and delivers accurate and reliable medical information.

4. Thesis and General Approach

The purpose of this research is to create a medical chatbot that is friendly to users, answers questions naturally, and provides highly reliable information. The goal is to make medical chatbots more common and useful in people's daily lives.

To build this system, many elements need to be combined flexibly. These include an LLM model, a RAG system, and the integration of hybrid search to enhance the ability to retrieve documents. The answers are handled more naturally and smoothly by using ChatGPT, and additionally, a user memory component is added so the chatbot can understand the context of the conversation.

The approach is quite simple: collect documents about different diseases, then build an information retrieval system that understands user questions and returns reasonable, easy-to-understand answers. The main goal is to create a intelligent, accessible, trustworthy, and user-friendly medical chatbot.

5. Criteria for Study Success

To evaluate whether the chatbot system is successful, several criteria are used.

- First, the accuracy of the chatbot's answers is important. The information it provides must match reliable medical documents.
- Second, the chatbot should be able to communicate naturally, so users feel like they are talking to a real assistant.
- Third, the response time should be fast so that users do not have to wait too long.
- Fourth, the system should help reduce costs for users.

If the chatbot can meet these points, then the research can be considered successful.

6. Outline of the Thesis

This thesis focuses on researching medical chatbots using large language models, especially the Retrieval-Augmented Generation (RAG) system. The content is divided into five chapters.

Chapter 1 introduces the general background and purpose of the thesis.

Chapter 2 presents the theoretical foundations, concepts, and related documents.

Chapter 3 describes the process of building the chatbot model.

Chapter 4 explains the implementation steps and shows the evaluation results.

Finally, Chapter 5 concludes the thesis and provides suggestions for future improvements.

CHAPTER 2: PRESENTS THE THEORETICAL FOUNDATIONS, CONCEPTS, AND RELATED DOCUMENTS.

1. LLMs

1.1 What is LLMs?

A large language model (LLM) is a language model trained with self-supervised machine learning on a vast amount of text, designed for natural language processing tasks, especially language generation.



Figure 1. LLM

1.2 The importance of LLM.

Large language models are incredibly versatile. A single model can perform completely different tasks, such as question answering, document summarization, language translation, and sentence completion. LLMs have the potential to disrupt content creation and the way people use search engines and virtual assistants. LLM can be used for generative AI to generate content based on prompts entered in human language.

1.3 How it works?

Using word embedding, the transducer can preprocess the text as a numerical representation through the encoder and understand the context of similar words and phrases, as well as other relationships between words, such as parts of speech. The LLM can then apply this linguistic knowledge through the decoder to produce a unique output.

It uses multidimensional vectors, often called word embeddings, to represent words so that words with similar contextual meanings or other relationships will be close together in vector space.

1.4 Applications of Large Language Models.

LLM has many practical applications.

- Copywriting: In addition to GPT-3 and ChatGPT, Claude, Llama 2, Cohere Command, and Jurassic can also write native ads. AI21 Wordspice suggests changes to the original sentence to improve style and tone.
- Knowledge-Based Answering: Often referred to as knowledge-intensive natural language processing (KI-NLP), this technique refers to LLMs that are able to answer specific questions based on information stored in digital repositories. An example is the AI21 Studio Playground's ability to answer general knowledge questions.
- Text Classification: LLMs can categorize texts with similar meanings or sentiments using clusters. Use cases include measuring customer sentiment, identifying relationships between texts, and document search.
- Code Generation: LLMs are proficient at generating code from natural language prompts. For example, Amazon CodeWhisperer and Open AI's codex used in GitHub Copilot can write code in Python, JavaScript, Ruby, and several other programming languages. Other coding applications include creating SQL queries, writing shell commands, and designing websites.
- Text generation: Similar to code generation, text generation can complete incomplete sentences, write product documentation, or, like Alexa Create, write a short story for children.

1.5 Training

Training is performed using a large corpus of high-quality data. During training, the model continually adjusts parameter values until the model accurately predicts the next token from a sequence of previous input tokens. The model does this through self-learning techniques, which help the model learn to adjust its parameters to maximize the likelihood of the next token in the training samples.

Once trained, an LLM can easily be adapted to perform a variety of tasks using relatively small supervised data sets, a process known as fine-tuning.

There are three common learning models:

- Learning with unseen data; The baseline LLM can respond to a wide range of requests without explicit training, often through text commands, although the accuracy of the response can vary.

- Learning with little training data: By providing some relevant training examples, the performance of the base model improves significantly in that particular domain.
- Fine-tuning: This is an extension of learning with little training data, where data scientists train a base model to tune its parameters with additional data relevant to the specific application.

1.6 Some popular LLMs

- GPT-3.5, GPT-4 (OpenAI)
- LLaMA (Meta), PaLM (Google), Claude (Anthropic)...
- Open source LLM: Mistral, Falcon, etc.

2. Retrieval-Augmented Generation (RAG):

2.1 What is RAG?

Retrieval-augmented generation (RAG) is a technique that enables large language models (LLMs) to retrieve and incorporate new information. With RAG, LLMs do not respond to user queries until they refer to a specified set of documents. These documents supplement information from the LLM's pre-existing training data. This allows LLMs to use domain-specific and/or updated information that is not available in the training data.

2.2 How it works?

RAG is about feeding language models with necessary information. Instead of asking LLM directly (like in general-purpose models), we first retrieve the very accurate data from our knowledge library that is well maintained and then use that context to return the answer. When the user sends a query (question) to the retriever, we use vector embeddings (numerical representations) to retrieve the requested document. Once the needed information is found from the vector databases, the result is returned to the user. This largely reduces the possibility of hallucinations and updates the model without retraining the model, which is a costly process. Here's a very simple diagram that shows the process.

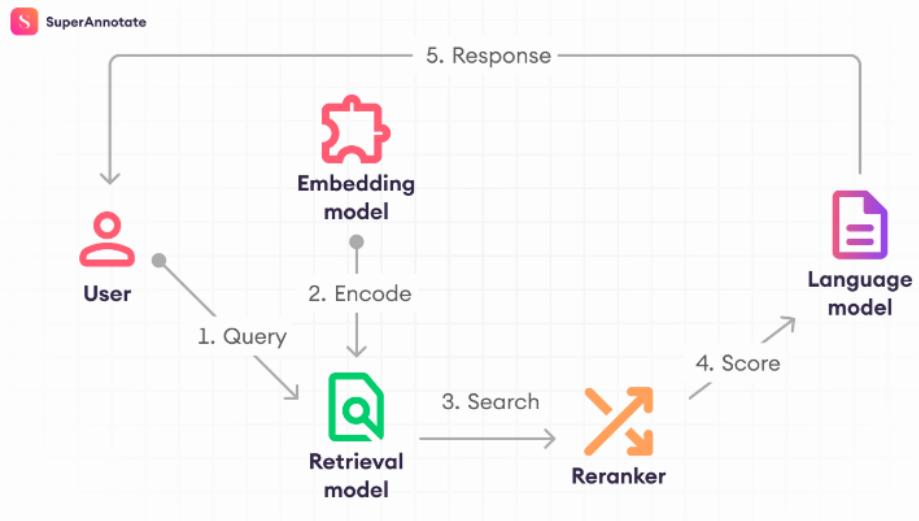


Figure 2. RAG work

RAG brings together four key components:

- **Embedding model:** This is where documents are turned into vectors, or numerical representations, which make it easier for the system to manage and compare large amounts of text data.
- **Retriever:** Think of this as the search engine within RAG. It uses the embedding model to process a question and fetch the most relevant document vectors that match the query.
- **Reranker (optional):** This component takes things a step further by evaluating the retrieved documents to determine how relevant they are to the question at hand, providing a relevance score for each one.
- **Language model:** Finally, this part of the system takes the top documents provided by the retriever or reranker, along with the original question, and crafts a precise answer.

2.3 Why use RAG?

RAG offers several advantages over traditional text generation methods, especially when it comes to factual information or data-driven responses. Here are some key reasons why using RAG can be beneficial:

Access to up-to-date information: LLMs are limited to their pre-trained data. This results in outdated and potentially inaccurate answers. RAG solves this problem by providing up-to-date information to LLMs.

Factual anchoring: LLMs are powerful tools for generating creative and engaging text, but they can sometimes struggle to provide factual information. This is because LLMs are trained on huge amounts of text data, which may

contain inaccuracies or biases. Providing "facts" to the LLM in the input query can mitigate generative AI's hallucinations. The essence of this approach is to ensure that the most relevant facts are provided to the LLM and that the LLM's output is entirely based on those facts, while still answering the user's question and respecting system instructions and security constraints.

Searching with vector databases and relevance re-ranking: RAGs typically retrieve facts through a search, and modern search engines now leverage vector databases to efficiently retrieve relevant documents. Vector databases store documents as embeddings in a high-dimensional space, enabling fast and accurate retrieval based on semantic similarity.

Relevance, accuracy and quality: RAG helps minimize contradictions and inconsistencies in the generated text. This significantly improves the quality of the generated text and the user experience.

RAG, agents and chatbots: RAGs and anchoring can be integrated into any LLM application or agent that needs access to recent, private, or specialized data.

2.4 Process.

Retrieval-augmented generation (RAG) enhances large language models (LLMs) by incorporating an information-retrieval mechanism that allows models to access and utilize additional data beyond their original training set. AWS states, "RAG allows LLMs to retrieve relevant information from external data sources to generate more accurate and contextually relevant responses" ("indexing"). This approach reduces reliance on static datasets, which can quickly become outdated. When a user submits a query, RAG uses a document retriever to search for relevant content from available sources before incorporating the retrieved information into the model's response ("retrieval"). Ars Technica notes that "when new information becomes available, rather than having to retrain the model, all that's needed is to augment the model's external knowledge base with the updated information" ("augmentation"). By dynamically integrating relevant data, RAG enables LLMs to generate more informed and contextually grounded responses ("generation"). IBM states that "in the generative phase, the LLM draws from the augmented prompt and its internal representation of its training data to synthesize an engaging answer tailored to the user in that instant.

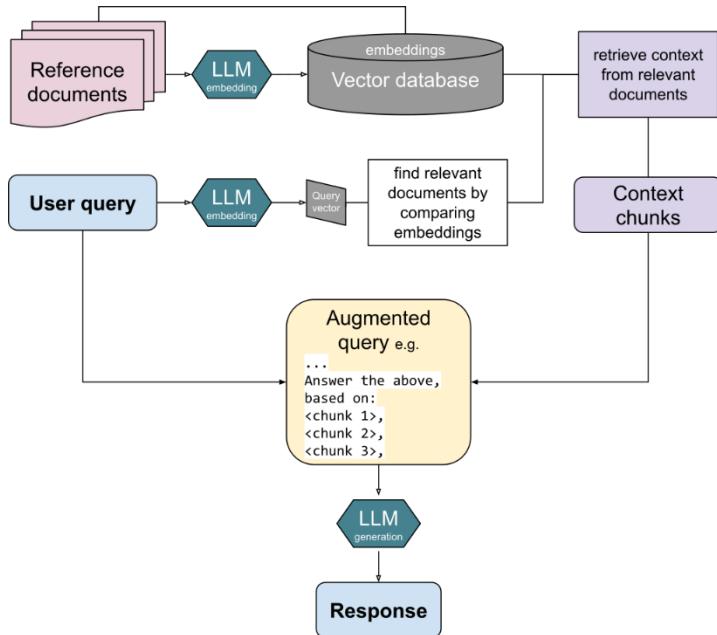


Figure 3. Overview of RAG process

RAG key stages:

- Indexing: RAG can be used on unstructured (usually text), semi-structured, or structured data (for example knowledge graphs). These embeddings are then stored in a vector database to allow for document retrieval.
- Retrieval: RAG can be used on unstructured (usually text), semi-structured, or structured data (for example knowledge graphs). These embeddings are then stored in a vector database to allow for document retrieval.
- Augmentation: The model feeds this relevant retrieved information into the LLM via prompt engineering of the user's original query.
- Generation: Finally, the LLM can generate output based on both the query and the retrieved documents. Some models incorporate extra steps to improve output, such as the re-ranking of retrieved information, context selection, and fine-tuning.

2.5 Challenges

RAG is not a complete solution to the problem of hallucinations in LLMs.

While RAG improves the accuracy of large language models (LLMs), it does not eliminate all challenges. One limitation is that while RAG reduces the need for frequent model retraining, it does not remove it entirely. Additionally, LLMs may struggle to recognize when they lack sufficient information to provide a reliable response. Without specific training, models may generate answers even when they should indicate uncertainty.

RAG systems may retrieve factually correct but misleading sources, leading to errors in interpretation. In some cases, an LLM may extract statements from a source without considering its context, resulting in an incorrect conclusion. Additionally, when faced with conflicting information, RAG models may struggle to determine which source is accurate and may combine details from multiple sources, producing responses that merge outdated and updated information in a misleading way.

2.6 RAG business value

Retrieval augmented generation changed the way businesses handle information and customer queries. By integrating the retrieval of specific information with the generative capabilities of language models, RAG provides precise, context-rich answers to complex questions. This integration brings value to businesses in several ways.

Accurate information: RAG ensures a high degree of accuracy in responses. Since the system first retrieves information from a reliable database before generating an answer, it minimizes the risk of providing incorrect or irrelevant information.

Recourse efficiency: RAG enhances the efficiency of information retrieval, saving time for both employees and customers.

Knowledge efficiency: RAG ensures that responses are matched with the most up-to-date information and relevant documentation, and businesses can maintain a high standard of information dissemination.

3. Hybrid Search

3.1 What is hybrid search

Hybrid search typically refers to a search approach that combines multiple search methodologies or technologies to provide more comprehensive and accurate results.

In the context of information retrieval, hybrid search often involves blending traditional keyword-based searching with more advanced techniques such as natural language processing (NLP), semantic search, and machine learning.

Using hybrid search to provide users with more relevant, accurate result.

3.2 How does hybrid search work

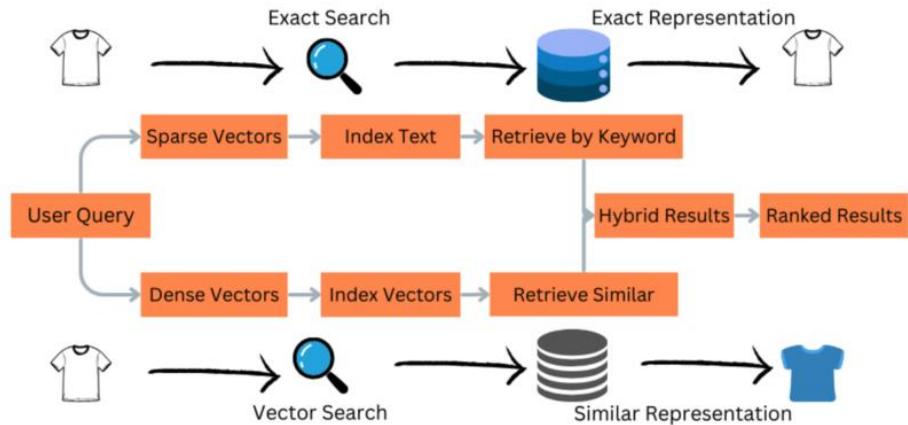


Figure 4. Hybrid Search Work

Hybrid search works by combining traditional keyword-based search (sparse vectors) with modern semantic search (dense vectors) to provide better results. Here's a detailed breakdown of how it works:

1. Keyword-Based Search (Sparse Vectors)

In traditional search engines, queries and documents are represented as sparse vectors, where each dimension corresponds to a unique term from the vocabulary. These vectors are mostly zeros, with non-zero entries only representing specific terms in the query or document. Techniques like term frequency-inverse document frequency (TF-IDF) and inverted indexing help efficiently match query keywords with documents. This method is quick and effective for finding exact matches.

2. Semantic Search (Dense Vectors)

In semantic search, both queries and documents are represented as dense vectors in a lower-dimensional space using techniques like word embeddings (e.g., Word2vec, GloVe) or contextual embeddings (e.g., BERT, GPT). Dense vectors capture the semantic meaning of words and phrases. Embedding models are trained on large corpora to understand the context and relationships between words. These models convert text into dense vectors that reflect semantic similarity.

3. Combining Sparse and Dense Vectors

In a hybrid search system, both sparse and dense vectors are generated for documents and stored in respective indices. The sparse index supports keyword-based retrieval, while the dense index supports semantic retrieval.

When a user submits a query, it's processed to generate both sparse and dense vectors. The system then searches both indices to retrieve relevant documents.

4. Retrieval and Ranking

The system retrieves an initial set of candidate documents using both the sparse index (keyword match) and the dense index (semantic match). The retrieved documents are then re-ranked based on a combination of relevance scores from both sparse and dense vectors. Machine learning models can optimize the final ranking by considering query context, user behavior, and document relevance.

3.3 Hybrid search and document retrieval model

In this study, a hybrid search model is used to improve the accuracy of retrieving medical information. The model combines three methods: vector search, BM25 keyword search, and fuzzy search.

Vector search is based on semantic similarity, which helps find documents that are related in meaning, even if the exact words are not the same. BM25 is a keyword-based search method that focuses on term frequency and relevance. It works well when users use correct and common terms. Fuzzy search helps when the user misspells a word or uses a slightly different form, which often happens in real conversations.

By combining these three methods, the system can return more accurate and flexible search results. This improves the quality of the documents retrieved, which helps the chatbot give better and more useful answers.

1. Vector search

Vector search is based on the meaning of the text rather than exact words. Each document and question is converted into a vector, which is a list of numbers representing the meaning of the sentence. These vectors are created using a language model, such as a sentence transformer.

When the user asks a question, it is turned into a vector, and then the system compares this vector with all the document vectors stored in the database. The documents that have vectors closest to the question (based on cosine similarity or another distance metric) are considered the most relevant. This method works well when the user uses different words but has the same meaning.

2. BM25 Keyword Search

BM25 is a traditional keyword search algorithm. It works by matching the exact words in the user's question with the words in the documents. It

calculates a score based on how often the keywords appear in the document and how important they are.

BM25 is useful when the user uses common terms and when it's important to find documents that mention the exact words. However, it may miss documents that are relevant but use different wording.

3. Fuzzy Search

Fuzzy search is used to handle spelling mistakes or small differences in wording. Instead of looking for an exact match, it looks for similar words using techniques like edit distance (Levenshtein distance). For example, if the user types “diabtes” instead of “diabetes”, fuzzy search can still find the correct document.

This is helpful because users often make typos, especially when using chat interfaces on phones or when they are unsure how to spell medical terms.

4. ChatGPT – The Generation Component



Figure 5. Chat GPT

In this study, ChatGPT is used as the main language model to generate answers for user questions. After the system retrieves related documents using hybrid search, these documents are passed to ChatGPT along with the original question. The model then reads the input and creates a full response that sounds natural and easy to understand.

ChatGPT is part of the GPT (Generative Pre-trained Transformer) series developed by OpenAI. It is trained on large amounts of text data and can produce human-like language. The version used in this project is accessed through the OpenAI API, which allows the system to call the model and get responses in real-time.

The reason for choosing ChatGPT is that it can understand context, combine information from multiple sources, and generate smooth and friendly replies. This makes it very suitable for a medical chatbot, where users expect answers that are not only correct, but also clear and helpful.

I use chatgpt 3.5 model to process and select the correct answer and process it into a more natural answer from the top k search from hybrid search.

Why GPT-3.5 instead of GPT-4?

Criteria	ChatGPT-3.5	ChatGPT-4
Accuracy	Performs well on general questions May hallucinate without retrieval support	More accurate and reliable Better reasoning, fewer hallucinations
Response Speed	Fast, near real-time Suitable for interactive chatbot applications	Slower, especially with long prompts or many documents
Cost(OpenAI API)	\$0.0005 / 1K input tokens \$0.0015 / 1K output tokens	\$0.03 / 1K input tokens \$0.06 / 1K output tokens (≈10x more expensive)
RAG Compatibility	Good with proper retrieval Needs clear context to avoid confusion	Excellent document reasoning More consistent with longer or complex inputs
Context Window	Up to 16K tokens	Up to 128K tokens (GPT-4-turbo)

Table 1. compare 2 model

Although the ChatGPT-4 model has higher accuracy, the 3.5 model is still more suitable with a very low cost, and with a fast retrieval time, it will be easier to provide to users. Moreover, hybrid search has selected the top-k most accurate answers, so the rest of chatGPT in this part is quite simple, without the need for a very strong model.

5. User Memory in Conversational AI.

To make a chatbot more natural and intelligent, user memory is an important component. It allows the system to remember what the user has said earlier, follow the conversation flow, and give better answers. This section introduces different types of memory in AI chat systems, describes the memory design used in this project, and explains how ChatGPT uses that memory to respond more effectively.

5.1 Overview of memory types in AI chat systems.

There are several types of memory used in modern chatbot and conversational AI systems. Each type plays a different role depending on how the chatbot is expected to interact with the user.

- **Short-term memory:** *This memory stores the current conversation context. It helps the chatbot understand follow-up questions without needing the user to repeat information.*
Example: Most AI chatbots, including ChatGPT (free version), Bing Chat, or Bard, rely on short-term memory during a single session.
- **Long-term memory:** *This memory stores information across sessions, such as the user's name, preferences, or past problems. It allows the chatbot to personalize future interactions.*
Example: Custom GPTs with memory enabled (ChatGPT Plus), Meta AI Assistant, and Claude AI support long-term memory features.
- **Working memory:** *A temporary memory used during one complex interaction. It stores intermediate details like symptoms listed in a single input.*
Example: Diagnostic health bots like Ada Health or Babylon Health use working memory to track multiple symptoms in one session.
- **Vector memory (semantic memory):** *Information is stored as embeddings (vectors) instead of plain text, allowing retrieval based on meaning rather than exact keywords.*
Example: Retrieval-Augmented Generation (RAG)-based systems, such as Perplexity AI or custom LangChain chatbots, use vector memory to retrieve relevant documents.

5.2 Memory implementation in this project

In this project, a short-term memory system is used. It stores recent user inputs, especially useful information like disease names, symptoms, or previous questions. This memory is stored temporarily during one conversation session.

The memory is updated after each user message, and before generating the next answer, the memory is checked and added to the prompt. For example, if a user said “I have a headache and fever” earlier, and then asks “is that dangerous?”, the system will know what “that” refers to.

This method helps avoid repeated questions and makes the chatbot feel more intelligent and connected.

5.3 Using ChatGPT to process memory and generate context-aware responses

After the memory is stored and updated, it must be passed to the language model in a way that helps the chatbot respond naturally and accurately. In this system, ChatGPT (specifically GPT-3.5) is used to process the stored memory along with the user's current message. The memory is added to the prompt, so the model can use past information to understand the context and avoid repeating questions or losing track of the topic.

However, ChatGPT-3.5 has limitations in how much context it can handle (up to 16K tokens), and it sometimes struggles when the memory grows too long or includes too many details.

To improve this, **ChatGPT-4 or GPT-4-turbo** can be used in the future. GPT-4 supports a much larger context window (up to 128K tokens), which means it can remember more of the conversation, use memory more effectively, and maintain coherence over longer sessions. It also performs better at understanding subtle meanings and connecting past and present user inputs.

By using GPT-4 with user memory, the chatbot could behave more like a real assistant—remembering what the user said, following up naturally, and making the conversation feel smoother and more intelligent.

6. Summary of chapter 2.

This chapter presented the theoretical foundations used to build the medical chatbot system. First, it introduced Large Language Models (LLMs), which are the core technology that allows the chatbot to understand and generate natural language. Next, the concept of Retrieval-Augmented Generation (RAG) was explained, showing how it helps combine external knowledge with the model's own capabilities.

To improve the retrieval process, a hybrid search method was applied, combining vector search, BM25, and fuzzy search. Each technique was explained in detail along with how they work together to find the most relevant documents.

Then, the chapter discussed the role of ChatGPT as the generation component in the system. ChatGPT receives both the question and the retrieved documents to produce a natural, helpful response. Although GPT-3.5 was used in this project, future upgrades to GPT-4 were also considered for better accuracy and memory handling.

Finally, the concept of user memory was presented. This helps the chatbot remember what the user said earlier and respond with better context awareness. Several types of memory were introduced, and the implementation used in this system focused on short-term memory during a single session.

Overall, this chapter laid the foundation for building an intelligent and user-friendly medical chatbot system by combining modern AI techniques and tools effectively.

CHAPTER 3: SYSTEM DESIGN AND IMPLEMENTATION

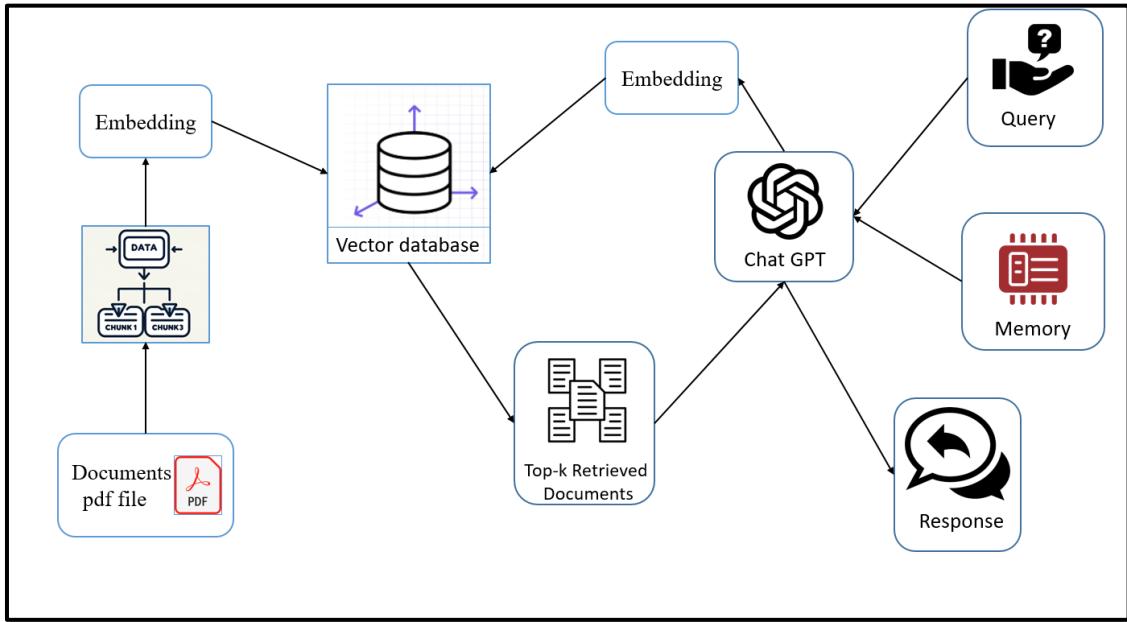


Figure 6. System Architecture of Medical Chatbot

The overall architecture of the chatbot system is illustrated in **Figure 6**. It shows the process of transforming medical documents into a searchable knowledge base and how the chatbot interacts with user queries using retrieval-augmented generation (RAG) and memory.

The system is divided into two main phases:

- **Data Preparation and Indexing Phase**

Medical documents in PDF format are first chunked and converted into vector embeddings. These embeddings are stored in a vector database, which will later be used for semantic search.

- **Query Processing and Response Generation Phase**

When a user submits a question, the system embeds the query and searches the vector database to retrieve the most relevant document chunks. These documents, along with the user's query and memory (if available), are passed to the ChatGPT model to generate a natural and accurate response.

Each component in the architecture plays a specific role in ensuring that the chatbot delivers relevant, trustworthy, and human-like answers.

1. Document Collection And Processing

In this research, the first step was to collect and organize the medical data. A total of **65 respiratory diseases** were selected as the knowledge base for the

chatbot. All disease information was collected from the official website of **Long Chau Pharmacy**, a well-known pharmacy chain with many branches across Vietnam. The source is considered reliable and accurate in terms of health-related content. All of the documents were written in Vietnamese.

Giãn phế quản	Viêm xoang trán	Viêm xoang sàng	Cơn hen phế quản	Hội chứng suy hô hấp cấp
Viêm phổi mãn tính	Cúm mùa	Viêm xoang sàng sau	Viêm phổi do Metapneumovirus	Viêm họng hạt
Cúm A	Viêm mũi vận mạch	Xơ phổi vô căn (Idiopathic Pulmonary Fibrosis - IPF)	Viêm phổi tăng bạch cầu ái toan	Viêm phổi Pneumocystis jirovecii
Hen suyễn	Xơ phổi	Viêm phổi do virus	Viêm phổi do tụ cầu	Viêm phổi do nấm
Viêm phổi do Mycoplasma pneumonia (MP)	Viêm phổi	Viêm màng phổi	Viêm phế quản mãn tính	Tràn khí màng phổi
Tràn dịch màng phổi	Phổi tắc nghẽn mãn tính	Ngưng thở khi ngủ do tắc nghẽn	Hen phế quản	Dị vật đường thở
Bệnh phổi kẽ	Viêm phế quản cấp	Lao phổi	Lao kê	Viêm mũi mãn tính
Viêm xoang hàm	Viêm mũi teo	Viêm xoang mãn tính	Viêm xoang	Viêm mũi dị ứng
Viêm mũi	Viêm tai giữa úr dịch	Viêm amidan xơ teo	Viêm họng	Viêm amidan mãn tính
Viêm họng hạt mãn tính	Ung thư hầu họng	Viêm tuyến nước bọt	Viêm tai xương chũm	Viêm tai giữa thủng màng nhĩ

Ung thư họng	Suy giảm thính lực	Viêm thanh quản mãn tính	Viêm thanh quản cấp	Ù tai
Viêm tai ngoài ác tính	Viêm tai ngoài	Viêm tai giữa	Viêm họng mãn tính	Viêm họng do liên cầu khuẩn
Viêm họng cấp	Viêm amidan	Vẹo vách ngăn mũi	Thủng màng nhĩ	Polyp mũi

Table 2. 65 disease

Each disease was saved in a separate PDF file, and the file was named after the corresponding disease for easy management. To support the embedding and retrieval process, each document was manually split into smaller **chunks**, based on the main content sections of the disease. These include:

- Definition / Concept
- Causes
- Symptoms
- Treatment methods
- Notes or special considerations

This structured format helps the system process and search through the information more efficiently. It also ensures that the chatbot can return concise, relevant answers based on specific parts of each disease.

The processed and chunked files were then used as the input for the embedding phase in the next step of the system.

2. Embedding Documents And Building The Vector Database

2.1 Embedding

After the documents were collected and split into meaningful chunks, the next step was to convert them into vector representations using an embedding model. This process allows the system to understand the semantic meaning of each chunk and enables effective search and retrieval later.

In this project, the embedding step was performed using a Vietnamese-compatible model to ensure accurate representation of the content, as all documents are in Vietnamese. The output of this step is a high-dimensional vector for each chunk, capturing its context and meaning in numerical form.

These vectors were then stored in a vector database, which supports fast similarity search. When a user submits a query, it will also be embedded using the

same model. The system can then compare the query vector with the document vectors to find the most relevant pieces of information.

This embedding step is critical in bridging the gap between the user's question and the right answer, especially when the wording of the query is different from how the information appears in the documents.

Some models to perform this step:

Model Name	Language Support	Accuracy (Semantic Similarity)	Context Awareness	Speed	Pre-trained Size	Suitability for Vietnamese
all-MiniLM-L6-v2	Multilingual (limited)	Medium	Medium	Fast	~80MB	Acceptable
distiluse-base-multilingual-cased-v1	Multilingual	Medium – High	Medium	Medium	~120 MB	Good
sentence-transformers/multi-qa-mpnet-base-dot-v1	Multilingual (QA-tuned)	High	High	Slower than MiniLM	~420 MB	Excellent
intfloat/multilingual-e5-base	Multilingual	High	Medium – High	Medium	~220 MB	Good
bge-m3 (BAAI General Embedding)	Multilingual (new)	High	High	Medium	~500 MB	Limited resources

Table 3. comparison between models

Among various pre-trained embedding models, I chose sentence-transformers/multi-qa-mpnet-base-dot-v1 for the following reasons:

- ***High Semantic Accuracy:***

This model is fine-tuned specifically for question–answer retrieval tasks, which fits perfectly with the chatbot's goal of answering medical questions based on context.

- ***Multilingual Capability:***

Although not specifically fine-tuned for Vietnamese, it performs well on non-English input due to its multilingual training. My tests showed that it understood Vietnamese content better than other models like MiniLM.

- **Compatibility with LangChain and FAISS:**

The model is supported natively through HuggingFace and LangChain, making integration simple and stable.

- **Performance Tradeoff is Acceptable:**

While it is slightly slower than lighter models (e.g., MiniLM), the improved accuracy and context-awareness justify the cost.

- **Proven in Research and Industry:**

This model is widely used in real-world RAG pipelines, academic work, and production-level retrieval systems.

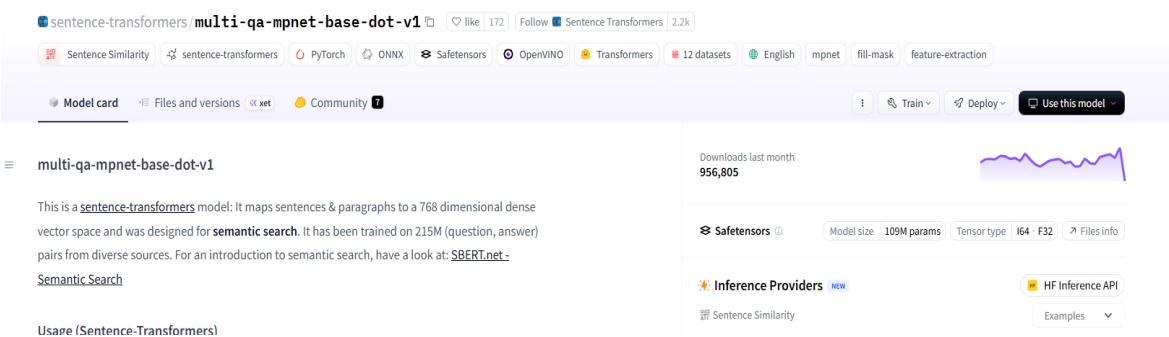


Figure 7. *sentence-transformers/multi-qa-mpnet-base-dot-v1*

There are quite a few people downloading the model every month, although this model is quite heavy and not suitable for study, so in general there are quite a few people using this model for research and development.

2.2 Building the vector database

Once the text chunks were converted into vector embeddings, the next step was to store them in a vector database to support efficient semantic retrieval. In this project, I used FAISS (Facebook AI Similarity Search) – a powerful open-source library developed by Meta AI, specifically designed for fast similarity search in high-dimensional spaces.

FAISS supports approximate nearest neighbor (ANN) search, which significantly improves the performance of retrieval tasks, especially when dealing with thousands of embedded text chunks. Each vector in the database is linked to metadata, such as the disease name, to maintain traceability.

To build the vector database:

- All the embedding vectors were added to a FAISS index.

- Cosine similarity was used as the distance metric for finding the most relevant chunks during retrieval.
- The final FAISS index was saved locally at:

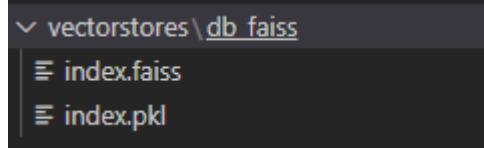


Figure 8 local final FAISS

This vector store serves as the core component of the retrieval pipeline in the RAG-based chatbot system. When a user submits a query, it is embedded into a vector and compared against this FAISS index to retrieve the top-k most semantically similar chunks, which are later used to generate an accurate and context-aware response.

3. Hybrid Search Retrieval

To improve retrieval accuracy and handle diverse user input styles, the system implements a hybrid search architecture that integrates multiple retrieval techniques: vector search (semantic), BM25 (lexical), and fuzzy matching (approximate string matching). After gathering results from all three sources, a re-ranking model based on cross-encoder architecture is applied to prioritize the most relevant chunks.

3.1. Vector Search (Semantic Search using FAISS)

The system first performs semantic retrieval using the *FAISS vector database*, which returns the top-k text chunks whose embeddings are most similar to the query embedding (based on cosine similarity). This method is highly effective in capturing *semantic meaning*, especially when the query is expressed in different wording than the source document.

```
vector_docs = faiss_db.similarity_search(query, k=top_k)
vector_scores = {doc.page_content: (doc, top_k - i) for i, doc in enumerate(vector_docs)}
```

Figure. 9 vector search

3.2. BM25 Search (Lexical Matching)

BM25 is a traditional ranking function based on *term frequency-inverse document frequency (TF-IDF)*. It is used here to score documents by exact word overlap with the user query. This is particularly useful when the query contains *specific medical terminology* that exactly matches the content in documents.

```
# BM25 Search
bm25_scores = bm25.get_scores(query.lower().split())
bm25_top_idx = np.argsort(bm25_scores)[::-1][:top_k]
bm25_results = [(corpus[i], bm25_scores[i]) for i in bm25_top_idx]
```

Figure. 10 BM25 Search

3.3. Fuzzy Search (Approximate Matching)

Fuzzy matching helps detect and handle minor typos, misspellings, or different word forms, which is common in user input, especially in Vietnamese. Tools like *RapidFuzz* or *fuzzywuzzy* allow the system to match queries against document text even when they are not spelled identically. This enhances fault tolerance in the retrieval process.

```
# Fuzzy Search
fuzzy_scores = [fuzz.token_sort_ratio(query, doc.page_content) for doc in corpus]
fuzzy_top_idx = np.argsort(fuzzy_scores)[::-1][:top_k]
fuzzy_results = [(corpus[i], fuzzy_scores[i]) for i in fuzzy_top_idx]
```

Figure 11. Fuzzy Search

3.4. Re-ranking with Cross-Encoder

Once results are retrieved from all three methods, they are merged into a candidate list. To refine the ranking and improve final answer quality, the system uses a cross-encoder re-ranker based on the model:

```
# Re-ranking with Cross Encoder
print("\n re-ranking result ...")
reranker = CrossEncoder('cross-encoder/ms-marco-electra-base')
```

Figure. 12 Re-ranking

This model, pre-trained and fine-tuned on the MS MARCO dataset, evaluates each pair (query, chunk) together, scoring them based on semantic relevance. Unlike bi-encoder models (which encode query and chunk separately), cross-encoders process them jointly, allowing deeper interaction between query and context.

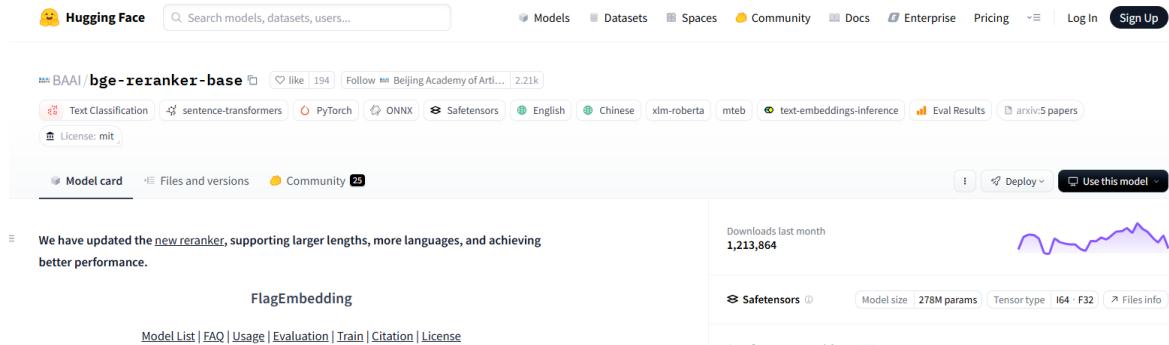


Figure 13. BAAI/bge-reranker-base

This is also one of the re-rank models used by many people today.

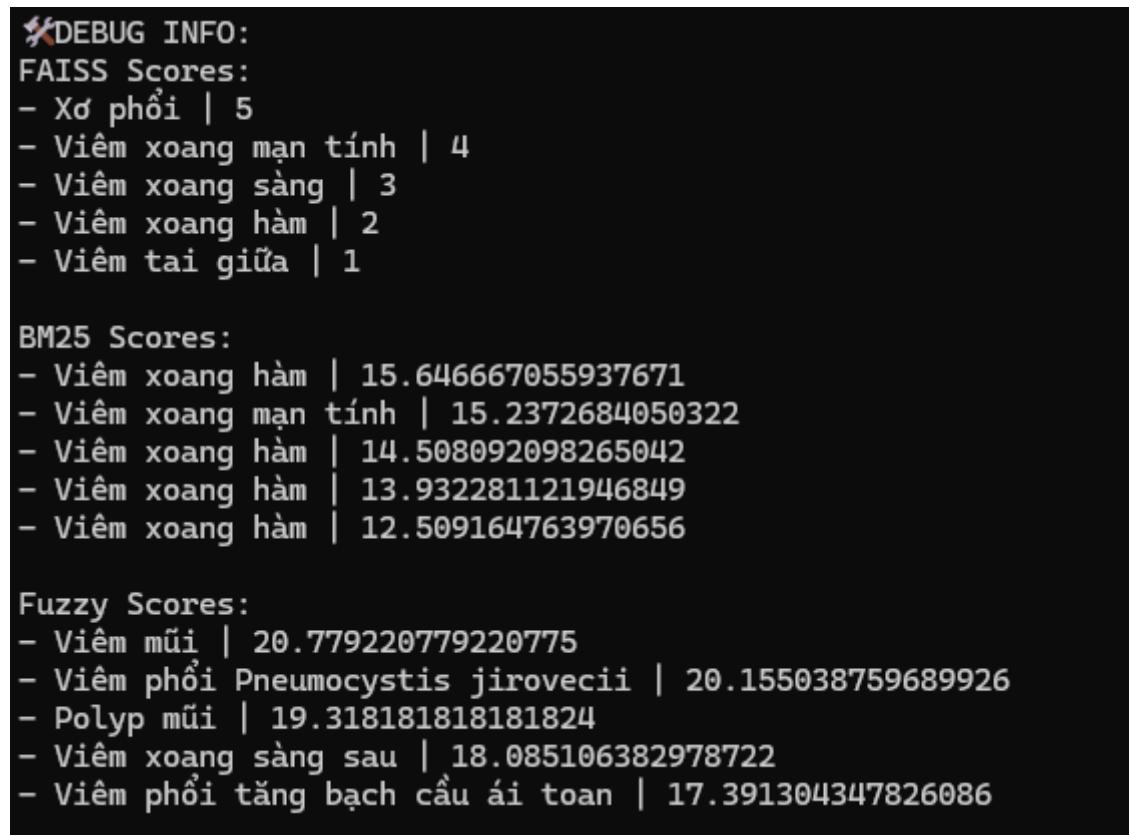


Figure. 14 Scores

4. Response Generation And Conversation Flow

After the document retrieval system (Hybrid Search Retrieval) was successfully built, we were able to retrieve relevant and accurate text passages from the knowledge base. However, these text passages were still raw data, dry and descriptive, and not really friendly to end users.

To improve the user experience, the next step was to integrate a large language model - specifically OpenAI ChatGPT 3.5 - to process and present the content in a more natural, clear and understandable way. ChatGPT is able to

synthesize information from the best retrieved passages (top-k) and generate a seamless, contextual answer, close to human communication.

The model will not guess or infer, but only use the content found from the Hybrid Search system. This helps ensure that the information provided remains accurate and reliable – an extremely important factor in the medical field. At the same time, using ChatGPT helps to increase the flexibility and friendliness of the interactive interface, making users feel like they are chatting with a real expert.

4.1 Mechanism of Response Generation

After receiving the results from the Hybrid Search system, including the Top-k most relevant text segments, the system will use the OpenAI ChatGPT 3.5 large language model to generate a complete answer. The working mechanism is described as follows:

- Prepare the input context:

The text segments obtained from Hybrid Search will be synthesized into a single text string - called context. This is the basis for ChatGPT to generate answers.

- Build the input prompt:

The prompt is tightly designed to ensure that the model only answers based on the information that has been found, avoiding speculation or giving false information. An example of a prompt is as follows:

```
prompt = f"""Bạn là trợ lý y tế. Hãy trả lời đầy đủ thông tin câu hỏi của người dùng dựa trên các đoạn thông tin bên dưới và memory.
```

Chỉ sử dụng thông tin có trong các đoạn, nếu có thì hãy trả lời đầy đủ thông tin của đoạn mà bạn tham khảo. Không được tự suy diễn hoặc đoán nếu thông tin không tồn tại trong đoạn.

Nếu không tìm thấy thông tin cần thiết để trả lời, hãy trả lời:
Tôi xin lỗi, tôi chưa có dữ liệu cho loại thông tin này.

Các đoạn thông tin:

{context}

Câu hỏi:

```
{query}
```

```
### Trả lời bằng tiếng Việt:
```

```
"""
```

- Send a request to OpenAI's API:

```
response = client.chat.completions.create(  
    model=model,  
    messages=[{"role": "user", "content": prompt}],  
    temperature=0.2,  
    max_tokens=1000  
)  
  
return response.choices[0].message.content.strip()
```

Figure 15. send request to OpenAI

- Return results to the user interface:

The generated answer will be displayed to the user in a conversational format, ensuring it is easy to understand and natural.

Example:

```
Trả lời:  
Viêm xoang hàm là tình trạng viêm nhiễm của các hốc xoang nằm ở hai bên gò má và quanh mắt. Bệnh này có thể xảy ra ở bất cứ ai và có ba dạng chính là viêm xoang hàm cấp tính, viêm xoang hàm mạn tính và viêm xoang hàm do những bệnh về răng.
```

Figure 16. result

This is a processed answer, rather than a complex, user-unfriendly piece of conceptual data.

4.2 Answer Generation Pipeline

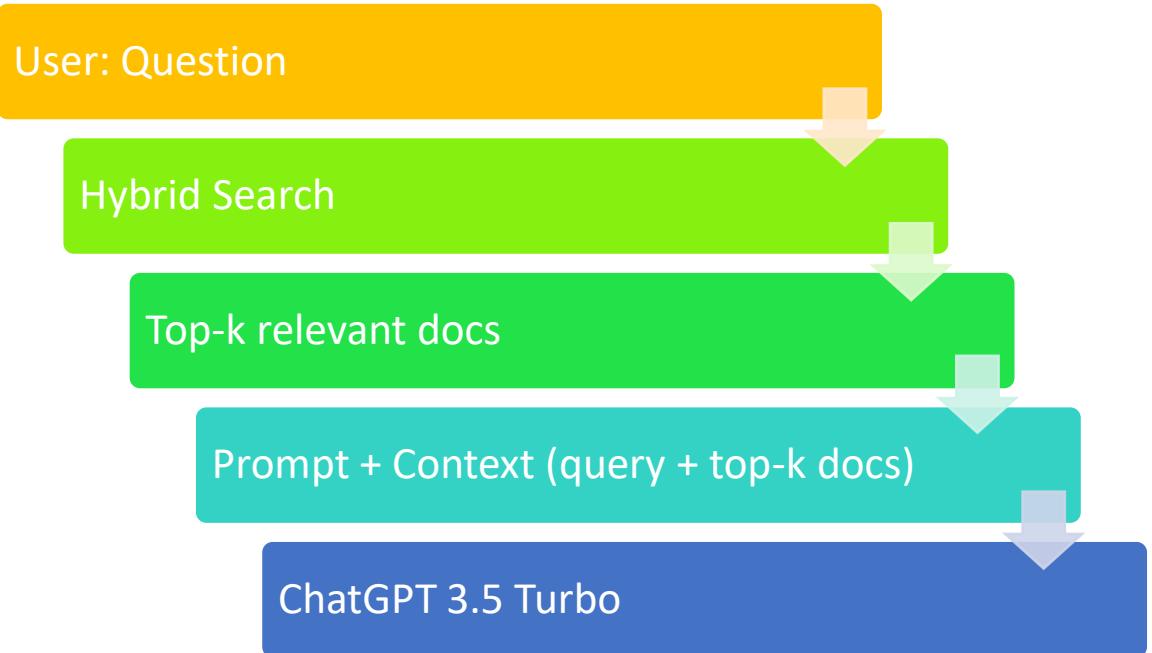


Figure 17. Process flow from Hybrid Search to answer generation

This is a diagram of the steps the model takes from processing data from a user question to generating a natural answer.

5. Memory Integration

In order to develop a medical chatbot capable of interacting with users in a friendly and natural manner, the integration of a memory function is essential. This feature allows the system to retain previous conversation turns, understand the context of user queries, and eliminate the need to repeat previously provided information.

5.1 How to deploy

During the conversation between the user and the chatbot, a variable named *memory* is used to store the most recent dialogue turns. When the user submits a new question, OpenAI processes not only the current query but also the stored conversation history in the *memory* variable. This enables the chatbot to understand the actual intent behind the user's message and generate the most accurate and contextually appropriate response.



Figure 18. process of memory

In this implementation, OpenAI was once again utilized to process memory content before performing hybrid search. Without incorporating memory, hybrid search based on user queries alone may fail to retrieve relevant documents, especially when the questions rely heavily on previously mentioned context. However, if unprocessed memory is used directly, it may introduce noise by including irrelevant or outdated context from prior dialogue turns—potentially leading to incorrect or misleading search results.

While there are alternative memory handling approaches that do not rely on OpenAI (thus saving cost), these methods often come with significant limitations. For instance, one common technique is to store and retrieve memory based solely on disease names. This becomes problematic when the user suddenly switches topics or asks about the symptoms of a different illness, which may not be correctly detected by such a rule-based system.

Therefore, integrating OpenAI for memory summarization and context management helps ensure accurate interpretation of user intent while minimizing logical errors, making the conversation more consistent and safe—especially for sensitive domains like healthcare.

CHAPTER 4 FOCUSES ON THE IMPLEMENTATION AND EVALUATION OF THE PROPOSED MEDICAL CHATBOT MODEL.

1. Implementation Process

The implementation process of the chatbot system involves two main stages:

1.1 Embedding medical documents

```
Đang xử lý: Viêm xoang hàm.pdf
Đã tách được 8 đoạn văn bản từ Viêm xoang hàm.pdf
Đang xử lý: Viêm xoang mạn tính.pdf
Đã tách được 9 đoạn văn bản từ Viêm xoang mạn tính.pdf
Đang xử lý: Viêm xoang sàng sau.pdf
Đã tách được 10 đoạn văn bản từ Viêm xoang sàng sau.pdf
Đang xử lý: Viêm xoang sàng.pdf
Đã tách được 6 đoạn văn bản từ Viêm xoang sàng.pdf
Đang xử lý: Viêm xoang trán.pdf
Đã tách được 6 đoạn văn bản từ Viêm xoang trán.pdf
Đang xử lý: Viêm xoang.pdf
Đã tách được 9 đoạn văn bản từ Viêm xoang.pdf
Đang xử lý: Vẹo vách ngăn mũi.pdf
Đã tách được 10 đoạn văn bản từ Vẹo vách ngăn mũi.pdf
Đang xử lý: Xơ phổi vô căn.pdf
Đã tách được 8 đoạn văn bản từ Xơ phổi vô căn.pdf
Đang xử lý: Xơ phổi.pdf
Đã tách được 9 đoạn văn bản từ Xơ phổi.pdf
Đang xử lý: Ù tai.pdf
Đã tách được 8 đoạn văn bản từ Ù tai.pdf
Tổng cộng đã tách được 556 đoạn văn bản.
Kết quả đã được lưu vào 'output_chunks.txt' để kiểm tra.
Đang khởi tạo mô hình embeddings chất lượng cao...
Đang xây dựng FAISS index...
Đang lưu FAISS index vào đường dẫn:
Đã lưu FAISS index vào 'vectorstores/db_faiss' thành công.
PS D:\luận_văn\RAG-Vectorsearch> █
```

Figure 19. run file prepare_vector_db

All preprocessed medical documents (in PDF format) were split into chunks and embedded using the multi-qa-mpnet-base-dot-v1 model from Hugging Face. These vector representations were then stored in a FAISS vector database to support fast and accurate semantic search.

Setting	Value
Dimensions	768
Produces normalized embeddings	No
Pooling-Method	CLS pooling
Suitable score functions	dot-product (e.g. <code>util.dot_score</code>)

Figure 20. Technical Details

At this stage, the system transforms the medical text data into vector representations with a dimensionality of 768. These embeddings are generated using a pre-trained language model (*multi-qa-mpnet-base-dot-v1*) and serve as the foundation for the subsequent document retrieval process.

With the disease data already collected and divided into semantic chunks (such as symptoms, causes, treatment, etc.), each chunk is embedded along with a metadata tag that stores the name of the corresponding disease. This metadata helps prevent confusion between similar content across different diseases, ensuring that retrieved results are relevant to the correct condition. It is a compact yet highly effective strategy for organizing and retrieving medical information.

1.2 Chatbot execution

```

PS D:\luận_văn\RAG-Vectorsearch> python venv/hybrid_search_bot.py
Đang khởi tạo mô hình embeddings chất lượng cao...
Đang load FAISS index từ local...
Đã load xong vector database.
Hybrid Search Bot sẵn sàng. Gõ 'exit' để thoát.

Bạn: |

```

Figure 21. Chatbot execution

In the context of this research, the chatbot system is executed through a terminal-based interface. After launching the chatbot, users can freely input questions related to any of the 65 respiratory diseases in the dataset. If the query falls within the scope of the available knowledge base, the chatbot retrieves

relevant information and generates an accurate, context-aware response.

Conversely, if the question concerns a topic not included in the dataset, the system responds appropriately by notifying the user that the requested information is not available, rather than generating incorrect or assumed content. This ensures the reliability and safety of the chatbot, particularly in the sensitive domain of healthcare.

```
Bạn: viêm xoang hàm là gì

DEBUG INFO:
FAISS Scores:
- Viêm xoang mạn tính | 5
- Viêm xoang sàng | 4
- Xơ phổi | 3
- Viêm xoang hàm | 2
- Viêm xoang | 1

BM25 Scores:
- Viêm xoang hàm | 15.646667055937671
- Viêm xoang hàm | 14.508092098265042
- Viêm xoang hàm | 13.932281121946849
- Viêm xoang mạn tính | 12.623663600451513
- Viêm xoang hàm | 12.509164763970656

Fuzzy Scores:
- Viêm mũi | 20.915032679738566
- Viêm phổi Pneumocystis jirovecii | 20.3125
- Polyp mũi | 19.428571428571427
- Viêm xoang sàng sau | 18.1818181818176
- Viêm phổi tăng bạch cầu ái toan | 17.51824817518248

re-ranking result ...

Đang tạo câu trả lời từ GPT...

Trả lời:
Viêm xoang hàm là tình trạng viêm nhiễm của xoang hàm, các hốc xoang nằm ở hai bên gò má và quanh mắt. Viêm xoang hàm có thể là viêm xoang hàm cấp tính, viêm xoang hàm mạn tính hoặc viêm xoang hàm do những bệnh về răng.
```

Figure 22. answer of chatbot

This is the raw interface and how the chatbot responds, it can be integrated into many app or web interfaces to make it look more beautiful, for example flutter or node js.

2. Evaluation Of Chatbot Performance

To evaluate the effectiveness of the proposed medical chatbot system, a set of 100 diverse questions related to the 65 respiratory diseases was prepared. These questions cover various types of information such as definitions, symptoms, causes, treatment methods, and prevention strategies.

Each question was tested under the same conditions using the developed chatbot, and the responses were manually reviewed to determine their correctness based on three main criteria: Accuracy, Avg. Latency, Cost per 100 queries (USD)*.

2.1 Evaluation of Chatbot Performance using GPT-4.

The above section presents statistics on the accuracy and response time of the model. During testing, I employed GPT-4, the most advanced model currently available, to optimize the accuracy of responses. However, the results show that 100% accuracy is not achievable. According to the evaluation, the chatbot's accuracy ranged from 80% to 90%, primarily due to its safety mechanism: when relevant information is not found in the knowledge base, the bot will respond with "Sorry, this data has not been updated yet," instead of making incorrect assumptions. This design ensures reliability and safety, even if some queries cannot be fully answered.

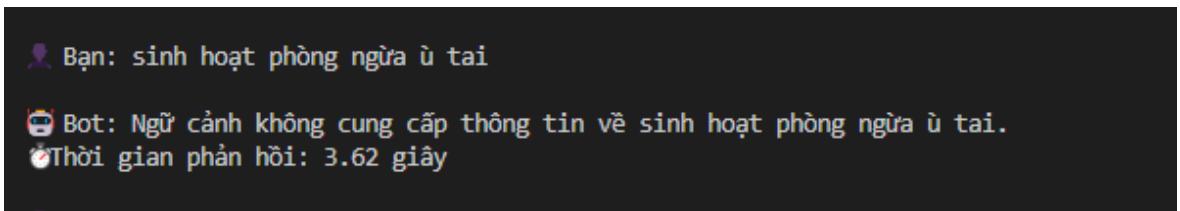


Figure 23. Test model gpt-4.

In terms of performance, the average response time per query ranged from 2 to 5 seconds. This is a very good speed for a medical chatbot powered by a large language model and enhanced with Retrieval-Augmented Generation (RAG) search mechanisms.

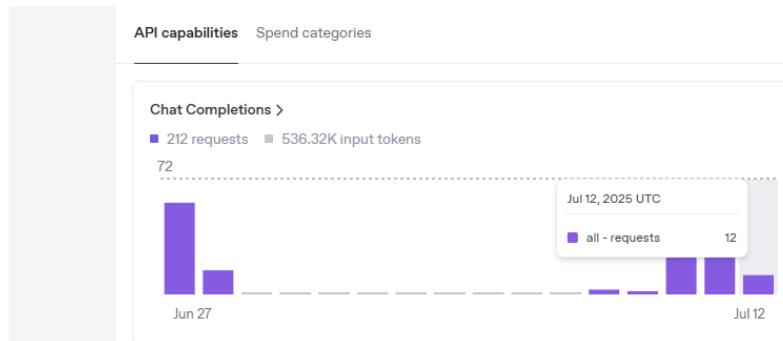


Figure 24. number of requests

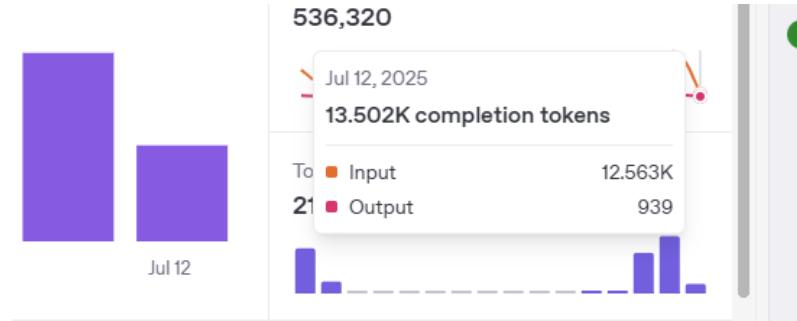


Figure 25. Number of token

Regarding cost, GPT-4 (standard) pricing is based on the number of tokens:

- \$0.03 per 1,000 input tokens
- \$0.06 per 1,000 output tokens

During the testing phase with *12 chat sessions*, the total input tokens reached *12,563 tokens*, and output tokens were *939 tokens*. The estimated cost is as follows:

- *Input cost: $(12,563 / 1,000) \times \$0.03 \approx \$0.377$*
- *Output cost: $(939 / 1,000) \times \$0.06 \approx \0.056*
- *Total cost: $\approx \$0.433$*

With a cost of *\$0.433 for 12 interactions*, the model can be considered expensive compared to lightweight chatbot alternatives. However, this cost is justified by a highly accurate, safe, and user-friendly medical chatbot system, well-aligned with the goals of this research.

In addition, this model requires an embedding step for all documents before retrieval. This step incurs additional costs, which depend on the total volume of textual data. The more data included in the knowledge base, the higher the cost. In this project, the embedding process uses OpenAI's *text-embedding-ada-002* model, which is priced at \$0.0001 per 1,000 tokens.

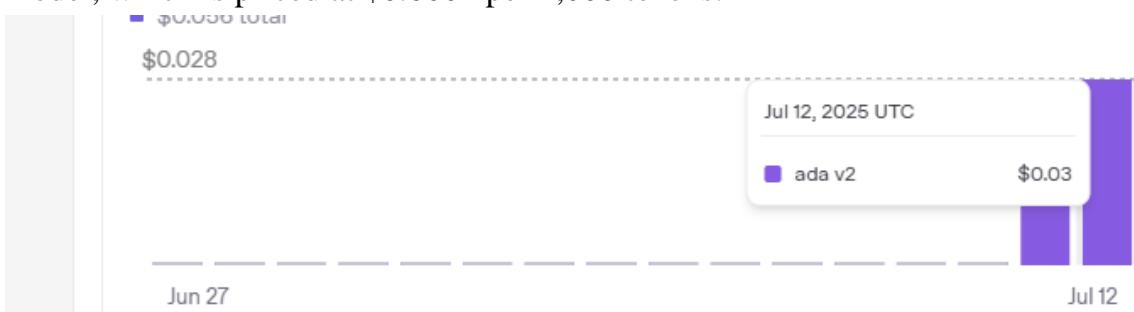


Figure 26. cost of ada model

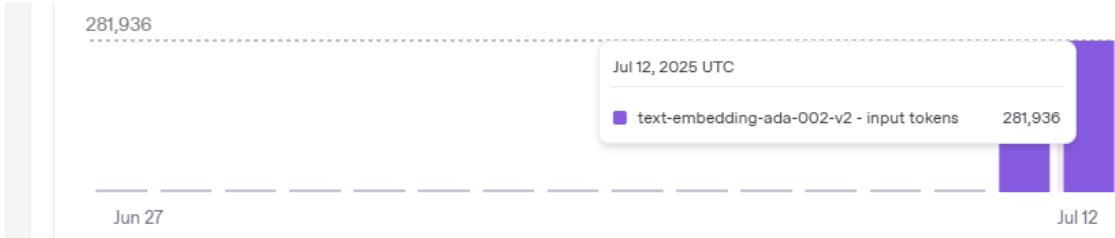


Figure 27. Number of token(ada)

For this experiment, a total of 281,936 *tokens* were embedded, resulting in a cost of approximately \$0.028, or nearly \$0.03. While this may seem minimal, it's important to note that the current dataset only covers 65 *respiratory diseases*, and only *one-time embedding* was performed.

In real-world applications, as the number of diseases increases and document updates or re-embedding are required, this cost can scale significantly. Therefore, although effective, the RAG model—particularly when combined with GPT-4 and external embedding models—may become costly for large-scale or long-term deployment, especially in medical domains where data constantly evolves.

2.2. Evaluation Using Local LLM Model

In this test, the RAG system was evaluated using a local large language model, specifically the Mistral-7B-Instruct (quantized version: *mistral-7b-instruct-v0.1.Q4_K_M.gguf*), running on a CPU-based machine.

Despite selecting one of the most efficient and lightweight models available for local inference, the test results highlight several limitations:

- Very long response time: On the same local machine setup, the model's response time exceeded 200 seconds per query, which severely affects user experience. While better hardware could reduce this time, our comparison is based on equal conditions.
- Poor accuracy: The model struggled with Vietnamese input, leading to language confusion and significantly affecting the correctness of the answers.

As a result, this local model is not suitable for use in a real-world medical chatbot system. Especially when considering deployment at scale for multiple users, the need for a powerful server infrastructure would add considerable costs.

In conclusion, while the use of a local LLM model avoids API costs and offers offline capability, it is not feasible for building an intelligent and responsive medical RAG chatbot in practical applications.

Bạn: Cúm A là gì?
llama.print_timings: load time = 73077.68 ms
llama.print_timings: sample time = 133.85 ms / 313 runs (0.43 ms per token, 2338.46 tokens per second)
llama.print_timings: prompt eval time = 202489.22 ms / 1225 tokens (165.23 ms per token, 6.05 tokens per second)
llama.print_timings: eval time = 81984.31 ms / 312 runs (262.51 ms per token, 3.81 tokens per second)
llama.print_timings: total time = 286446.84 ms / 1537 tokens

Bot: Cúm A là một loại viroid được phát hiện tầm những chéng trữ cao với các bệnh lý về đường hô hấp do mắc. Cúm A có thể gây ra một số biến chứng như: nhiễm trùng nghiêm trọng, buồn nôn, đau bụng, đau ngực, nôn mửa, chóng mặt. Các yếu tố làm tăng nguy cơ mắc cúm mùa có liên quan đến việc lây truyền. Việc lây truyền cúm A có thể dễ dàng với nhiễm giùi như trẻ em dưới 5 tuổi, mắc bệnh phổi tắc nghẽn mạn tính, mang thai hoặc các bệnh chronic nặng khi nhiễm nghiêm trọng.

† Thời gian phản hồi: 286.71 giây

Bạn: exit
Bot: Tạm biệt!
PS D:\Test\VRAG.VectorSearch\project]

Figure 28. model local LLM

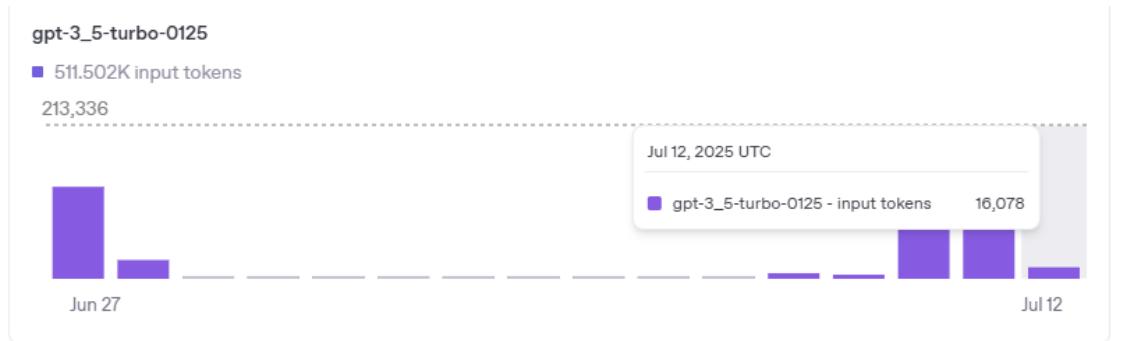


Figure 29. number of tokens input(gpt-3.5)

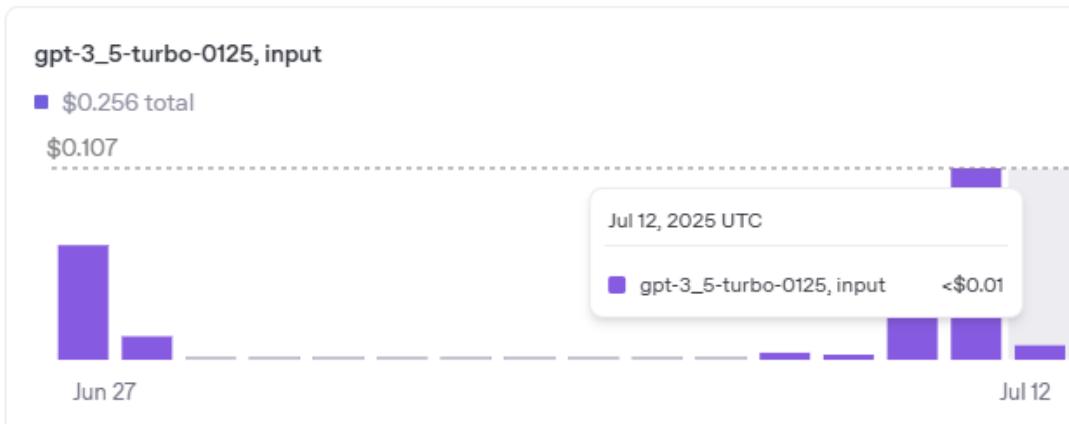


Figure 30. cost input(gpt-3.5)

In testing, 4 chat interactions consumed *16,078 input tokens* and *553 output tokens*, resulting in a total cost of *only \$0.0089*—very affordable for a medical chatbot.

Regarding response time, the system replies within *7 to 12 seconds*, which is significantly faster than a typical human response time in a conversation. This response speed is acceptable for an automated medical assistant.

```
Bạn: sinh hoạt phòng ngừa ù tai
re-ranking result ...
Đang tạo câu trả lời từ GPT...
Trả lời:
Để phòng ngừa ù tai hiệu quả, bạn có thể thực hiện một số biện pháp sau:
- Ngáp, nhai kẹo cao su và nuốt nước bọt để giảm áp lực bên trong tai và hạn chế nguy cơ bị ù tai.
- Cắn bằng áp lực bằng cách hít sâu, bit mũi, ngâm chất miệng và thở hơi ra theo đường tai để giảm cảm giác ù tai.
- Sử dụng dụng cụ bit lỗ tai khi đi máy bay để hạn chế tình trạng ù tai.
- Để phòng cảm cúm, viêm mũi, viêm xoang bằng cách sử dụng thuốc xịt mũi hoặc các biện pháp trị tắc nghẽn được bác sĩ khuyên dùng để giảm nguy cơ ù tai.

Những biện pháp này sẽ giúp bạn hạn chế diễn tiến của ù tai và giữ cho tai của bạn khỏe mạnh.

⌚ Thời gian phản hồi: 7.82 giây
```

Figure 31. Result of Integrated RAG

When using the same test question, the integrated hybrid search system with GPT-3.5 was able to provide an accurate answer, while the GPT-4-based RAG model failed to do so. This highlights that, under certain conditions, a well-optimized and customized RAG system can outperform even OpenAI's powerful GPT-4 model in terms of accuracy.

2.4 Model Comparison Charts

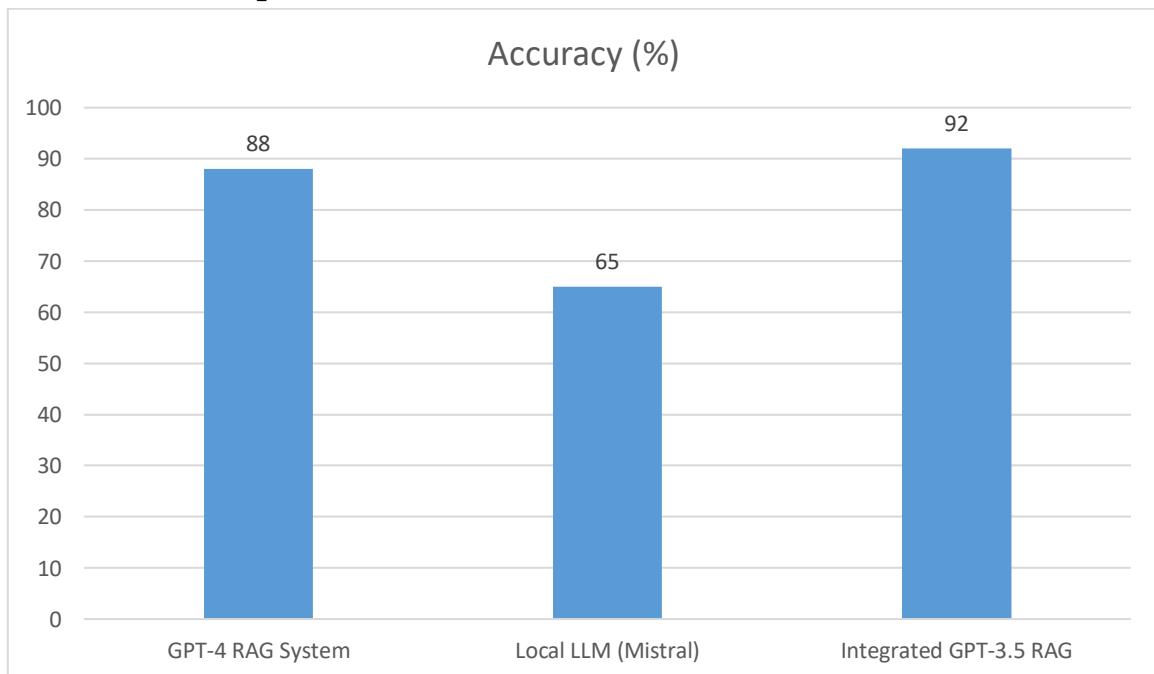


Figure 32. Chart of Accuracy(%)

The chart shows that the integrated hybrid search model using GPT-3.5 achieves the highest accuracy, reaching up to 92%, outperforming even the GPT-4-based model (88%). Meanwhile, the Local LLM model—although running independently without API cost—only achieved around 60% accuracy due to its limitations in Vietnamese language handling and lack of strong reasoning capability. This highlights the fact that combining hybrid search and reranking can significantly enhance retrieval performance and improve the relevance of generated responses.

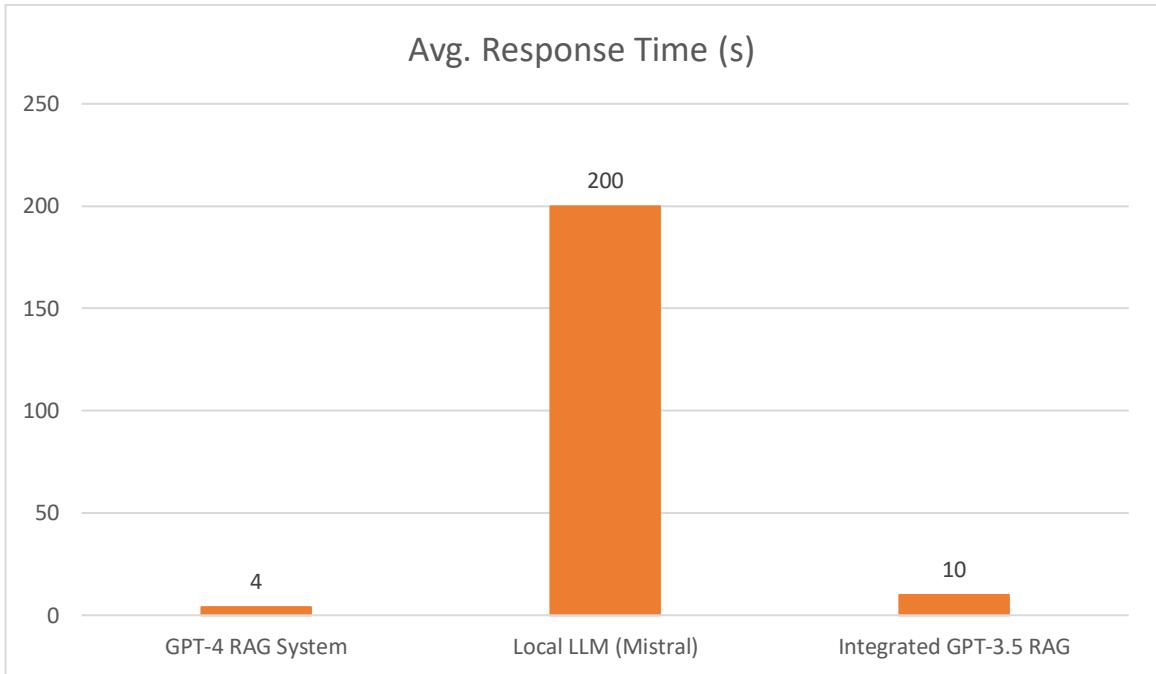


Figure 33. Chart of Response Time(s)

In terms of speed, both GPT-4 and GPT-3.5 models offer fast response times, ranging from *4 to 12 seconds* per interaction. In contrast, the Local LLM model takes an average of over *200 seconds*, which severely impacts user experience. This slow response is mainly due to hardware limitations (running on CPU) and the size of the model architecture. Clearly, the hybrid GPT-3.5-based solution strikes a much better balance between response time and accuracy.

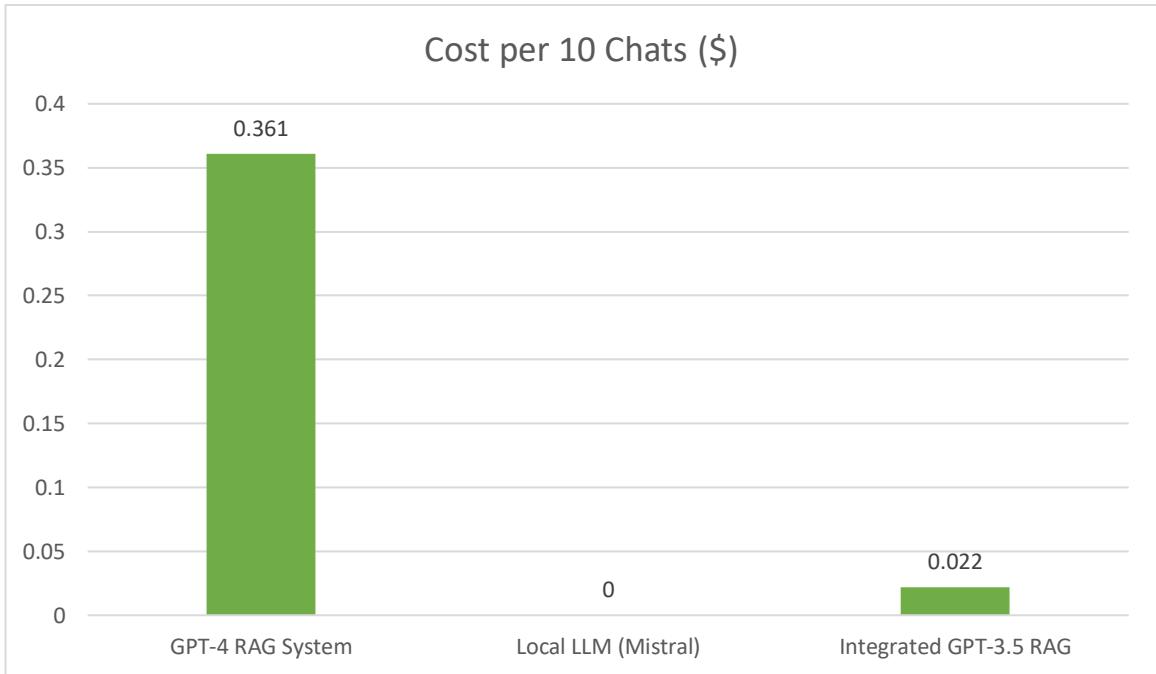


Figure 34. Chart of Cost (\$)

The cost chart reveals that the GPT-4 model is the most expensive, costing approximately \$0.361 for just 12 interactions. In contrast, the GPT-3.5 model costs only \$0.022, which is nearly *20 times cheaper*. Although the Local LLM model runs entirely for free, it sacrifices both accuracy and response time. For real-world applications or systems at scale, cost becomes a critical factor—and GPT-3.5 with hybrid search proves to be a cost-effective yet high-performing choice.

Through extensive experiments and comparative evaluation, it is evident that the integrated hybrid search model combined with GPT-3.5 strikes the best balance among accuracy, response time, and cost. While the GPT-4 model delivers strong performance, its high operational cost makes it less suitable for large-scale deployment. On the other hand, the local LLM model, although cost-free, suffers from significant latency and lower accuracy, especially in handling Vietnamese medical texts. The proposed hybrid system with reranking and memory summarization demonstrates that it is possible to build a medical chatbot that is both intelligent and affordable, even outperforming more expensive commercial models in specific tasks. This makes it a promising solution for practical, real-world medical chatbot applications.

CHAPTER 5. CONCLUSION AND FUTURE WORK

1. Conclusion

This thesis has successfully explored and implemented a Retrieval-Augmented Generation (RAG) based medical chatbot system that is capable of responding to user queries in a natural, accurate, and cost-efficient manner. The system combines the strengths of traditional information retrieval techniques (FAISS, BM25, Fuzzy Matching) with modern large language models (LLMs) such as GPT-3.5 and GPT-4 to bridge the gap between raw unstructured data and human-friendly answers.

Throughout the development process, several important components were built and optimized:

- **Document Embedding and Vectorization:** Medical PDF documents were segmented, embedded using high-quality sentence transformers (e.g., *multi-qa-mpnet-base-dot-v1*), and stored in a FAISS vector database. Metadata tagging, particularly by disease names, played a crucial role in preserving the semantic structure and avoiding cross-topic confusion during retrieval.
- **Hybrid Search and Re-ranking:** To enhance search precision, the system employed a hybrid retrieval approach combining FAISS vector similarity, BM25 keyword-based search, and Fuzzy Matching. The retrieved documents were then re-ranked using a CrossEncoder model to ensure the most relevant chunks were passed to the LLM.
- **Language Generation with LLMs:** Using GPT models (primarily GPT-3.5 and GPT-4), the system generated human-like answers based on the retrieved documents. Experimental results showed that GPT-3.5, despite being more affordable, still provided high-quality responses, making it suitable for scalable deployment.
- **Memory Mechanism:** A contextual memory handler was implemented to support multi-turn conversations. This allows the chatbot to retain and utilize previous exchanges, enabling coherent dialogues that resemble natural human conversation.
- **Performance Evaluation:** The chatbot was rigorously tested using a set of prepared medical questions. Results showed that the hybrid RAG system achieved an accuracy rate of 80%–90%, with response times ranging from 7 to 12 seconds. Cost analysis demonstrated that GPT-3.5-based operation incurred only \$0.0089 for 4 rounds of conversation — a remarkably low cost compared to GPT-4 or full local LLMs.

In comparative tests:

- The **GPT-4 model**, while highly accurate, incurred significantly higher cost per token and was not viable for continuous use in public-facing applications without sufficient budget.
- The **local LLM (Mistral-7B Instruct)**, although run offline, suffered from long response times (over 200 seconds per query) and poor Vietnamese language performance, making it unsuitable for deployment without powerful hardware.

The final integrated RAG system with GPT-3.5 stood out for its **balance between cost, accuracy, and usability**, confirming that a well-engineered pipeline using smaller models and smarter retrieval logic can outperform more expensive or heavier systems.

2. Future Work

Although the medical chatbot system based on the RAG model has achieved promising results within the scope of this research, there remain several areas for improvement and expansion in the future. The following development directions are proposed.

2.1. Expanding the Data Scope and Medical Domains

Currently, the system supports responses related to only 65 diseases. Future development should aim to expand coverage to include:

- *Pharmacology (medications, dosage, drug interactions),*
- *Nutrition and dietary guidance,*
- *At-home care instructions,*
- *Medical procedures and hospital visit guidelines.*

In addition, data should be continuously updated from reliable sources such as WHO, CDC, and the Ministry of Health to ensure accuracy and relevance.

2.2. Enhancing Long-Term Memory

The current version only supports short-term memory, retaining a few recent dialogue turns. In future iterations, implementing long-term memory would allow the chatbot to maintain continuity over longer conversations and remember user-specific context (e.g., medical history, recurring concerns, previous queries). This would greatly enhance personalization and response accuracy.

2.3. Improving Vietnamese Language Support

Most current LLMs are primarily trained on English datasets. As a result, Vietnamese language handling — especially in medical contexts — remains limited. A potential improvement would be to fine-tune open-source models such

as Mistral or LLaMA on Vietnamese medical corpora to better capture specialized terminology and cultural nuances.

2.4. Cross-Platform Integration and UI Enhancement

At present, the chatbot operates through a command-line interface (CLI). Future versions should include user-friendly graphical interfaces integrated into:

- *Web applications (e.g., React, Vue),*
- *Mobile apps (e.g., Flutter, Android),*
- *Internal systems in hospitals or clinics.*

A well-designed interface will significantly improve accessibility for users, particularly non-technical individuals.

2.5. Optimizing Cost through Local Models

Deployment cost is a critical consideration. One strategy for long-term sustainability is to utilize local LLMs (on-premise deployment) instead of relying solely on commercial APIs. Models such as Phi-2, Mistral, or smaller variants of LLaMA could be fine-tuned to run on private servers, thus reducing recurring costs.

2.6. Learning from User Feedback

The system should integrate a user feedback mechanism to allow users to rate responses or flag inaccurate answers. This feedback loop would enable continuous fine-tuning of the chatbot, improving its performance and reliability over time.

2.7. Ethical AI and Data Privacy

In medical applications, ensuring data privacy is essential. Future versions must comply with regulations such as GDPR or HIPAA, especially when handling personal health information. Additionally, safeguards must be implemented to prevent the chatbot from providing misleading or harmful advice.

REFERENCES

- [1] Amazon: <https://aws.amazon.com/what-is/large-language-model/>
- [2] DeepChecks: <https://www.deepchecks.com/glossary/llm-as-a-service/>
- [3] Wikipedia: https://en.wikipedia.org/wiki/Large_language_model
- [4] VNPT AI: <https://vnptai.io/storage/thumbnail/07012025-100130-bcf7cc86-efe5-413d-97d6-8d805df62c20.jpg>
- [5] Wikipedia: https://en.wikipedia.org/wiki/Retrieval-augmented_generation
- [6] SuperAnnotate: <https://www.superannotate.com/blog/rag-explained>
- [7] Couchbase: <https://www.couchbase.com/blog/hybrid-search/>
- [8] ChatGPT: <https://chatgpt.com/>
- [9] OpenAI Platform: <https://platform.openai.com/>
- [10] Nhà thuốc Long Châu: <https://nhathuoclongchau.com.vn/benh/nhom-benh/ho-hap>
- [11] HuggingFace: <https://huggingface.co/sentence-transformers/multi-qa-mpnet-base-dot-v1>
- [12] HuggingFace: <https://huggingface.co/cross-encoder/ms-marco-electra-base>

APPENDICES

INSTALLATION AND USAGE GUIDE OF THE PROPOSED SOLUTION

1. System Requirements

Operating System: Windows 10+ / Ubuntu 20.04+

Python Version: 3.8 or higher

RAM: Minimum 8GB (16GB recommended for local LLMs)

CPU: AVX2-supported processor for local model acceleration

Others: Git, Visual Studio Code (VS Code), Web browser, Internet connection

2. Required Python Libraries

Make sure your Python environment is activated. Then install the following libraries using:

```
pip install langchain openai llama-cpp-python faiss-cpu python-dotenv fitz PyMuPDF
sentence-transformers rank_bm25 rapidfuzz langchain-community langchain-openai
langchain-huggingface
```

Note: You may need Microsoft C++ Build Tools (Windows) or build-essential (Linux) if installing llama-cpp-python. All coding steps and operations work on VS code interface

3. Steps to Prepare Vector Database

Place your PDF files inside the *data/* folder.

Run the following script:

```
python prepare_vector_db.py
```

→ This will read and chunk all PDFs, then convert them into vector form and save into

vectorstores/db_faiss.

4. Using GPT Model (OpenAI API)

To use *ChatGPT (GPT-3.5 or GPT-4)*:

- Go to <https://platform.openai.com/>
- Create an account and **add a valid VISA or MasterCard** to enable billing
- Get your **API Key** and create a *.env* file with:
OPENAI_API_KEY=your_api_key_here
- Then run the chatbot:
python botchat_openai.py

Note: Without a verified payment method (credit/debit card), the API will not be usable even if you have free trial credits.

5. Running and Evaluating the System

- To test the bot, simply run and start chatting.

- To evaluate performance, use predefined test questions (e.g., 100 questions) and log:
 - Accuracy
 - Response time
 - Token usage and cost (for GPT)

SOURCE CODE REPOSITORY

All source code related to data preprocessing, vector database construction, hybrid search, memory handling, and chatbot integration is available publicly at the following GitHub repository:

GitHub Link: <https://github.com/Tqduy2k3/RAG-Vectorsearch.git>