Slide 1 – Title

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

Good morning/afternoon, respected committee members and everyone present.

My name is **Le Huynh Dang**, student ID **B2111977**, from the **Information Technology** major at **Can Tho University**.

Today, I am honored to present my graduation thesis titled:

“Applying Hybrid RAG Technology in a Chatbot for Consulting on Sexually Transmitted Diseases (STDs).”

In this presentation, I will briefly introduce the motivation for this topic, the main technologies used, the system design and implementation, and finally the experimental results and conclusions.

Slide 3 – Urgency of the Topic

Sexually transmitted diseases are still a **sensitive** and **difficult** topic for many people to talk about.

In reality, a lot of young people feel **embarrassed** or **afraid of being judged** when asking doctors or family about STDs.

As a result, they often search on the internet and may receive **incomplete, outdated or even incorrect** information.

Therefore, there is a clear need for a system that can provide **confidential**, **evidence-based**, and **user-friendly** consultations about STDs, with better control over the information source.

Slide 5

Now I will briefly introduce **Retrieval-Augmented Generation, or RAG**.

The basic idea of RAG is very simple: when the user asks a question, the system first **retrieves the most relevant passages** from a knowledge base, then **attaches those passages to the prompt**, and finally the **LLM generates an answer based on that context with citations**.

This brings several benefits: it **reduces hallucinations**, it allows us to **update the system just by updating the data source**, and it gives us **better control over what content and citations the model can use**.

However, the quality of RAG strongly depends on three key factors: **the quality of the data**, the **chunking and metadata strategy** we use to split and label the documents, and the **retrieval algorithm** that selects which chunks are sent to the model.

Slide 6 – BM25 and FAISS Indexing – 2.2

Data about medical articles will be recorded in CSV files, and these files are first **split into chunks of data** – each chunk is a small paragraph with its own metadata.

The chunks are then indexed in two ways.

First, the **BM25 retriever** builds a classic keyword index over these chunks, so it can quickly match exact terms like "chlamydia" or "condom use".

At the same time, the chunks are converted into **embedding vectors** and stored in the **FAISS vector store**.

When a user asks a question, the system doesn't let the LLM answer alone.

Instead, it sends the query to the retrieval layer.

Using the **nearest-neighbor search** over the FAISS store, combined with the BM25 signal, the system finds the **most relevant chunks** from the knowledge base.

These retrieved chunks are then **fed back into the LLM**, which uses them as grounded context to generate a safer and more accurate answer for the user.

Slide 7 – Hybrid Recursion & RRF Fusion – 2.3

To merge the results from BM25 and FAISS, I apply a **hybrid fusion method** based on **Reciprocal Rank Fusion, or RRF**.

The idea is to look at the **rank** of each document in each ranking list and sum them using the formula:

score_RRF = Σ 1 / (k + $rank_i$) -> "score R-R-F equals the sum over i of one over k plus rank sub i."

Here, documents that are **highly ranked in both BM25 and FAISS** will get a **higher final score**.

This approach **prioritizes consensus** between the two retrieval methods and **avoids problems with different score scales**.

In practice, this hybrid strategy helps the chatbot retrieve **more relevant and robust passages**, especially for **noisy or ambiguous user questions**.

Slide 8 – GraphRAG – 2.4

Besides the standard hybrid RAG, I also explore **GraphRAG**.

The main idea of GraphRAG is to represent the knowledge base as a **graph**, where **nodes** can be entities or chunks, and **edges** represent relationships such as "is related to", "explains", or "belongs to the same topic".

When answering a question, the system can first **identify relevant nodes** and then **expand along the graph**, aggregating context from **connected clusters** instead of treating every chunk independently.

This helps the chatbot generate answers that are not only **locally relevant**, but also **globally coherent**, especially for **multi-step or high-level questions** like prevention strategies or comparing different STDs.

Slide 9 – Data Pipeline & Chunking – 2.5

To support all of these retrieval methods, I designed a **data pipeline** with carefully chosen **chunking and metadata**.

First, I collect and clean data from **trusted medical websites** about STDs.
Then, I **split the documents into chunks** using a strategy that balances **length** and **semantic coherence**, for example by paragraph or section.

Each chunk is stored with **metadata** such as source, language, topic, and section heading.
These metadata fields are used for **filtering** and **better retrieval**, for example, focusing on prevention, symptoms, or treatment guidelines.

Finally, I build **BM25 indexes** for lexical search and **vector indexes in FAISS** for semantic search over these chunks.

Slide 10 – Overall Architecture – 3.1

First, the process starts with the user's query.
The query is stored together with the previous messages in the chat history, so that each conversation can be resumed later.
Then the current query, together with some recent turns, is sent to the backend built with FastAPI.

Inside the backend, we perform two small steps before retrieval: **pre-processing** and **scope filtering**.
Pre-processing means normalizing the text – for example, removing extra spaces, lower-casing, and cleaning special characters – so the input is consistent and ready for embedding.
Scope filtering decides **where** we are allowed to search in the knowledge base, for example only STI-related documents, only Vietnamese content, or a specific section such as "prevention".
By cleaning the text and narrowing the scope early, we make the later retrieval step both faster and more accurate.

Next, in the **embedding** step, the cleaned query is converted into a vector representation.
This vector is then compared with the vectors stored in the **vector database**.
Since the database contains embeddings of all chunks in the knowledge base, a **top-k retrieval** operation can find the chunks that are most similar to the user's query.

In the diagram you can also see a **GraphRAG** module: this represents an extended design where graph-based reasoning over a knowledge graph can be added on top of the standard vector search to provide more structured, global context.
Finally, the retrieved context and the **conversation memory** are sent to the LLM.
The LLM uses this grounded information to generate an answer, which is returned to the user interface and appended back to the chat history.

Slide 11 – Backend & API – 3.2

The backend is built with **FastAPI**, providing **RESTful endpoints** for functions such as:

- **User registration and login**, with **JWT-based authentication**.
- **Creating and managing conversations**
- **Sending a question to the chatbot includes text, voice, image**

Slide 12 – Chat Workflow – 3.3

This slide illustrates the **chat workflow** step by step.

1. The user sends a **question** through the web interface.
2. The backend **authenticates the user** and stores the message in the database.
3. The **hybrid retriever** uses BM25, FAISS, and GraphRAG to **find the most relevant chunks** from the knowledge base.
4. These chunks are **fused using RRF** to produce a final ranked list.
5. The backend **constructs a prompt** that includes the user's question, the retrieved context, and safety instructions.
6. The **LLM generates an answer**, which is then post-processed and stored.
7. Finally, the **answer is returned** to the user interface and displayed in the chat.

Slide 15 – Evaluation (Hit@k by intent)

On this slide, I present the **retrieval accuracy** of the system, measured by **section-level Hit@k** for different user intents.
I created **100 test queries** in total, divided equally into **5 intent groups**: Symptoms, Transmission, Testing, Treatment, and Prevention — so **20 queries per intent**.
For each query, I checked whether the correct section of the knowledge base appeared in the **top-1, top-3, or top-6 retrieved chunks**. This gives us the **Hit@1, Hit@3, and Hit@6** metrics shown in the table and the bar chart.
Looking at the results:

- For **Transmission**, the system performs very well: Hit@1 is **14 out of 20 (70%)**, and Hit@3 and Hit@6 both reach **100%**.

- For **Testing**, Hit@1 is **75%**, Hit@3 is **80%**, and Hit@6 goes up to **85%**.
- **Symptoms** is a bit lower, with **45%** at Hit@1 and **65%** at Hit@6.
- The weakest intents are **Treatment** and **Prevention**. Treatment only gets **35% at Hit@1** and **45% at Hit@6**, while Prevention starts at **20% at Hit@1** and improves to **50% at Hit@6**.

Overall, across all 100 queries, the system achieves **49% Hit@1**, **60% Hit@3**, and **69% Hit@6**.

This means that in about **two-thirds of the cases**, the correct section is already present within the **top-6 retrieved chunks**, which is a reasonable level for a first prototype, but it also clearly shows **where we still need to improve**, especially for treatment and prevention questions.

Slide 16 – Evaluation (Latency)

The second part of the evaluation looks at **latency**, because a medical chatbot also needs to be **responsive enough for interactive use**.

This chart compares the **measured latency** of my system with the **non-functional targets** I set at the beginning of the project.

I focus on two percentiles:

- the **median latency (p50)**, which represents a typical request, and
- the **95th percentile (p95)**, which represents **slow, worst-case requests**.

In practice, the **median latency** of the system is around **4–5 seconds**, while the target was about **3 seconds**.

For the **95th percentile**, the system is roughly **9 seconds**, compared to a target of around **7 seconds**.

So the system is **still usable** for an academic prototype, but it does **not fully meet** the original latency targets, especially in the tail.

This reflects the trade-off I mentioned earlier: when we use a richer retrieval pipeline and a large language model, we gain **answer quality**, but we also pay a cost in **response time**.

Improving this latency — for example by optimizing indexes, caching, or using a faster model — is an important direction for **future work**.

Slide 15 – Conclusion and Future Work

To conclude, my thesis makes **three main contributions**.

First, it builds a **bilingual STI chatbot** that can answer sensitive questions in both Vietnamese and English, using a carefully curated medical knowledge base as its main source of truth.

Second, on the retrieval side, it **designs and experimentally compares several strategies** — BM25 for lexical search, FAISS for semantic search, and a **hybrid BM25 + FAISS pipeline with RRF fusion**.
The results show that the **hybrid setup retrieves more relevant passages** and helps the model answer **more accurately** than using BM25 or FAISS alone.

Third, it implements a **complete end-to-end system** with a FastAPI backend, a MySQL database, JWT-based authentication and a custom web interface, so the chatbot can actually be deployed and extended in practice.

For future work, I would like to:
– Enrich the dataset with more **local and up-to-date medical sources**.
– Add **stronger safety filters and clearer disclaimers** to further reduce potential risks for users.
– And implement the planned **GraphRAG-style extension** and evaluate the system using **standardized benchmarks** or together with **medical experts**.

Overall, I hope this system is a small but meaningful step toward **safer and more accessible sexual-health education** for young people.

Slide 16 – Thank You / Q&A

That is the end of my presentation.

Thank you very much for your attention.
I will be happy to answer any questions or receive any comments from the committee.

Why you don’t save data in files?

Reduce the complexity of reading/writing files, managing paths, permissions, etc.

Easy to package and deploy: just run the code and all the data is available, no need to depend on external files.