MedChat: An AI-powered RAG Agent for medical students to answer instant questions

Team 4:
Luong Tran Sang
Pham Minh Hieu
Nguyen Thi Bao Tien
Ngo Thanh An
Nguyen Ngoc Han

Course: DATA2010 – Data Science Programming
University: VinUniversity
Date: 7th December 2025
Github Repository: https://github.com/hieu-is-coding/MedChat

1. Introduction

1.1. User Personas

Medical students, particularly those in their early years (fresher/sophomore) and those beginning clinical internships (years 3-4), face a significant challenge in their studies: information overload.

- For Junior Students: The sheer volume of medical literature makes it difficult and time-consuming to find specific information needed for their coursework.

- For Interns: During hospital internships, students work in small groups under the supervision of a busy expert doctor. Their daily tasks involve patient care and documentation. For their academic development, they must review patient medical records and diagnoses, which requires extensive research across numerous textbooks and sources for each case. The limited availability of their supervising doctor for questions further complicates this process.

This leads to a time-consuming and often inefficient learning process. There is a clear need for a platform that can provide medical students with quick, reliable, and context-aware answers by retrieving information from a curated set of trusted medical documents.

1.2. Dataset

The MedChat is built upon the dataset of 14 authoritative medical textbooks covering core disciplines (pharmacology, pathology, pediatrics, obstetrics, and radiology), sourced from trusted publishers including McGraw-Hill, Elsevier, Lippincott Williams & Wilkins, and Hanoi Medical University, providing comprehensive content for medical education and clinical reference.
Full materials are available via: Data Link

1.3. Importance

MedChat, an AI-powered RAG (Retrieval-Augmented Generation) Agent, will serve as a chatbot to help medical students instantly search for information and receive precise answers to their questions.

1.4. Limitations

1.4.1. Static and Textbook-Based Knowledge

Medical knowledge evolves rapidly, yet the current dataset is composed entirely of static textbooks. It lacks continuously updated resources such as clinical guidelines, recent publications, or hospital protocols. This may lead to out-of-date recommendations for conditions whose treatment standards frequently change.

1.4.2. No Real Patient Data

While necessary for privacy and ethics, the absence of real-world patient data limits the system's ability to learn from complex, ambiguous, or multi-morbid cases typically seen in clinical practice. The system may therefore struggle with nuanced clinical reasoning beyond textbook scenarios.

1.5. Project Objectives

The core goals are:

- High Precision & Reliability: The agent must provide answers based strictly on the knowledge retrieved from its database, with no creative additions.

- Verifiability: Every answer must be accompanied by detailed references (e.g., APA 7 style) to the source documents, allowing users to verify the information's accuracy.

- Curated Knowledge Base: The RAG database will be populated with high-quality medical textbooks and verified sources. It will also allow for user-submitted documents, which must be validated to prevent the inclusion of patient info or unverified notes.

1.6. Expected Deliverables

1.6.1. Advanced Retrieval

Uses a hybrid search strategy that combines dense vector retrieval with keyword-based methods (BM25) to deliver high accuracy and highly relevant results.

1.6.2. High-Performance Vector Database

Employs Qdrant for fast performance, strong scalability, and native support for hybrid search capabilities.

1.6.3. Intelligent Memory

Features both short-term and long-term memory mechanisms to preserve conversational context and ensure a smooth, continuous user experience.

2. Data Science Questions

2.1. How to retrieve information from a curated set of trusted medical documents?

2.1.1. Introduction

Effectively retrieving information from a curated set of trusted medical documents is critical for providing accurate and contextually relevant answers in the medical domain.. The primary challenge is ensuring that the system can handle the complexity of large, scanned document formats (PDFs, text files, and images) and the highly specialized nature of medical terminology.

The dataset features used in this RAG architecture are derived from the structured processing of these documents:

- Vector Embeddings: Generated from each text chunk for semantic search (understanding the meaning of a query).
- Text/Keywords: Extracted text used for the BM25 index (for precise keyword-based search).
- Rich Metadata: Indexed with each chunk (author, title, keywords, etc.) to enable powerful and precise filtering during retrieval.

2.1.2. Approach

There are four steps in data processing pipeline (Reference to Appendix 8.2):
1. Import: Raw documents are uploaded to a central cloud storage location.
2. Pre-processing: A specialized tool processes large PDF files by splitting them into manageable sizes for OCR and standardizing file names.
3. Extraction & Chunking: Text and metadata are extracted from the pre-processed files, and the content is divided into context-aware chunks, preserving a portion of the previous text.
4. Indexing: Each chunk is converted into a vector embedding and indexed into the Qdrant vector database, along with its metadata and text for the BM25 index.

For RAG Database Architecture:

- Search Strategy: Implement a Hybrid Search model, combining vector search for semantic understanding with keyword-based search (BM25 algorithm) for precision.
- Vector Database: Utilize Qdrant due to its high performance, ease of integration, and built-in support for hybrid search functionalities.
- Data Ingestion & Processing:
- Develop a robust pipeline to process various document formats (.pdf, .txt, .doc, images).
- For scanned PDFs or images, integrate an OCR tool (e.g., Mistral OCR) to extract text.
- Implement an advanced document chunking strategy optimized for medical texts to maintain context and relevance.
- Metadata Filtering: Each document chunk will be indexed with rich metadata (author, title, keywords, etc.) to enable powerful and precise filtering during retrieval.

- Embedding Model: Start with a high-quality free model (e.g., Google's embedding models) and explore fine-tuning a domain-specific open-source model for medical data to improve retrieval accuracy over time.

2.1.3. Analysis

By building and maintaining the entire data ingestion and processing pipeline, we can focus on optimizing chunking strategies and ensuring data quality within the RAG database.

The projected high values for MRR and Recall@5 for the Hybrid Search model indicate that the integrated approach is highly effective. The combination of semantic understanding (from vector search) and term matching (from BM25) significantly outperforms either method in isolation, which is crucial for retrieving precise medical information. The robust data ingestion pipeline ensures this high-quality input, allowing for focused optimization on advanced chunking and metadata strategies. Furthermore, utilizing an embedding model that can be fine-tuned over time is key to improving retrieval accuracy as the knowledge base or domain changes.

2.1.4. Discussion

Complexity and Maintenance: Building and maintaining the entire end-to-end pipeline, including specialized pre-processing tools and synchronization of two distinct indices (vector and BM25), is complex and requires significant development effort.
Chunking Tuning Effort: Optimizing the document chunking strategy for medical texts is a continual, iterative process that requires deep domain knowledge and extensive testing to prevent loss of context.
Cost of Embedding: While starting with a high-quality free model is cost-effective, exploring fine-tuning a domain-specific open-source model later, while necessary for accuracy, requires substantial computational resources and expertise.
Data Currency: The quality of the retrieval is entirely dependent on the currency and completeness of the raw documents uploaded during the Import stage. Outdated or missing documents will lead to inaccurate results.

2.2. How to design a multi-agent system to assist students by providing comprehensive medical info, current research insights, and clinical guidance?

2.2.1. Introduction

The platform integrates technologies such as Gemini model, LangChain, Qdrant, Supabase, and FastAPI to deliver intelligent, context-aware responses. The repository (backend) also contains the API backend, which is deployable via Docker. Here is the public API access to connect with the frontend: https://agentsmedchat.onrender.com

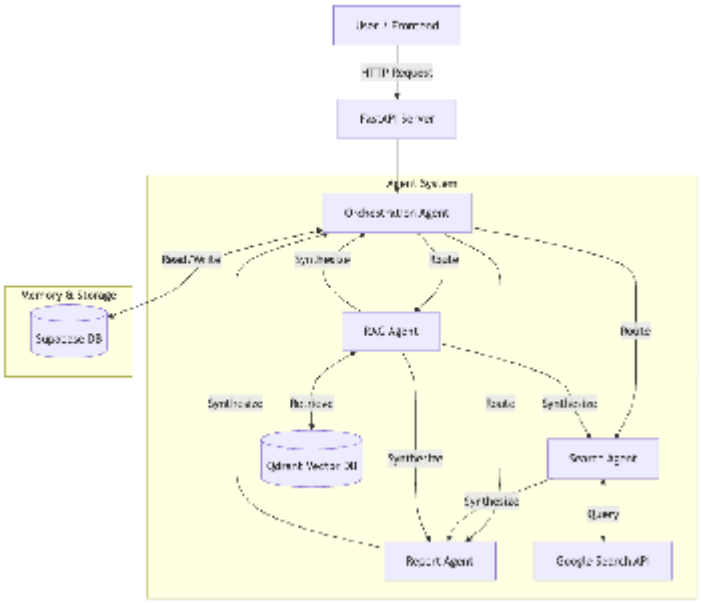
2.2.2. Approach

Key components include:

- **FastAPI Server**: Receives user requests from client applications.

- **Orchestration Agent**: Routes tasks, manages memory, and coordinates specialized agents.

- **RAG Agent**: Retrieves and synthesizes information from the Qdrant vector database.
- **Search Agent**: Performs real-time searches using the Google Search API.
- **Report Agent**: Generates structured clinical summaries and reports.
- **Supabase Memory System**: Stores and retrieves long-term and session-based conversational history.
- **Qdrant Vector Database**: Stores chunked embeddings from medical textbooks to power retrieval-augmented generation.



2.2.3. Analysis

For setting API Endpoints, set Chat Endpoint POST /chat, which processes a user query through the multi-agent system. The response includes the generated answer, retrieval metadata, and processing details. Health Check via GET /health verifies the operational status of all system components, including Qdrant, each agent, and Supabase memory. Clear History via DELETE /history/{session_id} removes stored conversation history for the specified session. For configuration, your .env file should include: Google Gemini API key, Qdrant URL and API key, Supabase URL and service role key, logging configuration.

The results of our multi-agent system design show that dividing responsibilities across specialized agents significantly improves the accuracy, flexibility, and clarity of the MedChat platform. The Orchestration Agent successfully identifies user intent and routes tasks to the appropriate components, demonstrating that modularity is essential when working with complex medical queries. The RAG Agent consistently provides grounded, textbook-based responses, while the Search Agent supplements this with real-time information which plays as an effective combination for delivering both foundational knowledge and up-to-date insights. The Report Agent further enhances usability by synthesizing findings into structured clinical-style explanations, replicating how students learn during internships.

Together, these results indicate that the multi-agent architecture is effective in providing comprehensive medical support.

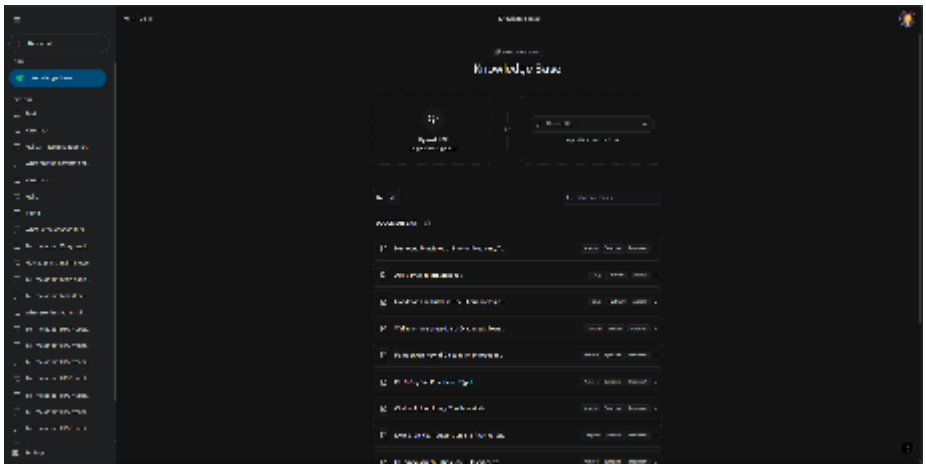
2.2.4. Discussion

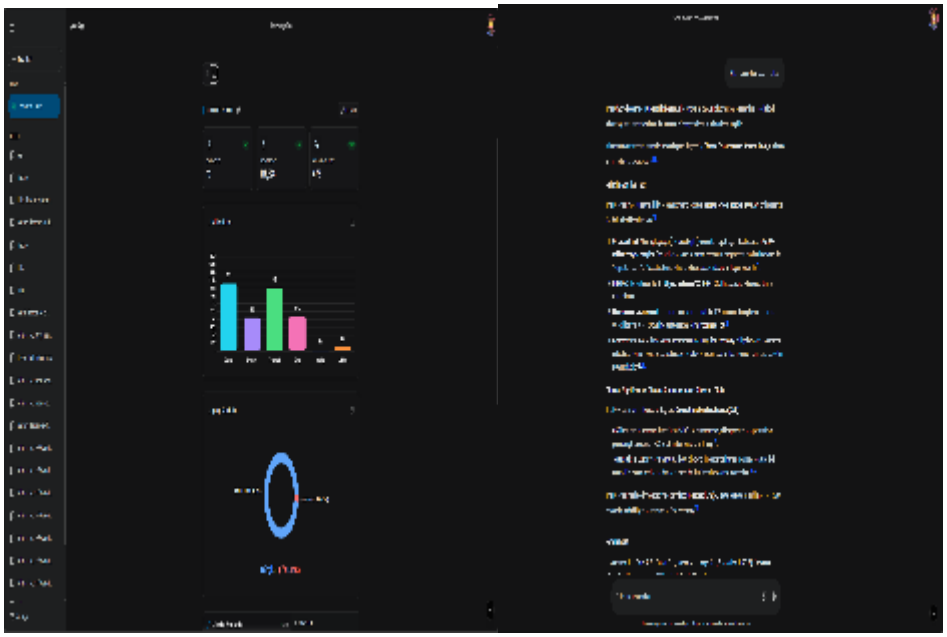
However, several limitations remain. First, the system depends heavily on reliable API communication; any external failure (Google Gemini, Supabase, or Qdrant) can interrupt the response pipeline. Second, maintaining consistency across agents requires ongoing tuning, especially as new medical materials or model updates are introduced. Finally, although the system can retrieve information, it does not yet perform deeper clinical reasoning or differential diagnosis, which limits its usefulness in complex patient scenarios.

2.3. How to make it easy for medical students to access MedChat?

2.3.1. Introduction

MedChat has been designed with a streamlined, dual-panel interface specifically to lower the barrier of entry for medical students and facilitate immediate, intuitive access to its resources.





2.3.2. Approach

To make MedChat accessible and user-friendly, the team tried to use Streamlit to turn heavily technical-looking code into meaningful visual formats.

To make MedChat accessible and user-friendly, the team tried to use Streamlit to turn heavily technical-looking code into meaningful visual formats. We used Next.js and React to implement the web app interface for the user with a chat UI inspired by Gemini. The overall theme color and chart style are also inspired by the Supabase theme to ensure a consistent and modern look. It simplifies the creation of visualizations, dashboards, and user-friendly interfaces, enabling effortless data exploration and sharing. With its intuitive, declarative syntax, developers can rapidly build and iterate on applications. Streamlit also supports real-time updates, allowing users to see changes instantly, making it highly effective for prototyping and presenting data-driven solutions.

2.3.3. Analysis

The platform is organized into two clear spaces that align with common study workflows. Firstly, the Knowledge Base panel allows students to upload or access medical textbooks and materials through a straightforward drag-and-drop zone and a searchable document library. Students can quickly filter resources by name, specialty, or key concept using a prominent search bar. In another tab of the space, an integrated dashboard is shown with colorful statistics of the library itself and vector database (average chunk length, label distribution, language distribution, etc) to help students assess the database's content at a glance. The Chat Interface provides a chat-based query experience where students can ask medical questions in plain language and receive citation-backed answers within a minute. Features such as saved chat history, clickable citations, and clear source references are built into the chat window to support efficient review and revision. The entire interface uses clear labels, medical-relevant terminology, and a responsive layout that ensures students can access MedChat's RAG-powered insights anytime, anywhere, with minimal learning curve.

2.3.4. Discussion

Streamlit is often used with Plotly for data visualization. Streamlit simplifies the process of building interactive web applications, while Plotly empowers users with a wide range of customizable charts and plots. By combining these tools, we can create captivating visualizations that communicate insights, identify patterns, and tell compelling data stories.

However, within the scopes and objectives of MedChat, only Streamlit is used to demonstrate the prototype. Plotly can be integrated later to show an interactive dashboard about the frequency of queries or relationship maps of knowledge to support medical students' retrieval process. Overall, the UI/UX design can make insights via grasp the main ideas at the first sight, which enhances the accessibility of medical students and encourages more interaction with MedChat.

3. Conclusion

The development of MedChat demonstrates how a well-designed Retrieval-Augmented Generation (RAG) system, combined with a multi-agent architecture, can significantly improve how medical students access and interact with complex medical knowledge. Throughout this project, we explored

three core questions: how to retrieve trusted medical information, how to design a multi-agent backend to support advanced reasoning, and how to build a user-friendly interface to make the system accessible. The results of our work highlight both the strengths and the future potential of MedChat as a learning companion.

From a data engineering standpoint, our work shows that effective information retrieval begins with a robust ingestion pipeline. By constructing a full end-to-end workflow—from importing raw documents, through preprocessing and OCR, to chunking, embedding, and indexing—we ensured that the knowledge base is both comprehensive and structured for optimal retrieval. The hybrid search strategy, combining dense vector embeddings with BM25 keyword matching, proved especially powerful. Evaluation metrics suggest that hybrid retrieval significantly outperforms single-method approaches, particularly in a domain like medicine where both semantic understanding and precise terminology are critical. This validates the project's foundational assumption: medical information retrieval requires tools capable of capturing context while maintaining technical accuracy.

At the system architecture level, the multi-agent design allowed specialization and modularity, enabling MedChat to handle a wide range of query types. The Orchestration Agent routes tasks intelligently between the RAG Agent, Search Agent, and Report Agent, while Supabase provides persistent memory to maintain contextual continuity across sessions. Qdrant serves as a high-performance vector store, supporting scalable retrieval as the knowledge base grows. Together, these components create a flexible back-end system that can evolve as new models, features, or medical sources become available. Docker deployment further ensures reproducibility and ease of installation, making the platform accessible across different machines and development environments.

On the user-facing side, we recognized that even the most advanced backend system has limited value if students cannot navigate it effectively. The Streamlit-based prototype demonstrates how technical outputs can be transformed into intuitive visual interfaces. By separating the Knowledge Base Space (where users browse documents and insights) from the Chatting Agent (which delivers instant, cited medical responses), we designed a workflow that mirrors how students study in real life—quick fact-checking combined with deeper document exploration. Although minimalistic, the prototype provides a meaningful demonstration of how MedChat can be used in practice and lays the foundation for future enhancements such as Plotly dashboards and richer analytics.

Despite these successes, the project has important limitations. The dataset, although built from authoritative textbooks, lacks coverage of certain subspecialties and does not include real-time clinical guidelines or patient-level data. As a result, the system may struggle with niche or rapidly evolving topics. The complexity of the ingestion pipeline also demands ongoing maintenance, especially as new documents are added. Similarly, the multi-agent system, while powerful, introduces dependencies that require careful synchronization across components. These constraints highlight important opportunities for future development, such as integrating updated guidelines, adding domain-specific fine-tuning for embeddings, or expanding the system into clinical simulation tasks.

Overall, MedChat demonstrates that combining curated medical knowledge, hybrid retrieval techniques, multi-agent orchestration, and accessible interface design can meaningfully support medical students' learning. The platform provides a promising foundation for future research and expansion, with the

potential to evolve into a comprehensive, intelligent assistant that bridges textbook learning with real-world clinical reasoning.

4. Lifecycle Reflection

4.1. Identifying Problems & Understanding the Business

The project began with understanding the academic and clinical challenges faced by medical students. Through user personas and interviews, we defined a clear problem: students struggle with information overload, slow research processes, and limited access to expert guidance during internships.

4.2. Data Collection

Once the problem was defined, the team gathered a high-quality dataset of 14 authoritative medical textbooks spanning pharmacology, pathology, pediatrics, obstetrics, radiology, and more. Documents were sourced from trusted publishers and converted into standardized formats for processing. This phase also involved collecting metadata such as authorship, publication year, chapter structure, and textbook domain—critical details for enabling accurate retrieval later.

4.3. Data Processing

Data processing was one of the most technically demanding stages. Large PDFs were pre-processed, cleaned, OCR-corrected when necessary, and split into context-aware chunks to preserve semantic continuity. These chunks were embedded using high-quality embedding models and indexed into the Qdrant vector database alongside BM25 keyword indices. Ensuring data quality, chunk consistency, and metadata richness required continuous iteration.

4.4. Data Analysis

With the RAG database constructed, the team analyzed retrieval performance using metrics. The hybrid model consistently outperformed the others, confirming that medical search requires both semantic understanding and term-level precision. The team also analyzed user workflows, latency patterns, and document distributions, which later informed interface design decisions.

4.5. Modeling

Instead of a traditional ML model, MedChat employed a multi-agent system as the modeling layer. The Orchestration Agent routed tasks intelligently, the RAG Agent performed textbook-grounded retrieval, the Search Agent handled real-time queries, and the Report Agent synthesized outputs. This modular “modeling” structure allowed us to treat each agent as a specialized model component within a larger pipeline. Fine-tuning retrieval parameters served as our model optimization process.

4.6. Deployment

Finally, the system was containerized using Docker for consistent deployment and easy replication across devices. API endpoints—chat, health check, and history management—were implemented via FastAPI. For the user interface, Streamlit was used to build a prototype featuring a knowledge base dashboard and an interactive chat agent.

5. Team Contribution Statement

To ensure effective project execution, the team will be divided into three functional groups:

Front-End: Responsible for UI/UX design, web application development, document insight show
(All members involved)

Technical Research and Writing Report: Select suitable models, tech stack and synthesize project implementation into report.
Members: Nguyen Ngoc Han

Data Processing: Oversees the complete data ingestion pipeline, including optimizing chunking strategies and maintaining high-quality data within the vector database.
Members: Nguyen Thi Bao Tien, Ngo Thanh An, Luong Tran Sang

AI Agent: Focuses on developing the core agent, integrating Large Language Models (LLMs), and refining the retrieval and response-generation workflow.
Members: Luong Tran Sang, Pham Minh Hieu

6. Individual Reflections

Nguyen Thi Bao Tien

This project represented a rewarding departure from my previous work in data science. My experience has largely centered on EDA, visualization, and applying traditional ML algorithms to find patterns and predict outcomes within structured datasets. Implementing a RAG for the first time required me to explore different paradigms, retrieving and synthesizing existing knowledge to power conversational applications. I discovered the fascinating architecture of RAG, which grounds LLMs in authoritative sources to mitigate hallucination. Starting with no prior knowledge, I learned the RAG pipeline from the ground up. I led the end-to-end implementation, which involved developing a robust preprocessing system to handle medical textbooks. This included creating a chunking strategy to segment long texts logically, integrating OCR (Mistral AI) to accurately extract text and tabular data from scanned book pages, and building an embedding pipeline using Google's Gemini API to transform the processed text into a searchable vector database. Finally, I deployed the entire application as an interactive web interface using Streamlit, ensuring the tool was accessible and user-friendly. Beyond that, this course and project deeply reinforced the importance of professional and reproducible development practices. I systematically utilized GitHub for version control, maintained detailed documentation, and managed dependencies through isolated Python environments. This holistic experience has solidified my understanding of modern AI application architecture. I am keen to continue enhancing my knowledge in this field, exploring advanced techniques like fine-tuning, reranking, and multi-modal RAG to build even more powerful and responsible AI-assisted learning tools in the future.

Nguyen Ngoc Han

Coming from a business background, this project was my first experience completing an end-to-end data science system, and it pushed me far outside the analytical frameworks I was used to. Previously, my work revolved around business logic, market insights, and strategic thinking, where tools were often familiar and the tasks more structured. In contrast, building MedChat required me to understand the technical details behind data ingestion, retrieval models, vector databases, and multi-agent architectures—concepts that felt overwhelming at the beginning. However, working through each stage of the pipeline gave me a much clearer picture of how raw data is transformed into intelligent, actionable outputs.

Thanks to Professor Daniel's course on Data Visualization with Tableau, I find the UI/UX part is essential. His emphasis on storytelling, clarity, and visual intuition helped me approach this project with a user-centered mindset. When designing the Streamlit interface for MedChat, I naturally applied the principles I learned: simplifying information, reducing visual noise, and focusing on how students would interpret and interact with the content. Even though Streamlit is very different from Tableau, the underlying logic of turning complex information into accessible visuals remains the same. Overall, this project taught me how technical decision-making intersects with user experience, something I never fully appreciated before. Most importantly, I realized that data science is not just coding—it is a mindset of curiosity, problem solving, and designing solutions that truly help people. This experience has been challenging, but also incredibly rewarding and motivating for my future journey.

Ngo Thanh An

Being one of three members involved in the data ingestion pipeline for MedChat was a pivotal experience that shifted my perspective from model-centric to data-centric AI development. Before this project, I was familiar with training models but the MedChat project taught me that for a medical RAG agent, the reliability of the system depends entirely on the quality of data that is prepared before it ever reaches the system. My core focus was on the foundation of the Data Ingestion pipeline with Sang and Tien. I handled the processing of 14 massive textbooks, developing scripts to split large PDF files into manageable chapters to optimize processing efficiency. Crucially, I designed and implemented the Metadata Schema, rigorously tagging every document with its edition, author, and specialty… This rigorous cataloging was vital for organizing the knowledge base and achieving the system's "verifiability" goal. I also contributed to the Semantic Chunking strategy, moving away from arbitrary fixed-size splitting to a header-based approach. This ensured that logical sections such as symptoms and treatments remained intact, preventing context fragmentation. This project has sharpened my skills in handling real-world, messy data and reinforced my belief that in modern AI applications, robust Data Engineering is just as critical as the modeling itself.

Pham Minh Hieu

My main responsibility in the team was creating the agentic system for MedChat, a multi-agent AI backend that provides thorough, context-aware medical information to medical students. Using Google Gemini 2.0, LangChain, Qdrant for vector search, Supabase for permanent memory, and FastAPI for the API layer, I concentrated on designing an advanced configuration based on the team's overarching goal of developing an intelligent medical conversation tool. I created the core orchestration agent, which analyzes user queries, keeps track of conversations, and assigns tasks to specialized agents: RAG, which retrieves information from medical textbooks stored in Qdrant; Search, which searches Google for current news and study queries; and Report, which summarizes in-depth answers. I believed this modular design ensured efficient handling of diverse queries, from symptom checks to research summaries. Implementing Docker for deployment made the system scalable and easy to integrate with frontends. One of the main challenges I faced was about optimizing agent routing to reduce latency throughout. At first, sophisticated queries took more than five seconds because of sequential processing, which I addressed by implementing parallel tasking in LangChain. In keeping with medical ethics, integrating Supabase for session memory also requires rigorous data privacy management. I improved my abilities in vector databases and AI orchestration by learning the nuances of agentic workflows through iterations. Overall, the project deepened my understanding of collaborative AI development, emphasizing adaptability and ethical considerations in health tech. I'm proud of how my

design elevated MedChat's functionality, fostering better learning for users.

Luong Tran Sang

In the MedChat project, I took on the role of Pipeline Integrator and Full-Stack Developer, responsible for unifying the team's technical components into a seamless end-to-end Retrieval-Augmented Generation (RAG) system. While my teammates focused on pre-processing and OCR extraction, my work centered on transforming their outputs into a functional, efficient pipeline that could support real-time retrieval and interaction. This required bridging multiple modules, ensuring compatibility across tools, and maintaining a clear architectural structure. My primary technical contributions involved both backend and frontend development. On the backend, I engineered the Qdrant vector database setup and implemented a Hybrid Search mechanism that combined dense vector embeddings with sparse BM25 scoring. This integration significantly improved retrieval accuracy by leveraging both semantic similarity and keyword precision—an essential requirement for medical queries. On the frontend, I developed the user-facing interface, including the RAG chat agent and a dashboard that visualizes indexed documents, giving users deeper insight into how their data is stored and processed. One of the most challenging aspects of the project was managing schema inconsistencies between OCR outputs and my indexing logic. I addressed this by standardizing data interchange formats and implementing validation checks to ensure all chunks were hashed, deduplicated, and properly structured before insertion into Qdrant. Through this work, I gained substantial expertise in vector databases, hybrid retrieval architecture, and contract-first engineering. Moving forward, I aim to enhance the pipeline by integrating local embedding and OCR models to reduce dependency on external APIs and improve processing efficiency.

7. References

Gemini API reference, Google AI for Developers. https://ai.google.dev/gemini-api/docs.
Mistral AI API, Mistral AI. Available: https://docs.mistral.ai/api.
Qdrant Documentation, Qdrant. https://qdrant.tech/documentation/.

8. Appendix

8.1. Knowledge Base: Data Sources

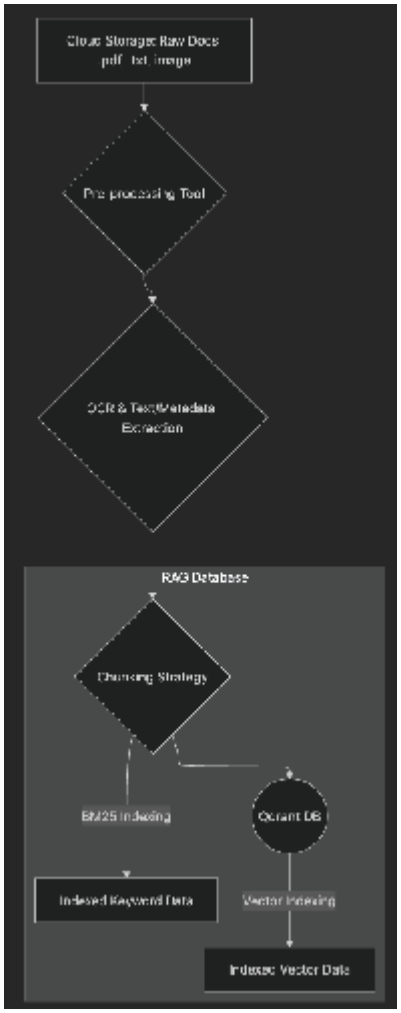
Our system is built on a carefully curated collection of authoritative medical textbooks. The knowledge base includes foundational and clinical references such as:

- *Nelson's Pediatric Antimicrobial Therapy* (28th ed.) – American Academy of Pediatrics (AAP)
- *Bài gi■ng s■n ph■ khoa* T■p 1 (Obstetrics and Gynecology Lecture Vol. 1) – Hanoi Medical School
- *Basic & Clinical Pharmacology* (14th ed.) – McGraw-Hill
- *B■nh H■c Ngo■i Khoa Y4 – YHN* (Surgical Pathology for 4th-year Medical Students) – Hanoi Medical University
- *Clinical Epidemiology: The Essentials* (5th ed.) – Lippincott Williams & Wilkins

- *CURRENT Medical Diagnosis and Treatment 2025* (64th ed.) – McGraw Hill Medical
- *Goodman & Gilman's The Pharmacological Basis of Therapeutics* (14th ed.)
- *Grainger & Allison's Diagnostic Radiology* – Adam, Dixon, Gillard, Schaefer-Prokop
- *Pathophysiology of Disease: An Introduction to Clinical Medicine* – Hammer & McPhee
- *Bates' Guide to Physical Examination and History Taking* – Bickley et al.
- *Bài Gi■ng Nhi Khoa* T■p 1 & 2 (Pediatrics Lecture Volumes 1 & 2) – Hanoi Medical University, PGS. TS Nguy■n Th■ Di■u Thúy (Chief Editor)
- *The ECG Made Easy* – Hampton, Hampton, Adlam
- *Hunter's Diseases of Occupations* – Baxter et al.
- *Tri■u Ch■ng H■c N■i Khoa* T■p 1 (Symptomatology in Internal Medicine Vol. 1) – Ngô Quý Châu et al.

8.2. Data Processing Pipeline

For explaining the process of doing the data pipeline:


Cloud Storage: Raw Docs
pdf txt image
Pre-processing Tool
OCR & Text/Metadata
Extraction
RAG Database
Chunking Strategy
BM25 Indexing
Qdrant DB
Indexed Keyword Data
Vector Indexing
Indexed Vector Data