TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
KHOA CÔNG NGHỆ THÔNG TIN

---------o0o---------

BÁO CÁO BTL CHUYÊN ĐỀ CÔNG NGHỆ THÔNG TIN

ĐỀ TÀI:
Xây dựng ứng dụng trợ lý ảo hỗ trợ sổ tay sinh viên

Giảng viên hướng dẫn: TS. Bùi Ngọc Dũng

Sinh viên thực hiện: Chu Văn Dũng

Mã sinh viên: 211210740

Lớp:: Kỹ sư CNTT 2 – K62

Hà Nội, tháng 12 năm 2025

Lời mở đầu

Với sự bùng nổ của cuộc Cách mạng công nghiệp 4.0, Trí tuệ nhân tạo (AI) và Xử lý ngôn ngữ tự nhiên (NLP) đang tạo ra những bước đột phá mạnh mẽ trong lĩnh vực giáo dục và chuyển đổi số trường học. Một trong những thách thức lớn nhất tại các trường đại học hiện nay là việc quản lý và truyền tải khối lượng thông tin khổng lồ từ các văn bản quy chế, quy định và hướng dẫn học vụ đến sinh viên. Các phương pháp tra cứu truyền thống dựa trên từ khóa (keyword search) thường gặp hạn chế lớn khi không hiểu được ngữ cảnh câu hỏi, dẫn đến việc sinh viên gặp khó khăn trong việc tiếp cận thông tin chính xác từ các tài liệu dài như "Sổ tay sinh viên".

Đề tài **"Nghiên cứu và xây dựng hệ thống Trợ lý ảo hỗ trợ tra cứu Sổ tay sinh viên sử dụng kỹ thuật RAG và Truy vấn đa phương thức"** được thực hiện nhằm giải quyết bài toán trên bằng cách ứng dụng các mô hình ngôn ngữ lớn (LLM) tiên tiến. Giải pháp đề xuất tập trung vào việc tích hợp kỹ thuật **Retrieval-Augmented Generation (RAG)** để đảm bảo tính chính xác của thông tin văn bản, kết hợp với mô hình thị giác tiếng Việt **Vintern-1B** để xử lý bài toán tìm kiếm hình ảnh (sơ đồ, biểu đồ) – một tính năng mà các hệ thống chatbot thông thường thường bỏ qua.

Điểm nhấn của nghiên cứu là việc tối ưu hóa quy trình truy xuất thông tin thông qua cơ thuật toán **Reranking theo thời gian thực**, giúp hệ thống ưu tiên các văn bản quy chế mới nhất, giải quyết vấn đề chồng chéo thông tin giữa các văn bản cũ và mới. Đồng thời, hệ thống được xây dựng trên kiến trúc module hóa với Flask và LangChain, cho phép cập nhật tri thức động mà không cần huấn luyện lại mô hình.

Nội dung báo cáo được trình bày trong 05 chương chính nhằm cung cấp cái nhìn toàn diện từ cơ sở lý thuyết, kiến trúc hệ thống đến kết quả thực nghiệm:

- Chương 1: Tổng quan về Trí tuệ nhân tạo tạo sinh và kỹ thuật RAG.
- Chương 2: Phân tích công nghệ và Lựa chọn mô hình.
- Chương 3: Thiết kế và Xây dựng kiến trúc hệ thống.
- Chương 4: Tối ưu hóa thuật toán Truy xuất và Xếp hạng (Reranking).
- Chương 5: Kết quả cài đặt và thử nghiệm hệ thống.

Mục lục

CHƯƠNG 1: TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO TẠO SINH VÀ KỸ THUẬT RAG

1.1. Sự phát triển của Trí tuệ nhân tạo tạo sinh (Generative AI)

Trong thập kỷ qua, lĩnh vực Trí tuệ nhân tạo (AI) đã chứng kiến sự chuyển dịch mạnh mẽ từ các mô hình phân loại (Discriminative models) sang các mô hình tạo sinh (Generative models). Nếu như AI truyền thống tập trung vào việc phân loại dữ liệu hoặc dự đoán các giá trị rời rạc, thì AI tạo sinh (Generative AI - GenAI) có khả năng tạo ra dữ liệu mới – bao gồm văn bản, hình ảnh, âm thanh và mã nguồn – dựa trên các mẫu đã học được từ tập dữ liệu huấn luyện khổng lồ.

Sự bùng nổ của GenAI được thúc đẩy bởi sự ra đời của kiến trúc **Transformer** (Vaswani et al., 2017), cho phép các mô hình xử lý dữ liệu tuần tự với cơ chế sự chú ý (Self-Attention mechanism) hiệu quả hơn vượt trội so với các kiến trúc RNN hay LSTM trước đó. Đây là nền tảng cho sự ra đời của các Mô hình ngôn ngữ lớn (Large Language Models - LLMs) như GPT (OpenAI), PaLM/Gemini (Google) hay Llama (Meta).

1.2. Mô hình ngôn ngữ lớn (Large Language Models - LLMs)

1.2.1. Định nghĩa và Cơ chế hoạt động

Mô hình ngôn ngữ lớn là các mô hình học sâu được huấn luyện trên lượng dữ liệu văn bản khổng lồ (hàng nghìn tỷ tham số). Cơ chế cốt lõi của LLM là dự đoán từ (hoặc token) tiếp theo trong một chuỗi văn bản dựa trên ngữ cảnh trước đó.

Trong phạm vi đề tài này, em sử dụng mô hình **Google Gemini** (phiên bản gemini-2.5-flash). Đây là thế hệ mô hình đa phương thức (multimodal) tiên tiến, có khả năng hiểu và xử lý thông tin phức tạp với tốc độ phản hồi nhanh và chi phí tối ưu hóa cho các ứng dụng thời gian thực.

1.2.2. Các hạn chế của LLM truyền thống

Mặc dù LLM có khả năng sinh văn bản trôi chảy, việc áp dụng trực tiếp LLM vào các bài toán tra cứu thông tin chuyên biệt (như Sổ tay sinh viên trường Đại học Giao thông Vận tải) gặp phải ba thách thức lớn:

1. **Hiện tượng ảo giác (Hallucination):** LLM có thể tự tin đưa ra các câu trả lời sai sự thật hoặc không có căn cứ khi không có đủ thông tin.

2. **Dữ liệu lỗi thời (Knowledge Cut-off):** LLM chỉ có kiến thức đến thời điểm được huấn luyện. Các quy chế, thông báo mới của nhà trường sẽ không được mô hình biết đến.
3. **Thiếu kiến thức miền riêng (Domain-specific Knowledge):** Các tài liệu nội bộ của trường UTC không nằm trong tập dữ liệu huấn luyện công khai của Google hay OpenAI.

Để giải quyết các vấn đề này, kỹ thuật **RAG (Retrieval-Augmented Generation)** đã ra đời và trở thành tiêu chuẩn cho các hệ thống hỏi đáp dựa trên tri thức (Knowledge-based Q&A).

1.3. Kỹ thuật Retrieval-Augmented Generation (RAG)

1.3.1. Khái niệm

Retrieval-Augmented Generation (RAG) là một kiến trúc kết hợp sức mạnh của mô hình ngôn ngữ lớn (LLM) với một hệ thống truy xuất thông tin (Information Retrieval). Thay vì dựa hoàn toàn vào "trí nhớ" nội tại (parametric memory) của mô hình, RAG cung cấp cho LLM khả năng tra cứu dữ liệu từ một nguồn kiến thức bên ngoài (non-parametric memory) trước khi sinh câu trả lời.

1.3.2. Kiến trúc và Quy trình hoạt động

Hệ thống RAG trong đề tài này được xây dựng dựa trên framework **LangChain**, hoạt động theo quy trình 3 bước chính:

1. **Truy xuất (Retrieval):** Khi người dùng đặt câu hỏi, hệ thống sẽ tìm kiếm các đoạn văn bản (chunks) có liên quan nhất trong cơ sở dữ liệu vector (Vector Store) chứa nội dung Sổ tay sinh viên.
2. **Tăng cường (Augmentation):** Các đoạn văn bản tìm được sẽ được ghép cùng với câu hỏi gốc và lịch sử trò chuyện để tạo thành một ngữ cảnh (Context) hoàn chỉnh.
3. **Sinh văn bản (Generation):** LLM nhận đầu vào là Prompt chứa ngữ cảnh đã được tăng cường và sinh ra câu trả lời chính xác dựa trên thông tin đó.

Trong hệ thống thực nghiệm, quy trình này được cụ thể hóa bằng chuỗi xử lý (Chain): retrieval_and_rerank | setup_and_retrieval | answer_generation.

1.4. Biểu diễn Vector và Cơ sở dữ liệu Vector (Vector Database)

1.4.1. Text Embedding

Để máy tính có thể tìm kiếm sự tương đồng về ngữ nghĩa (semantic similarity) thay vì chỉ khớp từ khóa (keyword matching), văn bản cần được chuyển đổi thành các vector số học. Quá trình này gọi là **Embedding**.

Đề tài sử dụng mô hình **bkai-foundation-models/vietnamese-bi-encoder**. Đây là mô hình embedding được tối ưu hóa riêng cho tiếng Việt, giúp nắm bắt tốt các đặc trưng ngôn ngữ và từ vựng chuyên ngành giáo dục tại Việt Nam, vượt trội hơn so với các mô hình đa ngôn ngữ thông thường.

1.4.2. Vector Store (ChromaDB)

Các vector sau khi được sinh ra cần được lưu trữ và đánh chỉ mục để truy xuất nhanh chóng. Đề tài lựa chọn **ChromaDB** – một cơ sở dữ liệu vector mã nguồn mở, hiệu năng cao, cho phép thực hiện các truy vấn tìm kiếm tương đồng (Similarity Search) dựa trên độ đo Cosine Similarity.

1.5. Truy xuất đa phương thức (Multimodal Retrieval)

Bên cạnh văn bản, thông tin trong Sổ tay sinh viên còn tồn tại dưới dạng biểu đồ, sơ đồ quy trình và hình ảnh minh họa. Các phương pháp RAG truyền thống thường bỏ qua nguồn dữ liệu này.

Để khắc phục, đề tài tích hợp mô hình **Vintern-Embedding-1B**. Đây là mô hình thị giác (Vision Model) tiên tiến dành cho tiếng Việt, có khả năng:

- Mã hóa hình ảnh thành vector đặc trưng.
- Hiểu mối liên hệ ngữ nghĩa giữa câu truy vấn văn bản và nội dung hình ảnh.

Việc kết hợp cả RAG văn bản và tìm kiếm hình ảnh tạo nên một hệ thống tra cứu toàn diện, hỗ trợ tối đa trải nghiệm của sinh viên.

Chương 2: Phân tích công nghệ và Lựa chọn mô hình.

2.1. Mô hình ngôn ngữ lớn: Google Gemini Flash

Trong kiến trúc RAG, Mô hình ngôn ngữ lớn (LLM) đóng vai trò là bộ não trung tâm, chịu trách nhiệm tổng hợp thông tin từ các tài liệu được truy xuất và sinh ra câu trả lời tự nhiên. Sau khi khảo sát các mô hình phổ biến hiện nay như GPT-4 (OpenAI), Llama 3 (Meta) và Claude 3 (Anthropic), đề tài lựa chọn dòng mô hình **Gemini** của Google, cụ thể là phiên bản **Gemini Flash**.

2.1.1. Lý do lựa chọn

Việc lựa chọn Gemini Flash dựa trên các tiêu chí kỹ thuật sau:

1. **Cửa sổ ngữ cảnh lớn (Long Context Window):** Gemini Flash hỗ trợ cửa sổ ngữ cảnh lên tới 1 triệu token. Điều này cho phép hệ thống nạp vào một lượng lớn thông tin quy chế (Context) trong một lần xử lý mà không bị cắt bớt dữ liệu, khắc phục điểm yếu của các mô hình nguồn mở nhỏ hơn.
2. **Tốc độ và Độ trễ thấp:** Hậu tố "Flash" biểu thị sự tối ưu hóa về tốc độ. Trong bài toán tra cứu thời gian thực cho sinh viên, độ trễ (latency) thấp là ưu tiên hàng đầu để đảm bảo trải nghiệm người dùng mượt mà.
3. **Khả năng lập luận (Reasoning):** Mặc dù là phiên bản tối ưu tốc độ, Gemini Flash vẫn duy trì khả năng lập luận logic tốt, giúp hiểu được các câu hỏi phức tạp hoặc nối tiếp (follow-up questions) của sinh viên.

Trong mã nguồn, mô hình được khởi tạo thông qua thư viện langchain-google-genai với tham số temperature=0.1 nhằm giảm thiểu tính ngẫu nhiên, đảm bảo câu trả lời luôn bám sát vào tài liệu được cung cấp.

2.2. Framework điều phối: LangChain

Để kết nối LLM với nguồn dữ liệu ngoài và quản lý lịch sử hội thoại, hệ thống sử dụng **LangChain** – một framework mã nguồn mở mạnh mẽ nhất hiện nay cho việc phát triển ứng dụng LLM.

2.2.1. LangChain Expression Language (LCEL)

Hệ thống tận dụng kiến trúc LCEL (LangChain Expression Language) để xây dựng các chuỗi xử lý (Chains) linh hoạt. Thay vì viết mã tuần tự (imperative), LCEL cho phép định nghĩa luồng xử lý theo dạng khai báo (declarative) với toán tử pipe (|).

Cụ thể, chuỗi xử lý RAG trong hệ thống được thiết kế bao gồm 3 module song song:

- retrieval_and_rerank: Truy xuất và sắp xếp lại tài liệu.
- setup_and_retrieval: Chuẩn bị ngữ cảnh và lịch sử chat.
- answer_generation: Sinh câu trả lời cuối cùng.

Việc sử dụng LangChain giúp hệ thống dễ dàng mở rộng, bảo trì và tích hợp các thành phần như bộ nhớ hội thoại (Chat History) mà không làm phức tạp hóa logic của ứng dụng.

2.3. Cơ sở dữ liệu Vector: ChromaDB

Trong bài toán RAG, việc tìm kiếm văn bản tương đồng yêu cầu một cơ sở dữ liệu chuyên dụng để lưu trữ các vector embeddings. Đề tài lựa chọn **ChromaDB** thay vì các giải pháp đám mây như Pinecone hay Weaviate.

2.3.1. Đặc điểm kỹ thuật

- **Mã nguồn mở và Triển khai cục bộ (Local Deployment):** ChromaDB cho phép chạy trực tiếp trên máy chủ (hoặc môi trường WSL) mà không phụ thuộc vào API bên thứ ba. Điều này giúp bảo mật dữ liệu nội bộ của nhà trường và giảm chi phí vận hành.
- **Persistent Storage:** Dữ liệu sau khi vector hóa được lưu trữ bền vững tại thư mục cục bộ (PERSIST_DIRECTORY), giúp hệ thống không phải tính toán lại embeddings mỗi khi khởi động lại.
- **Tích hợp sâu với LangChain:** ChromaDB tương thích hoàn toàn với LangChain VectorStore interface, cho phép thực hiện các truy vấn similarity_search một cách hiệu quả.

2.4. Mô hình Embedding tiếng Việt: BKAI Vietnamese Bi-Encoder

Chất lượng của hệ thống RAG phụ thuộc phần lớn vào bước truy xuất (Retrieval). Nếu vector hóa không tốt, hệ thống sẽ tìm sai tài liệu, dẫn đến câu trả lời sai. Các mô hình embedding đa ngôn ngữ (Multilingual) thường hoạt động kém hiệu quả với các ngôn ngữ có cấu trúc đặc thù như tiếng Việt.

Vì vậy, đề tài sử dụng mô hình **bkai-foundation-models/vietnamese-bi-encoder**. Đây là mô hình được phát triển bởi Viện Công nghệ thông tin Bách Khoa Hà Nội (BKAI), được huấn luyện chuyên sâu trên tập dữ liệu tiếng Việt lớn. Mô hình này giúp nắm bắt tốt hơn ngữ nghĩa của các thuật ngữ học vụ (ví dụ: "tín chỉ", "điểm rèn luyện", "học bổng khuyến khích") so với các mô hình chung của OpenAI hay Google.

2.5. Mô hình tìm kiếm đa phương thức: Vintern-1B

Một điểm đột phá của hệ thống là khả năng tìm kiếm hình ảnh dựa trên mô tả văn bản. Để thực hiện điều này, đề tài sử dụng mô hình **Vintern-Embedding-1B** (phát triển bởi 5CD-AI).

2.5.1. Kiến trúc và Cơ chế

Vintern-1B là một mô hình **VLM (Vision-Language Model)** được thiết kế để đưa cả hình ảnh và văn bản về cùng một không gian vector đa chiều (Embedding Space).

- **Cơ chế hoạt động:** Khi người dùng nhập "Sơ đồ tòa nhà A1", câu truy vấn này được chuyển thành vector V_{text}. Hệ thống sẽ so sánh V_{text} với các vector hình ảnh V_{image} đã được tính toán trước đó từ tài liệu PDF.
- **Thư viện hỗ trợ:** Hệ thống sử dụng transformers của Hugging Face để tải mô hình và torch (PyTorch) để thực hiện các phép tính toán ma trận trên CPU/GPU.

Việc sử dụng Vintern-1B giúp hệ thống vượt qua giới hạn của tìm kiếm từ khóa truyền thống, cho phép hiểu được ngữ nghĩa của hình ảnh mà không cần gán nhãn thủ công (Zero-shot retrieval).

2.6. Backend Framework: Flask

Để đóng gói các mô hình AI thành một ứng dụng web hoàn chỉnh, đề tài sử dụng **Flask**.

- **Lightweight:** Flask là một micro-framework nhẹ, linh hoạt, rất phù hợp để triển khai các ứng dụng AI (AI Inference Server) trên môi trường tài nguyên hạn chế.
- **Hệ sinh thái Python:** Do toàn bộ các thư viện AI (LangChain, PyTorch, ChromaDB) đều viết bằng Python, việc sử dụng Flask giúp tích hợp liền mạch backend và AI core mà không cần qua các cầu nối phức tạp.
- **Session Management:** Flask cung cấp cơ chế quản lý Session an toàn (FLASK_SECRET_KEY), cho phép lưu trữ lịch sử trò chuyện của từng phiên người dùng riêng biệt.

CHƯƠNG 3: THIẾT KẾ VÀ XÂY DỰNG KIẾN TRÚC HỆ THỐNG

3.1. Kiến trúc tổng thể hệ thống

Hệ thống được thiết kế theo mô hình **Client-Server** (Khách-Chủ) với kiến trúc phân lớp (Layered Architecture), giúp tách biệt rõ ràng giữa giao diện người dùng, logic nghiệp vụ và lớp dữ liệu.

Kiến trúc bao gồm 3 tầng chính:

1. **Presentation Layer (Client):** Giao diện Web tương tác với người dùng, gửi yêu cầu và hiển thị câu trả lời hoặc hình ảnh.
2. **Application Layer (Server):** Xử lý logic nghiệp vụ, bao gồm điều phối luồng RAG, quản lý phiên làm việc (Session) và giao tiếp với các mô hình AI.
3. **Data & Model Layer:** Lưu trữ dữ liệu vector (ChromaDB), các mô hình nhúng (Embedding Models) và mô hình ngôn ngữ lớn (Gemini API).

3.2. Quy trình xử lý dữ liệu đầu vào (Data Ingestion Pipeline)

Để hệ thống có khả năng trả lời chính xác, dữ liệu từ các tệp PDF "Sổ tay sinh viên" cần được xử lý và chuyển đổi sang dạng vector. Quy trình này được thực hiện chủ yếu ở phía quản trị (Admin) và bao gồm các bước sau:

3.2.1. Tải lên và Phân tách văn bản (Upload & Chunking)

- **Đầu vào:** Quản trị viên tải lên file PDF và nhập thông tin "Ngày ban hành" thông qua giao diện Admin.
- **Xử lý:**
    - Hệ thống sử dụng PyPDFLoader để đọc nội dung văn bản thô từ file PDF.
    - Sử dụng RecursiveCharacterTextSplitter để chia nhỏ văn bản thành các đoạn (chunks). Kích thước mỗi đoạn là 1000 ký tự với độ chồng lặp (overlap) 100 ký tự. Việc chồng lặp giúp giữ ngữ cảnh liên tục giữa các đoạn văn.

3.2.2. Gán nhãn dữ liệu (Metadata Tagging)

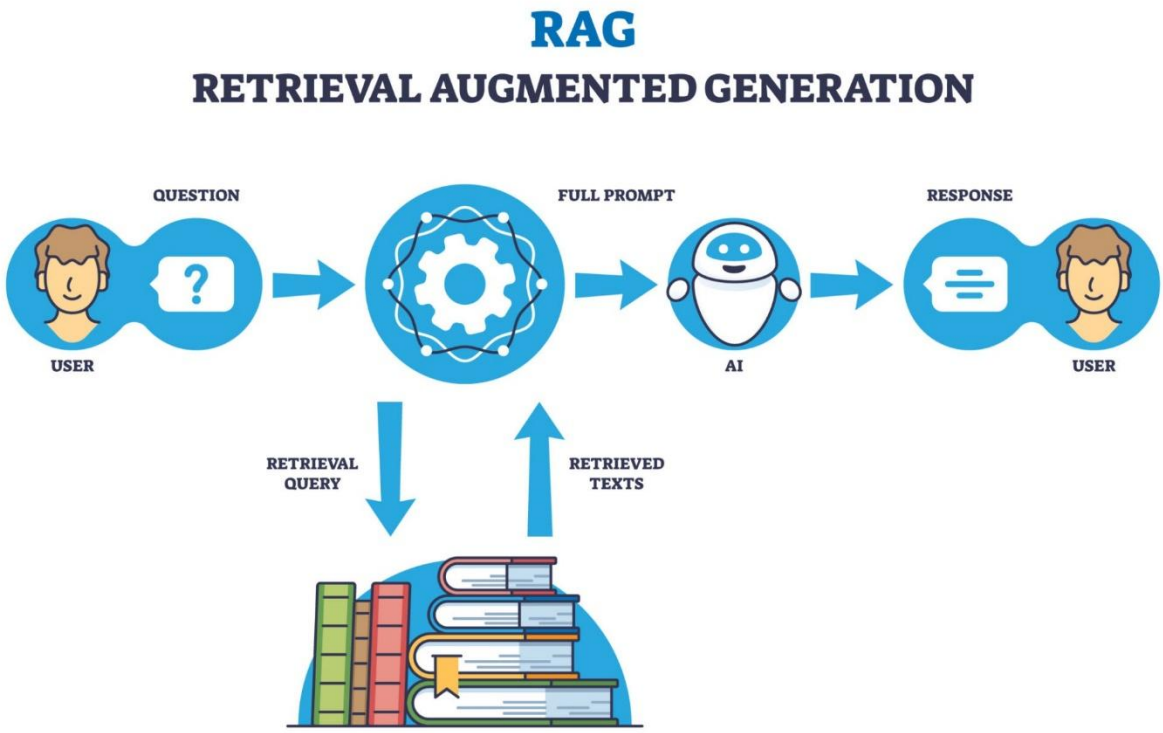
Một điểm cải tiến quan trọng của hệ thống là việc gán metadata thời gian cho từng đoạn văn bản. Biến upload_date được thêm vào metadata của mỗi chunk. Điều này phục vụ cho thuật toán Reranking sau này, giúp hệ thống phân biệt được quy chế cũ và mới.

3.2.3. Vector hóa và Lưu trữ (Embedding & Storage)

- Các đoạn văn bản được đưa qua mô hình bkai-foundation-models/vietnamese-bi-encoder để chuyển đổi thành vector số học.
- Vector và metadata tương ứng được lưu trữ bền vững vào **ChromaDB** tại thư mục cục bộ (PERSIST_DIRECTORY).

3.3. Thiết kế luồng xử lý Hỏi đáp văn bản (Text QA Workflow)

Đây là luồng xử lý chính khi sinh viên đặt câu hỏi. Hệ thống sử dụng kỹ thuật RAG kết hợp với bộ nhớ hội thoại.



Quy trình xử lý một câu hỏi diễn ra như sau:

1. **Tiếp nhận yêu cầu:** Người dùng gửi câu hỏi từ giao diện Web. Server nhận câu hỏi và lấy lịch sử trò chuyện (chat_history) từ Session hiện tại.
2. **Truy xuất (Retrieval):**
    - Hệ thống truy vấn ChromaDB để tìm 20 đoạn văn bản (k=20) có độ tương đồng cao nhất với câu hỏi.
3. **Sắp xếp lại (Reranking by Date):**
    - Các tài liệu tìm được sẽ đi qua hàm rerank_by_date. Hàm này sắp xếp các đoạn văn theo ngày ban hành (từ mới nhất đến cũ nhất) và lọc lấy 15 kết quả tốt nhất. Bước này đảm bảo thông tin trả về luôn là quy chế hiện hành.
4. **Sinh câu trả lời (Generation):**
    - Ngữ cảnh (Context) được tạo ra bằng cách ghép nội dung các đoạn văn bản đã sắp xếp.

    - Một PromptTemplate đặc tả vai trò "Trợ lý ảo UTC" được sử dụng để hướng dẫn mô hình Gemini sinh câu trả lời dựa trên Context và Lịch sử trò chuyện.
5. **Phản hồi:** Câu trả lời văn bản kèm theo nguồn tham khảo (Tên file, Số trang) được gửi trả lại Client.

3.4. Thiết kế luồng Tìm kiếm hình ảnh (Multimodal Retrieval)

Hệ thống cung cấp khả năng tìm kiếm hình ảnh (biểu đồ, sơ đồ) dựa trên mô tả văn bản.

1. **Khởi tạo:** Khi khởi động Server, hệ thống tải trước toàn bộ vector đặc trưng của các trang tài liệu (page_embeddings.npy) và danh sách ảnh gốc (page_list.pkl) vào RAM để tăng tốc độ truy xuất.
2. **Xử lý truy vấn:**
    - Câu truy vấn của người dùng được mã hóa bởi AutoProcessor và đưa qua mô hình Vintern-Embedding-1B để tạo ra vector truy vấn.
    - Hệ thống tính toán điểm tương đồng (Score) giữa vector truy vấn và toàn bộ vector ảnh trong cơ sở dữ liệu.
3. **Trả kết quả:**
    - Chọn ra top_k=2 hình ảnh có điểm số cao nhất.
    - Hình ảnh được mã hóa sang định dạng Base64 và gửi về Client để hiển thị trực tiếp trên khung chat.

3.5. Thiết kế Giao diện và Trải nghiệm người dùng (UX/UI)

Giao diện người dùng được thiết kế hướng tới sự đơn giản và hiệu quả, sử dụng công nghệ HTML5, CSS3 và JavaScript thuần.

- **Bố cục (Layout):** Sử dụng Flexbox để tạo bố cục chat linh hoạt. Thanh điều hướng (Header) chứa Logo UTC và nút Đăng nhập Admin.
- **Tương tác (Interaction):**
    - Sử dụng JavaScript fetch API để gửi yêu cầu bất đồng bộ (AJAX) đến Server, giúp trang web không bị tải lại khi gửi câu hỏi.
    - Hiệu ứng "Typing Indicator" (đang gõ) được thêm vào để báo hiệu hệ thống đang xử lý, nâng cao trải nghiệm người dùng.
    - Chế độ xem ảnh: Tích hợp Modal (cửa sổ bật lên) cho phép phóng to hình ảnh kết quả để xem rõ chi tiết sơ đồ/bảng biểu.

CHƯƠNG 4: TỐI ƯU HÓA THUẬT TOÁN TRUY XUẤT VÀ XẾP HẠNG

Trong các hệ thống RAG (Retrieval-Augmented Generation), thách thức lớn nhất không nằm ở khả năng sinh văn bản của LLM, mà nằm ở độ chính xác của dữ liệu đầu vào (Context) được truy xuất. Đối với miền dữ liệu đặc thù như Sổ tay sinh viên, nơi các quy chế thường xuyên được cập nhật, sửa đổi và bổ sung, việc truy xuất thuần túy dựa trên độ tương đồng (Similarity Search) là chưa đủ. Chương này trình bày các giải pháp tối ưu hóa thuật toán truy xuất và xếp hạng tài liệu để giải quyết bài toán trên.

4.1. Tối ưu hóa Chiến lược Phân mảnh văn bản (Chunking Strategy)

Kích thước của đoạn văn bản (Chunk size) ảnh hưởng trực tiếp đến ngữ cảnh mà mô hình nhận được. Nếu chunk quá nhỏ, thông tin bị manh mún; nếu chunk quá lớn, thông tin nhiễu sẽ làm giảm độ chính xác của câu trả lời.

Hệ thống sử dụng chiến lược **Recursive Character Text Splitter** với tham số cấu hình:

- **Chunk Size:** 1000 ký tự.
- **Chunk Overlap:** 100 ký tự.

Việc thiết lập độ chồng lặp (overlap) là 100 ký tự giúp bảo toàn ngữ nghĩa của các câu nằm ở ranh giới cắt, đảm bảo không có thông tin quan trọng nào bị mất đi giữa hai đoạn văn bản liền kề.

4.2. Thuật toán Reranking dựa trên yếu tố thời gian (Time-aware Reranking)

4.2.1. Vấn đề của Truy xuất vector thông thường

Các công cụ Vector Database mặc định xếp hạng tài liệu dựa trên điểm tương đồng Cosine (Cosine Similarity). Tuy nhiên, trong môi trường giáo dục, một quy chế "Học bổng" năm 2024 có thể có nội dung văn bản rất giống với quy chế năm 2020 (chỉ khác một vài chỉ số). Nếu chỉ dựa vào vector, hệ thống có thể truy xuất nhầm văn bản năm 2020 vì độ tương đồng ngữ nghĩa cao, dẫn đến việc cung cấp thông tin lỗi thời cho sinh viên.

4.2.2. Giải pháp Reranking tùy chỉnh

Để giải quyết vấn đề này, đề tài đã cài đặt một lớp logic trung gian (Middleware Logic) ngay sau bước truy xuất vector. Thuật toán được cài đặt trong hàm rerank_by_date hoạt động như sau:

1. **Retrieval:** Truy xuất tập hợp K = 50 tài liệu có độ tương đồng cao nhất từ ChromaDB.
2. **Metadata Parsing:** Trích xuất trường metadata date (ngày ban hành văn bản) từ các tài liệu này.
3. **Sorting:** Thực hiện sắp xếp lại danh sách tài liệu theo thứ tự thời gian giảm dần (từ mới nhất đến cũ nhất).
4. **Top-K Selection:** Chỉ giữ lại 15 tài liệu mới nhất để đưa vào ngữ cảnh (Context) cho LLM.

Đoạn mã giả (Pseudo-code) mô tả thuật toán:

```
def rerank_by_date(docs):
    # Sắp xếp document theo metadata 'date', ưu tiên ngày gần nhất
    sorted_docs = sorted(docs, key=lambda x: x.metadata['date'], reverse=True)
    # Cắt lấy top 15 document phù hợp nhất về ngữ nghĩa VÀ mới nhất về thời gian
    return sorted_docs[:15]
```

Phương pháp này đảm bảo rằng dù văn bản cũ có điểm tương đồng cao đến đâu, hệ thống vẫn ưu tiên cung cấp các quy định mới nhất cho sinh viên.

4.3. Tối ưu hóa Truy vấn Đa phương thức (Multimodal Optimization)

Hệ thống tích hợp mô hình **Vintern-Embedding-1B** để tìm kiếm hình ảnh. Do mô hình này có kích thước lớn (khoảng 1 tỷ tham số), việc tối ưu hóa hiệu năng tính toán là bắt buộc để đảm bảo tốc độ phản hồi.

4.3.1. Pre-computation (Tính toán trước)

Thay vì xử lý hình ảnh mỗi khi có truy vấn (On-the-fly), hệ thống thực hiện chiến lược tính toán trước (Offline processing):

- Toàn bộ các trang tài liệu PDF được chuyển đổi thành ảnh và đưa qua mô hình để tạo ra các vector đặc trưng (Embeddings).
- Kết quả được lưu trữ dưới dạng tệp nhị phân .npy (NumPy array) cho embeddings và .pkl (Pickle) cho danh sách ảnh.

4.3.2. In-Memory Search (Tìm kiếm trên RAM)

Khi khởi động ứng dụng (app.py), toàn bộ dữ liệu embeddings đã tính toán trước được tải trực tiếp vào bộ nhớ RAM (biến loaded_embeddings_np và image_embeddings_list_of_tensors).

Khi người dùng gửi truy vấn:

1. Chỉ có câu văn bản truy vấn cần được vector hóa (mất < 100ms).
2. Việc so khớp (tính điểm score_multi_vector) diễn ra hoàn toàn trên ma trận số học trong RAM, loại bỏ độ trễ do đọc/ghi đĩa.

Nhờ tối ưu hóa này, thời gian phản hồi cho một truy vấn hình ảnh giảm xuống dưới 1 giây (trên môi trường CPU), thay vì phải mất hàng chục giây nếu xử lý từ đầu.

4.4. Kỹ thuật Prompt Engineering (Tinh chỉnh câu lệnh)

Để kiểm soát chất lượng câu trả lời từ mô hình Gemini, hệ thống áp dụng kỹ thuật **Few-shot Prompting** kết hợp với chỉ dẫn vai trò (Role-playing).

Template (Mẫu câu lệnh) được thiết kế với các ràng buộc cứng:

- **Role definition:** "Bạn là một trợ lý ảo thân thiện và chuyên nghiệp của Trường Đại học Giao thông Vận tải (UTC)."
- **Constraints:** "Nếu ngữ cảnh mới không chứa thông tin, hãy nói: Rất tiếc, tôi không tìm thấy thông tin..." -> Giúp giảm thiểu ảo giác, ngăn chatbot bịa đặt thông tin không có trong sổ tay.
- **Context Injection:** Cấu trúc prompt chia rõ phần Lịch sử trò chuyện và Ngữ cảnh mới, giúp mô hình phân biệt được đâu là kiến thức thực tế (từ tài liệu) và đâu là luồng hội thoại.

4.5. Cơ chế quản lý phiên làm việc (Session Management)

Để tạo ra trải nghiệm hội thoại tự nhiên, hệ thống cần "nhớ" những gì người dùng vừa hỏi. Tuy nhiên, việc gửi toàn bộ lịch sử chat vào LLM sẽ làm tốn token và tăng chi phí.

Hệ thống tối ưu hóa bằng cửa sổ trượt (Sliding Window):

- Sử dụng biến k_memory = 3 trong logic xử lý.
- Chỉ lưu trữ và gửi đi 3 cặp câu hỏi-trả lời gần nhất trong session['chat_history'].

Việc giới hạn này đảm bảo LLM có đủ ngữ cảnh ngắn hạn để hiểu các câu hỏi tham chiếu (như "Còn môn kia thì sao?") nhưng không bị quá tải bởi các thông tin cũ không còn liên quan.

CHƯƠNG 5: KẾT QUẢ CÀI ĐẶT VÀ THỬ NGHIỆM HỆ THỐNG

Sau quá trình thiết kế và thực hiện, hệ thống Trợ lý ảo Sổ tay sinh viên (UTC Assistant) đã được triển khai trên môi trường giả lập Linux (WSL). Chương này trình bày chi tiết giao diện người dùng, kịch bản kiểm thử các chức năng và đánh giá hiệu năng cũng như các giới hạn hiện tại của hệ thống.

5.1. Môi trường triển khai thực nghiệm

Hệ thống được vận hành cục bộ với cấu hình như sau:

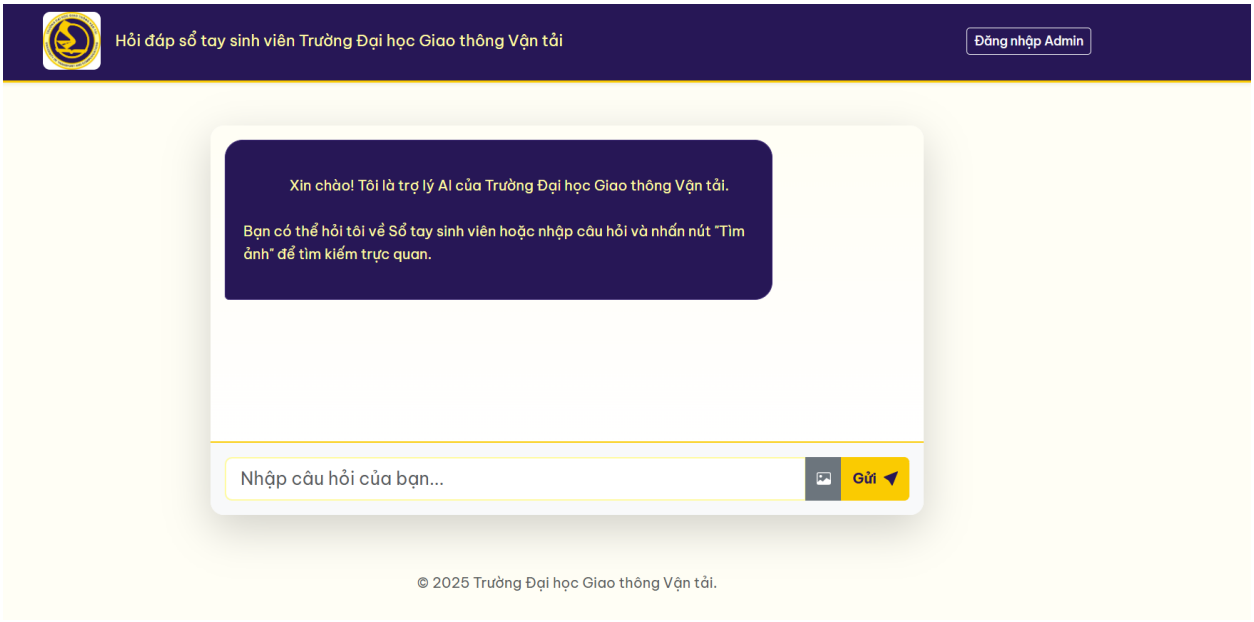
- **Hệ điều hành:** Ubuntu 22.04 chạy trên nền tảng WSL 2 (Windows Subsystem for Linux).
- **Web Server:** Flask (Python 3.10) phục vụ tại địa chỉ localhost:5000.
- **Cơ sở dữ liệu:** ChromaDB (Lưu trữ vector văn bản) và File System (Lưu trữ vector hình ảnh .npy).
- **Tài nguyên:** Hệ thống sử dụng CPU để thực thi mô hình (Inference), không sử dụng GPU rời.

5.2. Kết quả xây dựng Phân hệ Người dùng (Client)

Phân hệ này dành cho sinh viên tra cứu thông tin công khai. Giao diện được thiết kế tối giản, tập trung vào tính năng hội thoại.

5.2.1. Giao diện Trang chủ và Chatbot

Giao diện chính được bố cục với thanh Header chứa logo UTC và khu vực Chatbox rộng rãi. Màu sắc chủ đạo là xanh dương đậm (thương hiệu UTC) và vàng, giúp phân biệt rõ ràng giữa tin nhắn của Người dùng và Bot.

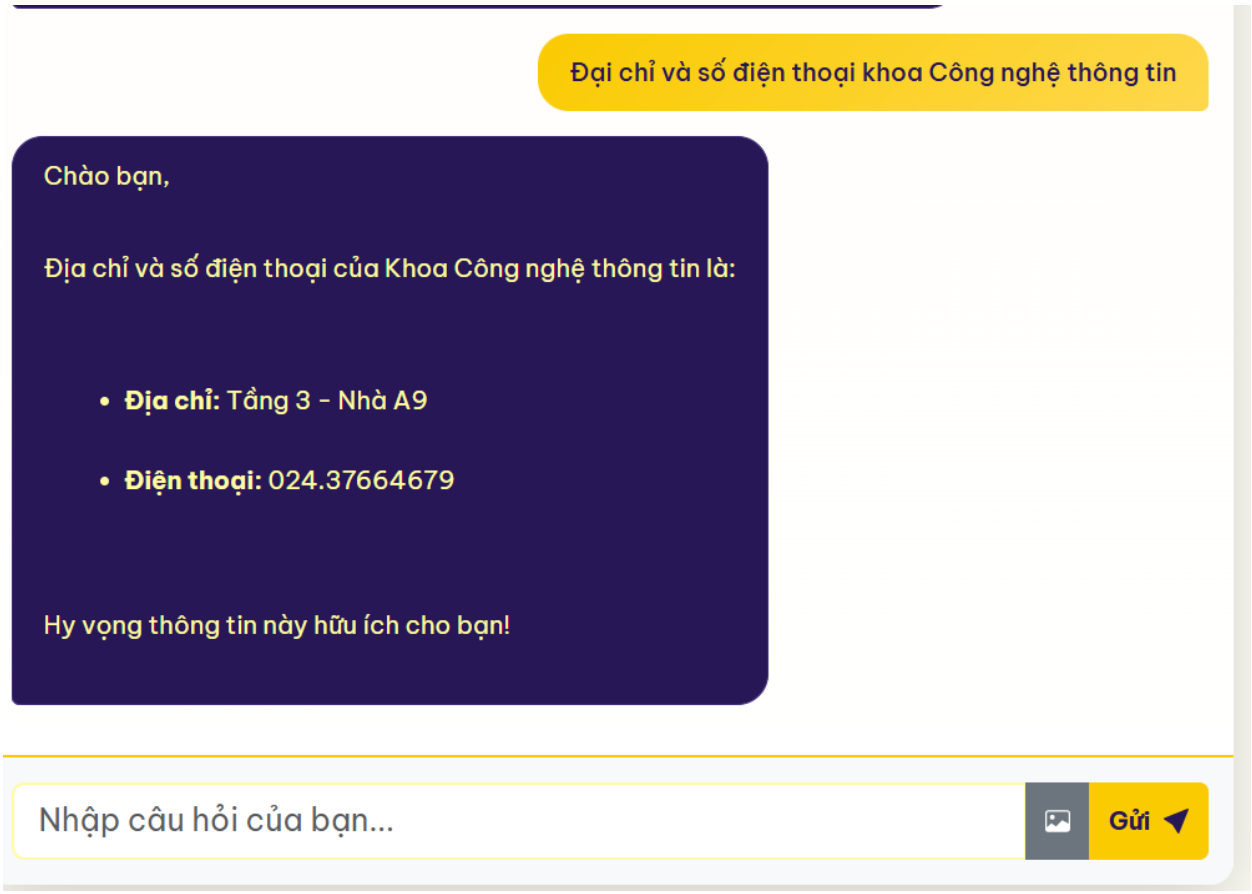


[HÌNH 5.1: Giao diện trang chủ hệ thống]

5.2.2. Thử nghiệm chức năng Hỏi đáp văn bản (Text QA)

Chức năng này sử dụng cơ chế RAG để tìm kiếm thông tin trong các file PDF đã được đánh chỉ mục.

- **Kịch bản:** Người dùng hỏi về địa chỉ các phòng ban (dựa trên thông tin file cũ) .
- **Kết quả:** Bot trả lời dựa trên văn bản, có trích dẫn tên file và số trang cụ thể. Dữ liệu văn bản được truy xuất bao gồm cả các file PDF mới nhất vừa được Admin cập nhật.



[HÌNH 5.2: Kết quả hỏi đáp văn bản có trích dẫn nguồn]

5.2.3. Thử nghiệm chức năng Tìm kiếm Hình ảnh (Image Retrieval)

Chức năng cho phép tìm kiếm các sơ đồ, biểu đồ trong tài liệu dựa trên mô tả.

- **Kịch bản:** Người dùng nhập từ khóa "Tìm các tuyến xe bus xung quanh".
- **Kết quả:** Hệ thống hiển thị ảnh thumbnail (thu nhỏ) chứa thông tin liên quan. Người dùng nhấp vào ảnh để xem phóng to (Image Modal).

Tìm các tuyến xe bus xung quanh

Trang 91 (Tương đồng: 17.6402)

CÁC TUYẾN XE BUÝT ĐẾN TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

- **26** Sân vận động quốc gia - Mai Động
- **27** Nam Thăng Long - Bến xe Yên Nghĩa
- **28** Đại học Mỏ - Bến xe Nước Ngầm
- **32** Nhổn - Bến xe Giáp Bát
- **34** Bến xe Mỹ Đình - TTHC huyện Gia Lâm
- **38** Nam Thăng Long - Mai Động
- **49** Nhổn - Trần Khánh Dư
- **E05** Long Biên - Khu đô thị Smart City
- **09A** Đại học Mỏ - Bờ Hồ
  (chạy từ thứ 2 đến 17h00 thứ 6)
- **09A CT** Đại học Mỏ - Trần Khánh Dư
  (chạy từ 17h00 thứ 6 đến hết Chủ nhật)
- **90** Sân bay Nội Bài - Hào Nam
- **146** Hào Nam - Nguyễn Thái Học - Hào Nam

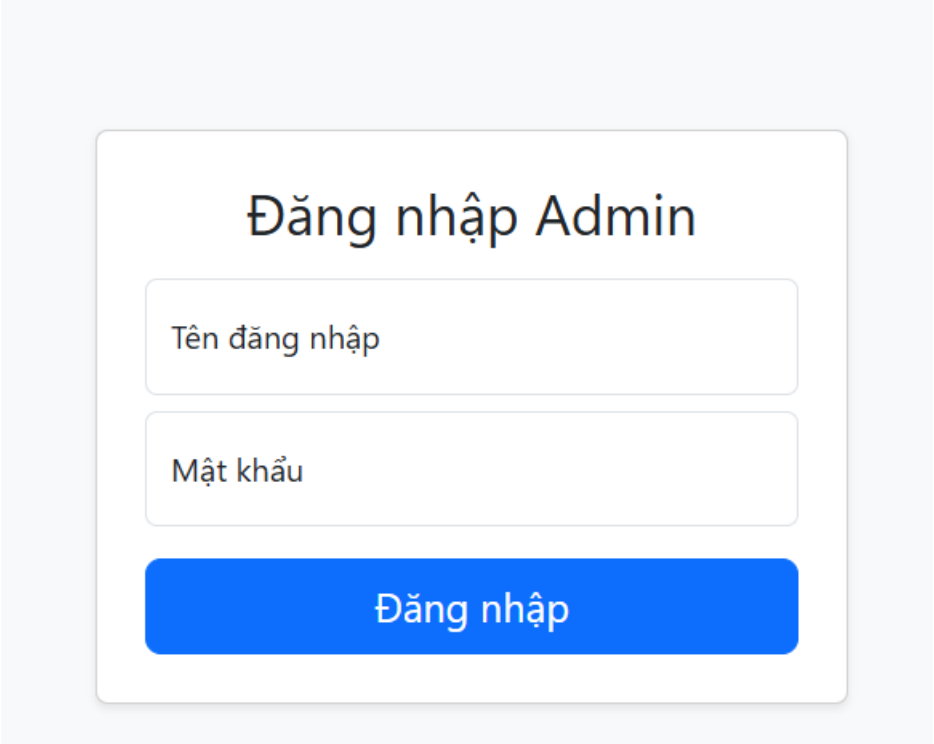
[HÌNH 5.3: Kết quả tìm kiếm hình ảnh và Giao diện phóng to

5.3. Kết quả xây dựng Phân hệ Quản trị (Admin)

Phân hệ này cung cấp quyền cập nhật tri thức cho hệ thống, được bảo vệ bằng cơ chế đăng nhập.

5.3.1. Đăng nhập và Xác thực

Admin truy cập thông qua đường dẫn /login hoặc nút trên thanh Header. Hệ thống kiểm tra tài khoản/mật khẩu từ biến môi trường server.

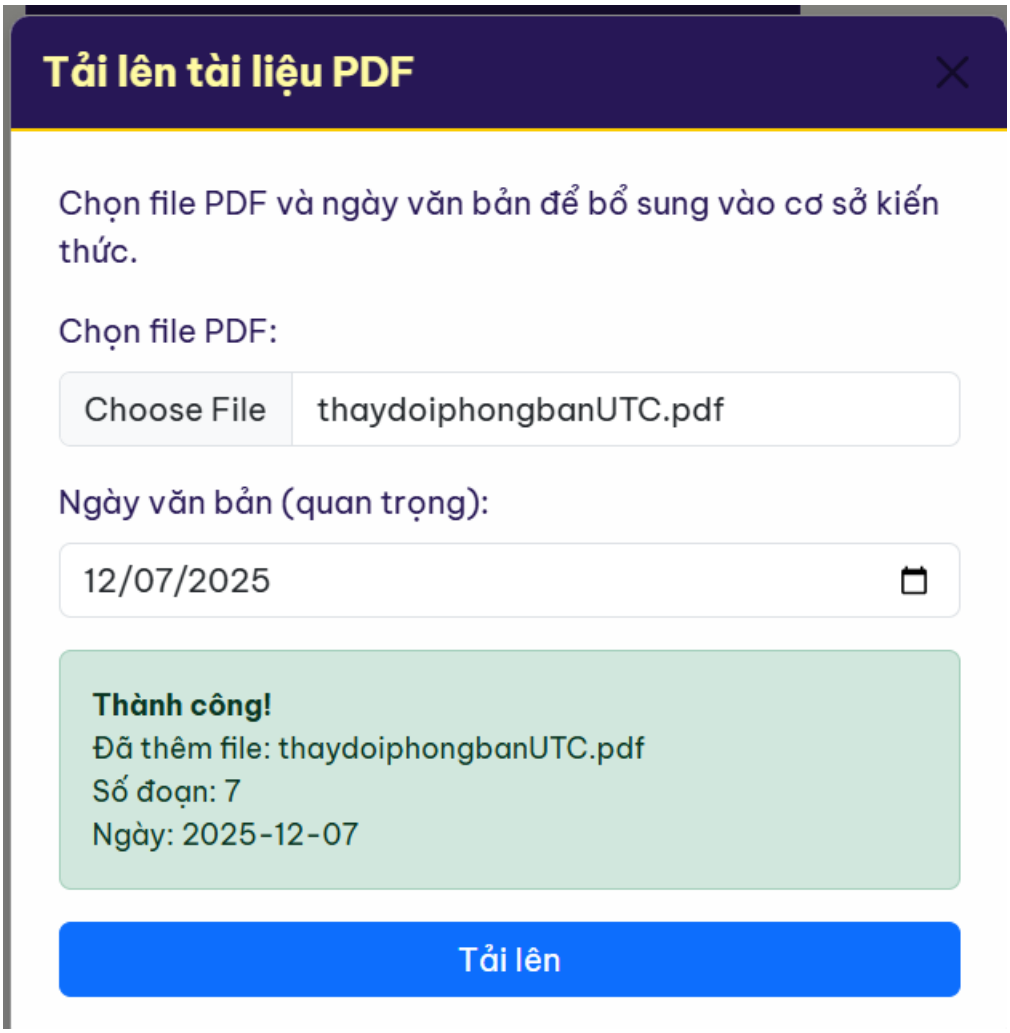


[HÌNH 5.4: Giao diện đăng nhập Admin]

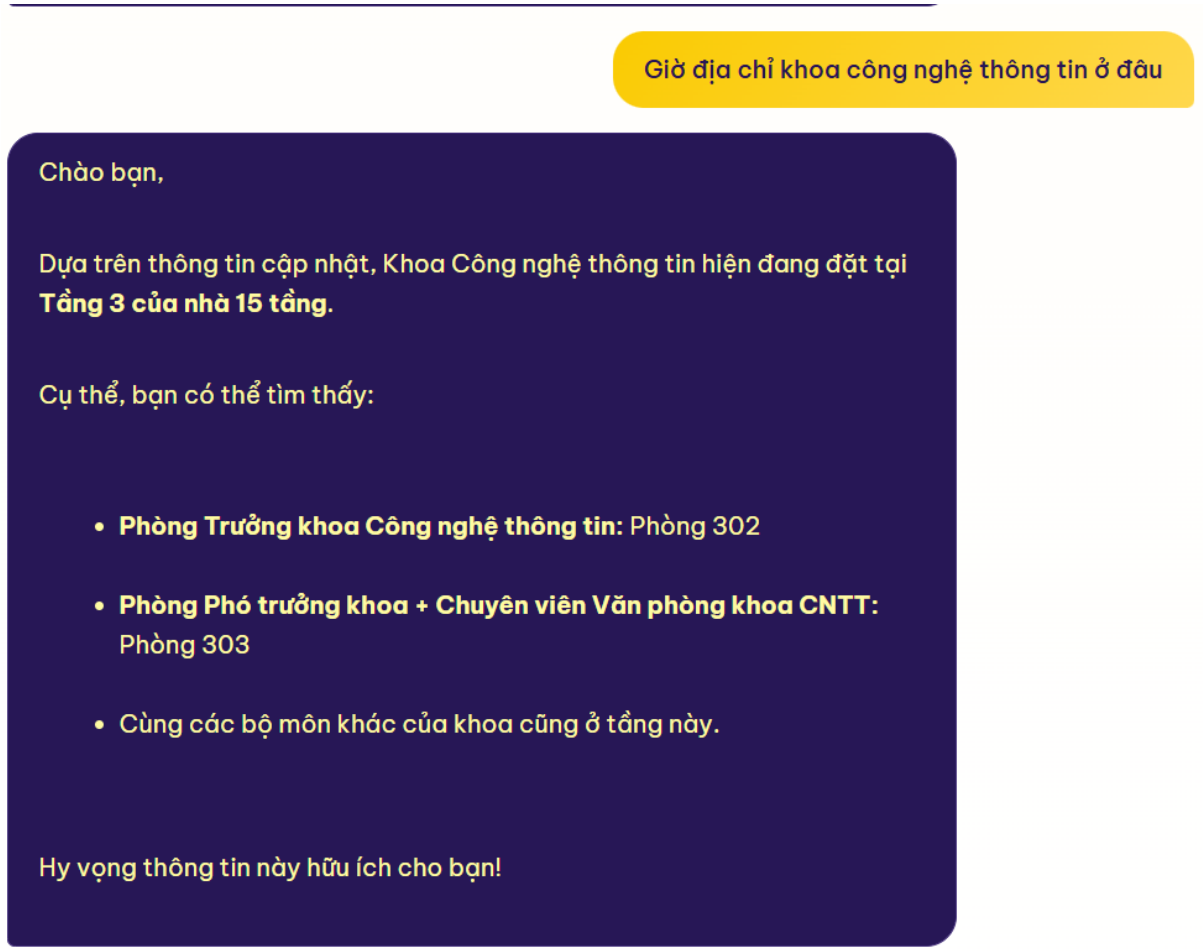
5.3.2. Chức năng Cập nhật tài liệu (Text Update)

Admin có thể tải lên file PDF mới kèm theo thông tin "Ngày ban hành".

- **Cơ chế:** Ngay khi tải lên thành công, hệ thống tự động phân mảnh (chunking) và thêm vector văn bản vào ChromaDB.
- **Kết quả:** Dữ liệu **văn bản** có hiệu lực tức thì. Sinh viên có thể hỏi về nội dung văn bản vừa upload ngay lập tức mà không cần khởi động lại server.



[HÌNH 5.5: Giao diện Upload PDF thành công]



[HÌNH 5.6: Tra thông tin theo dữ liệu mới]

5.4. Đánh giá và Phân tích giới hạn hệ thống

5.4.1. Đánh giá hiệu năng

- **Tốc độ phản hồi:** Trên môi trường CPU, thời gian trả lời câu hỏi văn bản trung bình từ 3-5 giây. Tìm kiếm hình ảnh mất khoảng 1-2 giây do sử dụng dữ liệu tính toán trước.
- **Độ chính xác:** Hệ thống ưu tiên tốt các văn bản mới nhất nhờ thuật toán Reranking theo thời gian.

5.4.2. Giới hạn cập nhật dữ liệu hình ảnh (Image Update Constraint)

Bên cạnh các kết quả đạt được, hệ thống hiện tại tồn tại một hạn chế kỹ thuật trong quy trình cập nhật dữ liệu hình ảnh:

- **Vấn đề:** Khi Admin tải lên file PDF mới, hệ thống **chỉ cập nhật dữ liệu văn bản** (Text Embeddings) để hỏi đáp ngay lập tức. Dữ liệu hình ảnh của file mới **chưa được cập nhật** vào tính năng tìm kiếm ảnh.
- **Nguyên nhân kỹ thuật:**
    1. **Kích thước mô hình:** Hệ thống sử dụng mô hình đa phương thức **Vintern-Embedding-1B** (khoảng 1 tỷ tham số). Việc trích xuất đặc trưng (Feature Extraction) cho toàn bộ các trang của một file PDF lớn đòi hỏi tài nguyên tính toán cao và thời gian xử lý dài.
    2. **Cơ chế tải trước (Pre-loading):** Để đảm bảo tốc độ tìm kiếm nhanh cho người dùng (Client), toàn bộ vector hình ảnh được nạp vào RAM ngay khi khởi động ứng dụng (loaded_embeddings_np). Việc cập nhật nóng (Hot-reload) tập dữ liệu lớn này trên môi trường giới hạn tài nguyên (WSL/CPU) có nguy cơ gây gián đoạn dịch vụ hoặc tràn bộ nhớ.
- **Giải pháp hiện tại:** Việc cập nhật chỉ mục hình ảnh đang được thực hiện theo cơ chế **Offline Batch Processing** (Xử lý lô ngoại tuyến). Admin cần chạy lại script huấn luyện và khởi động lại dịch vụ để dữ liệu ảnh mới có hiệu lực.

5.5. Kết luận

Chương 5 đã trình bày toàn cảnh kết quả triển khai hệ thống UTC Assistant. Hệ thống đáp ứng tốt các yêu cầu cơ bản về tra cứu văn bản thời gian thực và tìm kiếm hình ảnh trên dữ liệu có sẵn. Tuy nhiên, hạn chế về khả năng cập nhật hình ảnh thời gian thực (Real-time) đã được nhận diện và phân tích nguyên nhân, mở ra hướng nghiên cứu tối ưu hóa trong tương lai.

Tài liệu tham khảo

[1] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *NeurIPS*, 2020.

[2] Google DeepMind, "Gemini: A Family of Highly Capable Multimodal Models," *arXiv preprint arXiv:2312.11805*, 2023.

[3] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proceedings of EMNLP*, 2019.

[4] BKAI Foundation, "Vietnamese Bi-encoder," Hugging Face, 2024. [Online]. Available: https://huggingface.co/bkai-foundation-models/vietnamese-bi-encoder.

[5] 5CD-AI, "Vintern-Embedding-1B," Hugging Face, 2024. [Online]. Available: https://huggingface.co/5CD-AI/Vintern-Embedding-1B.

[6] T. Dettmers et al., "LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale," *arXiv preprint arXiv:2208.07339*, 2022.

[7] LangChain, "Introduction to LangChain," 2024. [Online]. Available: https://python.langchain.com/.