BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

BÁO CÁO ĐỒ ÁN TỐT NGHIỆP

NGÀNH KỸ THUẬT PHẦN MỀM

ĐỀ TÀI

NGHIÊN CỨU VÀ XÂY DỰNG CHATBOT ỨNG DỤNG TRÍ TUỆ NHÂN TẠO HỖ TRỢ HOẠT ĐỘNG TẠI PHÒNG ĐÀO TẠO TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

Giảng viên hướng dẫn	:	TS. Vũ Việt Thắng
Sinh viên	:	Hoàng Phúc Lâm
Mã sinh viên	:	2021606977
Lớp	:	2021DHKTPM04

Hà Nội - 06/2025

BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

BÁO CÁO ĐỒ ÁN TỐT NGHIỆP

NGÀNH KỸ THUẬT PHẦN MỀM

ĐỀ TÀI

NGHIÊN CỨU VÀ XÂY DỰNG CHATBOT ỨNG DỤNG TRÍ TUỆ NHÂN TẠO HỖ TRỢ HOẠT ĐỘNG TẠI PHÒNG ĐÀO TẠO TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

Giảng viên hướng dẫn	:	TS. Vũ Việt Thắng
Sinh viên	:	Hoàng Phúc Lâm
Mã sinh viên	:	2021606977
Lớp	:	2021DHKTPM04

Hà Nội - 06/2025

LỜI CẢM ƠN

Đồ án tốt nghiệp không chỉ là cơ hội để em vận dụng những kiến thức lý thuyết đã học vào thực tiễn mà còn là một hành trình rèn luyện tư duy phản biện, khả năng tự học, tự nghiên cứu và giải quyết vấn đề một cách hệ thống. Trong suốt quá trình thực hiện đề tài, em không chỉ được tiếp cận với các công nghệ mới và phương pháp luận hiện đại, mà còn học hỏi được nhiều điều quý báu về tinh thần trách nhiệm, tác phong làm việc chuyên nghiệp và khả năng làm việc độc lập trong môi trường học thuật.

Em xin bày tỏ lòng biết ơn chân thành và sâu sắc tới Ban Giám hiệu Trường Công nghệ Thông tin và Truyền thông - Trường Đại học Công nghiệp Hà Nội đã tạo điều kiện thuận lợi về cơ sở vật chất và môi trường học tập để em có thể hoàn thành đồ án này. Đặc biệt, em xin gửi lời cảm ơn trân trọng tới thầy Vũ Việt Thắng - người đã luôn tận tình định hướng, hỗ trợ chuyên môn và truyền cảm hứng trong suốt quá trình thực hiện đồ án. Những nhận xét sâu sắc và góp ý quý báu từ thầy không chỉ giúp em hoàn thiện đề tài, mà còn góp phần nâng cao tư duy học thuật và năng lực chuyên môn của bản thân.

Dù đã nỗ lực thực hiện với tinh thần nghiêm túc và trách nhiệm, em vẫn ý thức được rằng do hạn chế về thời gian và kinh nghiệm, đồ án không tránh khỏi những thiếu sót. Em rất mong nhận được những ý kiến đóng góp chân thành từ thầy cô và bạn đọc để có thể tiếp tục hoàn thiện đề tài trong tương lai.

Em xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

PHẦN I. MỞ ĐẦU

1. LÝ DO CHỌN ĐỀ TÀI

Trong bối cảnh cách mạng công nghiệp 4.0, các công nghệ trí tuệ nhân tạo (AI), học máy và xử lý ngôn ngữ tự nhiên (NLP) ngày càng cho thấy sức mạnh thay đổi căn bản cách thức vận hành và tương tác giữa người dùng với hệ thống thông tin. Tại Trường Đại học Công nghiệp Hà Nội, Phòng Đào tạo chịu trách nhiệm tiếp nhận, xử lý và phản hồi hàng loạt yêu cầu từ sinh viên về tuyển sinh, ghi danh, thời khóa biểu, lịch thi, kết quả học tập, thủ tục chuyển ngành… Với khối lượng thông tin lớn và tính chất lặp đi lặp lại của các câu hỏi, đội ngũ cán bộ phòng đào tạo phải dành phần lớn thời gian cho công tác đọc, phân loại và tìm kiếm thông tin để trả lời thủ công. Điều này không chỉ gây quá tải về nhân lực mà còn kéo theo hàng loạt nhược điểm: thông tin có thể bị trình bày không đồng nhất do nhiều người trả lời khác nhau, thời gian và khung giờ phục vụ bị giới hạn trong giờ hành chính, và các hình thức hướng dẫn truyền thống - như hòm thư chung, sổ tay hướng thường khó cập nhật kịp thời, không tận dụng được dữ liệu lịch sử để cá nhân hóa trải nghiệm người dùng.

Trước thực trạng đó, việc nghiên cứu và xây dựng một hệ thống ChatBot thông minh ứng dụng AI - NLP trở nên cấp thiết và đầy tiềm năng. ChatBot không chỉ tự động hoá phần lớn các câu trả lời cơ bản mà còn có khả năng học hỏi từ các tương tác, từ đó cải thiện dần chất lượng phản hồi và phân luồng những tình huống phức tạp đến đúng chuyên môn. Hệ thống này có thể xử lý hàng nghìn truy vấn đồng thời, giải phóng nguồn lực con người để cán bộ đào tạo tập trung vào các nhiệm vụ chuyên sâu hơn như hoạch định chính sách đào tạo và phân tích dữ liệu sinh viên; đồng thời bảo đảm tính nhất quán và chính xác nhờ sử dụng cơ sở dữ liệu chính thống, nâng cao uy tín trong giao tiếp với sinh viên.

Bên cạnh đó, ChatBot hoạt động 24/7, giúp sinh viên truy cập thông tin bất cứ lúc nào, giảm thiểu thời gian chờ đợi và đáp ứng nhu cầu của những bạn ở xa hoặc bận rộn. Hệ thống còn thu thập nhật ký tương tác để phân tích xu hướng, nhu cầu và điểm nghẽn trong quy trình phục vụ, từ đó đề xuất cải tiến và cá nhân hóa nội dung cho từng nhóm đối tượng như tân sinh viên, học viên cao học hay sinh viên quốc tế. Được xây dựng trên nền tảng RAG (Retrieval-Augmented Generation) kết hợp với mô hình ngôn ngữ lớn (LLM), giải pháp này dễ dàng mở rộng và tích hợp thêm các kênh giao tiếp như Messenger, Zalo hay cổng thông tin web, đồng thời triển khai các tính năng mới như nhắc lịch thi, thông báo thời khóa biểu mà không cần thiết kế lại toàn bộ hệ thống.

Tóm lại, đề tài "Nghiên cứu và xây dựng ChatBot ứng dụng trí tuệ nhân tạo hỗ trợ hoạt động tại Phòng Đào tạo Trường Đại học Công nghiệp Hà Nội" không chỉ giải quyết vấn đề thực tiễn cấp bách về hiệu quả và chất lượng phục vụ sinh viên mà còn mang tính khoa học cao, khi kết hợp giữa công nghệ AI tiên tiến và nhu cầu quản lý giáo dục, đồng thời mở ra cơ hội cho các nghiên cứu ứng dụng AI tại các đơn vị hành chính - giáo dục trong tương lai.

2. MỤC TIÊU CỦA ĐỀ TÀI

- Nghiên cứu các phương pháp xây dựng ChatBot sử dụng trí tuệ nhân tạo, đặc biệt là mô hình RAG (Retrieval-Augmented Generation).
- Thu thập, xử lý và tổ chức dữ liệu từ trang web tuyển sinh và đào tạo của trường một cách có cấu trúc.
- Xây dựng hệ thống ChatBot có khả năng hiểu ngữ nghĩa truy vấn của người dùng và đưa ra câu trả lời chính xác, dựa trên dữ liệu đã được lập chỉ mục.
- Tích hợp và triển khai sản phẩm trên giao diện web sử dụng Streamlit để người dùng có thể dễ dàng tương tác.

3. ĐỐI TƯỢNG VÀ PHẠM VI

a. Đối tượng

- Vấn đề quản lý thông tin và giao tiếp tự động tại Phòng Đào tạo Trường Đại học Công nghiệp Hà Nội.
- Các công nghệ ChatBot, mô hình ngôn ngữ lớn (LLM) và phương pháp kết hợp tìm kiếm - sinh câu trả lời (Retrieval-Augmented Generation - RAG).
- Các thư viện mã nguồn mở hỗ trợ xử lý ngôn ngữ tự nhiên, tạo vector database và triển khai giao diện người dùng.
- Các mô hình ngôn ngữ tiên tiến được ứng dụng để tạo câu trả lời tự động dựa trên truy vấn của người dùng.

b. Phạm vi

- Thu thập và xử lý dữ liệu từ trang tuyển sinh và thư viện tài liệu đào tạo của trường.
- Xây dựng hệ thống ChatBot tích hợp mô hình RAG và LLM để hiểu và phản hồi các truy vấn liên quan đến tuyển sinh, đào tạo …
- Thiết kế quy trình gồm các bước: thu thập, tiền xử lý, chunking, embedding, tìm kiếm ngữ nghĩa, sinh câu trả lời, hiển thị trên giao diện.

4. NỘI DUNG ĐỀ TÀI

Nội dung nghiên cứu của đề tài tập trung vào việc khảo sát thực trạng và nhu cầu tự động hóa trả lời câu hỏi tại Phòng Đào tạo Trường Đại học Công nghiệp Hà Nội thông qua phỏng vấn cán bộ, phân tích nhật ký tương tác và xác định các nhóm câu hỏi phổ biến. Trên cơ sở đó, đề tài nghiên cứu và đánh giá các giải pháp công nghệ chatbot hiện có, bao gồm các framework như Rasa, Botpress..., đồng thời tìm hiểu các công trình liên quan đến mô hình RAG (Retrieval-Augmented Generation) và LLM (Large Language Model). Tiếp theo, đề tài tiến hành thu thập và xử lý dữ liệu từ các nguồn như trang tuyển sinh, thư viện tài liệu đào tạo; xây dựng tập lệnh Python để crawl, làm sạch và tách đoạn văn bản. Sau đó, dữ liệu được mã hóa thành vector bằng kỹ thuật embedding và lưu trữ trong cơ sở dữ liệu vector kèm theo thông tin ngữ cảnh. Dựa trên dữ liệu này, đề tài thiết kế pipeline kết hợp tìm kiếm ngữ nghĩa và sinh câu trả lời thông qua mô hình RAG, tích hợp với mô hình ngôn ngữ lớn để tạo ra phản hồi phù hợp. Giao diện người dùng được triển khai bằng Streamlit nhằm đảm bảo tính thân thiện và dễ sử dụng. Cuối cùng, hệ thống được kiểm thử thông qua bộ câu hỏi chuẩn và khảo sát phản hồi người dùng để đánh giá độ chính xác, tốc độ phản hồi cũng như mức độ hài lòng, từ đó rút ra các bài học và đề xuất cải tiến.

5. PHƯƠNG PHÁP THỰC HIỆN

a. Nghiên cứu lý thuyết

- Nghiên cứu khái niệm, phân loại (rule-based, retrieval-based, generative), lịch sử phát triển và ứng dụng của ChatBot trong giáo dục.
- Tìm hiểu về Large Language Models (LLM) và phương pháp Retrieval-Augmented Generation (RAG), so sánh ưu - nhược điểm với các giải pháp thuần retrieval hoặc thuần generation.
- Đánh giá, so sánh các framework ChatBot (Rasa, Botpress, triển khai thủ công qua API) và các thư viện xử lý ngôn ngữ tự nhiên (NLTK, spaCy), embedding và tìm kiếm vector (FAISS, Milvus).
- Nghiên cứu các case study, công trình khoa học và báo cáo dự án tương tự trong lĩnh vực giáo dục để rút kinh nghiệm triển khai pipeline RAG và tối ưu prompt.

b. Nghiên cứu thực nghiệm

Cài đặt thực nghiệm Chatbot trên tập dữ liệu về tuyển sinh trường Đại học Công nghiệp Hà Nội và phân tích, đánh giá kết quả.

6. CẤU TRÚC CỦA BÁO CÁO

Nội dung báo cáo được trình bày với các phần chính sau:

Phần I: Mở đầu

Phần II: Nội dung báo cáo

Chương 1: Tổng quan cơ sở lý thuyết

Chương này trình bày các khái niệm nền tảng liên quan đến mô hình ngôn ngữ lớn (LLM), kỹ thuật Retrieval-Augmented Generation (RAG), embedding cho văn bản tiếng Việt và các phương pháp phân đoạn văn bản. Các nội dung này là cơ sở lý thuyết quan trọng để xây dựng và triển khai hệ thống Chatbot trong các chương sau.

Chương 2: Phương pháp thực nghiệm

Nội dung chương 2 sẽ trình bày toàn bộ quy trình xây dựng hệ thống Chatbot sử dụng kiến trúc Retrieval-Augmented Generation (RAG), bao gồm thiết lập thực nghiệm, lựa chọn mô hình và các bước tiến hành chi tiết từ thu thập, xử lý đến lập chỉ mục dữ liệu. Ngoài ra, chương cũng mô tả cách thiết kế luồng xử lý RAG cùng phương pháp đánh giá kết quả thông qua các tiêu chí phù hợp. Cuối cùng, các công cụ và thư viện được sử dụng trong quá trình triển khai hệ thống cũng được liệt kê và phân tích, làm cơ sở cho việc đánh giá và triển khai mô hình một cách hiệu quả.

Chương 3: Kết quả và thảo luận

Trong chương này trình bày kết quả thực nghiệm và đánh giá hiệu suất của hệ thống Chatbot sử dụng kiến trúc Retrieval-Augmented Generation (RAG) được phát triển trong khuôn khổ đề tài. Nội dung chương bao gồm việc mô tả kết quả triển khai chương trình, giao diện người dùng, và quá trình đánh giá chất lượng hệ thống thông qua bộ công cụ RAGAS. Đồng thời, chương cũng phân tích các ưu điểm, hạn chế hiện tại và đề xuất các hướng cải tiến nhằm nâng cao hiệu quả và khả năng mở rộng của hệ thống trong tương lai.

Phần III: Kết luận và kiến nghị

PHẦN II. NỘI DUNG

CHƯƠNG 1. TỔNG QUAN CƠ SỞ LÝ THUYẾT

1.1. MÔ HÌNH NGÔN NGỮ LỚN

1.1.1. Tổng quan về mô hình ngôn ngữ lớn

Mô hình ngôn ngữ lớn (Large Language Model – LLM) là một thuật toán học sâu có thể thực hiện nhiều tác vụ xử lý ngôn ngữ tự nhiên (NLP). Các mô hình ngôn ngữ lớn sử dụng kiến trúc Transformer và được huấn luyện trên các tập dữ liệu khổng lồ - vì vậy được gọi là "lớn". Điều này cho phép chúng nhận diện, dịch, dự đoán hoặc tạo ra văn bản hay nội dung khác.

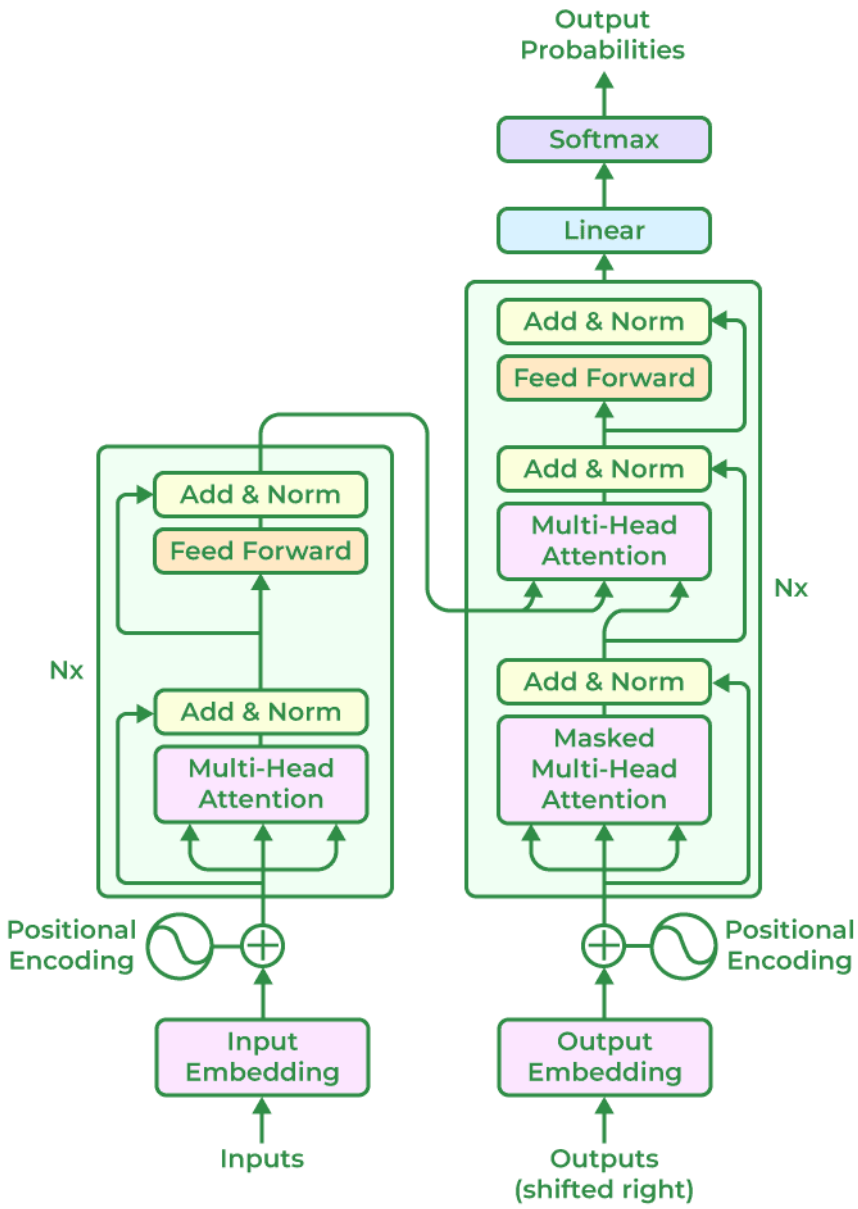
Hành trình phát triển mô hình ngôn ngữ lớn (LLM) bắt đầu từ những năm 1960 với Eliza và SHRDLU tại MIT, sử dụng kỹ thuật khớp mẫu đơn giản để mô phỏng đối thoại cơ bản. Đến cuối thập niên 1980, sự ra đời của mạng nơ-ron hồi tiếp (RNN) mở ra khả năng xử lý chuỗi dữ liệu, song phải đợi đến năm 1997 với Long Short-Term Memory (LSTM) mới khắc phục được vấn đề học lâu dài trong câu. Đến thập niên 2010, các bộ công cụ như Stanford CoreNLP và các tiến bộ về nhúng từ (word embeddings) đã chuẩn bị nền tảng cho một cuộc cách mạng thực sự. Bước ngoặt lớn nhất diễn ra vào năm 2017 với kiến trúc Transformer, cho phép xây dựng những mô hình sâu hơn, mạnh mẽ hơn. Và trong thập niên 2020, GPT-3 cùng các nền tảng như Hugging Face hay BARD đã nâng khả năng hiểu và sinh ngôn ngữ của máy lên tầm cao mới, kích thích hàng loạt ứng dụng AI đa dạng và phong phú.

1.1.2. Cách hoạt động của các mô hình ngôn ngữ lớn

Mô hình ngôn ngữ lớn (LLM) vận hành dựa trên một chuỗi các bước liên kết chặt chẽ:

- Nền tảng dữ liệu (foundation of data**):** LLM là kết quả của quá trình huấn luyện trên các bộ dữ liệu khổng lồ, thường bao gồm hàng petabyte văn bản. Dữ liệu này tạo thành nền móng mà LLM xây dựng khả năng ngôn ngữ của mình. Quá trình huấn luyện chủ yếu áp dụng phương pháp học không giám sát.
- Học từ vựng (word learning): Cốt lõi của LLMs nằm ở khả năng hiểu từ và các mối quan hệ phức tạp giữa chúng. Thông qua học không giám sát, LLMs bắt đầu hành trình khám phá từ, hiểu từ không chỉ riêng lẻ mà còn trong ngữ cảnh của câu và đoạn văn.

- Kiến trúc Transformer: Sự hình thành của mô hình ngôn ngữ lớn (LLMs) có thể được truy vết về kiến trúc transformer đột phá, một bước tiến quan trọng trong xử lý ngôn ngữ tự nhiên. Tại trái tim của đổi mới này là cơ chế chú ý, một khối xây dựng cơ bản đã tái định nghĩa cách mô hình hiểu và xử lý thông tin ngữ cảnh trong lượng lớn văn bản, thúc đẩy một sự chuyển đổi mô hình trong biểu diễn và hiểu ngôn ngữ.



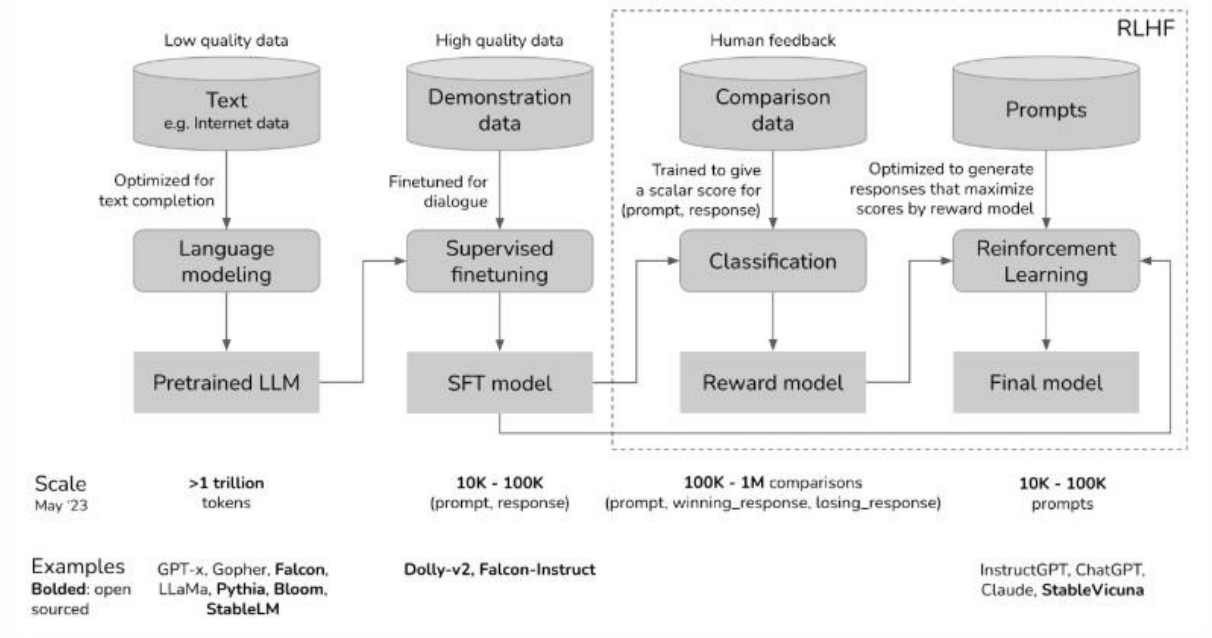
Hình 1.1. Kiến trúc Transformer

- Nhúng từ (word embedding): LLMs khởi đầu hành trình với nhúng từ, biểu diễn từ dưới dạng vector không gian nhiều chiều. Sự biến đổi này giúp nhóm các từ tương tự lại với nhau, hỗ trợ việc hiểu ngữ cảnh.

- Mã hóa vị trí (positional encoding): Đảm bảo mô hình nhận biết thứ tự từ và vị trí là rất quan trọng cho các nhiệm vụ như dịch thuật và tóm tắt. Nó không đi sâu vào nghĩa của từ mà chỉ theo dõi cấu trúc chuỗi.
- Tùy biến (customization): LLMs khả năng thích ứng – với việc tinh chỉnh (fine-tuning) và kỹ thuật thiết kế lời nhắc (prompt engineering) cho phép điều chỉnh chúng để phục vụ các mục đích cụ thể. Ví dụ, Salesforce Einstein GPT cá nhân hóa tương tác với khách hàng để nâng cao hành trình bán hàng và tiếp thị.
- Sinh văn bản (text generation): Đỉnh cao của khả năng LLMs nằm ở sinh văn bản. Sau quá trình huấn luyện và tinh chỉnh kỹ lưỡng, các mô hình này có thể tạo ra những phản hồi tinh vi dựa trên prompt. Tự hồi quy (autoregression), kỹ thuật sinh văn bản từng từ một, đảm bảo phản hồi phù hợp ngữ cảnh và mạch lạc.
- Các vấn đề đạo đức (ethical concerns): LLMs không tránh khỏi thách thức. Các cân nhắc đạo đức, bao gồm giảm thiểu thành kiến và khả năng giải thích, vẫn là những lĩnh vực nghiên cứu đang tiếp tục. Thành kiến, đặc biệt, phát sinh từ dữ liệu huấn luyện và có thể dẫn đến sự ưu tiên không công bằng trong kết quả của mô hình.

1.1.3. Quy trình huấn luyện một mô hình ngôn ngữ lớn

Mô hình ngôn ngữ lớn (large language model – LLM) hoạt động nhờ việc học cách dự đoán từ tiếp theo trong chuỗi văn bản dựa trên ngữ cảnh trước đó. Việc huấn luyện một mô hình ngôn ngữ lớn đòi hỏi một quy trình chặt chẽ, bao gồm ba giai đoạn chính: chuẩn bị dữ liệu (data Preparation), huấn luyện sơ bộ (pre-training) và huấn luyện hậu kỳ (post-training). Mỗi giai đoạn đảm bảo mô hình có thể học được kiến thức ngôn ngữ từ cấp độ nền tảng cho đến đáp ứng các yêu cầu cụ thể của ứng dụng thực tế.



Hình 1.2. Quy trình huấn luyện LLM

1.1.3.1. Chuẩn bị dữ liệu

Trong giai đoạn chuẩn bị dữ liệu, ý tưởng căn bản là tận dụng toàn bộ dữ liệu internet sạch để làm nguồn huấn luyện, tuy nhiên internet thực tế chứa rất nhiều dữ liệu không mong muốn và không đảm bảo cho chất lượng cần thiết. Đầu tiên, dữ liệu được thu thập quy mô lớn - ví dụ một bản sao Common Crawl với hơn 250 tỷ trang web, dung lượng vượt quá 1 PB - tiếp đó tiến hành trích xuất văn bản từ HTML, vượt qua những thách thức như công thức toán học hay boilerplate không liên quan. Tiếp theo, các đoạn nội dung độc hại (NSFW), thông tin cá nhân nhạy cảm (PII) hay quảng cáo gây nhiễu sẽ bị loại bỏ. Để tránh lặp lại, dữ liệu được khử trùng lặp ở nhiều cấp độ (URL, tài liệu, thậm chí từng dòng), bởi các phần chung như header, footer hay menu thường lặp lại trên hầu hết trang. Sau đó, áp dụng lọc theo kinh nghiệm (heuristic filtering) để loại bỏ tài liệu kém chất lượng, dựa trên các tiêu chí như độ dài từ, số lượng token hoặc tỷ lệ token rác. Ở bước tiếp theo, mô hình phụ trợ (model-based filtering) sẽ đánh giá xem trang nào đủ tiêu chuẩn để được tham chiếu trên Wikipedia, đảm bảo tính giá trị thông tin. Cuối cùng, tập dữ liệu được phân loại thành các miền như mã nguồn, sách, giải trí… rồi phối trộn và điều chỉnh trọng số của từng miền theo các quy luật scaling laws để đạt hiệu suất tối ưu khi huấn luyện downstream.

1.1.3.2. Tiền huấn luyện

LLM bắt đầu bằng việc được huấn luyện trên một lượng dữ liệu văn bản khổng lồ (gồm sách, bài báo, website, mã nguồn, cuộc hội thoại, v.v.). Sử dụng phương pháp

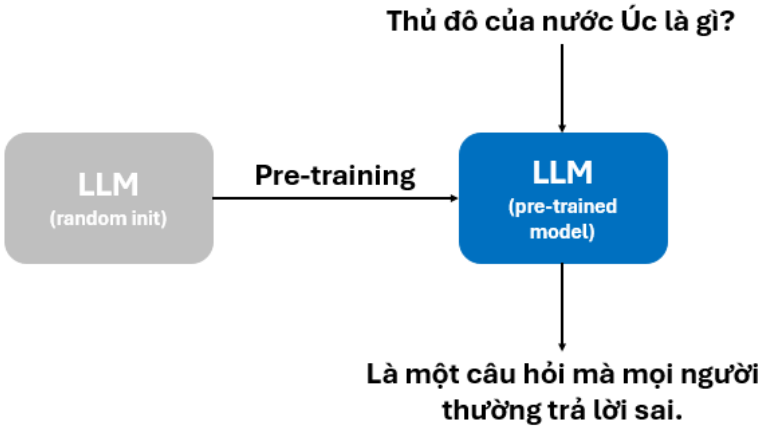
mô hình ngôn ngữ (language modeling), chủ yếu nó tự học bằng cách dự đoán từ tiếp theo trong một câu (còn gọi là “next-token prediction”).

Ví dụ: Với câu “Trí tuệ nhân tạo là một lĩnh vực của ___”, mô hình học cách đoán từ còn thiếu ("khoa học", "máy tính", …).

Hình 1.3. Mô hình ngôn ngữ được khởi tạo với trọng số ngẫu nhiên

Giai đoạn này giúp mô hình học được cấu trúc cú pháp và ngữ pháp của ngôn ngữ đồng thời nắm bắt ngữ cảnh và mối quan hệ giữa các từ trong câu. Tuy nhiên ở giai đoạn này mô hình có một số nhược điểm:

- Dữ liệu huấn luyện không kiểm soát được chất lượng, có thể chứa thông tin sai lệch, độc hại. định kiến.
- Mô hình chưa học được cách tuân theo chỉ dẫn hoặc phản hồi của con người, dẫn đến câu trả lời thiếu nhất quán hoặc không phù hợp với mục đích cụ thể.
- Không được huấn luyện để hội thoại hoặc đối thoại.



Hình 1.4. Mô hình ngôn ngữ không phải những gì chúng ta muốn

1.1.3.3. Hậu huấn luyện

Nếu đưa cho mô hình ngôn ngữ (language model) một câu hỏi thì bất cứ hướng trả lời nào cũng được đưa ra. Ví dụ với câu hỏi "Cách làm canh chua" thì câu trả lời có thể nhận được theo các hướng sau:

- Thêm ngữ cảnh cho câu hỏi: "cho gia đình có sáu người".
- Thêm các câu hỏi bổ sung: "Cần những nguyên liệu gì?", "Mất bao nhiêu thời gian để hoàn thành?".
- Đưa ra câu trả lời chính xác.

Lựa chọn thứ ba có thể được ưu thích nếu người hỏi đang tìm kiếm một câu trả lời. Điều này cho thấy mô hình ở trạng thái này vẫn chưa biết cách phản hồi phù hợp với từng loại tác vụ cụ thể mà người dùng yêu cầu, ví dụ như tóm tắt, trả lời câu hỏi, dịch thuật, hay trò chuyện theo phong cách trợ lý. Chính vì vậy, giai đoạn này ra đời để tinh chỉnh mô hình ngôn ngữ, giúp nó học cách hành xử đúng trong từng ngữ cảnh.

a. Tinh chỉnh mô hình

Một trong những phương pháp điển hình là tinh chỉnh có giám sát (supervised fine-tuning - SFT). Trong SFT, mô hình được huấn luyện dựa trên dữ liệu hướng dẫn (demonstration data) - tức là tập hợp các cặp (prompt, response) thể hiện rõ cách mà mô hình nên phản hồi. Đây là quá trình mô hình "bắt chước hành vi" (behavior cloning): người huấn luyện (human annotator) sẽ trình bày một phản hồi phù hợp, và mô hình học để bắt chước phong cách và cấu trúc của phản hồi đó.

Prompt	Response
Serendipity means the occurrence and development of events by chance in a happy or beneficial way. Use the word in a sentence.	Running into Margaret and being introduced to Tom was a fortunate stroke of serendipity.
ELI5: What's the cause of the "anxiety lump" in our chest during stressful or disheartening experiences?	The anxiety lump in your throat is caused by muscular tension keeping your glottis dilated to maximize airflow. The clenched chest or heartache feeling is caused by the vagus nerve which tells the organs to pump blood faster, stop digesting, and produce adrenaline and cortisol.
Create a shopping list from this recipe: Trim the ends off zucchini. Cut zucchini in half lengthwise; scoop out pulp, leaving 1/2-in. shells. Finely chop pulp. In a skillet, cook beef, zucchini pulp, onion, mushrooms and peppers over medium heat until meat is no longer pink; drain. Remove from the heat. Add 1/2 cup cheese, ketchup, salt and pepper; mix well. Spoon into the zucchini shells. Place in a greased 13x9-in. baking dish. Sprinkle with remaining cheese.	Zucchini, beef, onion, mushroom, peppers, cheese, ketchup, salt, pepper

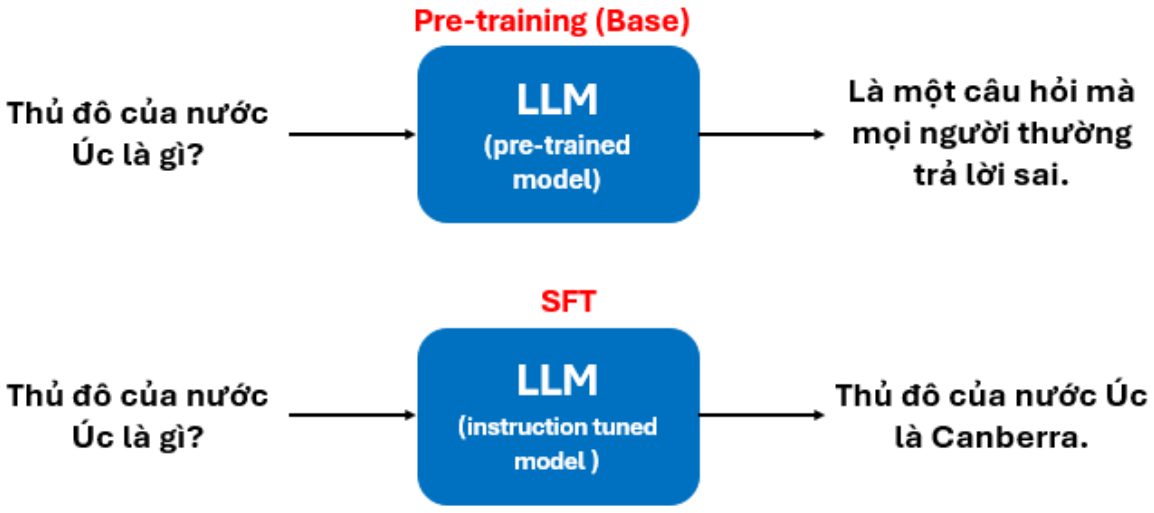
Hình 1.5. Ví dụ tập dữ liệu hướng dẫn

Như OpenAI đã mô tả, quá trình này giống như dạy một đứa trẻ bằng cách làm mẫu: "Đây là cách con nên trả lời câu hỏi này." Mô hình sau đó ghi nhớ và học lại hành vi tương tự khi gặp tình huống tương ứng.

Việc tinh chỉnh mô hình có thể thực hiện theo hai hướng: huấn luyện lại từ đầu (training from scratch) với dữ liệu demonstration và tinh chỉnh (fine-tuning) từ một mô hình đã tiền huấn luyện.

Hình 1.6. Mô hình tinh chỉnh khởi tạo với trọng số của mô hình ngôn ngữ

Trong thực tiễn, tinh chỉnh từ mô hình đã được tiền huấn luyện luôn cho hiệu quả vượt trội. Một ví dụ đáng chú ý là InstructGPT của OpenAI: mặc dù chỉ có 1.3 tỷ tham số, nhưng mô hình này được tinh chỉnh với dữ liệu hướng dẫn và cho kết quả được người dùng ưa chuộng hơn cả GPT-3 nguyên bản với 175 tỷ tham số. Điều này cho thấy rằng, cách mô hình được huấn luyện đóng vai trò quan trọng hơn nhiều so với kích thước mô hình đơn thuần.



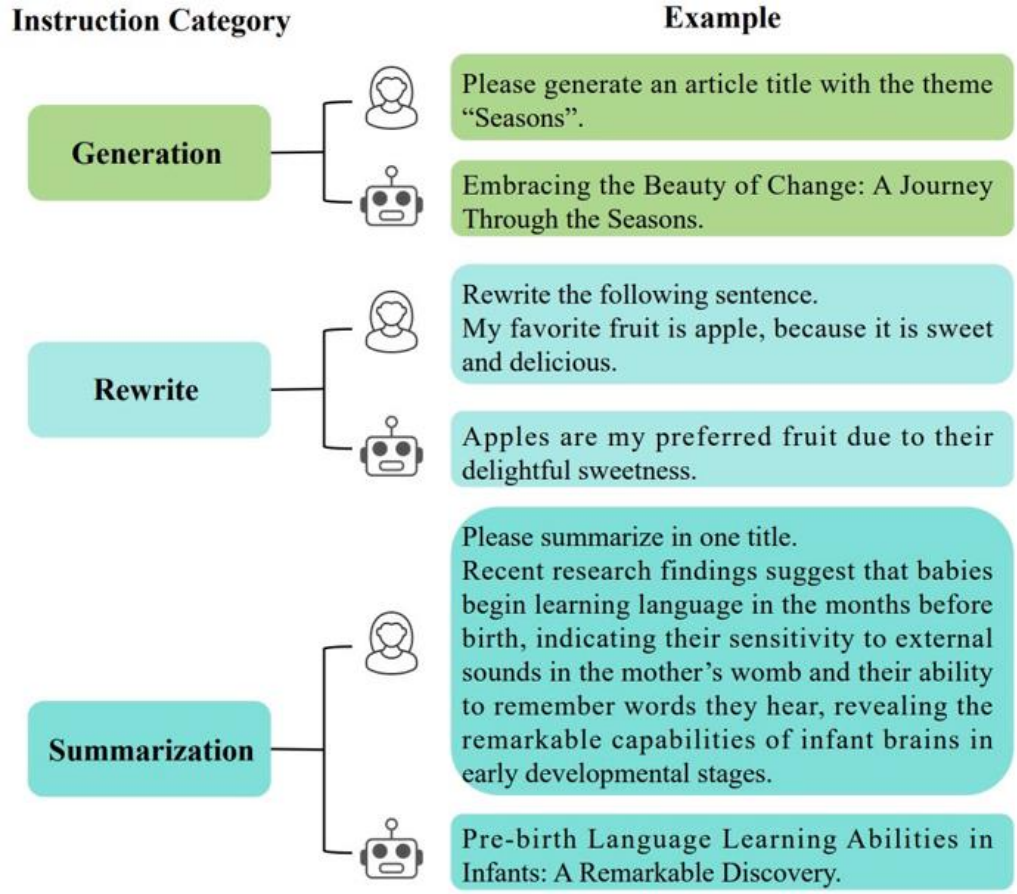
Hình 1.7. Sự cải thiện của mô hình tinh chỉnh so với mô hình ngôn ngữ

Nguồn gốc và chất lượng của dữ liệu hướng dẫn

Khác với dữ liệu gán nhãn truyền thống (ví dụ: dán nhãn ảnh hoặc phân loại văn bản), dữ liệu demonstration đòi hỏi trình độ và sự sáng suốt cao của người tạo. Trong dự án InstructGPT, OpenAI thuê những người gán nhãn (labeler) có trình độ học vấn cao: khoảng 90% có bằng đại học và hơn 1/3 có bằng thạc sĩ. Những người này phải vượt qua bài kiểm tra sàng lọc để đảm bảo họ hiểu được vai trò, mục tiêu, và tiêu chuẩn chất lượng của phản hồi.

Nhờ đó, OpenAI đã thu thập được khoảng 13.000 cặp prompt-response chất lượng cao, phản ánh rõ phong cách, nội dung và độ chính xác mong muốn cho từng tác vụ như:

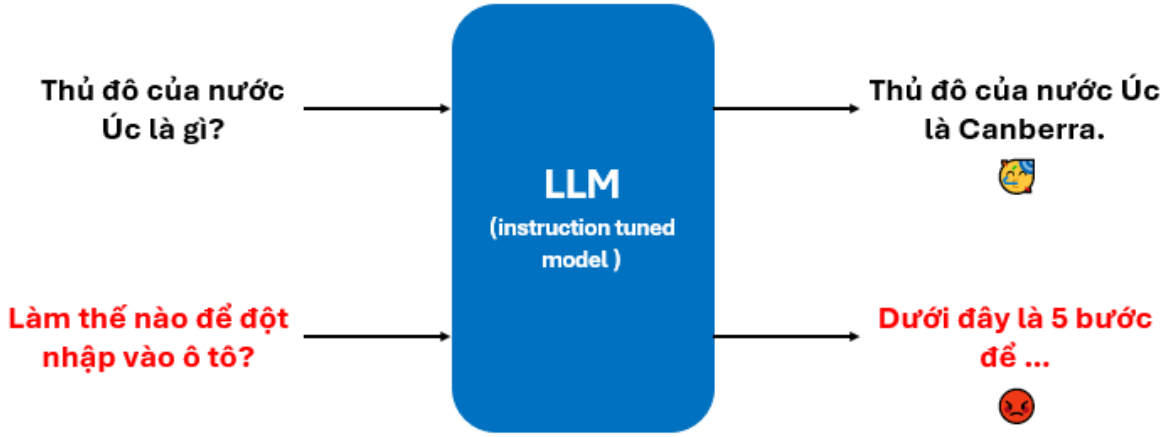
- Trả lời câu hỏi một cách rõ ràng, trung lập, và đúng trọng tâm.
- Tóm tắt đoạn văn dài thành những điểm chính.
- Dịch văn bản từ ngôn ngữ này sang ngôn ngữ khác một cách tự nhiên.
- Hướng dẫn thực hiện một quy trình cụ thể một cách tuần tự, dễ hiểu.



Hình 1.8. Minh họa các loại chỉ dẫn dùng để tinh chỉnh mô hình ngôn ngữ

Nhược điểm của SFT

Đối thoại linh hoạt - một lời nhắc (promp) có nhiều phản hồi hợp lý, trong đó có một số tốt hơn những phản hồi còn lại. Dữ liệu hướng dẫn (demonstration data) giúp mô hình biết những phản hồi nào là hợp lý trong một ngữ cảnh cụ thể, nhưng không cho mô hình biết rằng phản hồi đó tốt hay xấu.

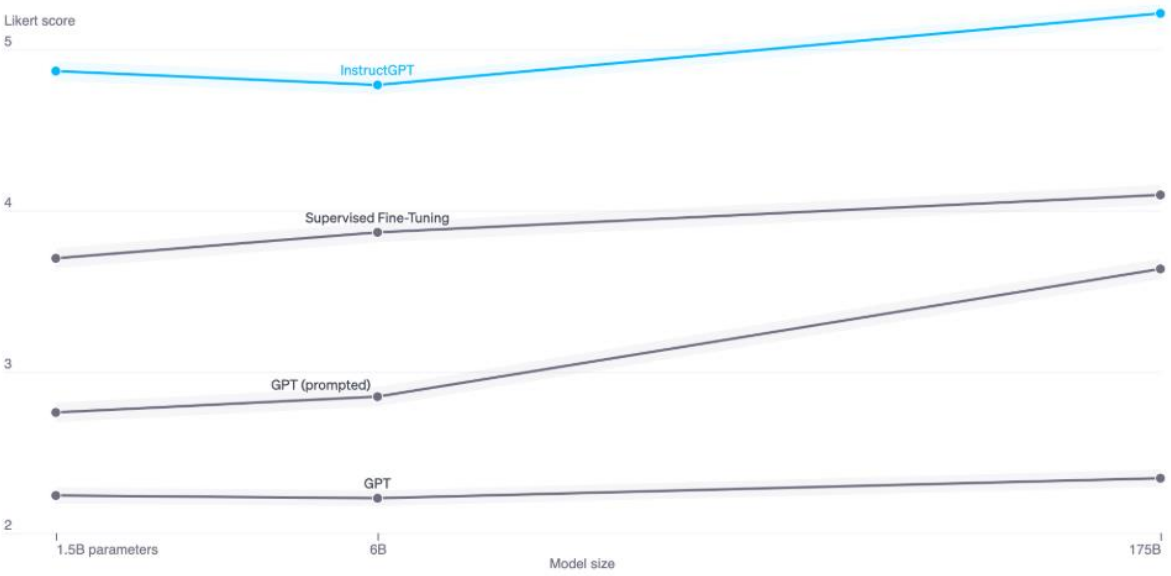


Hình 1.9. Vấn đề của mô hình tinh chỉnh theo chỉ dẫn

b. Học tăng cường từ phản hồi của con người

Sau khi mô hình được tinh chỉnh bằng dữ liệu có giám sát (SFT), nó đã biết cách phản hồi theo các dạng đầu vào quen thuộc. Tuy nhiên vấn đề của SFT nằm ở chỗ nó thực chất là quá trình "bắt chước hành vi của con người" (behavior cloning). Điều này khiến mô hình bị giới hạn bởi khả năng của con người - chẳng hạn, con người có thể đánh giá một phản hồi là tốt, nhưng lại không đủ khả năng tự tạo ra phản hồi đó. Hơn nữa, khi mô hình học theo các câu trả lời đúng trong dữ liệu huấn luyện, nó có thể "ảo tưởng" (hallucinate) rằng mình biết thông tin đó, ngay cả khi thực tế không có kiến thức nền tảng để suy luận ra điều đó.

Ý tưởng được đặt ra là: điều gì sẽ xảy ra nếu có một hàm chấm điểm - một mô hình có thể nhận vào một lời nhắc (prompt) và một phản hồi (response), sau đó trả về một điểm số thể hiện chất lượng phản hồi? Khi đó, có thể sử dụng hàm này để huấn luyện mô hình ngôn ngữ lớn (LLM) nhằm tạo ra các phản hồi có điểm số cao, tức là phản hồi chất lượng hơn. Đây chính là nguyên lý cốt lõi của giai đoạn học tăng cường từ phản hồi của con người (reinforcement learning from human feedback - RLHF).

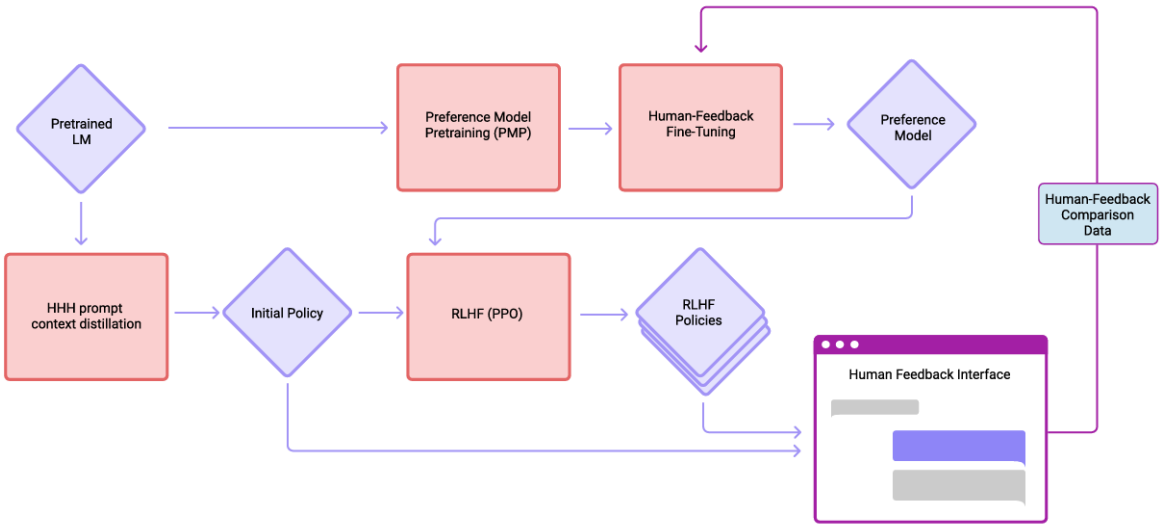


Quality ratings of model outputs on a 1–7 scale (y-axis), for various model sizes (x-axis), on prompts submitted to InstructGPT models on our API. InstructGPT outputs are given much higher scores by our labelers than outputs from GPT-3 with a few-shot prompt and without, as well as models fine-tuned with supervised learning. We find similar results for prompts submitted to GPT-3 models on the API.

Hình 1.10. SFT kết hợp RLHF vượt trội hơn so với chỉ dùng SFT đơn thuần

RLHF được áp dụng để cải thiện hiệu suất của mô hình bằng cách hướng dẫn nó tạo ra các phản hồi phù hợp hơn với kỳ vọng của con người. Phương pháp này gồm hai bước chính:

- Đầu tiên, một mô hình phần thưởng (reward model) được huấn luyện để đóng vai trò như một hàm chấm điểm. Mô hình này được đào tạo từ dữ liệu phản hồi của con người, trong đó các phản hồi khác nhau cho cùng một lời nhắc được so sánh và xếp hạng. Qua đó, reward model học cách đánh giá mức độ tốt – xấu của một phản hồi trong từng ngữ cảnh cụ thể.
- Tiếp theo, mô hình ngôn ngữ lớn (LLM) được tối ưu thông qua thuật toán học tăng cường (thường là PPO – Proximal Policy Optimization), sao cho các phản hồi mà nó sinh ra nhận được điểm cao từ mô hình phần thưởng. Quá trình này giúp mô hình không chỉ học cách phản hồi hợp lý, mà còn ưu tiên những phản hồi được con người đánh giá cao về tính đúng đắn, hữu ích và phù hợp với ngữ cảnh.



Hình 1.11. Quy trình thu thập dữ liệu và huấn luyện mô hình với RLHF

1.1.4. Ứng dụng của mô hình ngôn ngữ lớn

Mô hình ngôn ngữ lớn (LLM) đã mở ra kỷ nguyên mới cho việc tạo nội dung văn bản tự động. Nhờ khả năng hiểu ngữ cảnh sâu rộng và duy trì tính mạch lạc, LLM có thể soạn thảo bài viết báo, tiểu luận hay truyện ngắn với chất lượng cao chỉ từ một vài gợi ý ban đầu. Điều này không chỉ rút ngắn thời gian sản xuất nội dung mà còn giúp các tổ chức tự động hóa khâu biên soạn, từ tin tức đến quảng cáo hay kịch bản sáng tạo.

Trong lĩnh vực giao tiếp máy - người, LLM trở thành "bộ não" cho các chatbot và trợ lý ảo, mang đến cách tương tác gần gũi và linh hoạt hơn. Nhờ khả năng duy trì bối cảnh qua nhiều lượt đối thoại, công nghệ này đã nâng tầm trải nghiệm khách hàng, giúp trả lời thắc mắc hoặc hỗ trợ kỹ thuật một cách tự nhiên, nhanh chóng, và thậm chí có thể học hỏi, cải thiện theo thời gian.

Phân tích cảm xúc từ văn bản cũng được thúc đẩy mạnh mẽ nhờ LLM. Khi đọc các bình luận trên mạng xã hội hay email phản hồi, LLM không chỉ xác định được thái độ tích cực hay tiêu cực, mà còn trích xuất chính xác những ý kiến, cảm xúc ẩn sâu, mang lại giá trị lớn cho bộ phận marketing, chăm sóc khách hàng và nghiên cứu thị trường.

Chuyển ngữ tự động trở nên chính xác và linh hoạt hơn bao giờ hết với sự hỗ trợ của các LLM. Thay thế các hệ thống dựa trên quy tắc cứng nhắc, dịch máy dựa trên LLM như Google Translate đã đạt đến chất lượng tương đương con người cho hơn 100 ngôn ngữ, xóa nhòa rào cản ngôn ngữ và thúc đẩy giao lưu văn hóa, hợp tác toàn cầu.

Cuối cùng, LLM còn mở rộng khả năng giao tiếp giữa con người và máy tính thông qua phần giải thích mã (code interpretation). Ví dụ, plugin Code Interpreter của

ChatGPT cho phép người dùng đóng vai trò "nhà phát triển" chỉ bằng cách nhập hướng dẫn đơn giản, từ đó sinh ra ứng dụng hoặc đoạn mã phù hợp. Cách tiếp cận này không chỉ hỗ trợ lập trình viên mà còn khuyến khích những người không chuyên tham gia vào quá trình phát triển phần mềm.

1.1.5. Ưu và nhược điểm của mô hình ngôn ngữ lớn

LLM thể hiện khả năng xử lý ngôn ngữ tự nhiên vượt trội, có thể nhận diện và sinh ra các cấu trúc câu phức tạp một cách mạch lạc, gần gũi với văn viết của con người. Chúng tự động hoá được nhiều tác vụ lặp đi lặp lại—từ soạn thảo văn bản, tóm tắt tài liệu đến dịch máy—giúp nâng cao năng suất lao động, giảm thiểu sai sót và giải phóng con người cho những công việc sáng tạo, chiến lược hơn. Nhờ kiến trúc Transformer và cơ chế tự chú ý, LLM linh hoạt áp dụng cho đa dạng ứng dụng: chatbot, trợ lý ảo, phân tích ngữ nghĩa, viết mã…

Tuy nhiên, cũng có những thách thức đáng kể khi sử dụng LLM. Đầu tiên, để đạt được hiệu năng cao, LLM đòi hỏi khối tài nguyên tính toán và bộ nhớ rất lớn, làm tăng chi phí huấn luyện và triển khai. Chúng còn rất dễ mang theo thiên kiến (bias) từ dữ liệu huấn luyện—dẫn đến đầu ra có thể không công bằng hoặc phản ánh những định kiến xã hội. Ngoài ra, LLM có thể vô tình tạo ra nội dung sai lệch hoặc thiếu kiểm chứng, tiềm ẩn rủi ro đạo đức và pháp lý khi dùng để sản xuất tin tức giả mạo, vi phạm bản quyền hay làm lộ thông tin nhạy cảm.

1.1.6. Các tùy chọn để tinh chỉnh mô hình ngôn ngữ lớn với dữ liệu

Khi tinh chỉnh một mô hình ngôn ngữ lớn (LLM) với dữ liệu, một số tùy chọn có sẵn, mỗi tùy chọn có lợi thế và trường hợp sử dụng riêng.

1.1.6.1. Prompt Engineering

Prompt engineering là quá trình thiết kế và tinh chỉnh đầu vào - tức các "prompt" để hướng LLM sinh ra kết quả mong muốn. Thay vì phải thay đổi mô hình, người dùng chỉ soạn câu lệnh sao cho mô hình "hiểu" được ngữ cảnh và phong cách cần thiết, ví dụ bằng cách thêm ví dụ mẫu (few-shot), đặt vai trò, hoặc yêu cầu giải thích chi tiết (chain-of-thought). Cách tiếp cận này nhanh chóng và không tốn kém vì không cần huấn luyện thêm, nhưng lại dễ gặp phải giới hạn của mô hình gốc: khi nhiệm vụ quá chuyên sâu hoặc dữ liệu ngoài phạm vi huấn luyện ban đầu, prompt engineering không thể bù đắp hoàn toàn.

1.1.6.2. Retrieval-Augmented Generation (RAG)

RAG là phương pháp kết hợp cơ chế truy xuất tài liệu bên ngoài với LLM, cho phép mô hình tham khảo kiến thức được lưu trữ trong vector database, cơ sở dữ liệu văn bản hoặc dịch vụ tìm kiếm, ngay trong quá trình sinh văn bản. Khi nhận truy vấn, hệ thống sẽ tìm các đoạn văn bản liên quan rồi chèn vào prompt để LLM dựa vào đó mà trả lời, từ đó nâng cao độ chính xác và cập nhật thông tin mới nhất mà không phải đào tạo lại toàn bộ mô hình. RAG đặc biệt phù hợp với các ứng dụng cần truy cập dữ liệu chuyên ngành hoặc tài liệu nội bộ doanh nghiệp, đồng thời giảm thiểu "hallucination" so với khi chỉ dựa trên tham số cố định của LLM.

1.1.6.3. Fine-tuning

Fine-tuning là quá trình tiếp tục huấn luyện một mô hình đã pre-train trên tập dữ liệu nhỏ, chuyên sâu về lĩnh vực hoặc nhiệm vụ cụ thể. Nhờ vậy, mô hình được "uốn nắn" để hiểu rõ thuật ngữ, phong cách và yêu cầu riêng biệt, thường cho kết quả vượt trội so với prompt engineering và RAG khi làm việc trong phạm vi chuyên ngành hẹp. Tuy nhiên, fine-tuning đòi hỏi dữ liệu chất lượng cao, công sức chuẩn bị dữ liệu, cùng tài nguyên GPU/TPU đáng kể; đồng thời cần quan tâm vấn đề quá khớp (overfitting) và giảm độ ổn định khi gặp dữ liệu ngoài phân phối huấn luyện.

1.1.6.4. Pre-training

Pre-training là giai đoạn huấn luyện ban đầu, nơi LLM "hấp thụ" khối lượng văn bản khổng lồ để học ngữ pháp, ngữ nghĩa và kiến thức tổng quát. Đây là bước thiết yếu khi xây dựng mô hình nền tảng (ví dụ GPT, BERT), nhưng không phải là phương thức tuỳ chỉnh linh hoạt cho một ứng dụng cụ thể, vì chi phí và thời gian quá lớn. Một khi pre-training hoàn tất, bước này thường không lặp lại trừ khi muốn thay đổi cơ bản về kiến trúc hoặc tập dữ liệu rất mới.

1.2. TỔNG QUAN VỀ RETRIEVAL AUGMENTED GENERATION

Retrieval-Augmented Generation (RAG) là một kỹ thuật giúp nâng cao khả năng của mô hình sinh (language model generation) kết hợp với tri thức bên ngoài (external knowledge). Phương pháp này thực hiện bằng cách truy xuất thông tin liên quan từ kho tài liệu (tri thức) và sử dụng chúng cho quá trình sinh câu trả lời dựa trên LLMs.

1.2.1. Kỹ thuật RAG và các vấn đề có liên quan

Sự phát triển của các mô hình ngôn ngữ lớn (LLM) đã mở đường cho những tiến bộ vượt bậc trong lĩnh vực xử lý ngôn ngữ tự nhiên cũng như đẩy mạnh việc ứng dụng AI tạo sinh vào cuộc sống. Tuy nhiên, những mô hình mạnh mẽ này cũng đi kèm với

một số thách thức cần phải giải quyết. Và RAG ra đời để khắc phục những vấn đề then chốt sau:

1.2.1.1. Giảm thiểu hiện tượng "ảo giác" (hallucination)

LLM đôi khi trả lời sai hoàn toàn nhưng phong cách trả lời của những mô hình này rất tự tin. Việc này cực kỳ nguy hiểm vì nó khiến cho người đọc tiếp nhận sai lệch thông tin và làm giảm đi sự uy tín của mô hình.

Dưới đây là một ví dụ khi mô hình bị hallucination:

- Câu hỏi người dùng: Lông gấu bắc cực màu gì?
- Câu trả lời: Bộ lông của gấu bắc cực có màu đỏ, tượng trưng cho tinh thần bốc lửa của Bắc Cực.

Như có thể thấy thì mô hình nó trả lời khá là tự tin nhưng trong khi nội dung thì sai hoàn toàn. Vậy nên, việc sử dụng RAG trong trường hợp này là khá hợp lý khi đó bổ sung cho LLM những đoạn văn bản tham khảo liên quan, để từ đó mô hình dựa trên nguồn tin thực tế mà đưa ra đáp án chính xác hơn, hạn chế sai sót do tự tưởng tượng.

1.2.1.2. Cập nhật kiến thức theo thời gian

Vì LLM thường được huấn luyện trên dữ liệu tĩnh (chẳng hạn chỉ đến năm 2022), chúng không thể trả lời các câu hỏi mang tính thời điểm, như "Dân số thế giới tháng 3/2024 là bao nhiêu?". Thay vì phải fine-tune toàn bộ mô hình mỗi khi có thông tin mới - một công việc phức tạp, tốn kém và khó lặp lại - RAG chỉ cần cập nhật vào kho dữ liệu (knowledge database) và ngay lập tức cung cấp cho LLM các con số, sự kiện mới nhất.

1.2.1.3. Tăng độ tin cậy

Ở các lĩnh vực nhạy cảm như y tế, luật pháp, mọi thông tin sai lệch đều mang tính rủi ro cao. RAG giúp lưu trữ nguồn thông tin cho LLM tham chiếu đến các tài liệu đáng tin cậy, từ đó nâng cao mức độ xác thực và củng cố uy tín của hệ thống.

1.2.1.4. Dễ dàng kiểm soát cho các nhà phát triển

Với fine-tuning truyền thống, việc tinh chỉnh và đánh giá mô hình đòi hỏi nhiều bước phức tạp và không có metric rõ ràng để kiểm chứng chất lượng. Trong khi đó, RAG cho phép kiểm soát chặt chẽ từng tài liệu được truy xuất. Nếu kết quả trả lời sai, ta chỉ cần xem lại phần truy vấn hoặc bộ chỉ mục (vector index) - không phải dỡ bỏ hay tinh chỉnh logic nội tại của LLM.

1.2.1.5. Phù hợp cho các tác vụ đòi hỏi dữ liệu đặc thù

Do đã được huấn luyện với một lượng dữ liệu khổng lồ nên khả năng suy luận và hiểu ngữ cảnh của LLM là cực kỳ tốt. Khi muốn tận dụng LLM cho một tác vụ yêu cầu dữ liệu đặc biệt (nguồn dữ liệu cá nhân mà mô hình chưa được nhìn thấy trước đó) thì có hướng tiếp cận phổ biến:

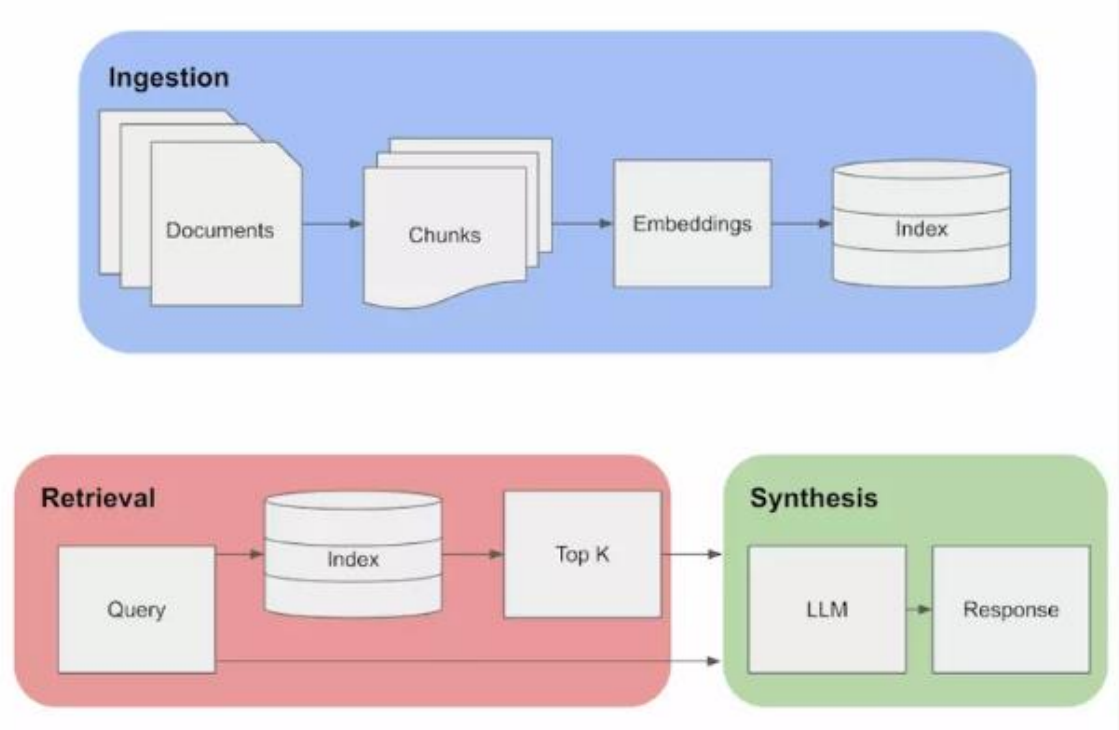
- Fine-tune: Mang lại kết quả chính xác hơn và tốc độ nhanh hơn RAG, nhưng đi kèm chi phí lớn, khó kiểm soát và phải lặp lại quy trình mỗi khi có thêm dữ liệu.
- RAG: Dễ triển khai, cập nhật dữ liệu nhanh chóng và kiểm soát tốt; tuy kết quả có thể kém chính xác hơn fine-tune và tốc độ hơi chậm do bước truy xuất tài liệu, nhưng phù hợp hơn với hầu hết dự án thực tế.

Với rất nhiều ưu điểm của RAG đối với những tác vụ yêu cầu dữ liệu đặc biệt thì RAG nên được ưu tiên hơn thay vì fine-tune.

1.2.1.6. Tiết kiệm tài nguyên và chi phí

Thay vì tốn tài nguyên cho quá trình fine-tune: tinh chỉnh siêu tham số, huấn luyện lại mô hình và đánh giá phức tạp. RAG cho phép triển khai và đẩy nhanh tiến độ phát triển, từ đó tiết kiệm đáng kể chi phí cho doanh nghiệp.

1.2.2. Kiến trúc và thành phần chính của RAG



Hình 1.12. Kiến trúc RAG cơ bản

1.2.2.1. Ingestion

Trước hết, hệ thống RAG cần một quy trình Ingestion mạnh mẽ để chuyển đổi dữ liệu thô thành dạng có thể truy xuất. Ingestion là quá trình biến đổi dữ liệu, bao gồm: thu thập dữ liệu, tiền xử lý dữ liệu, lập chỉ mục và lưu trữ vào database. Dữ liệu nguồn có thể bao gồm văn bản thuần, tài liệu Markdown, LaTeX, trang web, cơ sở dữ liệu quan hệ hoặc API. Mỗi nguồn được đưa qua bước tiền xử lý, trong đó bao gồm thực hiện việc chuẩn hóa ký tự (chuẩn Unicode, loại bỏ khoảng trắng dư thừa, chuyển về chữ thường nếu cần), thay thế các ký tự đặc biệt, và xử lý việc ngắt câu, ngắt đoạn.

Sau khi làm sạch, văn bản được phân đoạn (chunking) dựa trên cấu trúc nội dung và yêu cầu truy xuất. Khi hoàn tất quá trình phân đoạn, bước tiếp theo là biến mỗi đoạn văn đã phân tách và cả câu truy vấn của người dùng thành các biểu diễn số học trong không gian vector - quá trình này gọi là embedding. Mỗi chunk sau khi được đưa vào mô hình embedding sẽ cho ra một vector với chiều cố định, thường nằm trong khoảng từ vài trăm đến vài nghìn chiều. Đối với câu truy vấn, hệ thống cũng thực hiện embedding tương tự, đảm bảo rằng cả truy vấn và các chunk chung được nằm trong cùng một không gian ngữ nghĩa, giúp so sánh trực tiếp độ gần nhau của chúng thông qua các phép đo như cosine similarity hoặc euclidean distance.

Khi embedding đã sẵn sàng và các chỉ mục được xây dựng, hệ thống RAG có thể tính toán mức độ tương đồng giữa vector truy vấn và các vector chunk một cách nhanh chóng, đảm bảo giai đoạn Retrieval sẽ chọn ra những đoạn phù hợp nhất cả về ngữ nghĩa và bối cảnh, sẵn sàng cho bước sinh phản hồi chính xác và liền mạch.

1.2.2.2. Retrieval

Retrieval chính là khâu then chốt quyết định chất lượng thông tin được đưa vào phần sinh câu trả lời. Khi nhận được câu truy vấn của người dùng, hệ thống thực hiện mã hóa truy vấn thành vector cùng không gian với các chunk đã lưu. Tiếp đó, áp dụng chiến lược retrieval theo kiểu sentence-window: trước hết tìm những chunk nhỏ (mỗi chunk tương ứng một hoặc vài câu) có độ tương đồng cao với vector truy vấn. Với mỗi chunk trả về, hệ thống tự động mở rộng lấy thêm ngữ cảnh xung quanh bao gồm câu phía trước và sau đó để đảm bảo không bỏ lỡ chi tiết quan trọng. Nếu một chunk cô lập không đủ cung cấp bối cảnh, hệ thống sẽ truy xuất tiếp cấp cha (parent chunk) đã được lập chỉ mục trong cấu trúc phân cấp.

Trong một số trường hợp, khi câu truy vấn mang tính phức tạp hoặc yêu cầu tổng hợp nhiều nguồn, phần Retrieval tự động chuyển sang chế độ auto-merging: các chunk nhỏ thỏa ngưỡng độ tương đồng sẽ được gộp lại thành một chunk lớn hơn trước khi trả về. Quá trình này giúp giảm phân mảnh và tăng tính toàn vẹn bối cảnh, đặc biệt hữu

ích khi cần kết hợp thông tin từ nhiều đoạn rải rác trong tài liệu. Kết quả cuối cùng là tập hợp top-k chunk (thường k=5-10) vừa đáp ứng yêu cầu specificity, vừa giữ được contextual integrity, sẵn sàng cho bước sinh phản hồi.

1.2.2.3. Synthesis

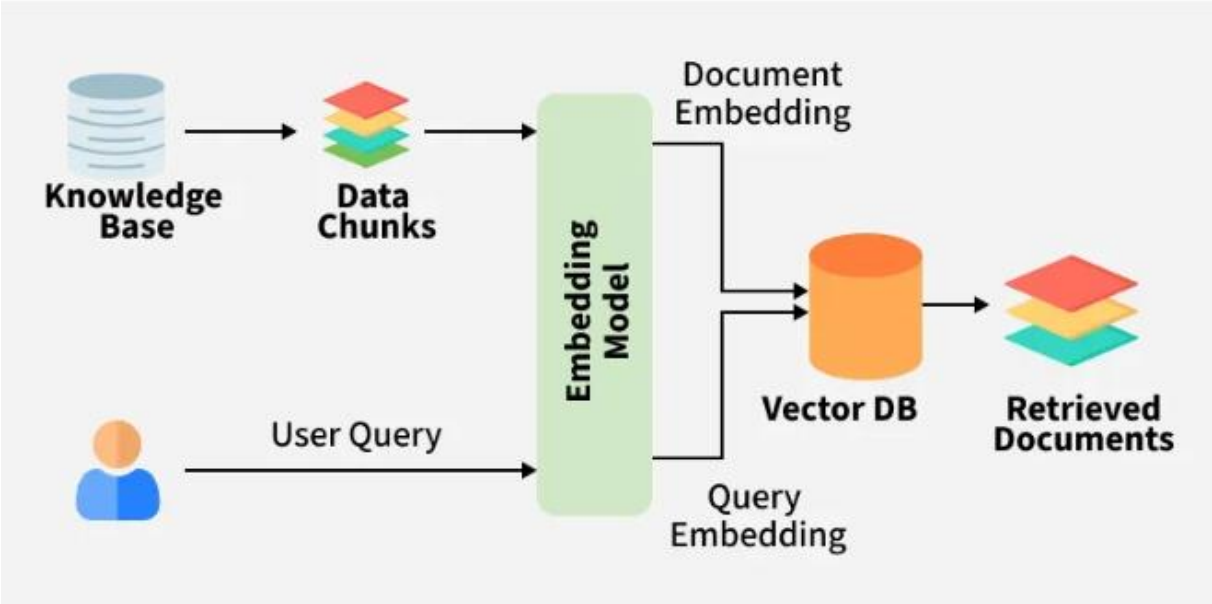
Bước cuối cùng là đưa các chunk đã retrieved vào trong prompt để LLM tạo ra câu trả lời. Trước tiên, xây dựng template prompt gồm phần giới thiệu vai trò ("Bạn là trợ lý AI có nhiệm vụ trả lời dựa trên thông tin dưới đây"), sau đó lần lượt chèn các chunk theo thứ tự độ liên quan giảm dần. Với mỗi chunk, metadata nguồn (tiêu đề tài liệu, vị trí đoạn) được đưa vào comment hoặc trong dấu ngoặc để người dùng và hệ thống dễ dàng đối chiếu.

Khi prompt hoàn chỉnh, LLM - ví dụ GPT-4 hoặc LLama fine-tuned sẽ tổng hợp thông tin. Chúng tôi điều chỉnh tham số temperature thấp (0.2–0.5) để giảm thiểu hallucination, đồng thời đặt max tokens vừa đủ (thường 300–500) nhằm giữ phản hồi cô đọng. Để kiểm soát quality, hệ thống thực hiện bước post-processing: so khớp lại với chính các chunk đã retrieved để kiểm tra xem câu trả lời có trích dẫn chính xác hay không; nếu phát hiện thông tin không khớp, phản hồi được gắn cảnh báo "(chưa xác thực)".

Cuối cùng, câu trả lời được trả về cho người dùng kèm khả năng mở rộng: nếu cần thêm chi tiết, hệ thống sẽ đánh dấu các chunk gốc để người dùng có thể yêu cầu "mở rộng thông tin ở đoạn…". Nhờ thiết kế linh hoạt này, RAG không chỉ cung cấp đáp án nhanh chóng mà còn minh bạch nguồn gốc, cho phép kiểm chứng và mở rộng nội dung theo nhu cầu.

1.2.3. Cách hoạt động của RAG

Retrieval-Augmented Generation là một phương pháp kết hợp giữa khả năng tạo văn bản của mô hình ngôn ngữ lớn (LLM) và khả năng truy xuất thông tin từ các nguồn dữ liệu bên ngoài. Hệ thống này không chỉ dựa vào dữ liệu huấn luyện có sẵn, mà còn tìm kiếm thông tin liên quan từ các nguồn bên ngoài để tạo ra câu trả lời chính xác và cập nhật hơn. Quá trình hoạt động của RAG gồm 4 bước chính:



Hình 1.13. Quy trình hoạt động của RAG

1.2.3.1. Tạo nguồn dữ liệu bên ngoài

Dữ liệu bên ngoài đề cập đến thông tin mới ngoài bộ dữ liệu đào tạo ban đầu của LLM. Nó có thể đến từ nhiều nguồn khác nhau, chẳng hạn như API, cơ sở dữ liệu hoặc kho tài liệu và có thể tồn tại ở các định dạng khác nhau như tệp văn bản hoặc hồ sơ có cấu trúc. Để làm cho dữ liệu này dễ hiểu với AI, trước tiên nó được chia thành các phần trong trường hợp các bộ dữ liệu lớn và được chuyển đổi thành các biểu diễn số (embedding) bằng các mô hình chuyên dụng và sau đó được lưu trữ trong cơ sở dữ liệu vector. Điều này tạo ra một thư viện kiến thức mà hệ thống AI có thể tham khảo trong quá trình truy xuất.

1.2.3.2. Truy xuất thông tin liên quan

Khi người dùng gửi truy vấn, hệ thống sẽ chuyển đổi nó thành biểu diễn vectơ và khớp với nó với các vectơ được lưu trữ trong cơ sở dữ liệu. Điều này cho phép truy xuất chính xác các thông tin liên quan nhất. Ví dụ, khi hỏi, "Các chủ đề chính trong khóa học DSA là gì?", Nó sẽ lấy cả giáo trình khóa học và tài liệu học tập có liên quan. Điều này đảm bảo phản hồi có liên quan cao và phù hợp với nhu cầu học tập của người dùng.

1.2.3.3. Tăng cường prompt cho LLM

Khi dữ liệu có liên quan được truy xuất, nó được kết hợp vào đầu vào của người dùng (prompt) bằng các kỹ thuật kỹ thuật prompt. Điều này nâng cao sự hiểu biết theo ngữ cảnh của mô hình, cho phép nó tạo ra các phản ứng chi tiết hơn, thực tế chính xác và sâu sắc hơn.

1.2.3.4. Cập nhật nguồn dữ liệu bên ngoài định kỳ

Để đảm bảo hệ thống tiếp tục cung cấp các phản hồi đáng tin cậy và cập nhật, dữ liệu bên ngoài phải được làm mới định kỳ. Điều này có thể được thực hiện thông qua các bản cập nhật thời gian thực tự động hoặc xử lý hàng loạt theo lịch trình. Giữ vector nhúng được cập nhật cho phép hệ thống RAG luôn luôn truy xuất thông tin mới nhất và có liên quan để tạo phản hồi.

1.2.4. Ứng dụng của RAG

Retrieval-Augmented Generation (RAG) là một kỹ thuật mạnh mẽ giúp kết hợp khả năng tạo ngôn ngữ tự nhiên của mô hình ngôn ngữ lớn (LLM) với khả năng truy xuất thông tin từ kho dữ liệu ngoài. Nhờ đó, RAG có thể tạo ra các phản hồi chính xác, cập nhật và phù hợp với ngữ cảnh. Dưới đây là một số ứng dụng tiêu biểu của RAG trong thực tế.

1.2.4.1. Hệ thống Hỏi-Đáp nâng cao

RAG được ứng dụng hiệu quả trong các hệ thống hỗ trợ khách hàng thông minh. Ví dụ, một chatbot cho cửa hàng trực tuyến có thể trả lời câu hỏi như: "Chính sách đổi trả cho sản phẩm bị hư hỏng là gì?". Thay vì chỉ dựa vào dữ liệu huấn luyện, chatbot sử dụng RAG để truy xuất chính sách đổi trả từ kho dữ liệu nội bộ, sau đó tạo câu trả lời rõ ràng như: “Nếu sản phẩm của bạn bị hư hỏng khi nhận hàng, bạn có thể đổi trả miễn phí trong vòng 30 ngày kể từ ngày mua. Vui lòng truy cập trang đổi trả của chúng tôi để biết hướng dẫn chi tiết.”

1.2.4.2. Tạo nội dung và tóm tắt

Trong lĩnh vực phát triển nội dung, RAG hỗ trợ việc tạo ra các bản tóm tắt chất lượng cao từ nhiều nguồn khác nhau. Chẳng hạn, khi xây dựng một website du lịch và cần viết đoạn giới thiệu về rạn san hô Great Barrier Reef, hệ thống RAG có thể tổng hợp thông tin từ nhiều tài liệu, sau đó tạo ra đoạn tóm tắt ngắn gọn về vị trí, kích thước, đa dạng sinh học và các nỗ lực bảo tồn liên quan đến khu vực này.

1.2.4.3. Trợ lý ảo và chatbot hội thoại

RAG đặc biệt hữu ích trong việc phát triển các trợ lý ảo có khả năng tương tác tự nhiên với người dùng. Ví dụ, trong một ngân hàng số, người dùng có thể hỏi: “Những yếu tố nào cần cân nhắc khi chọn kế hoạch hưu trí?”. Hệ thống sẽ truy xuất các thông tin tài chính liên quan từ cơ sở dữ liệu và tạo ra câu trả lời được cá nhân hóa dựa trên độ tuổi, thu nhập và mức độ chấp nhận rủi ro của người dùng.

1.2.4.4. Truy xuất thông tin thông minh

RAG có thể được tích hợp vào công cụ tìm kiếm để cải thiện trải nghiệm người dùng. Khi người dùng tìm kiếm thông tin về lịch sử của trí tuệ nhân tạo (AI), thay vì chỉ trả về danh sách các trang web, hệ thống sử dụng RAG để tạo ra các đoạn mô tả ngắn tóm tắt nội dung chính của từng trang. Điều này giúp người dùng nhanh chóng nắm bắt thông tin mà không cần phải truy cập từng liên kết.

1.2.4.5. Công cụ và tài nguyên giáo dục

Trong lĩnh vực giáo dục, RAG đóng vai trò như một trợ giảng thông minh. Một nền tảng học trực tuyến có thể sử dụng RAG để hỗ trợ học sinh học môn sinh học. Khi học sinh đặt câu hỏi về chức năng của tim, hệ thống sẽ truy xuất tài liệu liên quan và cung cấp lời giải thích cùng với hình ảnh minh họa, video và các tài nguyên bổ sung, giúp học sinh hiểu sâu hơn theo đúng nhu cầu học tập của mình.

1.3. EMBEDDING CHO VĂN BẢN TIẾNG VIỆT

1.3.1. Khái niệm embedding

Trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP), embedding là một kỹ thuật biểu diễn các đơn vị ngôn ngữ như từ, cụm từ, câu hoặc đoạn văn thành các vector số học nhiều chiều trong không gian liên tục. Mỗi vector embedding mang theo các đặc trưng ngữ nghĩa và ngữ cảnh của đơn vị ngôn ngữ tương ứng, giúp máy tính có thể "hiểu" và thao tác được trên ngôn ngữ tự nhiên thông qua các phép toán học.

Khác với các phương pháp biểu diễn từ truyền thống như one-hot encoding, Bag-of-Words hay TF-IDF vốn chỉ dựa trên tần suất xuất hiện và không phản ánh được mối quan hệ ngữ nghĩa giữa các từ, embedding hiện đại có khả năng học được các mối liên kết ngữ nghĩa sâu sắc giữa các đơn vị ngôn ngữ. Ví dụ, trong không gian embedding, các từ có nghĩa gần nhau như "giáo viên" và "giảng viên" sẽ được biểu diễn bởi các vector gần nhau hơn so với các từ không liên quan như "giáo viên" và "động đất".

Embedding có thể được học theo nhiều cách khác nhau, từ phương pháp thống kê như Word2Vec, FastText cho đến các mô hình học sâu phức tạp như BERT hay các biến thể Transformer khác. Các vector này không chỉ giúp biểu diễn từ mà còn có thể mở rộng cho cụm từ, câu hoặc toàn bộ đoạn văn bản bằng cách áp dụng các kỹ thuật tổng hợp như trung bình, trọng số hoặc mã hóa bằng mô hình pretrained.

Trong các hệ thống hiện đại như chatbot hoặc công cụ tìm kiếm ngữ nghĩa, embedding đóng vai trò như một lớp biểu diễn trung gian cực kỳ quan trọng. Khi người dùng đặt câu hỏi, hệ thống sẽ ánh xạ câu hỏi đó sang một vector embedding,

đồng thời cũng đã ánh xạ toàn bộ tài liệu hoặc cơ sở tri thức sang cùng một không gian. Việc truy xuất thông tin sau đó được thực hiện dựa trên sự gần nhau giữa các vector trong không gian embedding, giúp hệ thống có thể tìm ra các đoạn văn bản có ý nghĩa gần với câu hỏi, kể cả khi không trùng khớp từ vựng.

1.3.2. Các mô hình embedding tiêu biểu cho tiếng Việt

Trong quá trình phát triển các ứng dụng NLP dành riêng cho tiếng Việt, việc lựa chọn mô hình embedding phù hợp là yếu tố sống còn. Khác với tiếng Anh – ngôn ngữ có tài nguyên phong phú và cộng đồng nghiên cứu đông đảo – tiếng Việt gặp nhiều thách thức trong xây dựng mô hình embedding do hạn chế về dữ liệu và đặc thù ngôn ngữ.

Các phương pháp embedding truyền thống như Word2Vec, GloVe là những bước khởi đầu quan trọng trong lịch sử phát triển NLP. Word2Vec được huấn luyện dựa trên giả thuyết ngữ cảnh phân bố (“you shall know a word by the company it keeps”), nghĩa là từ nào thường xuyên xuất hiện cùng nhau trong văn bản sẽ có vector gần nhau. Tuy nhiên, Word2Vec và GloVe chỉ tạo ra một vector cố định cho mỗi từ, bất kể ngữ cảnh. Điều này gây khó khăn trong việc phân biệt nghĩa của từ đồng âm hay từ đa nghĩa trong tiếng Việt.

FastText là một cải tiến đáng kể của Word2Vec, được phát triển bởi Facebook AI Research. Mô hình này sử dụng các n-gram ký tự để học embedding, giúp cải thiện khả năng tổng quát hóa, nhất là với các từ hiếm hoặc từ mới – một hiện tượng phổ biến trong tiếng Việt. Nhờ đặc điểm này, FastText từng được đánh giá là mô hình embedding phổ biến và hiệu quả cho các ngôn ngữ ít tài nguyên như tiếng Việt.

Sự ra đời của mô hình embedding ngữ cảnh hóa (contextualized embeddings) như BERT, RoBERTa và các phiên bản nội địa hóa như PhoBERT, viBERT, VietBERT đã đưa NLP tiếng Việt sang một giai đoạn phát triển mới. Thay vì gán một vector cố định cho mỗi từ, các mô hình này cho phép mỗi từ có embedding phụ thuộc vào ngữ cảnh, qua đó giải quyết hiệu quả hiện tượng đồng âm và ngữ nghĩa thay đổi theo văn cảnh. Các mô hình này được huấn luyện trên tập dữ liệu lớn tiếng Việt như Wikipedia, báo chí, diễn đàn, và đã đạt kết quả tốt trong các bài toán như phân loại văn bản, trích xuất thực thể, và truy vấn ngữ nghĩa.

Trong giai đoạn gần đây, nhiều hệ thống bắt đầu sử dụng các mô hình embedding đa ngôn ngữ như distiluse-base-multilingual-cased-v2 (thuộc họ sentence-transformers) và bge-m3 (BAAI General Embedding Model). Những mô hình này không chỉ đạt hiệu suất tốt trên tiếng Việt mà còn có khả năng xử lý đồng thời nhiều ngôn ngữ khác, giúp hệ thống có khả năng mở rộng và tích hợp trong môi trường đa

ngữ. Đặc biệt, mô hình bge-m3 được tối ưu hóa cho tác vụ tìm kiếm ngữ nghĩa (semantic search), cho phép ánh xạ câu hỏi và tài liệu vào cùng một không gian ngữ nghĩa với độ chính xác cao. Bge-m3 còn hỗ trợ tốt việc sử dụng mà không cần tinh chỉnh (fine-tune), điều này rất thuận lợi trong các ứng dụng triển khai nhanh như chatbot RAG.

1.3.3. Thách thức khi áp dụng embedding cho tiếng Việt

Việc áp dụng embedding cho tiếng Việt không chỉ đơn thuần là chọn mô hình, mà còn phải đối mặt với nhiều thách thức xuất phát từ đặc trưng ngôn ngữ và hạ tầng dữ liệu.

Một trong những đặc điểm đáng chú ý nhất của tiếng Việt là cấu trúc đơn lập và đa âm tiết rời rạc. Trong tiếng Việt, các từ thường được viết rời theo từng âm tiết nhưng lại tạo thành đơn vị ngữ nghĩa lớn hơn. Ví dụ, từ "sinh viên" được viết thành hai từ tách biệt "sinh" và "viên". Nếu không được xử lý đúng (ví dụ: word segmentation), hệ thống embedding có thể hiểu nhầm "sinh" như một từ riêng biệt, dẫn đến việc biểu diễn không chính xác về ngữ nghĩa.

Tiếng Việt còn chứa nhiều hiện tượng ngôn ngữ đặc trưng như từ láy, từ ghép, đồng âm khác nghĩa, và ngữ pháp linh hoạt. Những yếu tố này khiến việc học embedding trở nên phức tạp hơn vì không chỉ cần hiểu được nghĩa của từng từ mà còn phải xử lý tốt sự kết hợp và thay đổi ý nghĩa theo ngữ cảnh. Ngoài ra, ngôn ngữ nói và viết trong tiếng Việt thường có sự khác biệt rõ rệt, đòi hỏi mô hình phải thích nghi linh hoạt với cả hai dạng này trong các ứng dụng như chatbot.

Một trở ngại khác là sự hạn chế của tài nguyên ngữ liệu tiếng Việt. So với các ngôn ngữ như tiếng Anh, tiếng Việt thiếu các tập dữ liệu lớn, chuẩn hóa và đa dạng về chủ đề để huấn luyện hoặc tinh chỉnh các mô hình embedding. Việc huấn luyện mô hình từ đầu thường tốn kém thời gian và tài nguyên, trong khi các mô hình pretrained hiện có đôi khi chưa đủ mạnh để xử lý các tác vụ chuyên biệt hoặc ngữ cảnh hẹp.

Cuối cùng, các công cụ tiền xử lý (như tokenizer, segmenter) cho tiếng Việt đôi khi vẫn chưa hoàn hảo, có thể gây ảnh hưởng đến chất lượng embedding. Trong một hệ thống như RAG, nếu văn bản không được token hóa đúng hoặc câu hỏi không được xử lý ngữ nghĩa chính xác, embedding đầu ra sẽ thiếu nhất quán và làm giảm độ chính xác của toàn bộ hệ thống.

1.3.4. Tầm quan trọng của embedding

Embedding đóng vai trò nền tảng và không thể thiếu trong hầu hết các hệ thống xử lý ngôn ngữ tự nhiên hiện đại, đặc biệt là trong các hệ thống Chatbot ứng dụng kiến

trúc RAG (Retrieval-Augmented Generation). Việc biểu diễn văn bản dưới dạng vector số học có khả năng nắm bắt ngữ nghĩa giúp các mô hình máy học vượt qua giới hạn của biểu diễn rời rạc truyền thống, từ đó nâng cao khả năng hiểu và xử lý ngôn ngữ tự nhiên một cách chính xác hơn.

Trước đây, các phương pháp biểu diễn văn bản như Bag-of-Words hay TF-IDF thường thiếu khả năng biểu diễn ngữ cảnh và không thể hiện được mối quan hệ ngữ nghĩa giữa các từ. Ví dụ, hai câu có ý nghĩa tương đồng nhưng sử dụng từ vựng khác nhau có thể được biểu diễn hoàn toàn khác biệt theo các phương pháp cũ. Trong khi đó, embedding hiện đại như Word2Vec, FastText, hay các mô hình ngữ cảnh hóa như BERT, PhoBERT, và bge-m3 cho phép ánh xạ các câu hoặc đoạn văn bản có cùng hoặc gần nghĩa đến các không gian vector gần nhau, từ đó cải thiện đáng kể hiệu quả trong các tác vụ phân loại, truy xuất thông tin, và sinh câu trả lời.

Trong kiến trúc RAG, embedding là cầu nối giữa phần truy xuất và phần sinh ngôn ngữ. Cụ thể, embedding giúp chuyển đổi câu hỏi của người dùng và toàn bộ tập văn bản dữ liệu thành các vector có cùng không gian. Việc tìm kiếm các đoạn văn bản liên quan sẽ được thực hiện dựa trên khoảng cách giữa các vector này (ví dụ: cosine similarity), thay vì so khớp từ khóa thuần túy. Điều này cho phép hệ thống tìm ra các đoạn có liên quan về ngữ nghĩa, kể cả khi câu hỏi và đoạn văn không sử dụng từ ngữ trùng khớp. Nhờ đó, khả năng hiểu và phục vụ nhu cầu người dùng được nâng cao rõ rệt.

Đối với tiếng Việt, embedding còn có vai trò đặc biệt quan trọng trong việc xử lý các hiện tượng ngôn ngữ đặc thù như từ ghép, từ láy, hoặc các biến thể hình thái của từ. Một mô hình embedding tốt sẽ giúp hệ thống hiểu rằng các từ như “học sinh”, “học trò” hay “sinh viên” tuy khác nhau về hình thức nhưng có thể gần nghĩa trong các ngữ cảnh nhất định. Điều này đặc biệt có ích trong việc xây dựng hệ thống Chatbot hoạt động trong môi trường học thuật hoặc cơ quan hành chính, nơi mà việc đảm bảo độ chính xác ngữ nghĩa là yếu tố then chốt.

Ngoài ra, embedding còn là yếu tố quyết định đến hiệu suất và hiệu quả của hệ thống trong các khía cạnh như tốc độ truy vấn, khả năng mở rộng hệ thống, và tính thích ứng với dữ liệu mới. Một hệ thống embedding được huấn luyện tốt có thể mã hóa hiệu quả cả những câu hỏi chưa từng gặp trước đó, từ đó giảm thiểu nhu cầu phải huấn luyện lại mô hình thường xuyên.

Tóm lại, embedding không chỉ là bước tiền xử lý đơn thuần, mà là một thành phần cốt lõi quyết định mức độ thông minh và hiệu quả của toàn bộ hệ thống Chatbot. Việc lựa chọn mô hình embedding phù hợp, tối ưu hóa cho ngôn ngữ tiếng Việt, và

tích hợp tốt với kiến trúc tổng thể sẽ tạo tiền đề vững chắc cho sự thành công của hệ thống trí tuệ nhân tạo trong thực tiễn.

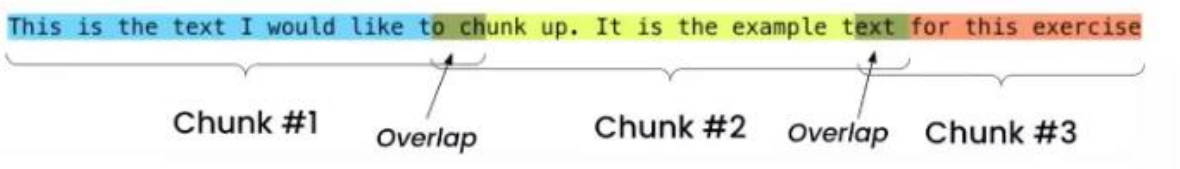
1.4. CÁC PHƯƠNG PHÁP PHÂN ĐOẠN VĂN BẢN

Chunking (phân đoạn) là quá trình chia prompts hoặc documents thành nhiều chunks, hoặc gọi là segments nhỏ hơn nhưng vẫn có nghĩa. Các chunks này có thể được cắt theo một kích thước cố định, chẳng hạn như số lượng ký tự, câu hoặc đoạn văn cụ thể. Trong RAG, mỗi chunk được mã hóa (encode) thành 1 vector embedding để truy vấn (retrieval). Phân tách các chunks đủ nhỏ, chính xác sẽ giúp kết quả truy xuất thông tin giữa câu truy vấn của người dùng và nội dung chunk chính xác hơn. Việc phân tách chunks quá lớn có thể bao gồm nhiều thông tin không liên quan, gây nhiễu và có khả năng làm giảm độ chính xác của retrieval. Bằng các việc kiểm soát kích thước của các chunks, RAG có thể cân bằng giữa lượng thông tin đầy đủ và tính chính xác.

Tuy nhiên, việc chunk data thường rất phức tạp và phụ thuộc nhiều vào cấu trúc của loại dữ liệu đó, không có một phương pháp nào tốt nhất cho tất cả. Việc lựa chọn kích thước chunking trong RAG là rất quan trọng. Nó cần đủ nhỏ để đảm bảo tính liên quan và giảm nhiễu nhưng cũng cần đủ lớn để duy trì được đủ thông tin, giữ được ngữ cảnh. Một vài phương pháp chunking phổ biến:

1.4.1. Fixed-Size Chunking

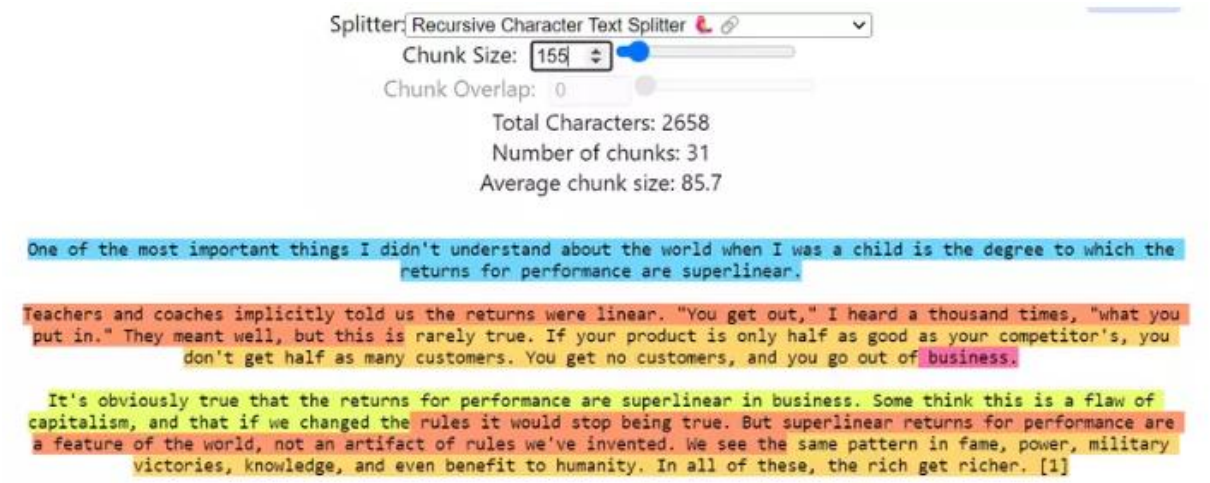
Phương pháp Fixed-Size Chunking đơn giản chia văn bản thành những đoạn có độ dài cố định, tính theo số ký tự hoặc số từ, hoàn toàn không xét đến ngữ nghĩa, cấu trúc hay nội dung bên trong. Việc chia này thường được xác định thông qua hai tham số cơ bản: độ dài chunk (chunk size) và độ chồng lắp giữa các đoạn (chunk overlap). Chunk size thể hiện số ký tự tối đa cho mỗi chunk, ví dụ 128 hay 256 ký tự, trong khi chunk overlap quy định số ký tự được lặp lại giữa các chunk liên tiếp, nhằm hạn chế mất mát thông tin ở ranh giới. Cách chia này dễ triển khai và được hỗ trợ sẵn trong các framework như LangChain (CharacterTextSplitter) hay LlamaIndex (SentenceSplitter). Tuy nhiên, phương pháp này dễ cắt ngang ý, tách đôi câu hay đoạn, dẫn đến việc mỗi chunk có thể không mang đủ ngữ cảnh để mô hình hiểu và truy xuất chính xác.



Hình 1.14. Ví dụ về Fixed Chucking

1.4.2. Recursive Chunking

Recursive Chunking cải thiện nhược điểm của Fixed-Size Chunking bằng cách phân tách văn bản dựa trên thứ tự ưu tiên của các ký tự phân tách (separators), ví dụ ngắt đoạn kép (\n\n), ngắt đoạn đơn (\n), khoảng trắng hay ký tự trống. Thuật toán bắt đầu với separator có độ ưu tiên cao nhất, cố gắng chia văn bản thành các chunk càng dài càng tốt mà không vượt quá kích thước cho phép và vẫn giữ được tính toàn vẹn của phân đoạn chứa separator đó. Nếu không thể tách được tại separator ưu tiên, thuật toán sẽ lùi xuống separator tiếp theo, chẳng hạn từ ngắt dòng xuống khoảng trắng, để chia đoạn. Qua nhiều lần đệ quy, thuật toán phân tách văn bản ở những điểm ngữ nghĩa quan trọng trước, chỉ dùng cách cắt thô bằng ký tự khi không còn lựa chọn. Nhờ đó, Recursive Chunking vừa duy trì cấu trúc ngữ nghĩa, vừa đảm bảo giới hạn kích thước.



Hình 1.15. Ví dụ về Recursive Chunking

1.4.3. Document-Based Chunking

Document-Based Chunking dựa hoàn toàn vào cấu trúc sẵn có của tài liệu. Đối với định dạng Markdown, hệ thống sẽ nhận diện các tiêu đề (H1-H6), dấu gạch ngang hoặc dấu trang ngang để tách thành mục; các vùng code Python được chia theo từ khóa như class, def hoặc ngắt đoạn kép; trong HTML chunking theo các thẻ như <p>, <section> hoặc các thẻ tiêu đề <h1> - <h6> giúp giữ nguyên cấu trúc tài liệu. Với PDF, metadata như tiêu đề, mục lục, subtitle và nội dung chính được trích xuất để định nghĩa ranh giới chunk. Phương pháp này đảm bảo mỗi chunk là một đơn vị nội dung logic, không làm xáo trộn bố cục gốc, giúp việc truy xuất thông tin và tái sử dụng đoạn văn dễ dàng hơn.

1.4.4. Semantic Chunking

Khác với những cách chỉ dựa vào định dạng hay ngắt ký tự, Semantic Chunking phân tách dựa trên ngữ nghĩa của câu. Đầu tiên văn bản được tách thành các câu riêng lẻ bằng biểu thức chính quy nhận diện dấu chấm, dấu hỏi hoặc dấu chấm than. Để giảm nhiễu, các câu liền kề được ghép lại theo một tham số buffer_size (ví dụ ghép mỗi câu với một câu trước và một câu sau), tạo thành chuỗi câu lớn hơn. Tiếp đó, mỗi chuỗi câu được chuyển thành embedding và tính toán độ tương đồng lẫn nhau bằng cosine similarity. Bất cứ khi nào khoảng cách (distance = 1 - similarity) giữa hai chuỗi vượt qua ngưỡng breakpoint nhất định (ví dụ 0.8 hoặc 0.95), chúng được tách ra thành hai chunk riêng biệt. Mỗi lần tính toán độ tương đồng tương ứng với một điểm dữ liệu trên biểu đồ; điểm có distance vượt ngưỡng được đánh dấu là vị trí ngắt (break point). Nhờ vậy, Semantic Chunking tạo ra những đoạn chứa thông tin đồng nhất về mặt ngữ nghĩa, tránh việc cắt ngang ý một cách thiếu tự nhiên.

CHƯƠNG 2. PHƯƠNG PHÁP THỰC NGHIỆM

2.1. PHƯƠNG PHÁP THỰC NGHIỆM

Đề tài này áp dụng phương pháp thực nghiệm để xây dựng Chatbot hỗ trợ hoạt động tại phòng đào tạo trường Đại học Công nghiệp Hà Nội. Trọng tâm của phương pháp là quá trình thu thập dữ liệu tự động từ trang web và tích hợp kỹ thuật Retrieval-Augmented Generation (RAG).

2.1.1. Thiết lập thực nghiệm

Để hiện thực hóa đề tài xây dựng Chatbot hỗ trợ hoạt động tại phòng đào tạo, việc lựa chọn ngôn ngữ lập trình, thư viện và công cụ phát triển phù hợp là yếu tố then chốt, ảnh hưởng trực tiếp đến hiệu quả, khả năng mở rộng và chất lượng của hệ thống. Trong khuôn khổ đề tài này, Python được lựa chọn là ngôn ngữ lập trình chính nhờ vào tính phổ biến, cú pháp dễ tiếp cận và hệ sinh thái phong phú trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP) và trí tuệ nhân tạo.

Trong quá trình thực nghiệm, thư viện Huggingface Transformers được sử dụng để triển khai các mô hình embedding văn bản. Cụ thể, các đoạn văn bản thu thập từ trang web và truy vấn của người dùng được mã hóa thành các vector ngữ nghĩa thông qua các mô hình tiền huấn luyện như Sentence Transformers. Việc biểu diễn văn bản dưới dạng vector giúp hệ thống có thể so sánh độ tương đồng giữa truy vấn và ngữ liệu, từ đó phục vụ cho quá trình truy xuất thông tin hiệu quả và chính xác.

Bên cạnh đó, đề tài sử dụng framework Langchain làm nền tảng xây dựng kiến trúc Chatbot theo hướng tiếp cận Retrieval-Augmented Generation (RAG). Langchain cho phép tích hợp linh hoạt giữa các mô hình ngôn ngữ lớn (LLM) và hệ thống truy xuất thông tin, như bộ mã hóa embedding và cơ sở dữ liệu vector. Thông qua đó, hệ thống có thể xử lý truy vấn của người dùng, thực hiện tìm kiếm ngữ cảnh phù hợp, và tạo ra câu trả lời giàu ngữ nghĩa, chính xác và dễ mở rộng. Langchain không chỉ giúp đơn giản hóa quá trình kết nối các thành phần mà còn tạo điều kiện thuận lợi cho việc xây dựng các luồng xử lý phức tạp, góp phần nâng cao hiệu suất và chất lượng phản hồi của Chatbot.

Sự kết hợp giữa các công cụ chính là nền tảng cốt lõi giúp tạo nên một hệ thống Chatbot thông minh, linh hoạt và hiệu quả trong việc hỗ trợ người dùng tra cứu thông tin học vụ một cách tự động và chính xác.

2.1.2. Lựa chọn mô hình

2.1.2.1. Mô hình nhúng

Mô hình hiieu/halong_embedding được phát triển như một giải pháp embedding văn bản chuyên sâu cho tiếng Việt, đồng thời vẫn giữ khả năng đa ngôn ngữ nhờ tiền huấn luyện trên cơ sở intfloat/multilingual-e5-base. Trọng tâm của mô hình này là hỗ trợ các hệ thống Retrieval-Augmented Generation (RAG) và tối ưu hóa hiệu suất sản xuất khi phải xử lý khối lượng lớn dữ liệu văn bản.

Trong quá trình huấn luyện, đội ngũ phát triển đã sử dụng một tập dữ liệu nội bộ gồm khoảng 100.000 ví dụ ghép giữa câu hỏi và tài liệu liên quan. Mỗi cặp dữ liệu này không chỉ giúp mô hình nắm bắt mối quan hệ ngữ nghĩa giữa truy vấn và đoạn văn tham chiếu, mà còn đảm bảo độ chính xác cao khi áp dụng vào các bài toán tìm kiếm ngữ nghĩa, so sánh tương đồng câu hay khai thác paraphrase.

Điểm nổi bật trong cơ chế huấn luyện của hiieu/halong_embedding chính là việc áp dụng Matryoshka Loss – một lớp hàm mất mát cho phép tạo ra nhiều cấp độ embedding khác nhau từ cùng một mô hình 768 chiều. Nhờ đó, người dùng có thể cắt ngắn embedding xuống các kích thước nhỏ hơn (ví dụ 512 hay 256 chiều) mà chỉ chịu giảm nhẹ về hiệu suất. Khả năng này đặc biệt hữu ích trong những môi trường yêu cầu độ trễ thấp, tiết kiệm bộ nhớ hoặc khi triển khai trên các thiết bị tài nguyên hạn chế.

Về mặt kỹ thuật, hiieu/halong_embedding chấp nhận đầu vào tối đa 512 token và sinh ra vector 768 chiều, sau đó thường được chuẩn hóa L2 để các embedding có độ dài chuẩn bằng 1, hỗ trợ tốt cho việc so sánh bằng cosine similarity. Sự kết hợp giữa kiến trúc Bi-Encoder của Sentence-Transformers và quá trình fine-tune từ intfloat/multilingual-e5-base giúp mô hình vừa đảm bảo tốc độ sinh embedding nhanh, vừa đạt độ chính xác cao.

Được cấp phép theo Apache-2.0, hiieu/halong_embedding không chỉ phù hợp cho nghiên cứu học thuật mà còn sẵn sàng để đưa vào các hệ thống thương mại. Bên cạnh mã nguồn chính, tác giả còn cung cấp bộ script đánh giá (eval) và hướng dẫn fine-tune, cùng với tài liệu seminar chi tiết, giúp người dùng dễ dàng triển khai và tùy biến cho nhu cầu riêng.

2.1.2.2. Mô hình ngôn ngữ lớn

Trong đề tài này, phiên bản *Gemini 1.5 Flash* được sử dụng làm mô hình ngôn ngữ lớn (LLM) chính để thực hiện các tác vụ xử lý ngôn ngữ tự nhiên trong hệ thống. Đây là một sản phẩm LLM do Google phát triển – nhờ khả năng cân bằng hiệu quả giữa tốc độ xử lý, chi phí vận hành và hiệu suất tổng thể. Đây là biến thể được thiết kế

chuyên biệt cho các ứng dụng yêu cầu phản hồi nhanh, xử lý thời gian thực, đồng thời vẫn duy trì được chất lượng suy luận ở mức ổn định.

Gemini 1.5 Flash nổi bật với khả năng xử lý đa phương thức (multimodal), cho phép mô hình tiếp nhận và phân tích các loại dữ liệu khác nhau như văn bản, hình ảnh, âm thanh và video trong cùng một lượt xử lý. Tính năng này đặc biệt hữu ích trong các ứng dụng hội thoại nâng cao hoặc hệ thống tư vấn tích hợp nhiều nguồn thông tin.

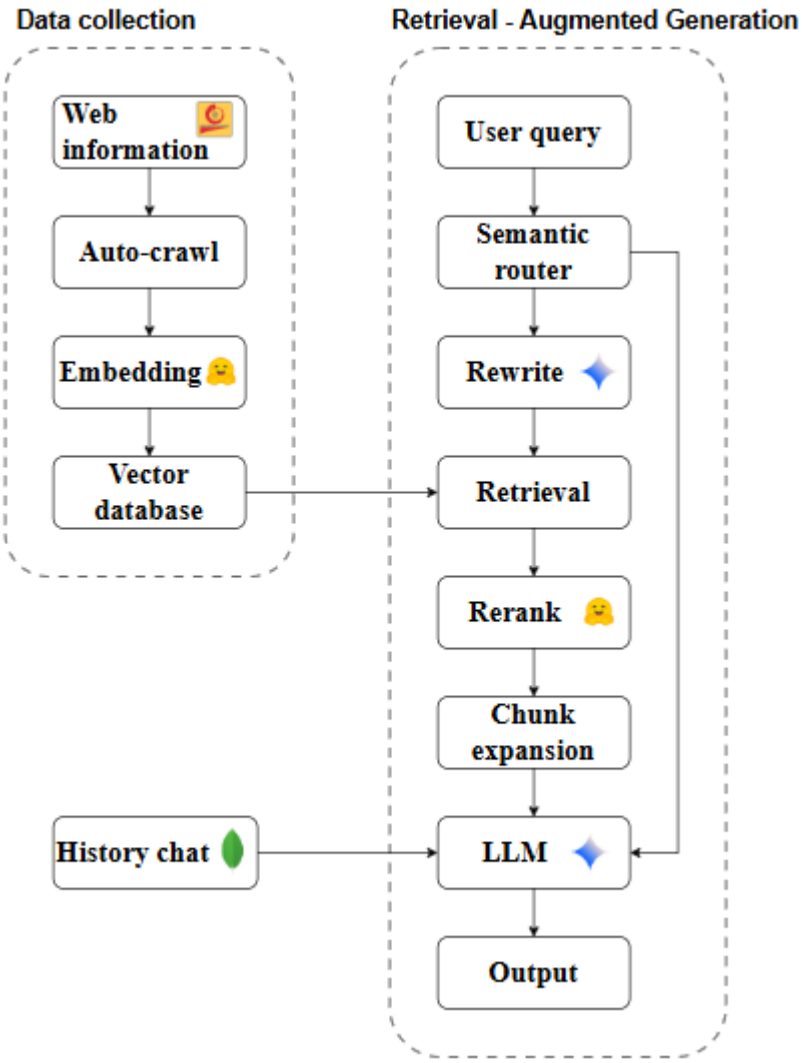
Về mặt hiệu năng, Gemini 1.5 Flash đạt tốc độ sinh đầu ra lên đến 194 token mỗi giây, vượt trội so với các mô hình cùng phân khúc, chẳng hạn như GPT-4o Mini chỉ đạt khoảng 99 token mỗi giây. Đây là yếu tố then chốt giúp nâng cao trải nghiệm người dùng trong các hệ thống yêu cầu phản hồi tức thì. Bên cạnh đó, mô hình còn hỗ trợ cửa sổ ngữ cảnh (context window) lên đến 1 triệu token trong phiên bản API, cho phép xử lý toàn bộ kho dữ liệu lớn như mã nguồn, tài liệu kỹ thuật, hoặc lịch sử hội thoại kéo dài – một lợi thế rõ rệt trong các hệ thống RAG (Retrieval-Augmented Generation) hoặc các ứng dụng chuyên sâu về hiểu ngữ cảnh.

Một trong những điểm mạnh đáng kể khác của Gemini 1.5 Flash là chi phí vận hành cực kỳ thấp. Với mức giá chỉ 0.1 USD cho mỗi 1 triệu token, mô hình này rẻ hơn gấp ba lần so với các đối thủ như GPT-4o Mini và thấp hơn rất nhiều so với Gemini 1.5 Pro (2.2 USD/1M tokens). Điều này giúp giảm đáng kể chi phí triển khai cho các hệ thống quy mô lớn, đồng thời mở ra cơ hội ứng dụng trong các dự án có giới hạn tài nguyên.

Mặc dù Gemini 1.5 Flash có điểm số 81% trên bộ benchmark MMLU (Massive Multitask Language Understanding) – thấp hơn một số mô hình cao cấp như Gemini 1.5 Pro (86%) – nhưng hiệu suất này vẫn nằm trong mức tốt và hoàn toàn đáp ứng được yêu cầu trong các tác vụ thông thường. Việc đánh đổi một phần nhỏ về khả năng suy luận để đạt được tốc độ, khả năng mở rộng và chi phí tối ưu là lựa chọn hợp lý đối với hệ thống mà dự án đang xây dựng.

Tóm lại, Gemini 1.5 Flash là mô hình phù hợp cho các ứng dụng AI cần tốc độ cao, chi phí thấp và khả năng xử lý đa dạng ngữ cảnh. Với cấu hình mạnh mẽ và khả năng thích ứng linh hoạt, mô hình này là lựa chọn chiến lược cho các hệ thống hội thoại và truy xuất tri thức thông minh hiện đại.

2.2. TIẾN HÀNH THỰC NGHIỆM



Hình 2.1. Kiến trúc hệ thống Chatbot

Để xây dựng một hệ thống ChatBot thông minh có khả năng truy xuất và tổng hợp thông tin chính xác từ dữ liệu thực tế, đề tài áp dụng một quy trình nghiên cứu gồm ba bước chặt chẽ, bao gồm: thu thập và xử lý dữ liệu, biến đổi và lập chỉ mục dữ liệu, và ứng dụng luồng xử lý RAG. Mỗi bước đóng vai trò thiết yếu trong việc đảm bảo chất lượng đầu vào, khả năng truy xuất hiệu quả và độ chính xác trong phản hồi đầu ra của hệ thống.

2.2.1. Thu thập và xử lý dữ liệu

Trong bối cảnh ngày càng có nhiều nhu cầu phân tích dữ liệu trực tuyến, việc tự động hoá việc thu thập và làm sạch thông tin từ các trang web trở thành một phần không thể thiếu. Đối với chuyên mục "Tin tức tuyển sinh" và "Hỏi đáp" trên trang chính thức của Trường Đại học Công nghiệp Hà Nội, lượng bài viết được cập nhật đều đặn và chứa đựng nhiều thông tin quan trọng phục vụ cho công tác tư vấn, nghiên cứu

hoặc nhu cầu truyền thông. Hệ thống được thiết kế nhằm tự động thu thập toàn bộ bài viết trong chuyên mục này, sau đó áp dụng các kỹ thuật xử lý ngôn ngữ tự nhiên để loại bỏ các thành phần nhiễu, đảm bảo dữ liệu đầu ra có chất lượng cao và đồng nhất về cấu trúc. Giải pháp sử dụng ngôn ngữ lập trình Python, tích hợp các thư viện như BeautifulSoup và API của Google để thực hiện việc trích xuất, tinh chỉnh và lưu trữ nội dung. Dữ liệu sau khi được xử lý sẽ được định dạng dưới dạng tệp JSON, phục vụ cho các mục đích phân tích hoặc nghiên cứu. Điều này hệ thống hóa việc trích xuất và tinh chỉnh nội dung để tạo ra một bộ dữ liệu có cấu trúc. Dưới đây là tóm tắt cách tiếp cận.

2.2.1.1. Quy trình thu thập dữ liệu

Quá trình thu thập bắt đầu từ một trang tin gốc và được thiết kế để chỉ thu thập các liên kết nội bộ, đảm bảo rằng tất cả nội dung đều nằm trong cùng một miền.

Tổng hợp liên kết: hệ thống lần lượt truy cập từng trang trong một danh sách trang đã định sẵn, từ đó trích xuất các liên kết bài viết thông qua việc phân tích cấu trúc HTML. Tiếp đó, hệ thống kiểm tra tính hợp lệ của từng đường dẫn bằng cách so sánh tên miền của liên kết đó với tên miền gốc. Chỉ những liên kết nội bộ nằm cùng miền mới được đưa vào danh sách thu thập, nhằm loại bỏ những đường dẫn dẫn ra ngoài phạm vi website tuyển sinh. Danh sách các liên kết hợp lệ này sẽ được dùng để thu thập nội dung của các bài viết.

Trích xuất nội dung: sau khi xác định được danh sách các bài viết cần thu thập, hệ thống lần lượt truy cập từng trang và trích xuất nội dung chính. Cụ thể, tiêu đề bài viết được xác định từ khối chứa tiêu đề, trong khi phần nội dung văn bản được lấy từ các thẻ HTML phổ biến như <p>, và <table> sử dụng BeautifulSoup.

2.2.1.2. Xử lý dữ liệu

Toàn bộ văn bản đã được thu thập từ các trang web sẽ được chuyển đến một mô hình ngôn ngữ lớn (Generative AI) của Google mang tên Gemini 1.5 Flash thông qua API. Mục tiêu của bước này không phải là rút gọn hay tóm tắt nội dung, mà đơn thuần là loại bỏ những thành phần thừa thãi chẳng hạn như dấu câu dư, các từ ngữ lặp lại hoặc không mang thông tin hữu ích giúp cho văn bản trở nên cô đọng, mạch lạc và dễ đọc hơn. Nhờ việc ứng dụng kỹ thuật xử lý ngôn ngữ tự nhiên của mô hình, bộ dữ liệu có thể duy trì nguyên vẹn các thông tin quan trọng, bao gồm cả dữ liệu dạng bảng, trong khi đảm bảo tính nhất quán về mặt ngôn từ.

Sau khi văn bản đã được làm sạch, hệ thống sẽ tổ chức kết quả dưới dạng một tệp JSON duy nhất. Mỗi phần tử trong mảng JSON gồm ba thành phần chính: đường dẫn

của trang nguồn, tiêu đề bài viết và đoạn nội dung đã được tinh chỉnh. Cấu trúc dữ liệu này được thiết kế để tối ưu cho việc truy xuất và phân tích tự động, cho phép các công cụ downstream hoặc các nhà phân tích có thể dễ dàng lọc, truy vấn và trực quan hóa thông tin theo nhu cầu.

2.2.1.3. Bộ dữ liệu cuối cùng

Kết quả bộ dữ liệu thu được bao gồm:

- Các liên kết URL của các bài viết đã thu thập.
- Tiêu đề và nội dung đã được làm sạch và có cấu trúc từ bài viết đó.

Các kết quả này được lưu trong tệp JSON để sử dụng cho các nghiên cứu hoặc phân tích tiếp theo.


```
[
  {
    "link": "https://tuyensinh.haui.edu.vn/tin-tuc/thong-tin-tuyen-sinh-dai-hoc-nam-2025/680f9d53f721616a54f6495f",
    "title": "Thông tin tuyển sinh đại học năm 2025",
    "content": "THÔNG TIN TUYỂN SINH ĐẠI HỌC NĂM 2025\n\nI. ..."
  },
  {
    "link": "https://tuyensinh.haui.edu.vn/tin-tuc/ket-qua-xet-tuyen-dao-tao-tu-xa-trinh-do-dai-hoc-dot-2-nam-2025/67f648ccc052c",
    "title": "Kết quả xét tuyển đào tạo từ xa trình độ đại học đợt 2 năm 2025",
    "content": "Trường Đại học Công nghiệp Hà Nội thông báo kết quả xét tuyển đào tạo từ xa trình độ đại học đợt 2 năm 2025 "
  },
  {
    "link": "https://tuyensinh.haui.edu.vn/tin-tuc/ket-qua-xet-tuyen-dao-tao-tu-xa-trinh-do-dai-hoc-dot-thang-01-nam-2025/67bbfb2",
    "title": "Kết quả xét tuyển đào tạo từ xa trình độ đại học đợt tháng 01 năm 2025",
    "content": "Trường Đại học Công nghiệp Hà Nội thông báo kết quả xét tuyển đào tạo ..."
  }
]
```


Hình 2.2. Minh họa cấu trúc tệp JSON

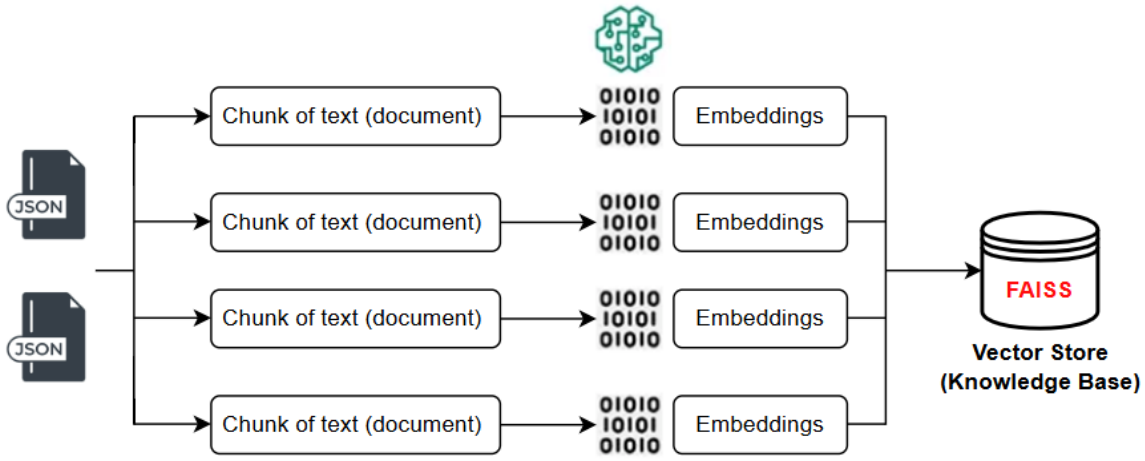
Bên cạnh đó hệ thống cũng tự động tổng kết lại hai chỉ số quan trọng: tổng số liên kết đã thu thập, tổng số ký tự của nội dung văn bản đã thu thập và tinh chỉnh.


```
{
    "status": "completed",
    "total_links_crawled": 10,
    "total_characters_crawled": 51577,
    "error": null
}
```


Hình 2.3. Trạng thái hệ thống thu thập dữ liệu

Phương pháp này đảm bảo dữ liệu được thu thập có liên quan, nhất quán và được định dạng để phân tích thêm trong khi vẫn duy trì hiệu quả và khả năng mở rộng.

2.2.2. Biến đổi và lập chỉ mục dữ liệu



Hình 2.4. Quy trình biến đổi và lưu trữ dữ liệu

2.2.2.1. Phân đoạn dữ liệu

Sau khi được lưu vào tệp JSON, dữ liệu tiếp tục được chia thành các đoạn nhỏ hơn, mỗi đoạn chứa khoảng 150–350 token, dựa trên phương pháp phân đoạn đệ quy (recursive chunking) để đảm bảo tính nhất quán về độ dài và ngữ cảnh. Phương pháp này khởi đầu bằng cách tách văn bản thành các khối lớn hơn theo trình tự ưu tiên: đầu tiên là các tiêu đề và mục lớn, sau đó là các đoạn văn, và cuối cùng mới đến các câu. Tại mỗi cấp độ, nếu khối văn bản vượt quá giới hạn token, nó sẽ được chia tiếp thành các phần con bằng cách tìm điểm ngắt thích hợp (ví dụ: kết thúc câu hoặc dấu chấm câu gần nhất). Quá trình chia nhỏ tiếp tục đệ quy cho đến khi mọi đoạn con đều nằm trong khoảng 150–350 token. Các đoạn kết quả không những có độ dài đồng đều mà còn giữ được tính mạch lạc về mặt cấu trúc nội dung. Mỗi đoạn sau khi được tạo sẽ được lưu cùng với tham chiếu đến tài liệu gốc để dễ dàng truy vết.

2.2.2.2. Chuẩn hóa embeddings

Sau khi dữ liệu đã được phân mảnh, mỗi đoạn này được nhúng vào một vectơ dày đặc (dense) bằng mô hình hiieu/halong_embedding thông qua thư viện HuggingFace Embeddings. Quá trình này rất quan trọng đối với khả năng của RAG trong việc truy xuất thông tin phù hợp nhất từ kho khi thức với câu truy vấn của người dùng. Cuối cùng các vectơ nhúng sẽ là nền tảng cho các tác vụ lập chỉ mục, tìm kiếm và truy xuất ngữ nghĩa trong các bước xử lý tiếp theo.

2.2.2.3. Lưu trữ và quản lý vectorstore

Sau khi các vectơ embeddings đã được chuẩn hóa và gắn metadata tương ứng với từng đoạn văn, bước tiếp theo là lưu trữ và quản lý chúng trong một VectorStore nhằm phục vụ việc tra cứu ngữ nghĩa hiệu quả. Trong báo cáo này, FAISS (Facebook AI Similarity Search) được lựa chọn làm nền tảng chính, nhờ khả năng xử lý quy mô lớn và tốc độ truy vấn cao.

FAISS là một thư viện mã nguồn mở được phát triển để hỗ trợ tìm kiếm k-nearest neighbors (k-NN) trên các tập dữ liệu vector rất lớn. Về bản chất, FAISS cung cấp nhiều cấu trúc chỉ mục (index) khác nhau, từ phương pháp duyệt toàn phần (brute-force) cho đến các thuật toán phân vùng không gian như IVF (Inverted File) và đồ thị HNSW (Hierarchical Navigable Small World). Tùy theo yêu cầu về độ chính xác và tốc độ trả lời, có thể lựa chọn hoặc kết hợp các index sao cho cân bằng được giữa chi phí bộ nhớ, thời gian xây dựng chỉ mục và tốc độ truy vấn.

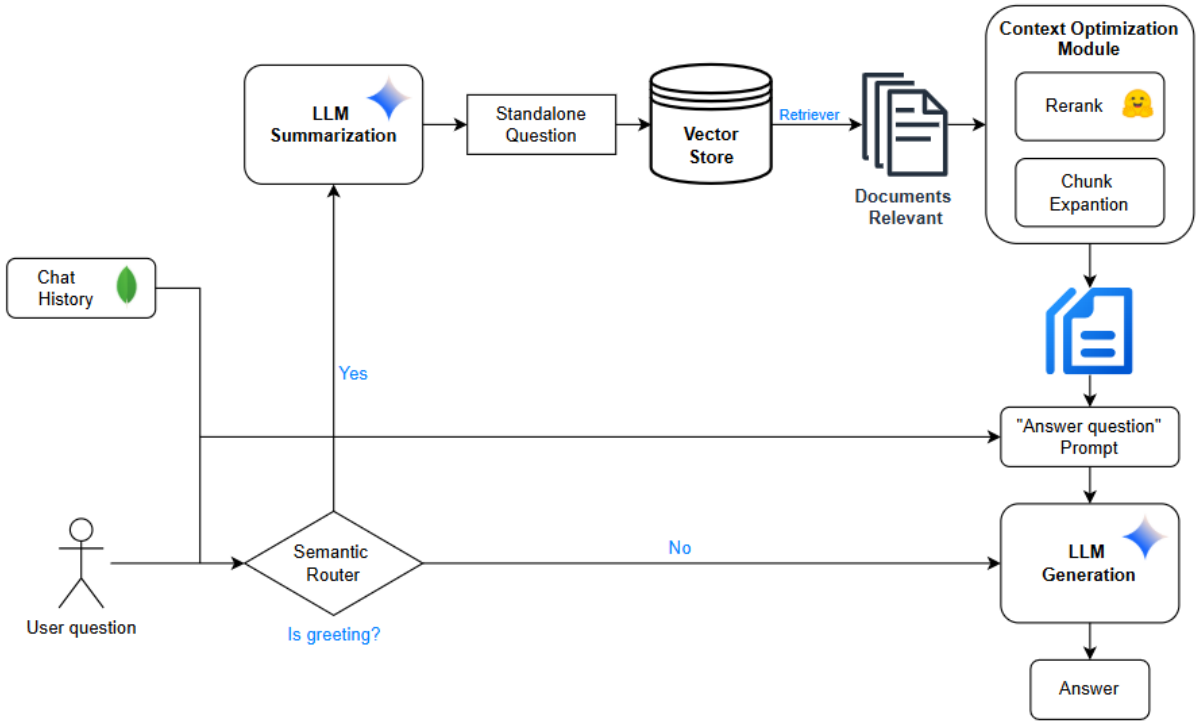
Quy trình xây dựng VectorStore khởi đầu bằng việc khởi tạo index. Để sử dụng IVF, cần thực hiện bước huấn luyện (train) với một tập mẫu embeddings nhằm phân chia không gian vector thành các cụm (clusters) thông qua thuật toán k-means. Khi index đã được huấn luyện, toàn bộ embeddings chuẩn hóa sẽ được thêm (add) vào FAISS, kèm theo các chỉ số tham chiếu đến tài liệu gốc. Quá trình này cho phép FAISS nhanh chóng xác định nhóm cụm chứa vectơ truy vấn, từ đó giới hạn phạm vi so sánh nội bộ, giúp tăng tốc độ tìm kiếm.

Ở khâu tra cứu, mỗi truy vấn đầu tiên được biến đổi thành embedding với cùng quy trình chuẩn hóa L2 như dữ liệu gốc. Embedding truy vấn sau đó được nộp vào index FAISS để thực hiện phép tìm k vectơ gần nhất (k-NN search). Kết quả trả về là danh sách các chỉ số vector và khoảng cách cosine tương ứng, cho phép tra cứu metadata kèm theo như chunk_id, tên tài liệu gốc, và vị trí trong văn bản. Nhờ vậy, hệ thống có thể nhanh chóng tái tạo ngữ cảnh và trả về các đoạn văn phù hợp nhất với truy vấn người dùng.

Trong quá trình vận hành, vectorStore cần được cập nhật khi có thêm dữ liệu mới hoặc khi cần loại bỏ các tài liệu không còn phù hợp. FAISS hỗ trợ việc thêm (add) và xóa (remove) vector động mà không phải xây dựng lại hoàn toàn chỉ mục. Tuy nhiên, sau nhiều lần cập nhật, hiệu quả phân vùng của index có thể giảm xuống; do đó, định kỳ có thể thực hiện lại bước huấn luyện và xây dựng index mới để tối ưu hiệu suất. Ngoài ra, các cơ chế sao lưu (backup) và ghi nhật ký (logging) cần được triển khai đồng thời để đảm bảo tính toàn vẹn dữ liệu và dễ dàng khôi phục khi xảy ra sự cố.

Nhờ FAISS, vectorstore trở thành tầng hạ tầng vững chắc cho mọi tác vụ tìm kiếm ngữ nghĩa và truy xuất thông tin trong hệ thống Retrieval-Augmented Generation, đáp ứng tốt cả yêu cầu về tốc độ, độ chính xác và khả năng mở rộng.

2.2.3. Thiết kế luồng xử lý của RAG



Hình 2.5. Luồng xử lý của RAG

Luồng xử lý của hệ thống Retrieval-Augmented Generation (RAG) tuân theo thiết kế dạng module để xử lý các truy vấn của người dùng và tạo ra phản hồi phù hợp theo ngữ cảnh:

- Semantic router: Khi người dùng nhập một truy vấn (query) bằng tiếng Việt, truy vấn này sẽ được xác định xem có cần được truy vấn hay không thông qua bộ định tuyến ngữ nghĩa (semantic router). Nếu không cần truy vấn, bộ định tuyến sẽ gửi truy vấn trực tiếp đến mô hình ngôn ngữ lớn (LLM) để tạo phản hồi.
- Ngữ cảnh hóa câu truy vấn: Để cho quá trình truy xuất đạt được hiệu quả cao hơn trong cuộc trò chuyện, việc liên kết với lịch sử cuộc trò chuyện là một giải pháp tốt. Khi đó, câu truy vấn của người dùng sẽ được tổng hợp cùng với lịch sử cuộc trò chuyện trong một công cụ prompt (prompt engine) và tích hợp vào mô hình ngôn ngữ lớn (LLM) - cụ thể là Google Gemini API nhằm thực hiện việc viết lại truy vấn. Lợi ích chính của việc viết lại truy vấn là làm tăng tính rõ ràng, độ cụ thể và độ phù hợp ngữ nghĩa của truy vấn, nhờ đó hệ

thống truy xuất có thể tìm được các đoạn thông tin liên quan với độ chính xác cao hơn.

- Embedding truy vấn: Câu truy vấn sau khi được viết lại sẽ nhúng vào mô hình embedding để chuyển đổi thành vector. Mô hình này đã được sử dụng trong quá trình tạo ra cơ sở dữ liệu tri thức (vectorstore) ở bước Biến đổi và lập chỉ mục dữ liệu, điều này là cần thiết vì khi đó câu truy vấn sẽ được đồng bộ với cùng quy trình chuẩn hóa như dữ liệu gốc.
- Tìm kiếm tương đồng: Vector truy vấn sẽ được sử dụng để thực hiện tìm kiếm tương đồng (L2) với tất cả các vector trong cơ sở dữ liệu đã được lập chỉ mục.
- Truy xuất tài liệu: Hệ thống xác định và đưa ra k đoạn văn bản có vector gần nhất (tương đồng nhất về ngữ nghĩa) với vector truy vấn. Đây là tài liệu liên quan nhất được truy xuất ra từ cơ sở dữ liệu.
- Tối ưu hóa ngữ cảnh: Để tăng mức độ uy tín cho các đoạn tài liệu đã truy xuất, một module tối ưu hóa ngữ cảnh được sử dụng. Hai thành phần chính là xếp hạng lại tài liệu liên quan (rerank) và mở rộng thông tin (chunk expansion) đóng vai trò then chốt trong việc cải thiện độ chính xác và tính đầy đủ của thông tin truy xuất trước khi đưa vào mô hình ngôn ngữ lớn (LLM)
- Tạo Prompt tăng cường: Sau khi xác định được các đoạn tài liệu liên quan phù hợp, một prompt hoàn chỉnh được tổng hợp bao gồm câu hỏi người dùng (query), lịch sử cuộc trò chuyện (chat history) và tài liệu liên quan (relevant documents).
- Sinh phản hồi: Prompt tăng cường sẽ được chuyển đến LLM, mô hình này sẽ nhận trách nhiệm tổng hợp phản hồi, để sinh ra một câu trả lời mạch lạc, đầy đủ, đảm bảo nội dung phù hợp với ngữ cảnh.

Toàn bộ quy trình xử lý trên được hiện thực hóa dưới dạng một chuỗi tích hợp trong LangChain, cho phép hệ thống hoạt động theo hướng module hóa – mỗi thành phần đảm nhận một chức năng chuyên biệt trong việc xử lý truy vấn và tạo phản hồi. Thiết kế dạng module này không chỉ giúp tăng khả năng mở rộng và linh hoạt của hệ thống, mà còn nâng cao hiệu quả xử lý và độ chính xác trong việc trả lời các câu hỏi mang tính ngữ cảnh cao. Các thành phần trong chuỗi sẽ được trình bày chi tiết ở các mục tiếp theo để làm rõ vai trò, cơ chế hoạt động và đóng góp của từng module trong luồng xử lý RAG.

2.2.3.1. Bộ định tuyến ngữ nghĩa

Bộ định tuyến ngữ nghĩa (semantic router) là một thành phần quan trọng giúp xác định xem truy vấn của người dùng có cần truy xuất thông tin hay không. Mục tiêu

chính của nó là cải thiện thời gian phản hồi và nâng cao sự hài lòng của người dùng bằng cách xử lý truy vấn một cách hiệu quả. Hai phương pháp đã được xem xét cho module này.

Phương pháp đầu tiên sử dụng mô hình hiieu/halong_embedding để nhúng các lời chào phổ biến được xác định trước (ví dụ: “xin chào”, “tôi tên là”, v.v.) thành các biểu diễn vector. Khi người dùng gửi một truy vấn, embedding của truy vấn đó sẽ được so sánh với các vector này bằng cách tính khoảng cách trung bình. Nếu khoảng cách vượt quá ngưỡng được xác định, truy vấn sẽ được phân loại là lời chào phổ biến; nếu không, truy vấn sẽ được đánh dấu là cần truy xuất.

Phương pháp thứ hai sử dụng một công cụ prompt được hỗ trợ bởi cùng một mô hình LLM được sử dụng để tạo ra phản hồi cuối cùng (Google Gemini API). Công cụ này tận dụng khả năng suy luận tiên tiến của LLM để phân tích truy vấn của người dùng và quyết định xem có cần truy xuất thông tin hay không.

Cuối cùng, phương pháp thứ hai đã được lựa chọn do tính đa dạng của cách diễn đạt lời chào, vốn có thể thay đổi đáng kể tùy theo ngữ cảnh và người dùng. Trong khi phương pháp đầu tiên phù hợp với các hệ thống có dữ liệu được kiểm soát chặt chẽ và được xử lý kỹ, phương pháp thứ hai mang lại sự linh hoạt và độ chính xác cao hơn trong việc xử lý các truy vấn người dùng linh động.

2.2.3.2. Viết lại truy vấn

Viết lại truy vấn (query rewriting) đóng vai trò quan trọng trong việc cải thiện chất lượng truy vấn đầu vào, từ đó nâng cao hiệu quả truy xuất thông tin trong toàn bộ hệ thống RAG. Truy vấn do người dùng nhập vào có thể mang tính tự nhiên, không rõ ràng, hoặc chưa được tối ưu về mặt ngữ nghĩa. Điều này gây khó khăn cho quá trình tìm kiếm chính xác trong cơ sở dữ liệu vector. Vì vậy, việc cải biên truy vấn trước khi đưa vào bước truy xuất là cần thiết.

Trong hệ thống này, phương pháp được lựa chọn là sử dụng một công cụ prompt (prompt engine) bao gồm câu truy vấn và lịch sử trò chuyện sẽ được tích hợp với mô hình ngôn ngữ lớn (LLM) - cụ thể là Google Gemini API nhằm thực hiện việc viết lại truy vấn. Mô hình này không chỉ tái cấu trúc câu chữ mà còn diễn giải ý định của người dùng theo cách rõ ràng, mạch lạc và phù hợp hơn với ngữ cảnh tìm kiếm. Việc khai thác khả năng hiểu ngôn ngữ sâu và suy luận logic của LLM giúp đảm bảo rằng truy vấn sau khi viết lại sẽ phản ánh chính xác mục tiêu tìm kiếm, đồng thời phù hợp hơn với cấu trúc dữ liệu được lưu trữ dưới dạng vector.

Lợi ích chính của module viết lại truy vấn là làm tăng tính rõ ràng, độ cụ thể và độ phù hợp ngữ nghĩa của truy vấn, nhờ đó hệ thống truy xuất có thể tìm được các đoạn thông tin liên quan với độ chính xác cao hơn. Ngoài ra, việc chuẩn hóa truy vấn theo một phong cách diễn đạt nhất quán còn giúp cải thiện khả năng tương tác giữa nhiều module trong pipeline.

Tuy nhiên, phương pháp này cũng có những nhược điểm đáng chú ý. Mỗi lần viết lại truy vấn yêu cầu một lần gọi bổ sung đến API của mô hình LLM, làm tăng độ trễ xử lý và chi phí vận hành. Đây là một yếu tố quan trọng cần cân nhắc, đặc biệt trong các hệ thống đòi hỏi phản hồi nhanh hoặc có tần suất truy vấn cao. Việc cân bằng giữa độ chính xác của truy vấn và hiệu suất hệ thống là một thách thức thường gặp khi triển khai thực tế module này.

2.2.3.3. Xếp hạng lại tài liệu liên quan

Xếp hạng lại tài liệu liên quan (Rerank Chunks) đóng vai trò thiết yếu trong việc nâng cao chất lượng dữ liệu truy xuất bằng cách đánh giá lại và sắp xếp các đoạn thông tin theo mức độ liên quan với truy vấn người dùng. Trong các hệ thống Retrieval-Augmented Generation (RAG), truy xuất ban đầu thường dựa trên phương pháp tìm kiếm ngữ nghĩa, tuy hiệu quả nhưng đôi khi vẫn chưa đủ chính xác trong việc xác định mức độ phù hợp sâu sắc giữa truy vấn và nội dung văn bản. Do đó, một bước xếp hạng lại bằng mô hình học sâu được bổ sung nhằm lọc ra những đoạn có độ liên quan cao nhất, giúp tăng chất lượng đầu vào cho bước tổng hợp cuối cùng bằng LLM.

Trong dự án này, quá trình xếp hạng lại được thực hiện bằng cách sử dụng mô hình ViRanker, một mô hình reranker tiếng Việt do nhóm nghiên cứu tại PTIT phát triển và công bố trên Hugging Face với tên gọi namdp-ptit/ViRanker. Đây là một mô hình được huấn luyện dựa trên kiến trúc ColBERT (Contextualized Late Interaction over BERT), với trọng tâm là tối ưu hóa việc đánh giá độ tương thích giữa truy vấn và từng đoạn văn bản bằng cơ chế tương tác sâu giữa các token. ViRanker có quy mô khoảng 600 triệu tham số và được tinh chỉnh đặc biệt cho các tác vụ ngôn ngữ tiếng Việt trong môi trường tìm kiếm văn bản học thuật, báo chí và dữ liệu chính thống.

Quy trình xếp hạng lại bắt đầu bằng việc mở rộng tập hợp các đoạn truy xuất từ k lên 2k đoạn, tức gấp đôi số lượng ban đầu. Việc này nhằm đảm bảo rằng hệ thống có đủ dữ liệu để chọn lọc những đoạn thực sự chất lượng. Sau đó, từng đoạn trong tập 2k này được kết hợp với truy vấn đầu vào để tạo thành các cặp (chunk, query). Mô hình ViRanker sẽ xử lý từng cặp và tính toán điểm số độ liên quan dựa trên mức độ tương tác ngữ nghĩa giữa các token trong truy vấn và đoạn văn bản.

Kết quả chấm điểm của mô hình được sử dụng để chọn ra top k đoạn có điểm số cao nhất. Các đoạn này không chỉ được chọn lọc lại mà còn được xếp theo thứ tự từ cao đến thấp về mức độ liên quan, đảm bảo rằng những đoạn có thông tin hữu ích nhất sẽ được ưu tiên đưa vào bước xử lý tiếp theo của pipeline, cụ thể là bước mở rộng và tổng hợp bằng LLM.

Tuy nhiên, phương pháp này cũng đặt ra một số thách thức đáng kể về mặt hiệu năng. Việc xử lý 2k cặp chunk-query với một mô hình có kích thước lớn như ViRanker đòi hỏi tài nguyên tính toán đáng kể, đồng thời làm tăng thời gian xử lý tổng thể của hệ thống. Điều này đặc biệt quan trọng trong các ứng dụng thời gian thực hoặc yêu cầu phản hồi nhanh. Bên cạnh đó, mô hình ViRanker chủ yếu được huấn luyện trên dữ liệu tiếng Việt, do đó cần đảm bảo tính tương thích khi hệ thống có thể hoạt động trên truy vấn đa ngôn ngữ hoặc nội dung phi cấu trúc.

Mặc dù có chi phí tính toán cao, hiệu quả mà module xếp hạng lại này mang lại là rõ ràng: nó giúp loại bỏ nhiễu từ bước truy xuất ban đầu và đảm bảo rằng chỉ những thông tin sát nghĩa và hữu ích nhất mới được đưa đến bước tổng hợp, từ đó góp phần nâng cao độ chính xác và chất lượng của phản hồi cuối cùng do mô hình LLM tạo ra.

2.2.3.4. Mở rộng thông tin

Mở rộng thông tin tài liệu liên quan đóng vai trò then chốt trong việc làm phong phú thêm ngữ cảnh của mỗi đoạn truy xuất bằng cách bổ sung các đoạn liền kề. Quy trình này đảm bảo rằng các đoạn được chọn chứa đủ ngữ cảnh để tạo ra phản hồi đầy đủ và mạch lạc trong các bước tiếp theo.

Trong dự án này, mỗi đoạn được truy xuất sẽ được mở rộng bằng cách thêm tối đa bốn đoạn lân cận - cụ thể là hai đoạn ở bên trái và hai đoạn ở bên phải. Quá trình mở rộng này cho phép hệ thống nắm bắt được ngữ cảnh xung quanh mà có thể không xuất hiện trong đoạn truy xuất ban đầu. Bằng cách tổng hợp các đoạn liền kề, hệ thống có thể cung cấp một cái nhìn đầy đủ hơn về thông tin, điều này đặc biệt hữu ích khi xử lý các truy vấn phức tạp cần hiểu rõ bối cảnh rộng hơn.

Mặc dù việc mở rộng này làm tăng độ phong phú của ngữ cảnh, nhưng nó cũng có những hạn chế tiềm tàng. Việc bổ sung các đoạn thông tin sẽ làm tăng lượng dữ liệu cần xử lý ở các bước sau, điều này có thể dẫn đến chi phí tính toán cao hơn và thời gian xử lý dài hơn. Tuy nhiên, những nhược điểm này thường được bù đắp bằng lợi ích từ độ chính xác ngữ cảnh được cải thiện, đặc biệt trong các trường hợp mà ngữ cảnh đóng vai trò then chốt để tạo ra phản hồi có ý nghĩa.

2.3. PHƯƠNG PHÁP ĐÁNH GIÁ VÀ KIỂM TRA KẾT QUẢ THỰC NGHIỆM

Trong bối cảnh xây dựng các hệ thống hỏi đáp sử dụng kiến trúc Retrieval-Augmented Generation (RAG), việc đánh giá chất lượng đầu ra của chatbot là một bước quan trọng nhằm đảm bảo tính chính xác, hữu ích và độ tin cậy của các phản hồi được tạo ra. Không giống như các hệ thống thuần sinh văn bản, mô hình RAG kết hợp giữa tìm kiếm thông tin và mô hình ngôn ngữ lớn, do đó cần có các phương pháp đánh giá chuyên biệt, có khả năng xem xét mối quan hệ giữa truy vấn, tài liệu được truy xuất và câu trả lời sinh ra.

Để hiện thực hóa việc đánh giá này, đề tài sử dụng RAGAS (Retrieval-Augmented Generation Assessment), một công cụ mã nguồn mở được thiết kế để đo lường chất lượng của các hệ thống RAG. RAGAS hoạt động bằng cách sử dụng mô hình ngôn ngữ lớn để phân tích và chấm điểm các cặp truy vấn – ngữ cảnh – câu trả lời đầu ra dựa trên nhiều tiêu chí khác nhau. Đây là công cụ hiện đại, phù hợp với yêu cầu đánh giá bán tự động và tiết kiệm thời gian, đồng thời vẫn đảm bảo mức độ tin cậy tương đương với phương pháp đánh giá thủ công.

Bốn tiêu chí chính được sử dụng trong đề tài để đánh giá chatbot bao gồm: Tính trung thực (Faithfulness), Mức độ liên quan của câu trả lời (Answer Relevancy), Độ bao phủ ngữ cảnh (Context Recall), Mức độ liên quan của ngữ cảnh (Context Relevancy). Các chỉ số này phản ánh ba khía cạnh cốt lõi để đảm bảo chatbot không chỉ trả lời đúng mà còn dựa trên thông tin đã truy xuất và sát với truy vấn của người dùng.

2.3.1. Tính trung thực (Faithfulness)

Faithfulness là thước đo đánh giá mức độ mà câu trả lời sinh ra bởi mô hình trung thành với thông tin đã được truy xuất từ cơ sở dữ liệu tri thức. Nói cách khác, chỉ số này phản ánh liệu chatbot có "bịa đặt" thông tin hay không. Một câu trả lời được xem là có độ trung thực cao nếu mọi thông tin trong đó đều có thể được xác minh từ các đoạn ngữ cảnh đã truy xuất.

Trong hệ thống RAG, đây là tiêu chí quan trọng hàng đầu vì mô hình sinh có xu hướng "hallucinate" – nghĩa là tạo ra nội dung không có trong dữ liệu. RAGAS sử dụng mô hình ngôn ngữ lớn để đối chiếu từng phần trong câu trả lời với ngữ cảnh nhằm phát hiện sai lệch, từ đó cho điểm trên thang 0–1. Giá trị càng gần 1 càng thể hiện mức độ trung thực cao.

2.3.2. Mức độ liên quan của câu trả lời (Answer Relevancy)

Answer Relevancy là tiêu chí đánh giá mức độ mà câu trả lời phản ánh đúng nội dung truy vấn của người dùng. Chỉ số này tập trung vào việc đo lường xem câu trả lời có đúng trọng tâm và giải quyết đúng vấn đề được đặt ra hay không, bất kể nó có được sinh từ ngữ cảnh chính xác hay không.

Đây là một khía cạnh phản ánh khả năng hiểu đúng câu hỏi và sinh câu trả lời phù hợp về nội dung. RAGAS sẽ sử dụng mô hình ngôn ngữ để so sánh ngữ nghĩa giữa câu truy vấn và câu trả lời, từ đó chấm điểm dựa trên sự tương thích về ý nghĩa và mức độ hài lòng của phản hồi. Kết quả đánh giá giúp phát hiện các trường hợp mô hình trả lời chung chung, lan man hoặc không liên quan.

2.3.3. Độ bao phủ ngữ cảnh (Context Recall)

Context Recall là chỉ số đánh giá mức độ đầy đủ của các thông tin được truy xuất so với yêu cầu của truy vấn. Khác với Context Relevancy – vốn tập trung vào sự phù hợp về chủ đề giữa truy vấn và các đoạn ngữ cảnh, Context Recall lại tập trung vào việc đo lường xem liệu các đoạn ngữ cảnh có bao hàm đầy đủ thông tin cần thiết để mô hình sinh ra được câu trả lời chính xác hay không.

Trong hệ thống RAG, việc truy xuất đúng tài liệu là điều kiện cần, nhưng truy xuất đủ tài liệu liên quan là điều kiện đủ để đảm bảo chất lượng đầu ra. Nếu thiếu mất một phần thông tin quan trọng trong ngữ cảnh truy xuất, ngay cả khi mô hình sinh hoạt động tốt, câu trả lời cũng có nguy cơ bị thiếu sót hoặc sai lệch.

Để tính toán chỉ số này, RAGAS sử dụng mô hình ngôn ngữ lớn để phân tích mối quan hệ giữa truy vấn, câu trả lời đầu ra và các ngữ cảnh được truy xuất. Hệ thống sẽ đánh giá xem liệu những thông tin thiết yếu để sinh ra câu trả lời có thực sự tồn tại trong các đoạn ngữ cảnh đã được hệ thống cung cấp hay không. Nếu phần lớn nội dung câu trả lời không thể được đối chiếu với các đoạn truy xuất, chỉ số Context Recall sẽ bị đánh giá thấp.

Chỉ số Context Recall có ý nghĩa quan trọng trong việc đánh giá hiệu quả của quá trình tìm kiếm thông tin, đặc biệt là trong việc thiết kế và tinh chỉnh vector database, lựa chọn mô hình embedding, cũng như tối ưu thuật toán truy vấn. Một hệ thống có điểm Context Recall cao chứng tỏ khả năng thu thập thông tin đầy đủ và chính xác, tạo nền tảng vững chắc cho giai đoạn sinh câu trả lời.

2.3.4. Độ chính xác của ngữ cảnh (Context Relevancy)

Context Relevancy là tiêu chí đo lường chất lượng của các đoạn văn bản được truy xuất bởi hệ thống, cụ thể là mức độ phù hợp giữa truy vấn và ngữ cảnh. Mặc dù các mô hình RAG có thể sinh câu trả lời từ ngữ cảnh, chất lượng của ngữ cảnh truy xuất đóng vai trò then chốt vì nó ảnh hưởng trực tiếp đến khả năng sinh thông tin chính xác.

Trong đánh giá bằng RAGAS, hệ thống sẽ xem xét xem liệu các tài liệu được truy xuất có chứa thông tin hữu ích, phù hợp với truy vấn hay không. Nếu hệ thống truy xuất sai chủ đề hoặc tài liệu không liên quan, điểm số sẽ thấp. Điều này đặc biệt hữu ích để đánh giá chất lượng của bộ vector store và thuật toán tìm kiếm.

2.4. CÔNG CỤ VÀ TÀI NGUYÊN SỬ DỤNG

2.4.1. Python

Python là ngôn ngữ lập trình chính được sử dụng trong quá trình xây dựng chatbot nhờ vào cú pháp đơn giản, khả năng diễn đạt rõ ràng và hệ sinh thái phong phú trong lĩnh vực trí tuệ nhân tạo và xử lý ngôn ngữ tự nhiên. Với cộng đồng phát triển rộng lớn và nhiều thư viện hỗ trợ chuyên sâu, Python cho phép rút ngắn thời gian phát triển, đồng thời đảm bảo hiệu quả khi triển khai các hệ thống học sâu phức tạp.

Trong đề tài này, Python được sử dụng để xử lý toàn bộ pipeline từ thu thập dữ liệu, tiền xử lý văn bản, mã hóa ngữ nghĩa, đến xây dựng các chuỗi xử lý logic cho mô hình ngôn ngữ. Ngoài ra, Python cũng đóng vai trò cầu nối giữa các thành phần như vector store, mô hình ngôn ngữ lớn và giao diện người dùng, từ đó hiện thực hóa khả năng hỏi đáp tài liệu một cách tự động, thông minh và có khả năng mở rộng linh hoạt. Với khả năng tích hợp mạnh mẽ và hỗ trợ tốt từ cộng đồng, Python là nền tảng cốt lõi cho việc triển khai kiến trúc Retrieval-Augmented Generation trong hệ thống chatbot.

2.4.2. Huggingface

Huggingface là một nền tảng mã nguồn mở nổi bật trong lĩnh vực trí tuệ nhân tạo, đặc biệt là xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP). Được phát triển với mục tiêu phổ cập các mô hình ngôn ngữ mạnh mẽ đến cộng đồng nghiên cứu và phát triển ứng dụng AI, Huggingface cung cấp một hệ sinh thái phong phú với nhiều thư viện nổi bật như transformers, datasets, tokenizers và sentence-transformers. Trong khuôn khổ đề tài này, thư viện transformers và sentence-transformers được sử dụng làm công cụ chính để triển khai các mô hình nhúng (embedding) văn bản, phục

vụ cho giai đoạn truy xuất ngữ nghĩa trong kiến trúc Retrieval-Augmented Generation (RAG).

Thư viện transformers cho phép truy cập nhanh chóng đến hàng nghìn mô hình ngôn ngữ tiền huấn luyện, từ các mô hình tổng quát như BERT, RoBERTa, GPT đến các mô hình chuyên biệt được tinh chỉnh cho từng tác vụ như phân loại văn bản, sinh ngôn ngữ, trả lời câu hỏi hay nhận diện thực thể. Khả năng tải mô hình chỉ với một dòng lệnh và tích hợp dễ dàng vào pipeline xử lý giúp tăng tốc quá trình thử nghiệm và triển khai thực tế. Đặc biệt, với kiến trúc hướng mô-đun, các mô hình trong thư viện này có thể dễ dàng hoán đổi, tùy biến theo yêu cầu cụ thể mà không cần viết lại toàn bộ hệ thống.

Bên cạnh đó, thư viện sentence-transformers, vốn được xây dựng dựa trên nền tảng transformers, cung cấp giao diện thuận tiện để nhúng các đoạn văn bản thành vector ngữ nghĩa với độ chính xác cao. Thay vì đơn thuần mã hóa từ cấp độ token như trong các mô hình truyền thống, sentence-transformers sử dụng các kỹ thuật huấn luyện đặc biệt như Siamese Network và Triplet Loss để tối ưu hóa vector biểu diễn ở cấp độ câu, giúp tăng cường khả năng so sánh và đánh giá mức độ tương đồng giữa các câu hỏi và tài liệu. Trong hệ thống chatbot, các vector này là đầu vào cho bước tìm kiếm vector (semantic retrieval), đóng vai trò quan trọng trong việc lựa chọn ngữ cảnh phù hợp để sinh phản hồi.

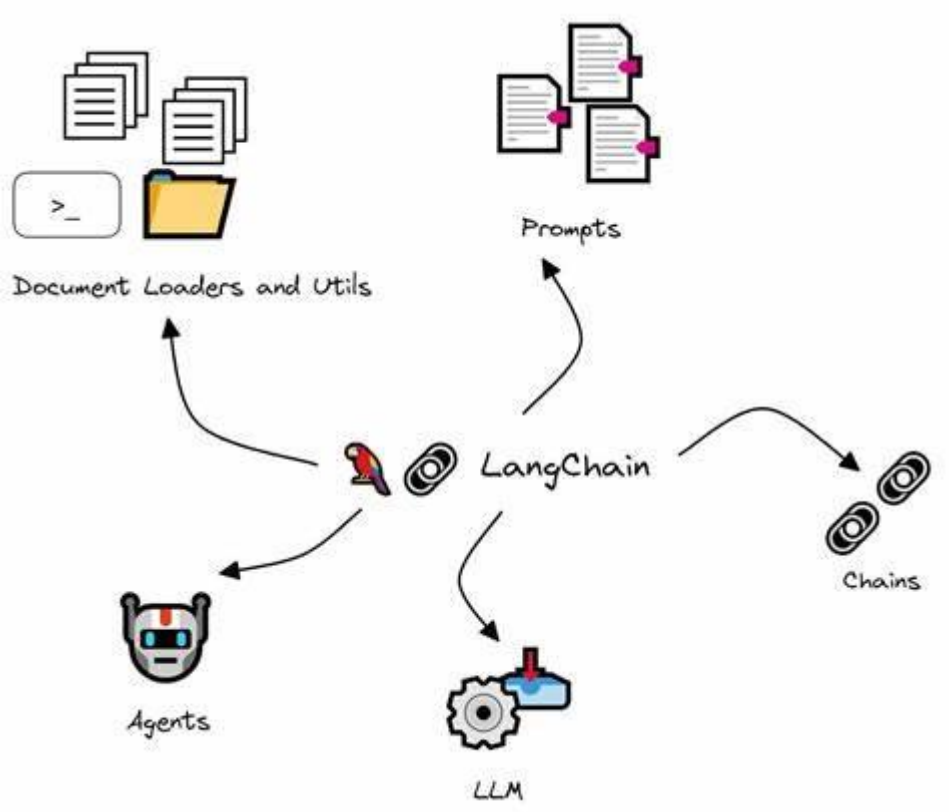
Không chỉ dừng lại ở việc cung cấp mô hình, Huggingface còn phát triển nền tảng Huggingface Hub – một kho lưu trữ trực tuyến nơi các tổ chức và cá nhân có thể chia sẻ mô hình, bộ dữ liệu, và pipeline xử lý. Điều này tạo ra một cộng đồng hợp tác mở, nơi các kết quả nghiên cứu mới nhất có thể được áp dụng ngay vào thực tế thông qua các công cụ tiện ích. Ngoài ra, Huggingface cũng hỗ trợ việc tối ưu hóa mô hình trên nhiều nền tảng phần cứng như CPU, GPU, và TPU, cũng như khả năng tương thích cao với các công cụ huấn luyện và triển khai như PyTorch, TensorFlow và ONNX.

Tóm lại, việc sử dụng Huggingface trong đề tài không chỉ mang lại hiệu quả vượt trội trong quá trình xử lý và nhúng văn bản, mà còn tạo điều kiện thuận lợi cho việc mở rộng, cập nhật và tái sử dụng mô hình trong tương lai. Nền tảng này đóng vai trò then chốt trong việc đảm bảo độ chính xác, tính linh hoạt và khả năng tương tác của hệ thống chatbot với người dùng.

2.4.3. Langchain

LangChain là một framework mã nguồn mở được thiết kế nhằm hỗ trợ quá trình phát triển các ứng dụng tận dụng mô hình ngôn ngữ lớn (LLMs) một cách có cấu trúc,

linh hoạt và dễ mở rộng. Thay vì đơn thuần chỉ gửi truy vấn đến một mô hình và nhận kết quả trả về, LangChain cung cấp một hệ sinh thái đầy đủ các thành phần giúp xây dựng những chuỗi xử lý phức tạp, kết hợp giữa mô hình ngôn ngữ và các nguồn dữ liệu hoặc công cụ xử lý bên ngoài.



Hình 2.6. Kiến trúc tổng quan của Langchain

LangChain được tổ chức theo hướng mô-đun hóa, nơi mà mỗi thành phần trong pipeline xử lý đóng một vai trò cụ thể, có thể được tùy biến và kết hợp lại để tạo ra ứng dụng hoàn chỉnh. Trong số đó, một số thành phần cốt lõi thường xuyên được sử dụng bao gồm: LLMs, Prompts, Chains, Tools, Agents, Memory, và Retrievers.

- Mô hình ngôn ngữ (LLM) là trung tâm của mọi hệ thống được xây dựng bằng LangChain. Đây có thể là các mô hình như OpenAI GPT-3.5/GPT-4, Google Gemini, Anthropic Claude hoặc các mô hình mã nguồn mở như LLaMA, Mistral, Falcon. LangChain hỗ trợ đa dạng các API, cho phép dễ dàng chuyển đổi giữa các nhà cung cấp mà không cần viết lại toàn bộ mã.
- Prompt trong LangChain được trừu tượng hóa thành PromptTemplate. Đây là thành phần cho phép định nghĩa các mẫu gợi ý (prompt) một cách linh hoạt và tái sử dụng. Thay vì viết prompt cố định dạng chuỗi, người phát triển có thể tạo ra các template với biến động, từ đó tùy chỉnh nội dung đầu vào gửi đến mô hình theo ngữ cảnh cụ thể của người dùng.

- Chain là khái niệm cốt lõi trong LangChain. Đây là một chuỗi các bước xử lý, nơi đầu ra của bước này sẽ được truyền sang đầu vào của bước tiếp theo. Ví dụ, một chuỗi đơn giản có thể bao gồm: nhận câu hỏi từ người dùng, chèn vào prompt, gửi tới mô hình ngôn ngữ và trả về kết quả. Với các chuỗi phức tạp hơn, hệ thống có thể bao gồm nhiều mô hình, truy xuất tài liệu, ghi nhớ ngữ cảnh và thậm chí là thực hiện logic điều kiện. Một ví dụ tiêu biểu là LLMChain, kết hợp một PromptTemplate và một mô hình ngôn ngữ để tạo thành một đơn vị xử lý tự động hóa.
- Retriever là thành phần cho phép hệ thống truy xuất thông tin từ kho dữ liệu ngoài mô hình, thường là các kho tri thức dưới dạng vector. LangChain hỗ trợ nhiều loại vector store như FAISS, Pinecone, Chroma, Milvus,... Các tài liệu trước khi đưa vào hệ thống sẽ được chia nhỏ, nhúng thành vector và lưu trữ. Khi có truy vấn, Retriever sẽ tìm kiếm các đoạn văn bản gần nghĩa nhất với câu hỏi, từ đó cung cấp bối cảnh cần thiết cho mô hình sinh câu trả lời chính xác hơn.
- Tool là các hàm, dịch vụ hoặc công cụ bên ngoài mà mô hình có thể gọi tới khi cần. Đây có thể là máy tính, trình duyệt web, API tra cứu thời tiết, hay bất kỳ logic xử lý nào được đóng gói dưới dạng callable. Các tool này đặc biệt hữu ích khi kết hợp với Agent, thành phần có khả năng lên kế hoạch và ra quyết định.
- Agent là một tác nhân thông minh có thể sử dụng mô hình ngôn ngữ để phân tích truy vấn đầu vào và chọn công cụ phù hợp để thực hiện tác vụ. LangChain cung cấp một số loại agent như ReAct, Self-ask with search, Conversational Agent,... Những agent này được huấn luyện theo các phương pháp cho phép lập luận, ghi nhớ các hành động đã thực hiện, và đưa ra hành vi tiếp theo dựa trên kết quả trung gian. Điều này giúp mô hình không chỉ phản hồi văn bản mà còn biết cách hành động, ví dụ như tìm kiếm thông tin, tính toán, hoặc gọi API.
- Memory là thành phần giúp hệ thống lưu trữ trạng thái cuộc hội thoại, thông tin lịch sử hoặc các giá trị trung gian. Đây là yếu tố quan trọng để chatbot duy trì tính nhất quán trong giao tiếp, chẳng hạn như nhớ tên người dùng, các câu hỏi đã hỏi trước đó, hoặc kết quả từ bước truy xuất tài liệu. LangChain hỗ trợ nhiều loại bộ nhớ như ConversationBufferMemory, SummaryMemory hay VectorStoreRetrieverMemory, tùy theo mức độ chi tiết và loại dữ liệu cần lưu trữ.

Khi được kết hợp lại, các thành phần trên cho phép LangChain trở thành một framework mạnh mẽ trong việc xây dựng các hệ thống trí tuệ nhân tạo hội thoại, có khả năng truy cập tri thức bên ngoài, ghi nhớ ngữ cảnh và thực hiện chuỗi hành động logic phức tạp. Trong dự án chatbot, LangChain giúp hiện thực hóa khả năng hỏi đáp theo tài liệu (retrieval-augmented generation - RAG), tích hợp nhiều nguồn thông tin khác nhau, và mở rộng ứng dụng một cách dễ dàng khi cần thêm chức năng mới.

2.4.4. Streamlit

Streamlit là một framework mã nguồn mở được phát triển nhằm hỗ trợ các nhà khoa học dữ liệu và kỹ sư AI trong việc xây dựng nhanh chóng các giao diện người dùng cho ứng dụng máy học và trí tuệ nhân tạo. Không giống như các framework truyền thống như Flask hay Django, Streamlit cho phép người dùng triển khai giao diện với cấu trúc lập trình tuyến tính, gần gũi với cách viết một tập lệnh Python thông thường. Điều này giúp tiết kiệm đáng kể thời gian phát triển và giảm độ phức tạp trong khâu tổ chức mã nguồn.

Về mặt kỹ thuật, Streamlit hoạt động dựa trên mô hình phản ứng với sự thay đổi của trạng thái. Mỗi lần người dùng tương tác với giao diện (chẳng hạn như chọn từ một hộp thả, nhập văn bản hay bấm nút), toàn bộ tập lệnh Python sẽ được thực thi lại, và kết quả hiển thị được cập nhật tương ứng. Để đạt hiệu quả cao trong việc hiển thị và xử lý, Streamlit cung cấp một loạt các hàm API trực quan như st.text_input(), st.button(), st.markdown() và đặc biệt là st.chat_message() dành riêng cho các ứng dụng hội thoại như chatbot. Ngoài ra, framework này còn hỗ trợ tốt việc tích hợp các thư viện phổ biến trong lĩnh vực khoa học dữ liệu như pandas, matplotlib, seaborn, plotly, hoặc các mô hình học sâu thông qua PyTorch và TensorFlow.

Một điểm mạnh nổi bật của Streamlit là khả năng triển khai nhanh chóng ứng dụng dưới dạng web mà không cần kiến thức chuyên sâu về lập trình frontend. Với một vài dòng lệnh đơn giản, người dùng có thể chạy ứng dụng cục bộ hoặc triển khai lên nền tảng đám mây thông qua Streamlit Community Cloud hoặc các công cụ như Docker, Heroku. Trong dự án chatbot, Streamlit đóng vai trò là nền tảng để tạo ra giao diện trò chuyện giữa người dùng và hệ thống, cho phép gửi truy vấn văn bản và hiển thị phản hồi từ mô hình một cách trực quan và mượt mà.

CHƯƠNG 3. KẾT QUẢ VÀ THẢO LUẬN

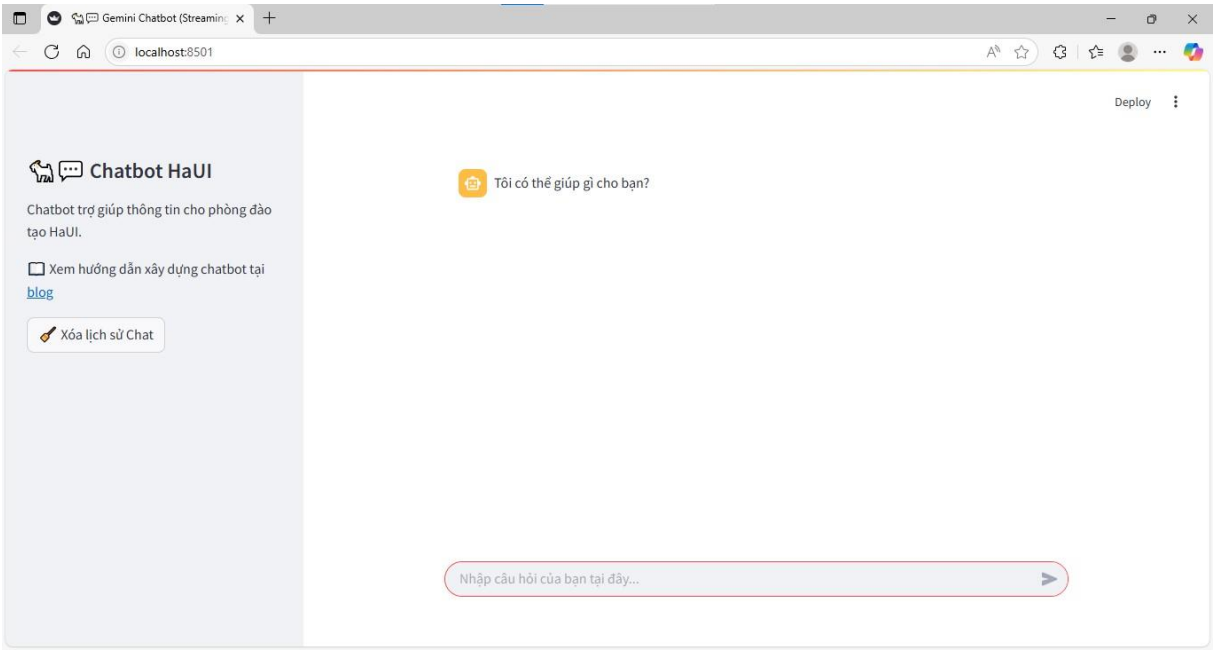
3.1. KẾT QUẢ THỰC NGHIỆM

3.1.1. Kết quả chương trình

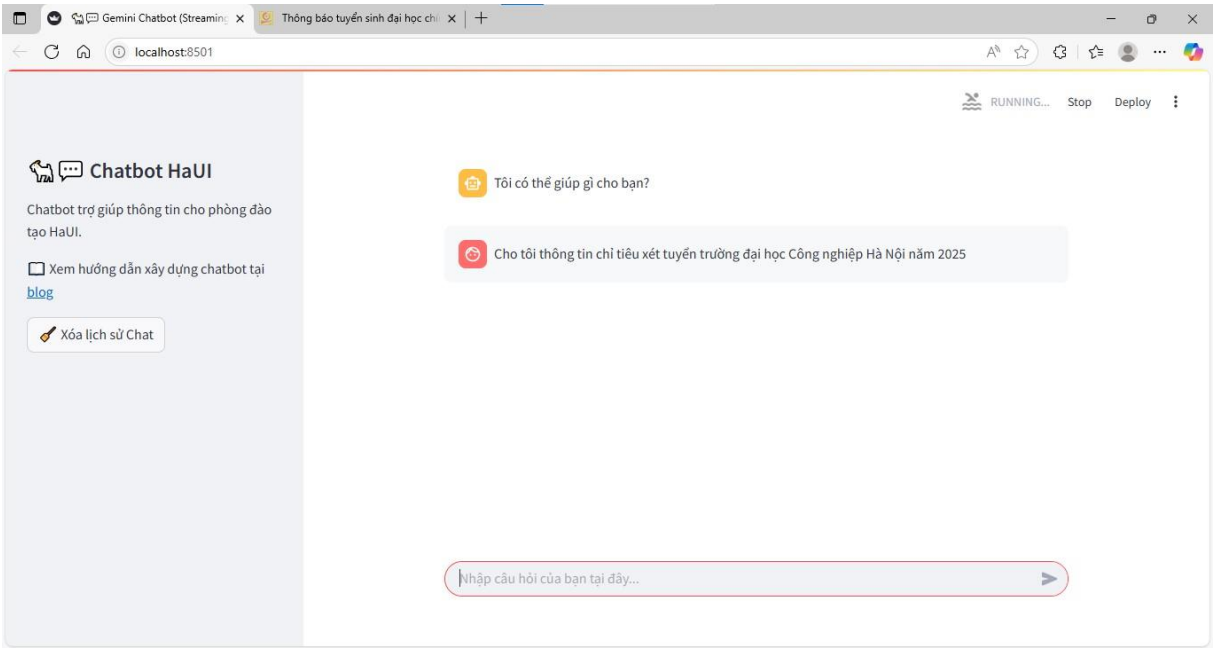
Sau quá trình thực nghiệm và phát triển, Chatbot được triển khai thành công sử dụng hệ thống Retrieval-Augmented Generation (RAG) tích hợp với quy trình AutoCrawl nhằm giải quyết vấn đề truy xuất thông tin động và tạo phản hồi. Những thành tựu chính của nghiên cứu bao gồm:

- Thu thập dữ liệu hiệu quả: quy trình AutoCrawl tự động thu thập và nhúng thông tin liên quan vào cơ sở dữ liệu vector.
- Truy xuất và tạo phản hồi hiệu quả: sự kết hợp của tìm kiếm ngữ nghĩa, nhiều module như viết lại (rewrite), xếp hạng lại (rerank) và tổng hợp phản hồi sử dụng Google Gemini API đã cung cấp các câu trả lời chính xác và phù hợp với ngữ cảnh cho các truy vấn của người dùng.
- Khả năng mở rộng: kiến trúc hỗ trợ việc tích hợp các cải tiến trong tương lai, như tìm kiếm thưa thớt (sparse search) và truy xuất dựa trên đồ thị tri thức (knowledge graph-based).

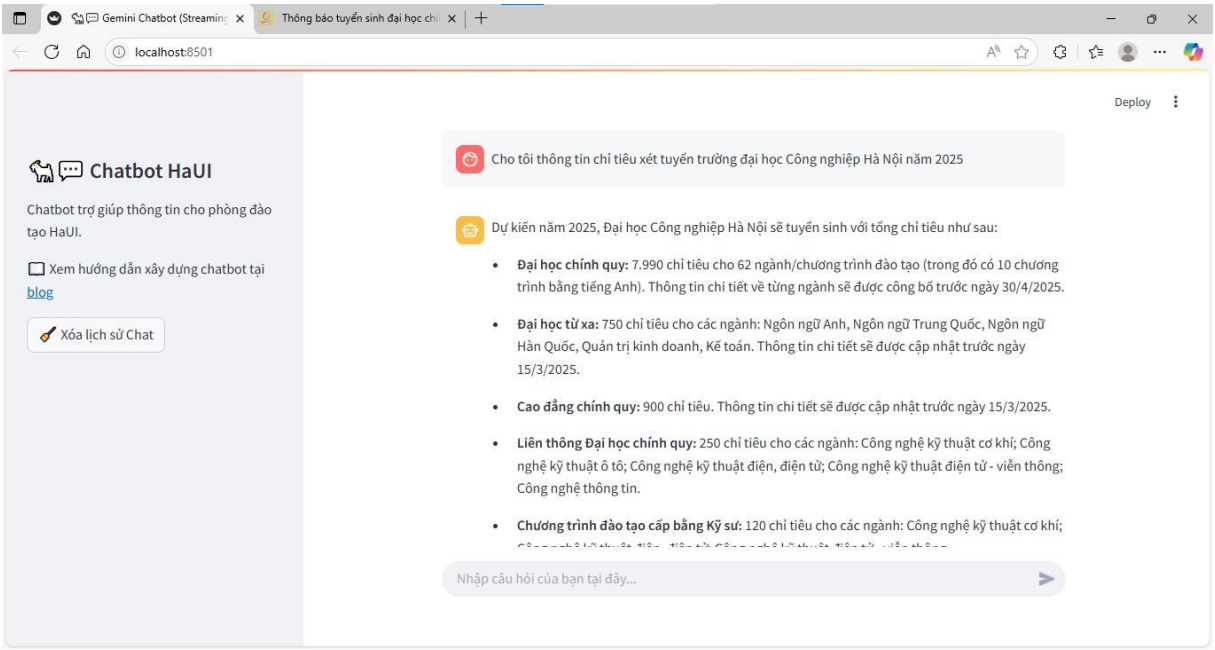
Để hoàn thiện kết quả thực nghiệm, ứng dụng chatbot được triển khai thông qua giao diện người dùng sử dụng Streamlit, cho phép người dùng dễ dàng tương tác với hệ thống thông qua trình duyệt web. Giao diện thân thiện và đơn giản này giúp người dùng có thể nhập câu hỏi trực tiếp vào một ô nhập liệu, đồng thời nhận phản hồi bằng văn bản dưới dạng hội thoại.



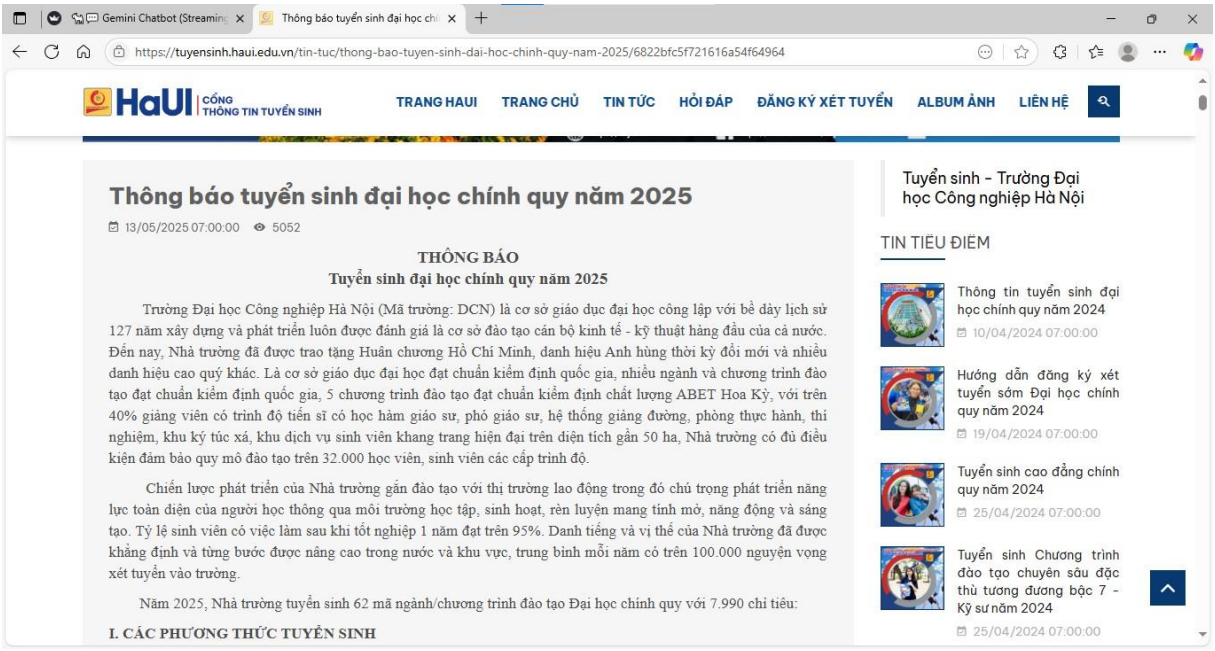
Hình 3.1. Giao diện Chatbot hỗ trợ hoạt động tại phòng đào tạo



Hình 3.2. Giao diện ứng dụng khi đang suy luận



Hình 3.3. Giao diện ứng dụng khi đã sinh câu trả lời



Hình 3.4. Thông tin tuyển sinh chính thức của trường HaUI

Nhờ vào các kiến trúc đã được trình bày, một sản phẩm chatbot được phát triển với thiết kế đơn giản nhưng đáp ứng được các yêu cầu cơ bản về kỹ thuật và chức năng. Chatbot có khả năng xử lý các truy vấn tìm kiếm thông tin một cách nhanh chóng và chính xác nhờ việc kết hợp giữa công nghệ tìm kiếm ngữ nghĩa và mô hình sinh văn bản. Sản phẩm hoạt động ổn định, phản hồi sát với ngữ cảnh, đồng thời hỗ trợ trích dẫn nguồn dữ liệu rõ ràng để tăng độ tin cậy cho người dùng. Mặc dù còn đơn giản, chatbot đã thể hiện được khả năng tự động hóa quy trình tra cứu thông tin và mang lại trải nghiệm tương tác mượt mà.

3.1.2. Hiệu suất hệ thống

Để đánh giá chất lượng hệ thống chatbot được phát triển dựa trên kiến trúc Retrieval-Augmented Generation (RAG), đề tài sử dụng công cụ RAGAS (Retrieval-Augmented Generation Assessment). Đây là một framework đánh giá hiện đại, cho phép kiểm tra hiệu quả của các hệ thống RAG theo hướng bán tự động, vừa tiết kiệm thời gian vừa đảm bảo tính tin cậy tương đương với phương pháp đánh giá thủ công.

Quá trình đánh giá được chia thành hai giai đoạn chính tương ứng với hai thành phần cốt lõi trong kiến trúc RAG: giai đoạn sinh câu trả lời (generation) và giai đoạn truy xuất ngữ cảnh (retrieval). Mỗi giai đoạn được đánh giá thông qua hai tiêu chí cụ thể nhằm phản ánh chất lượng của hệ thống từ nhiều góc độ.

Trong giai đoạn retrieval, hai tiêu chí đánh giá lần lượt là độ chính xác của ngữ cảnh (context precision) và độ bao phủ ngữ cảnh (context recall). Context precision phản ánh tỷ lệ thông tin hữu ích so với nhiễu trong tập ngữ cảnh mà hệ thống truy xuất. Trong khi đó, context recall thể hiện khả năng truy xuất đầy đủ các đoạn ngữ cảnh cần thiết để cung cấp câu trả lời chính xác cho câu hỏi của người dùng.

Đối với giai đoạn generation, đánh giá tập trung vào hai khía cạnh. Thứ nhất là tính trung thực (faithfulness), thể hiện mức độ chính xác của câu trả lời sinh ra dựa trên thông tin đã được truy xuất. Chỉ số này cho biết liệu câu trả lời có "bịa đặt" thông tin hay không. Thứ hai là mức độ liên quan của câu trả lời (answer relevancy), đo lường mức độ mà nội dung phản hồi thực sự giải quyết được truy vấn của người dùng, bất kể nó có đúng theo dữ liệu hay không.

Bảng 3.1. Kết quả đánh giá của hệ thống

	Context precision	**Context recall**	**Faithfulness**	**Answer relevancy**
RAGAS	0.70	0.73	0.76	0.72

Kết quả đánh giá hệ thống chatbot thông qua RAGAS cho thấy hiệu suất tổng thể của mô hình là khá tốt, với các chỉ số dao động từ 0.70 đến 0.76. Cụ thể, điểm Faithfulness đạt 0.76 – là chỉ số cao nhất trong bốn tiêu chí. Điều này cho thấy mô hình sinh phản hồi với mức độ trung thực tương đối cao, phần lớn các thông tin trong câu trả lời đều có thể được truy xuất rõ ràng từ các đoạn văn bản được cung cấp. Đây là một tín hiệu tích cực vì nó phản ánh rằng hệ thống đã giảm thiểu được hiện tượng "hallucination" – tức là sinh ra thông tin không tồn tại trong dữ liệu.

Context recall đạt 0.73, cho thấy hệ thống truy xuất được phần lớn các ngữ cảnh quan trọng cần thiết để trả lời câu hỏi. Tuy nhiên, vẫn còn một tỷ lệ nhỏ thông tin cần thiết không được truy xuất, có thể là do giới hạn của bộ vector store hoặc thuật toán tìm kiếm chưa tối ưu. Việc cải thiện recall có thể giúp hệ thống cung cấp thông tin đầu vào phong phú hơn cho mô hình sinh.

Với Context precision đạt 0.70, ta có thể thấy rằng ngữ cảnh được truy xuất phần lớn là phù hợp với truy vấn, tuy nhiên vẫn tồn tại một lượng nhiễu nhất định – tức là có các đoạn văn bản không thực sự cần thiết hoặc không liên quan sát đến nội dung câu hỏi. Điều này ảnh hưởng gián tiếp đến chất lượng của câu trả lời và làm giảm hiệu quả xử lý của mô hình.

Cuối cùng, chỉ số Answer relevancy đạt 0.72 phản ánh rằng câu trả lời nhìn chung có liên quan đến truy vấn đầu vào, nhưng vẫn còn một số trường hợp mô hình phản hồi chưa thực sự đúng trọng tâm hoặc diễn đạt còn lan man. Điều này có thể xuất phát từ việc ngữ cảnh chưa đủ cụ thể, hoặc do mô hình sinh chưa hiểu đúng ý định người dùng.

Tổng thể, các chỉ số cho thấy hệ thống đã hoạt động ổn định và có khả năng đáp ứng yêu cầu cơ bản của một chatbot hỏi đáp dựa trên RAG. Tuy nhiên, để đạt chất lượng cao hơn, cần tập trung cải thiện hai khía cạnh chính: nâng cao độ chính xác và độ bao phủ của quá trình truy xuất ngữ cảnh (retrieval), đồng thời tinh chỉnh mô hình sinh để tăng độ liên quan và rõ ràng của phản hồi.

3.2. HƯỚNG PHÁT TRIỂN

Mặc dù hệ thống RAG hiện tại đã cho thấy kết quả khả quan, nhưng vẫn có một số lĩnh vực được xác định cần phát triển và cải tiến thêm. Những nâng cấp này tập trung vào việc hoàn thiện các module hiện có và khám phá các chức năng bổ sung nhằm tối ưu hóa hiệu suất của hệ thống.

- Nâng cao lọc dữ liệu trong AutoCrawl: Cải thiện quá trình thu thập dữ liệu bằng cách áp dụng các kỹ thuật lọc tiên tiến để loại bỏ nội dung không liên quan hoặc chất lượng thấp, đảm bảo độ chính xác cao hơn trong kết quả truy xuất.
- Tinh chỉnh mô hình xếp hạng lại (Rerank): Một thách thức được xem xét là mô hình rerank, namdp-ptit/ViRanker, đôi khi hoạt động chưa tối ưu, đặc biệt trong một số ngữ cảnh hoặc loại truy vấn cụ thể. Để khắc phục, một hướng cải tiến khả thi là tinh chỉnh mô hình rerank sử dụng dữ liệu đặc thù của phòng đào tạo.

- Thêm module bổ sung - Phân tách truy vấn (Query Decomposition): Cuối cùng, mở rộng khả năng của hệ thống bằng cách tích hợp thêm các module như Phân tách truy vấn, giúp chia nhỏ các truy vấn phức tạp thành các truy vấn con đơn giản hơn, xử lý từng phần riêng biệt trước khi tổng hợp kết quả. Phương pháp này có thể cải thiện khả năng xử lý các truy vấn phức tạp, đa chiều và cung cấp các câu trả lời chính xác, chi tiết hơn.

Những cải tiến đề xuất này nhằm nâng cao tính ổn định, khả năng sử dụng và khả năng mở rộng của hệ thống, giúp hệ thống trở nên linh hoạt hơn cho các ứng dụng thực tiễn.

PHẦN III. KẾT LUẬN VÀ KIẾN NGHỊ

Qua quá trình nghiên cứu và triển khai, đề tài đã hoàn thành các mục tiêu trọng tâm đề ra, góp phần khẳng định tiềm năng ứng dụng của trí tuệ nhân tạo – đặc biệt là mô hình ngôn ngữ lớn (LLMs) kết hợp với kiến trúc Retrieval-Augmented Generation (RAG) – trong việc xây dựng hệ thống hỗ trợ truy xuất thông tin chuyên biệt. Sản phẩm chatbot được phát triển không chỉ hoạt động ổn định mà còn đáp ứng hiệu quả các yêu cầu về kỹ thuật, trải nghiệm người dùng và tính thực tiễn trong môi trường giáo dục.

Hệ thống đã tích hợp quy trình AutoCrawl nhằm tự động thu thập và cập nhật cơ sở tri thức, cho phép chatbot phản hồi chính xác và sát ngữ cảnh các truy vấn từ người dùng. Việc sử dụng các kỹ thuật hiện đại như nhúng văn bản, viết lại truy vấn, xếp hạng lại và sinh phản hồi thông qua mô hình ngôn ngữ lớn đã tạo ra một luồng xử lý trơn tru, góp phần đảm bảo chất lượng đầu ra. Thử nghiệm cho thấy hệ thống đạt được mức độ trung thực (faithfulness) tương đối cao và có khả năng truy xuất được phần lớn ngữ cảnh cần thiết, khẳng định tính khả thi của kiến trúc RAG trong bối cảnh sử dụng tiếng Việt.

Tuy vậy, trong quá trình thực hiện, đề tài cũng nhận diện rõ các hạn chế cần khắc phục. Vẫn còn tồn tại tỷ lệ nhiễu nhất định trong ngữ cảnh truy xuất, ảnh hưởng đến độ chính xác của phản hồi. Đồng thời, một số câu trả lời vẫn chưa hoàn toàn bám sát trọng tâm, đặc biệt trong các truy vấn phức tạp hoặc mơ hồ về ngữ nghĩa. Những vấn đề này phản ánh thách thức chung trong việc đảm bảo tính tin cậy của phản hồi sinh ra bởi LLM, kể cả khi ngữ cảnh đầu vào tương đối đầy đủ.

Từ kết quả nghiên cứu và đánh giá thực nghiệm, đề tài đề xuất một số hướng phát triển cụ thể trong tương lai. Trước hết, cần nâng cao chất lượng dữ liệu đầu vào bằng cách cải tiến quy trình AutoCrawl với các kỹ thuật lọc ngữ nghĩa và loại bỏ nhiễu tự động. Bên cạnh đó, việc tinh chỉnh lại mô hình xếp hạng lại (rerank) trên tập dữ liệu chuyên biệt của phòng đào tạo sẽ giúp cải thiện đáng kể mức độ liên quan và tập trung của phản hồi. Đặc biệt, tích hợp thêm module phân tách truy vấn (query decomposition) là một hướng đi triển vọng để nâng cao khả năng xử lý các truy vấn dài hoặc đa tầng ý nghĩa – vốn rất phổ biến trong ngữ cảnh giáo dục.

Về dài hạn, hệ thống có thể mở rộng tích hợp các yếu tố như đồ thị tri thức, đa phương thức tương tác (như nhận dạng giọng nói), và cá nhân hóa dựa trên thông tin người dùng để tăng mức độ thông minh và linh hoạt. Đồng thời, việc triển khai hệ thống trong môi trường thực tế, thu thập phản hồi người dùng và tiến hành cải tiến vòng lặp sẽ là bước tiếp theo quan trọng nhằm hoàn thiện sản phẩm.

Tóm lại, đề tài đã bước đầu chứng minh được giá trị thực tiễn của mô hình RAG khi ứng dụng trong xây dựng chatbot hỗ trợ tra cứu thông tin giáo dục. Với định hướng cải tiến liên tục và đầu tư đúng mức, hệ thống hứa hẹn sẽ phát triển thành một công cụ hữu ích, đồng hành cùng tiến trình chuyển đổi số trong giáo dục đại học tại Việt Nam.

TÀI LIỆU THAM KHẢO

[1] Aavache, "Llm-based web crawler," Github, 2023. [Online]. Available: https://github.com/Aavache/LLMWebCrawler/tree/.

[2] WenhaoHuang, ChenghaoPeng, ZhixuLi, JiaqingLiang, YanghuaXiao, LiqianWen, ZulongChen, "AutoCrawler: A Progressive Understanding Web Agent for Web Crawler Generation," *arXiv,* 2024.

[3] Huyen, "RLHF: Reinforcement Learning from Human Feedback," Chip Huyen, 2023. [Online].

Available: https://huyenchip.com/2023/05/02/rlhf.html#rlhf_overview.

[4] Yuntao Bai, Andy Jones, Kamal Ndousse, "Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback," *arXiv,* 2022.

[5] gkamradt, "Full Stack Retrieval," Github, 2024. [Online].

Available: https://github.com/FullStackRetrieval-com/RetrievalTutorials.

[6] Aymeric Roucher, "Advanced RAG on Hugging Face documentation using LangChain," Hugging Face, 2024. [Online].

Available: https://huggingface.co/learn/cookbook/advanced_rag.

[7] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, Zhaofeng Liu, "Evaluation of Retrieval-Augmented Generation: A Survey," *arXiv,* 2024.

[8] Bangoc123, "Vietnamese Retrieval Backend: RAG + MongoDB + Gemini 1.5 Pro + Semantic Router + Reflection," Github, 2024. [Online].

Available: https://github.com/bangoc123/retrieval-backend-with-rag.

[9] Yunfan Gao, Yun Xiong, Meng Wang, Haofen Wang, "Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks," Github, 2024. [Online]. Available: https://arxiv.org/pdf/2407.21059.

[10] Ngo Hieu, "Halongembedding: A vietnamese text embedding," Hugging Face, 2024. [Online]. Available: https://huggingface.co/hiieu/halong_embedding.

[11] Nam Dang Phuong, "Viranker: A cross-encoder model for vietnamese text ranking," Hugging Face, 2024. [Online].

Available: https://huggingface.co/namdp-ptit/ViRanker.