ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Chatbot Ẩm Thực

Cao Hải An - 23001818
Đặng Thế Anh - 23001821
Phạm Minh Cương - 23001840
Đỗ Minh Đức - 23001864
Phạm Nhật Quang - 23001920

Mã học phần: MAT1206E
Học kỳ 1, Năm học 2025-2026

Thông tin Dự án

Học phần: MAT1206E – Nhập môn Trí tuệ Nhân tạo
Học kỳ: Học kỳ 1, Năm học 2025-2026
Trường: VNU-HUS (Đại học Quốc gia Hà Nội – Trường Đại học Khoa học Tự nhiên)
Tên dự án: Chatbot Ẩm Thực
Ngày nộp: 30/11/2025
Báo cáo PDF: Báo cáo dự án Chatbot Ẩm Thực
Slide thuyết trình: Slide thuyết trình dự án Chatbot Ẩm Thực
Kho GitHub: `https://github.com/HaianCao/FoodChatbot`

Thành viên nhóm

Họ tên	Mã sinh viên	Tên GitHub	Đóng góp
Cao Hải An	23001818	HaianCao	Xây dựng pipeline chatbot
Đặng Thế Anh	23001821	DangTAnh	Thu thập dữ liệu
Phạm Minh Cương	23001840	mcnb2005	Phát triển giao diện web
Đỗ Minh Đức	23001864	minhhhduc	Xây dựng module truy xuất dữ liệu
Phạm Nhật Quang	23001920	NhatquangPham	Tiền xử lý dữ liệu

Danh sách hình vẽ

Danh sách bảng

Mục lục

Chương 1

Giới thiệu

1.1 Tóm tắt

Dự án xây dựng một hệ thống chatbot ẩm thực thông minh có khả năng cung cấp thông tin toàn diện về món ăn, bao gồm cách chế biến, thành phần nguyên liệu, giá trị dinh dưỡng và các gợi ý ẩm thực cá nhân hóa. Hệ thống được phát triển dựa trên kiến trúc Retrieval-Augmented Generation (RAG), kết hợp mô hình ngôn ngữ lớn Gemini và cơ sở dữ liệu vector để nâng cao khả năng truy xuất và tổng hợp tri thức. Dữ liệu ẩm thực được chuẩn hóa, mã hóa thành vector nhúng và lưu trữ trong cơ sở dữ liệu vector để đảm bảo việc tìm kiếm thông tin nhanh chóng và chính xác. Khi người dùng đặt câu hỏi, hệ thống tự động truy xuất các tài liệu liên quan và sử dụng Gemini LLM để tạo ra câu trả lời tự nhiên, mạch lạc và đáng tin cậy. Kết quả cho thấy chatbot có khả năng hiểu ngữ cảnh tốt, hỗ trợ người dùng lựa chọn món ăn, gợi ý công thức dựa trên nguyên liệu sẵn có và cung cấp thông tin dinh dưỡng theo cách thân thiện và dễ sử dụng. Dự án là minh chứng cho việc kết hợp RAG và LLM trong việc xây dựng các hệ thống tư vấn thông minh trong lĩnh vực ẩm thực.

1.2 Bài toán đặt ra

Trong bối cảnh công nghệ số phát triển mạnh mẽ, thói quen tiếp cận thông tin của con người đã thay đổi đáng kể. Người dùng ngày càng có xu hướng tìm kiếm giải pháp nhanh, chính xác và mang tính cá nhân hóa cho các nhu cầu hằng ngày, trong đó việc lựa chọn món ăn và tìm kiếm công thức nấu nướng là những nhu cầu phổ biến nhất. Mặc dù nguồn thông tin ẩm thực trên Internet vô cùng phong phú, người dùng vẫn gặp nhiều khó khăn do dữ liệu bị phân tán, chất lượng không đồng đều và không phải lúc nào cũng phù hợp với điều kiện thực tế. Điều này đặt ra nhu cầu cần có một hệ thống trợ lý ảo có khả năng cung cấp thông tin ẩm thực đáng tin cậy, rõ ràng và được cá nhân hóa.

1.2.1 Khả năng hiểu ngôn ngữ tự nhiên trong bối cảnh ẩm thực

Người dùng có thể đưa ra những câu hỏi rất đa dạng, từ đơn giản như "Làm mì Ý thế nào?" đến phức tạp như "Tôi có gà, nấm và phô mai – có thể nấu món gì dưới 30 phút và ít calo?". Do đó, hệ thống cần có khả năng phân tích ý định, nhận diện thực thể như nguyên liệu, món ăn, phương pháp chế biến, đồng thời hiểu rõ ngữ cảnh của câu hỏi. Bài toán đặt ra là tận dụng sức mạnh của mô hình ngôn ngữ lớn (LLM) để xử lý ngôn ngữ tự nhiên một cách linh hoạt và chính xác trong lĩnh vực ẩm thực.

1.2.2 Khả năng truy xuất tri thức chính xác và nhanh chóng

Kho tri thức ẩm thực rất rộng và bao gồm nhiều loại thông tin như nguyên liệu, dinh dưỡng, cách chế biến, các biến thể của món ăn và kỹ thuật nấu nướng. Hệ thống cần tổ chức và chuẩn hóa dữ liệu theo dạng có thể tìm kiếm hiệu quả. Việc sử dụng cơ sở dữ liệu vector (vector database) cho phép truy vấn dựa trên độ tương đồng ngữ nghĩa thay vì tìm kiếm từ khóa truyền thống.

Thách thức kỹ thuật bao gồm:

- Chuẩn hóa và làm sạch dữ liệu không đồng nhất.
- Mã hóa tri thức bằng các vector nhúng có chất lượng cao.
- Thiết kế cơ chế truy vấn tối ưu nhằm đảm bảo kết quả trả về chính xác và phù hợp nhất với truy vấn.

1.2.3 Sinh câu trả lời tự nhiên, mạch lạc và đáng tin cậy

Không chỉ đơn thuần truy xuất thông tin, chatbot phải tạo ra câu trả lời mạch lạc và hữu ích. Điều này bao gồm hướng dẫn nấu ăn theo từng bước rõ ràng, giải thích lý do sử dụng nguyên liệu hoặc kỹ thuật cụ thể, phân tích khẩu vị, mức độ khó, thời gian chế biến, dinh dưỡng, cũng như đề xuất điều chỉnh món ăn theo nhu cầu sức khỏe của người dùng.

Việc kết hợp kiến trúc Retrieval-Augmented Generation (RAG) và mô hình Gemini LLM đảm bảo rằng câu trả lời vừa dựa trên tri thức chính xác vừa có tính tự nhiên và dễ hiểu.

1.2.4 Cá nhân hóa theo nhu cầu và ràng buộc thực tế của người dùng

Một hệ thống chatbot thông minh cần cung cấp gợi ý phù hợp với từng cá nhân, bao gồm:

- Sở thích cá nhân như mức độ cay, khẩu vị hoặc phong cách ẩm thực.
- Các chế độ dinh dưỡng đặc thù (ăn chay, keto, low-carb, không gluten).
- Dị ứng hoặc hạn chế trong ăn uống.
- Nguyên liệu hiện có trong bếp.
- Mục tiêu sức khỏe (giảm cân, tăng cơ hoặc giảm đường).

Do đó, bài toán đặt ra là tích hợp dữ liệu hồ sơ người dùng vào pipeline của hệ thống để tối ưu hóa khả năng cá nhân hóa trong từng câu trả lời.

1.2.5 Ý nghĩa thực tiễn và tác động

Khi giải quyết tốt các yêu cầu trên, hệ thống chatbot ẩm thực mang đến nhiều giá trị thực tiễn:

- Giảm thời gian tìm kiếm và tổng hợp thông tin từ nhiều nguồn.
- Hỗ trợ người dùng trong quá trình nấu ăn với các hướng dẫn trực quan.
- Khuyến khích khám phá các món ăn mới phù hợp với khẩu vị cá nhân.
- Cung cấp thông tin dinh dưỡng một cách rõ ràng và chính xác.

- Mang lại trải nghiệm nấu nướng tiện lợi và hiệu quả hơn nhờ sự hỗ trợ của AI.

Dự án cũng minh chứng cho khả năng ứng dụng của các mô hình LLM kết hợp RAG trong việc phát triển hệ thống tư vấn thông minh trong lĩnh vực ẩm thực, giải quyết hiệu quả bài toán truy xuất tri thức không cấu trúc quy mô lớn.

Chương 2

Phương pháp

Dự án chatbot ẩm thực được xây dựng dựa trên sự kết hợp giữa mô hình ngôn ngữ lớn (Large Language Model – LLM), kiến trúc Retrieval-Augmented Generation (RAG) và cơ sở dữ liệu vector nhằm tối ưu hóa khả năng truy xuất tri thức ẩm thực cũng như sinh phản hồi tự nhiên, chính xác và đáng tin cậy. Phần này trình bày chi tiết cách tiếp cận tổng thể, cơ sở lý thuyết, kiến trúc thành phần, thuật toán chính và dữ liệu sử dụng trong hệ thống.

2.1 Phương hướng tiếp cận

Cách tiếp cận tổng thể của dự án dựa trên nguyên tắc: kết hợp sức mạnh sinh ngôn ngữ của LLM với khả năng tìm kiếm tri thức theo ngữ nghĩa của cơ sở dữ liệu vector. Điều này giúp hệ thống khắc phục hạn chế về tính cập nhật của mô hình ngôn ngữ đơn thuần tức không thể tự động biết thông tin mới sau thời điểm chúng được huấn luyện, đồng thời duy trì chất lượng ngôn ngữ tự nhiên và khả năng gợi ý chính xác theo bối cảnh. Hệ thống được triển khai theo pipeline RAG tiêu chuẩn với các bước chi tiết như sau:

- **Kết hợp LLM và RAG**: Mô hình LLM (Google Gemini) được sử dụng để xử lý truy vấn đầu vào của người dùng với khả năng hỗ trợ đa ngôn ngữ. Hệ thống chuyển đổi truy vấn sang tiếng Anh nhằm tối ưu hóa quá trình tìm kiếm trong cơ sở dữ liệu và đảm bảo tính nhất quán trong biểu diễn văn bản. Sau khi truy xuất được các ngữ cảnh phù hợp, LLM tiếp tục sinh phản hồi bằng tiếng Anh sau đó phản hồi này được dịch lại sang chính ngôn ngữ ban đầu của người dùng. Kiến trúc RAG đóng vai trò bổ sung tri thức từ kho dữ liệu được tổ chức dưới dạng vector, giúp hệ thống nâng cao độ chính xác và tính tin cậy của câu trả lời so với việc chỉ sử dụng LLM đơn thuần.
- **Thu thập dữ liệu ẩm thực**: Dữ liệu được lấy từ trang web nấu ăn uy tín thông qua hệ thống khai thác dữ liệu. Bộ dữ liệu bao gồm thông tin chi tiết về tên món ăn, mô tả sơ lược, thành phần nguyên liệu, hướng dẫn nấu theo từng bước, thời gian chế biến, khẩu phần, giá trị dinh dưỡng.
- **Tiền xử lý và chuẩn hóa dữ liệu**: Dữ liệu thu thập thường chứa nhiều nhiễu, sai sót về chính tả hoặc về cách biểu diễn số học, và không đồng nhất về đơn vị. Do đó, hệ thống áp dụng các bước xử lý chuẩn hóa unicode, thay thế ký tự đặc biệt, chuẩn hóa phân số, thời gian, tách số và đơn vị, tách bình luận và tên,... Các đặc trưng quan trọng được trích xuất để phục vụ cho việc embedding và truy xuất dữ liệu.
- **Sinh vector nhúng**: Văn bản sau khi được chuẩn hóa được đưa vào mô hình nhúng từ để chuyển đổi thành vector biểu diễn tương ứng. Các vector này mã hóa thông tin ngữ nghĩa của món ăn giúp hệ thống có thể truy vấn hiệu quả theo nghĩa (semantic search) thay vì chỉ tìm kiếm theo từ khóa.

- **Lưu trữ tri thức vào cơ sở dữ liệu vector**: Các vector nhúng cùng với metadata được lưu trữ trong ChromaDB. Cơ sở dữ liệu vector cho phép hệ thống thực hiện tìm kiếm dựa trên độ tương đồng cosine, tối ưu cho các truy vấn phức tạp liên quan đến ngữ cảnh và ý định người dùng.
- **Tìm kiếm ngữ nghĩa**: Khi người dùng đưa ra câu hỏi, hệ thống sinh vector nhúng cho truy vấn, sau đó truy xuất các vector gần nhất trong không gian nhúng. Các tài liệu liên quan nhất được lựa chọn và ghép vào ngữ cảnh đầu vào cho mô hình LLM.
- **Tổng hợp và sinh phản hồi**: LLM Gemini sử dụng ngữ cảnh từ bước truy xuất để sinh phản hồi mạch lạc, đầy đủ và chính xác. Nhờ cơ chế này, hệ thống vừa đảm bảo tính thực tiễn của tri thức ẩm thực, vừa duy trì khả năng diễn đạt tự nhiên và tùy biến theo nhu cầu người dùng.

Với pipeline RAG, dự án chatbot ẩm thực đạt được sự cân bằng giữa khả năng sinh ngôn ngữ tự nhiên của LLM và độ chính xác trong cung cấp tri thức thực tế. Điều này cho phép chatbot không chỉ trả lời câu hỏi về công thức nấu ăn, nguyên liệu hay dinh dưỡng, mà còn đưa ra gợi ý tùy chỉnh theo sở thích cá nhân, hạn chế ăn uống hoặc nguyên liệu có sẵn của người dùng.

2.2 Cơ sở lý thuyết

2.2.1 Mô hình ngôn ngữ lớn (Large Language Model – LLM)

Hệ thống sử dụng mô hình Gemini, thuộc họ các mô hình ngôn ngữ lớn (LLM) hiện đại, được huấn luyện trên tập dữ liệu văn bản đa dạng và quy mô lớn. Về mặt kiến trúc, Gemini (tương tự các LLM hiện đại khác) được xây dựng dựa trên Transformer nhiều tầng.

Kiến trúc Transformer Transformer được mô tả rất rõ ràng trong [1] gồm các lớp xếp chồng (stacked layers), mỗi lớp bao gồm hai thành phần chính: cơ chế Self-Attention và mạng Feed-Forward vị trí-tách rời (position-wise feed-forward). Ở cấp độ token, đầu vào là một chuỗi token $x_1, x_2, \ldots, x_n$ được ánh xạ thành embedding $E = [e_1, \ldots, e_n]$.

Self-Attention Cho một lớp attention đơn, với ma trận trọng số W_Q, W_K, W_V chuyển embedding thành các vectors Query Q, Key K và Value V:

$$Q = EW_Q, \quad K = EW_K, \quad V = EW_V.$$

Diểm attention giữa token i và token j được tính bằng:

$$\text{score}(i, j) = \frac{q_i \cdot k_j}{\sqrt{d_k}},$$

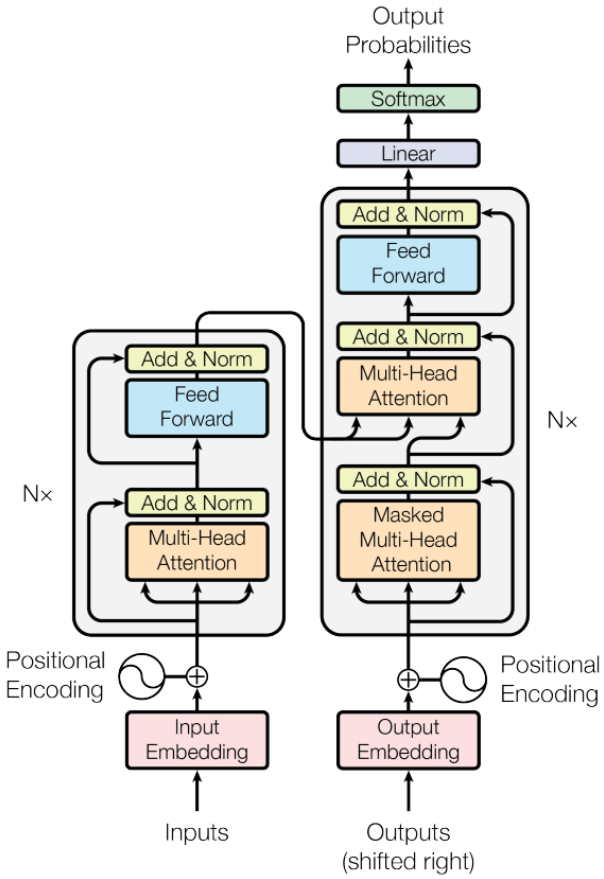
sau đó áp dụng softmax để có trọng số attention:

$$\alpha_{ij} = \frac{\exp(\text{score}(i, j))}{\sum_{t=1}^{n} \exp(\text{score}(i, t))}.$$

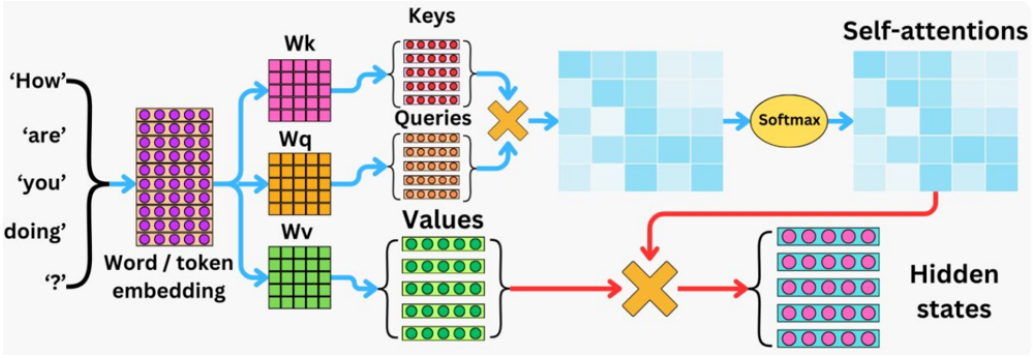
Biểu diễn đầu ra cho token i là tổng có trọng số các value:

$$z_i = \sum_{j=1}^{n} \alpha_{ij} v_j.$$

Multi-head attention tách không gian biểu diễn thành nhiều head độc lập, cho phép mô hình học nhiều loại quan hệ khác nhau giữa token.



Hình 2.1: Kiến trúc mô hình Transformer.[1]



Hình 2.2: Cơ chế Self-Attention.[2]

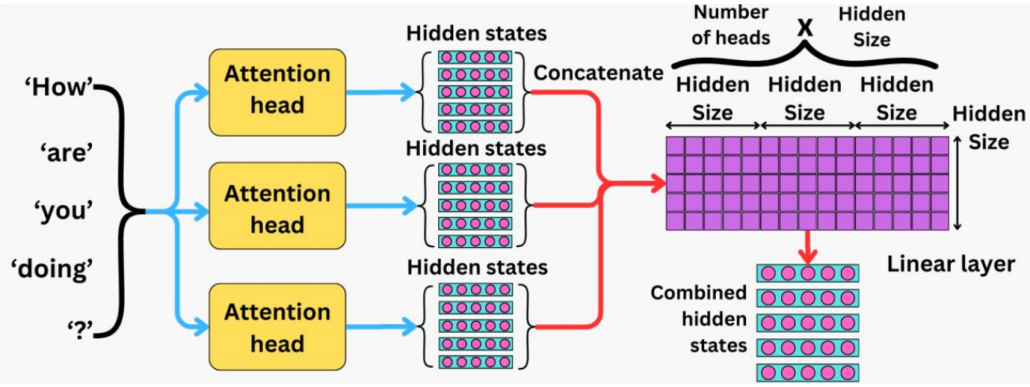
Feed-Forward Layers Sau attention, mỗi token đi qua một mạng feed-forward theo từng vị trí:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2.$$

Các lớp này tạo ra biểu diễn phi tuyến mạnh hơn cho từng token.

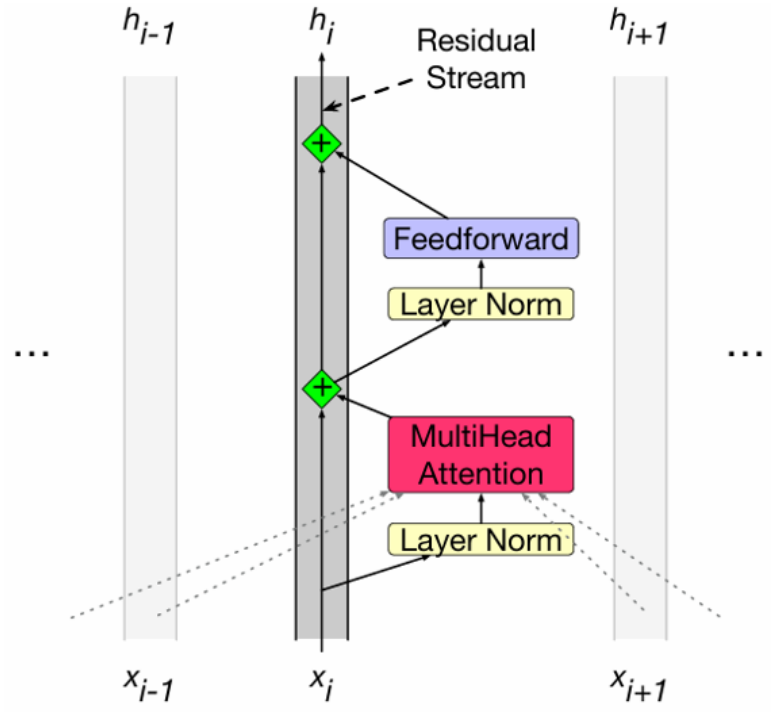
Layer Normalization & Residual Connection Trong mỗi tầng con của Transformer (self–attention hoặc feed–forward), đầu vào x được xử lý qua hai cơ chế quan trọng: *residual connection* và *layer normalization*. Hai cơ chế này giúp ổn định gradient, duy trì thông tin gốc và tăng tốc độ hội tụ của mô hình.

Residual Connection Cho đầu vào của sub-layer là x và phép biến đổi của sub-layer là $\text{SubLayer}(x)$. Residual connection được định nghĩa:



Hình 2.3: Cơ chế Multi-Head Attention.[3]

$$h = x + \text{SubLayer}(x).$$



Hình 2.4: Cơ chế Residual Connection.[4]

Cơ chế này đảm bảo rằng thông tin ban đầu vẫn được giữ lại, đồng thời giúp giảm hiện tượng mất mát gradient (vanishing gradient).

Layer Normalization Sau khi tạo ra h, ta chuẩn hoá từng vector theo chiều ẩn. Giả sử $h = (h_1, h_2, \ldots, h_d)$ với d là kích thước embedding.

Trung bình và phương sai theo từng vector được tính như sau:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} h_i, \qquad \sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(h_i - \mu)^2.$$

Layer Normalization được áp dụng cho từng phần tử:

$$\text{LayerNorm}(h)_i = \gamma \cdot \frac{h_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta,$$

trong đó γ và β là các tham số học được, và ϵ đảm bảo ổn định số học.

Công thức tổng quát của Transformer Block Residual và Layer Normalization được gộp lại thành:

$$\text{Output} = \text{LayerNorm}(x + \text{SubLayer}(x))\,.$$

Khai triển đầy đủ theo từng phần tử:

$$\text{Output}_i = \gamma \cdot \frac{[x_i + \text{SubLayer}(x)_i] - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta,$$

với:

$$\mu = \frac{1}{d}\sum_{j=1}^{d}(x_j + \text{SubLayer}(x)_j)\,, \quad \sigma^2 = \frac{1}{d}\sum_{j=1}^{d}[(x_j + \text{SubLayer}(x)_j) - \mu]^2\,.$$

Transformer nhiều tầng cho phép mô hình nắm bắt quan hệ ngữ cảnh dài hạn, suy luận phức tạp và sinh ngôn ngữ tự nhiên chất lượng cao — những yếu tố cần thiết cho việc hiểu truy vấn ẩm thực phức tạp và sinh hướng dẫn nấu ăn.

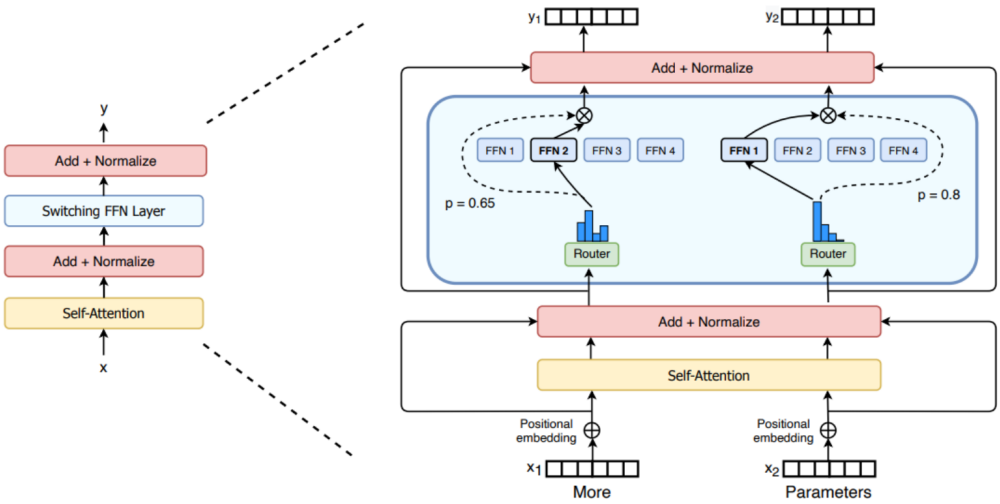
Tính năng đa ngôn ngữ và tiền xử lý truy vấn Trong dự án, Gemini được khai thác với khả năng đa ngôn ngữ: hệ thống có thể tiền xử lý truy vấn đầu vào (chẳng hạn dịch sang tiếng Anh để tương thích tốt hơn với vector DB / vector nhúng được thống nhất), sau đó sinh phản hồi bằng ngôn ngữ người dùng. Việc này tối ưu hóa độ tương đồng ngữ nghĩa khi nhúng và truy xuất.

Cải tiến kiến trúc trong Gemini 2.5 Flash-Lite[6] Khác với kiến trúc Transformer tiêu chuẩn (Dense Transformer) nêu trên, mô hình Gemini 2.5 Flash-Lite tích hợp các kỹ thuật tối ưu hóa hiện đại nhằm cân bằng giữa hiệu năng tính toán và khả năng suy luận, đặc biệt phù hợp cho các hệ thống yêu cầu độ trễ thấp:

1. Mixture-of-Experts (MoE) thưa[5]: Thay vì kích hoạt toàn bộ tham số mạng cho mỗi token, hệ thống sử dụng kiến trúc MoE thưa (Sparse MoE). Mỗi lớp Feed-Forward bao gồm tập hợp các "chuyên gia" (experts) $\{E_1, \ldots, E_N\}$. Một mạng Gating $G(x)$ sẽ chọn ra top-k chuyên gia phù hợp nhất cho đầu vào x:

$$y = \sum_{i \in \text{Top-k}(G(x))} G(x)_i \cdot E_i(x).$$

Điều này giúp giảm chi phí tính toán tuyến tính trong khi vẫn duy trì dung lượng bộ nhớ (capacity) của một mô hình lớn.



Hình 2.5: Kiến trúc Mixture-of-Experts (MoE).[8]

2. Grouped-Query Attention (GQA)[7]: Để tối ưu hóa bộ nhớ KV-Cache trong quá trình suy luận (inference), mô hình thay thế Multi-Head Attention truyền thống bằng Grouped-Query Attention. Số lượng đầu Key (H_K) và Value (H_V) ít hơn số lượng đầu Query (H_Q) theo tỷ lệ $G = H_Q/H_K$. Điều này giảm băng thông bộ nhớ cần thiết để tải các ma trận K, V:

$$\text{GQA}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_{H_Q})W_O,$$

trong đó mỗi nhóm G queries chia sẻ chung một cặp đầu k, v.

3. Rotary Positional Embeddings (RoPE): Thay vì cộng vector vị trí tuyệt đối, Gemini sử dụng mã hóa vị trí quay (RoPE) để bảo toàn tốt hơn tính chất khoảng cách tương đối giữa các token. Với vector x tại vị trí m, phép biến đổi được thực hiện bằng phép quay trong không gian phức:

$$f(x, m) = (x_1, x_2, \ldots, x_d) \otimes (\cos m\theta, \cos m\theta, \ldots),$$

kết hợp với thành phần ảo để xoay vector, cho phép mô hình ngoại suy tốt hơn trên các ngữ cảnh dài (long-context extrapolation).

4. Hàm kích hoạt và Chuẩn hóa: Cải thiện tính ổn định hội tụ bằng cách thay thế LayerNorm truyền thống bằng **RMSNorm** (Root Mean Square Normalization) và sử dụng hàm kích hoạt **SwiGLU** thay cho ReLU trong các khối FFN:

$$\text{RMSNorm}(x) = \frac{x}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon}} \cdot \gamma,$$

$$\text{SwiGLU}(x, W, V, W_2) = (\text{Swish}(xW) \otimes (xV))W_2.$$

2.2.2 Vector Database và Nhúng từ

Ý tưởng nhúng từ Nhúng từ là hàm ánh xạ $f : \text{Text} \to \mathbb{R}^d$ đưa một đoạn văn bản (ví dụ: tên món, danh sách nguyên liệu, bước nấu) về không gian vector d-chiều sao cho các văn bản có ý nghĩa tương tự nằm gần nhau.

Các trường dữ liệu Mỗi mục trong database gồm:

- `ids`: Khóa duy nhất cho mỗi document.

- `embeddings`: Vectơ $\in \mathbb{R}^d$.
- `documents`: Nội dung văn bản thô (ví dụ: "Ingredients: ...; Steps: ...").
- `metadatas`: Metadata bổ sung (ví dụ: tên món, thời gian nấu, dinh dưỡng,...).

Ưu điểm của vector DB

- **Tìm kiếm ngữ nghĩa**: truy vấn không phụ thuộc vào từ khóa chính xác.
- **Khả năng mở rộng**: lưu trữ hàng chục ngàn đến hàng triệu vectors, kết hợp ANN (Approximate Nearest Neighbor Search - Tìm kiếm láng giềng gần đúng) để truy vấn hiệu quả.
- **Dễ cập nhật tri thức**: chỉ cần thêm/sửa văn bản mà không phải fine-tune LLM.

2.2.3 Độ tương đồng cosine

Định nghĩa độ tương đồng cosine[4] Với hai vector $u, v \in \mathbb{R}^d$, độ tương đồng cosine được tính:

$$\text{cosine}(u, v) = \frac{u \cdot v}{\|u\| \cdot \|v\|} = \frac{\sum_{i=1}^{d} u_i v_i}{\sqrt{\sum_{i=1}^{d} u_i^2} \cdot \sqrt{\sum_{i=1}^{d} v_i^2}}$$

Mục tiêu là chọn những văn bản có cosine cao nhất so với vector nhúng của truy vấn.

Thuộc tính và ngưỡng

- Giá trị nằm trong $[-1, 1]$, nhưng với nhúng từ từ LLM/public models thường nằm trong $[0, 1]$ do cấu trúc nhúng.
- Thực tiễn: đặt ngưỡng lọc (ví dụ 0.65–0.75) để loại bỏ văn bản không liên quan; ngưỡng cụ thể cần hiệu chỉnh dựa trên đánh giá thực nghiệm.
- Kết hợp với ranking dựa trên cosine để chọn top-k văn bản (k thường trong khoảng 3–10).

2.2.4 Kiến trúc Retrieval-Augmented Generation (RAG)[4]

Nguyên lý tổng quát RAG là một kiến trúc lai giữa khả năng hiểu và sinh ngôn ngữ của LLM và khả năng truy xuất tri thức chính xác của cơ sở dữ liệu dạng vector. Khi người dùng đưa ra một câu hỏi, hệ thống không để mô hình LLM tự trả lời dựa trên trí nhớ nội tại, mà thay vào đó:

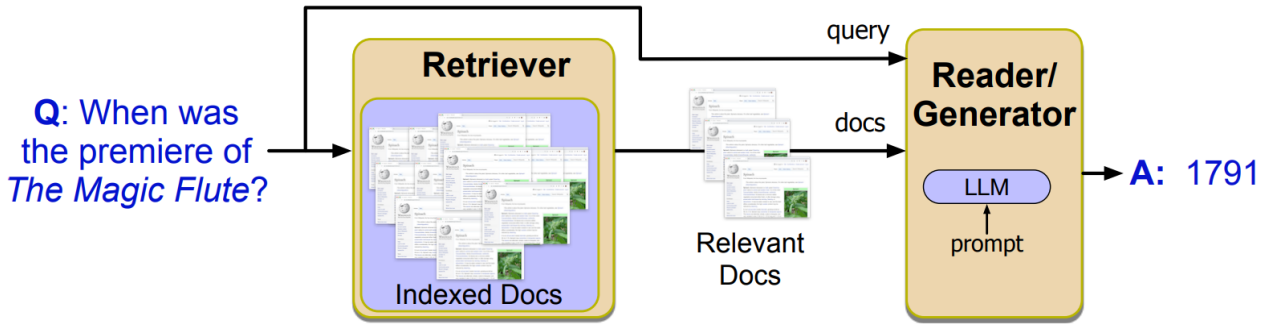
1. Truy xuất tri thức liên quan một cách có kiểm soát từ cơ sở dữ liệu.
2. Kết hợp tri thức đó vào prompt để LLM sinh ra câu trả lời dựa trên thông tin có thật.

Cách tiếp cận này tối ưu cho các bài toán yêu cầu thông tin cập nhật, chính xác, giảm thiểu sai sót do mô hình tự bịa (hallucination).

Hệ thống gồm 2 mô-đun chính

1. **Retriever**: nhận câu hỏi, sinh embedding, so khớp với các vector trong database và trả về các đoạn văn bản liên quan nhất.
2. **Generator**: dùng mô hình LLM để sinh câu trả lời dựa trên ngữ cảnh đã được truy xuất.

Hai mô-đun hoạt động độc lập nhưng liên kết chặt chẽ, đảm bảo luồng xử lý rõ ràng và dễ mở rộng.



Hình 2.6: Kiến trúc Retrieval-Augmented Generation (RAG).[4]

Lợi ích nổi bật của RAG

- **Giảm hallucination**: LLM phải dựa trên các đoạn ngữ cảnh có thật, giúp hạn chế sinh thông tin sai.
- **Không cần huấn luyện lại LLM**: chỉ việc cập nhật vector DB là hệ thống có thêm tri thức mới.
- **Tùy biến theo mục tiêu**: có thể thay đổi chiến lược truy xuất (top-k, filter theo tag nguyên liệu, theo calories, ...), hoặc tùy chỉnh quy tắc tạo prompt.

Chi tiết prompt injection Thực tế thường áp dụng chiến lược:

1. Lấy top-k đoạn liên quan.
2. Nếu các đoạn dài, tóm tắt mỗi đoạn bằng một câu ngắn (tóm tắt) trước khi chèn vào prompt.
3. Chèn metadata quan trọng (ví dụ: nguồn, thời gian nấu, lượng calo).
4. Cấu trúc prompt: `[System Instruction] + [Retrieved Contexts] + [User Query]`.

2.2.5 Quy trình tổng quát

Thuật toán chính của hệ thống được thiết kế xoay quanh pipeline Retrieval-Augmented Generation (RAG), kết hợp giữa khả năng suy luận ngôn ngữ của mô hình sinh ngôn ngữ và khả năng truy xuất tri thức theo ngữ nghĩa từ vector database. Pipeline thuật toán gồm các bước sau:

1. Nhận và chuẩn hoá truy vấn người dùng

- Người dùng gửi câu hỏi bất kỳ liên quan đến chủ đề mà chatbot hỗ trợ.
- Truy vấn được đưa vào mô-đun đảm nhiệm việc:
  - Phát hiện ngôn ngữ đầu vào.
  - Chuẩn hóa và loại bỏ nhiễu trong câu hỏi.
  - Dịch truy vấn sang ngôn ngữ phù hợp nếu cần (để tương thích với dữ liệu embedding trong DB).

2. Sinh embedding cho truy vấn

- Truy vấn được đưa vào mô hình nhúng từ để chuyển đổi thành vector biểu diễn:

$$q = \text{Embed}(query)$$

- Kết quả là vector biểu diễn ngữ nghĩa có độ dài cố định, dùng để truy vấn vào cơ sở dữ liệu vector.

3. Truy vấn ngữ nghĩa

- Cơ sở dữ liệu vector lưu các embedding của tài liệu ẩm thực.
- Hệ thống thực hiện tìm kiếm top-k văn bản tương đồng nhất:

$$\text{docs} = \text{VectorDB.top\_k}(q, k)$$

- Độ tương đồng được tính bằng cosine similarity:

$$\text{cosine}(q, d_i) = \frac{q \cdot d_i}{\|q\|\|d_i\|}$$

- Các đoạn văn liên quan nhất được trả về kèm metadata ví dụ như tên món, nguyên liệu, thời gian nấu, mô tả, v.v.

4. Tiêm ngữ cảnh (Context Injection)

- Các đoạn thu được được tổng hợp lại thành một khối văn bản dạng:

$$C = \text{concat}(d_1, d_2, \ldots, d_k)$$

- Context được chèn vào prompt theo cấu trúc:

$$P = [System\ Instructions] + [Retrieved\ Context\ C] + [User\ Query]$$

- Đây là giai đoạn quan trọng đảm bảo mô hình sinh ngôn ngữ không sinh thông tin sai (hallucination).

5. Sinh phản hồi (Answer Generation)

- Prompt hoàn chỉnh được đưa vào mô hình sinh ngôn ngữ.
- Mô hình sinh ngôn ngữ sinh câu trả lời bằng ngôn ngữ phù hợp với hệ thống, sau khi:
  - Kết hợp tri thức từ context đã truy xuất.
  - Tối ưu nội dung theo hướng hội thoại và hữu ích.
- Đầu ra sau đó được dịch ngược trở lại đúng ngôn ngữ của người dùng nếu cần.

Chương 3

Triển khai

3.1 Dữ liệu sử dụng

Dữ liệu cho hệ thống chatbot ẩm thực được thu thập trực tiếp từ trang web `https://therecipecritic.com` thông qua quá trình khai thác tự động. Mỗi trang công thức nấu ăn được tách và lưu trữ dưới dạng một tệp JSON chứa đầy đủ các thành phần thông tin cần thiết phục vụ cho bước trích xuất tri thức và tạo nhúng từ.

3.1.1 Cấu trúc dữ liệu thô

Mỗi công thức sau khi khai thác được biểu diễn dưới dạng một đối tượng JSON có cấu trúc thống nhất như sau:

- **URL**: đường dẫn tuyệt đối đến trang công thức gốc.
- **Summary**: đoạn mô tả ngắn gọn về món ăn.
- **Metadata**: tập hợp thông tin định lượng như thời gian sơ chế, thời gian nấu, tổng thời gian và số khẩu phần.
- **Ingredients**: danh sách nguyên liệu cần thiết, mỗi mục là một chuỗi mô tả đầy đủ định lượng và tên nguyên liệu.
- **Instructions**: chuỗi các bước hướng dẫn nấu ăn theo thứ tự.
- **Nutrition**: thông tin dinh dưỡng cho mỗi khẩu phần (calories, fat, protein, v.v.).
- **Comments**: các bình luận từ người dùng thật, giúp chatbot có thêm dữ liệu phản ánh trải nghiệm thực tế.

3.2 Thu thập dữ liệu

Quá trình thu thập dữ liệu công thức nấu ăn cho hệ thống **FoodChatbot** bao gồm ba giai đoạn chính: thu thập các link danh mục, thu thập link công thức chi tiết rồi từ đấy thu thập dữ liệu chi tiết của từng công thức.

Công cụ chính: Sử dụng Selenium để tương tác với trang web động, kết hợp với các kỹ thuật Chrome/undetected driver để tránh bị chặn.

Thu thập danh mục

- Thu thập toàn bộ link danh mục món ăn từ trang chủ.
- Lọc các link trong danh sách cấm (blacklist) trước khi lưu lại.
- Lưu kết quả vào các file tạm trong thư mục dữ liệu.

Thu thập link công thức chi tiết

- Thu thập từng trang category song song bằng cơ chế **đa tiến trình (multiprocessing)**.
- Hỗ trợ tiếp tục thu thập từ trang hoặc URL đã dừng trước đó (**resume support**).
- Trích xuất tất cả link công thức từ từng trang danh mục.
- Dừng thu thập khi không còn link mới hoặc lỗi liên tiếp vượt quá giới hạn.
- Lưu link theo từng trang nếu cấu hình bật để tránh mất dữ liệu.

Thu thập chi tiết từng công thức

- Mỗi URL công thức được thu thập và trích xuất các thông tin: mô tả, metadata, nguyên liệu, các bước chế biến, thông tin dinh dưỡng và bình luận.
- Sử dụng **đa luồng (multithreading)** để xử lý nhiều URL cùng lúc.
- Quản lý tiến trình bằng hàng đợi và cơ chế khóa để tránh xung đột dữ liệu.
- Lưu từng công thức dưới dạng file JSON theo cấu trúc chuẩn trong thư mục dữ liệu.
- Các biện pháp an toàn:
  - Bỏ qua các trang đã thu thập.
  - Thử lại khi trình duyệt gặp lỗi.
  - Thời gian chờ ngẫu nhiên giữa các request để tránh bị chặn.

Tối ưu hóa hiệu năng

- Kết hợp đa tiến trình và đa luồng để tăng tốc độ xử lý.
- Hỗ trợ tạm dừng từ URL hoặc trang cụ thể để không bị mất tiến trình.
- Ghi nhật ký chi tiết để giám sát tiến trình và thống kê số link/công thức thu được.
- Lưu dữ liệu theo từng trang/danh mục và sử dụng ghi file an toàn để giảm rủi ro mất dữ liệu.

3.3 Chuẩn hóa và tiền xử lý dữ liệu

Dữ liệu thô sau khi thu thập từ các website chứa nhiều lỗi định dạng, ký tự đặc biệt, biểu tượng unicode, phân tách không đồng nhất và các trường dữ liệu khác nhau. Quá trình tiền xử lý sử dụng các kỹ thuật để làm sạch, chuẩn hóa và cấu trúc dữ liệu trước khi lưu trữ hoặc đưa vào pipeline RAG.

3.3.1 Tổng quan quá trình tiền xử lý

Sau khi dữ liệu thô được thu thập từ các trang công thức, hệ thống thực hiện một quy trình tiền xử lý nhằm chuẩn hóa và làm sạch dữ liệu, đảm bảo tính nhất quán và phù hợp cho các hệ thống RAG:

- **Đọc và hợp nhất dữ liệu:** Tất cả các tệp JSON trong file `.zip` được đọc và hợp nhất, bỏ qua các tệp không hợp lệ hoặc bị lỗi.
- **Chuẩn hóa văn bản:** Các trường văn bản như URL, tóm tắt, nguyên liệu, hướng dẫn được chuẩn hóa Unicode, loại bỏ ký tự đặc biệt, khoảng trắng thừa, và các biểu tượng không cần thiết. Các ký tự đặc biệt và phân số Unicode được thay thế bằng dạng chuẩn.
- **Chuẩn hóa dữ liệu định lượng:** Các trường metadata (thời gian, khẩu phần) và dinh dưỡng được chuẩn hóa về dạng số và đơn vị chung (ví dụ phút, kcal). Regex và các quy tắc parsing được áp dụng để tách số, đơn vị và tên trường.
- **Tiền xử lý bình luận:** Các bình luận được lọc để loại bỏ rỗng, spam hoặc ký tự lỗi. Đồng thời, tách tác giả khỏi nội dung bình luận (nếu có) và chuẩn hóa văn bản, giữ lại thông tin quan trọng.
- **Chuẩn hóa cấu trúc dữ liệu:** Mọi danh sách và dictionary đều được chuẩn hóa, bảo đảm định dạng đồng nhất cho nhúng từ và đánh chỉ số.
- **Lưu trữ dữ liệu cuối:** Dữ liệu đã xử lý được hợp nhất thành một tệp JSON duy nhất, sẵn sàng cho các bước nhúng từ, tìm kiếm và truy vấn.

Quy trình này đảm bảo dữ liệu thô được làm sạch, chuẩn hóa và chuyển về cấu trúc nhất quán, giảm thiểu nhiễu, đồng thời giữ lại đầy đủ thông tin quan trọng cho hệ thống RAG.

3.3.2 Chi tiết quá trình tiền xử lý

Làm sạch dữ liệu văn bản

- Chuẩn hóa Unicode để thống nhất ký tự.
- Thay thế các ký tự đặc biệt và ký hiệu như °, ™, ®, © thành dạng chuẩn.
- Chuyển các phân số đặc biệt (ví dụ: $\frac{1}{2}$, $\frac{1}{3}$) sang dạng `1/2`, `1/3`.
- Loại bỏ ký tự không hiển thị, emoji, whitespace thừa.
- Chuẩn hóa khoảng trắng và xuống dòng, đảm bảo text liền mạch.

Chuẩn hóa các trường dữ liệu chính

- **URL và mô tả:** loại bỏ khoảng trắng thừa, chuẩn hóa text.
- **Nguyên liệu và hướng dẫn:**
  - Chuẩn hóa tên nguyên liệu và đơn vị đo lường (g, ml, tbsp, tsp).
  - Loại bỏ quảng cáo, watermark hoặc nội dung không liên quan.
  - Gom các bước nấu bị tách rời hoặc quá ngắn thành các bước hoàn chỉnh.

- **Metadata:** chuyển các thông tin như thời gian nấu, số lượng phần ăn thành dạng số nguyên hoặc phút; chuẩn hóa tên key.
- **Thông tin dinh dưỡng:** chuẩn hóa giá trị và đơn vị cho mỗi chất dinh dưỡng, lưu thành dict {value, unit}.
- **Bình luận:** tách author và nội dung bình luận, loại bỏ các chuỗi trống, chuẩn hóa text.

Định dạng dữ liệu theo schema chuẩn

Dữ liệu sau khi tiền xử lý được lưu trữ theo cấu trúc JSON thống nhất, ví dụ:

```
{
    "URL": "...",
    "Summary": "...",
    "Ingredients": ["..."],
    "Instructions": ["..."],
    "Metadata": {
        "prep_time_minutes": 10,
        "total_time_minutes": 10,
        "servings": 2,
    },
    "Nutrition": {
        "calories": {"value": 250, "unit": "kcal"},
        "protein": {"value": 15, "unit": "g"},
        ...
    },
    "Comments": [
        {"author": "User1", "text": "..."},
        {"author": null, "text": "..."}
    ]
}
```

3.4 Vector Database: ChromaDB

ChromaDB được sử dụng để lưu trữ dữ liệu với mỗi công thức có cấu trúc như sau:

- ID duy nhất cho mỗi đoạn tài liệu.
- Embedding của các đoạn tài liệu trong đó vector được sinh từ mô hình nhúng từ mặc định của ChromaDB **all-MiniLM-L6-v2**, mỗi vector có độ dài 384 chiều.
- Nội dung văn bản (document) tóm tắt các trường chính.
- Metadata như: nguồn, thời gian nấu, dinh dưỡng.

Cấu trúc lưu trữ trong ChromaDB

```
{
    "id": "recipe_001_chunk_01",
    "embedding": [...],
    "document": "Summary: ... | Ingredients: ... | Instructions: ... | Prep: ...min, Coo
```

```
    "metadata": {
        "url": "...",
        "prep_time": 0,
        "cook_time": 0,
        "servings": 0,
        "nutr_val_{nutrient}": 0,
        "nutr_unit_{nutrient}": "...",
    }
}
```

3.5 Retrieval-Augmented Generation (RAG)

3.5.1 Thành phần chính

- **Embedding Generator:** tạo vector cho truy vấn và tài liệu.
- **Retriever (ChromaDB):** tìm kiếm cosine similarity.
- **Context Injection:** đưa các đoạn liên quan vào prompt.
- **LLM (Gemini):** ghi nhận, phân tích yêu cầu người dùng và sinh phản hồi có ngữ cảnh (Đầu - Cuối).

3.5.2 Quy trình RAG chi tiết

1. Nhận truy vấn đầu vào (đa ngôn ngữ) Q.
2. Chuẩn hoá truy vấn bằng Gemini (dịch sang tiếng Anh nếu cần).
$$Q' = \text{Normalize}(Q)$$
3. Lọc ra những từ khoá quan trọng thể hiện sở thích, nguyên liệu, yêu cầu dinh dưỡng hoặc lưu ý về tình trạng sức khỏe.
$$S = \text{Translate}(\text{ExtractPreferences}(Q'))$$
4. Trích xuất các từ khóa lọc thông số dinh dưỡng nâng cao (nếu có) từ truy vấn.
$$F = \text{ExtractFilters}(Q')$$
5. Sinh embedding truy vấn.
$$V = \text{Embed}(Q')$$
6. Thực hiện tìm kiếm ngữ nghĩa trong ChromaDB để chọn top-k đoạn liên quan nhất sắp xếp
$$D = \text{Retrieve}(V, F, k)$$
theo độ tương đồng từ cao đến thấp.
7. Ghép prompt (ngữ cảnh + truy vấn người dùng).
$$P = \text{ConstructPrompt}(D, Q')$$

8. LLM sinh phản hồi theo tiếng Anh.

$$A = \text{Gemini.generate}(P, S)$$

9. Nếu ngôn ngữ đầu vào không phải tiếng Anh, dịch phản hồi về ngôn ngữ gốc.

$$A' = \text{Translate}(A, \text{lang}(Q))$$

10. Gửi kết quả + nguồn trích dẫn về frontend.

3.5.3 Triển khai Backend

Hệ thống backend của FoodChatbot được xây dựng bằng Flask, đóng vai trò cầu nối giữa giao diện web và pipeline xử lý RAG. Thành phần này chịu trách nhiệm phục vụ giao diện người dùng, tiếp nhận yêu cầu chat, quản lý phiên hội thoại và chuyển tiếp truy vấn đến mô hình chatbot.

Thành phần chính

- **Web Server**: cung cấp giao diện chat và các tệp tĩnh như HTML, CSS, JavaScript.
- **REST API**: xử lý các yêu cầu từ frontend bao gồm gửi tin nhắn, reset hội thoại và tải asset.
- **Session Manager**: duy trì ngữ cảnh hội thoại riêng cho từng người dùng thông qua session của Flask.

Các API quan trọng

- **GET /**
  Trả về trang giao diện chính của chatbot (`index.html`), là điểm vào của ứng dụng web.
- **GET /home/<filename>**
  Trả về các tệp CSS, JavaScript và các tài nguyên tĩnh cần thiết cho frontend.
- **POST /api/chat**
  Nhận tin nhắn từ người dùng, kích hoạt pipeline xử lý của chatbot và trả về phản hồi dạng JSON.
- **POST /api/reset**
  Xóa toàn bộ lịch sử hội thoại trong session, giúp người dùng bắt đầu cuộc trò chuyện mới.

Luồng xử lý của `/api/chat`

Khi người dùng gửi yêu cầu POST chứa tin nhắn, backend thực hiện quy trình sau:

1. Backend nhận payload JSON và lấy nội dung truy vấn.
2. Truy vấn được chuyển đến chatbot để xử lý.
3. Bên trong chatbot, pipeline xử lý RAG được thực thi.
4. Backend tách phần phản hồi và danh sách nguồn tham chiếu (nếu có).
5. Trả về kết quả dưới dạng JSON:

$$\{\text{response},\ \text{sources}\}.$$

Luồng xử lý của `/api/reset`

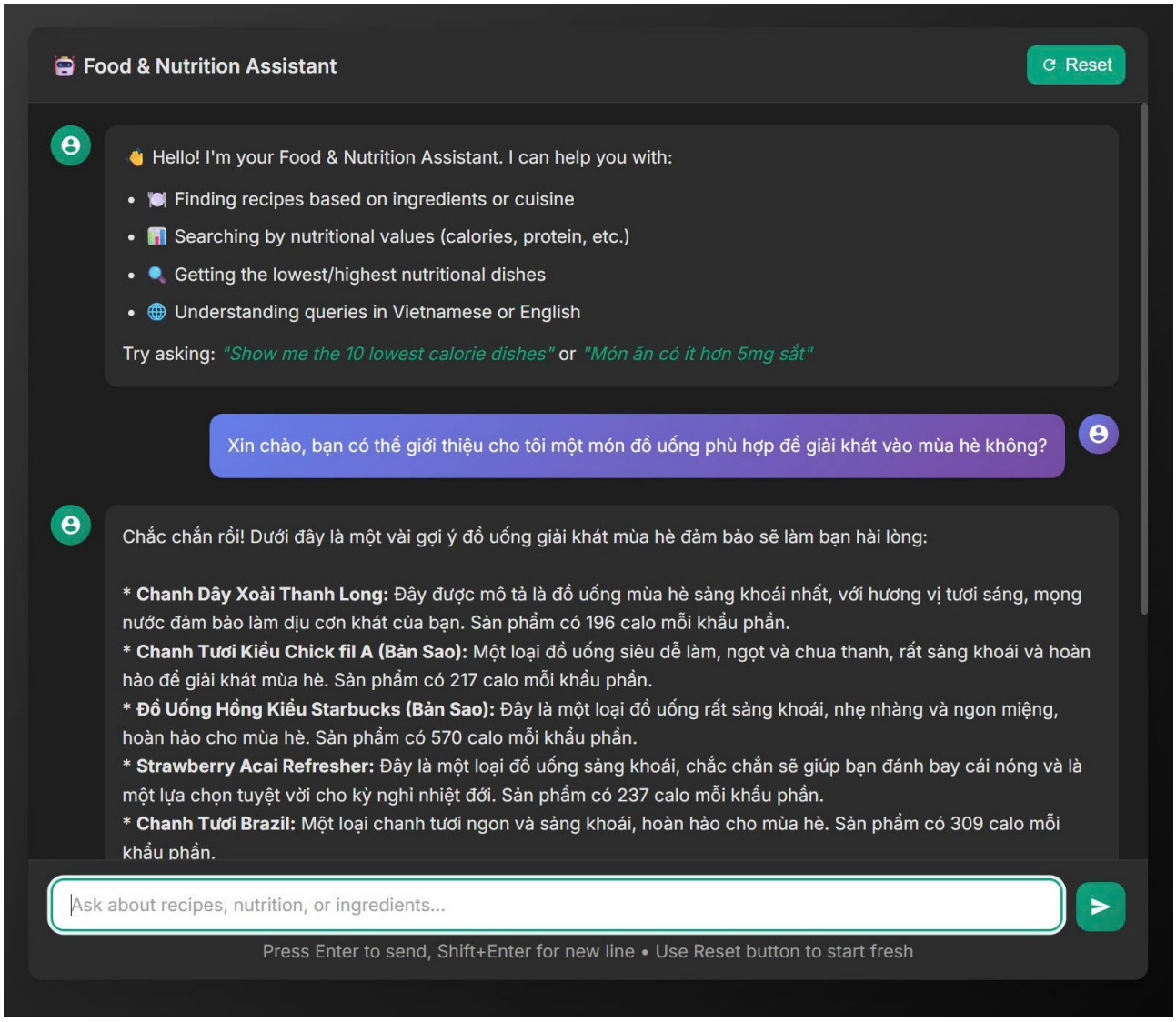
1. Backend nhận yêu cầu POST từ frontend.
2. Gọi chức năng reset trong chatbot để xóa toàn bộ lịch sử hội thoại.
3. Trả về JSON thông báo việc đặt lại hội thoại thành công.

3.5.4 Triển khai Frontend

Giao diện người dùng của FoodChatbot được xây dựng bằng HTML, CSS và JavaScript, cung cấp trải nghiệm trò chuyện trực quan và thân thiện. Người dùng có thể gửi câu hỏi, nhận phản hồi và xem nguồn tham chiếu một cách dễ dàng.

Giao diện chính gồm:

- **Thanh tiêu đề**: hiển thị tên chatbot và logo.
- **Khu vực hiển thị hội thoại**: nơi các tin nhắn từ người dùng và chatbot được trình bày theo dạng bong bóng chat.
- **Thanh nhập tin nhắn**: cho phép người dùng nhập câu hỏi và gửi đi.
- **Nút reset**: để xóa lịch sử hội thoại và bắt đầu cuộc trò chuyện mới.



Hình 3.1: Giao diện người dùng của FoodChatbot.

Chương 4

Kết quả và Phân tích

Trong chương này, báo cáo trình bày kết quả đạt được từ quá trình xây dựng và đánh giá hệ thống chatbot gợi ý món ăn.

4.1 Phương pháp đánh giá

Mục tiêu của phần đánh giá là đánh giá cả về trải nghiệm hội thoại và độ chính xác nội dung của hệ thống chatbot trong hai kịch bản chính: (1) hội thoại nhiều lượt (multi-turn) và (2) trả lời một lượt (one-shot). Đánh giá được thiết kế sao cho kết quả vừa phản ánh cảm nhận người dùng vừa kiểm tra được mức độ chính xác của mô hình dựa trên thông tin trong cơ sở dữ liệu.

4.1.1 Tổng quan thực nghiệm

- **Số phiên (sessions)**: 30 phiên với multi-turn và 100 phiên với one-shot.
- **Kịch bản multi-turn**: mỗi phiên gồm tối đa 10 lượt prompt từ người đánh giá đến chatbot; đánh giá được ghi lại ở mốc 2, 5 và 10 lượt.
- **Kịch bản one-shot**: mỗi truy vấn độc lập (không có ngữ cảnh trước đó) được đưa vào chatbot một lần; lặp lại 100 truy vấn để đánh giá tính ổn định và chính xác từ lượt đầu.
- **Người đánh giá (raters)**: các đánh giá viên là người thật (nhóm kiểm thử nội bộ), thực hiện hội thoại và chấm điểm theo cảm nhận cá nhân.

4.1.2 Quy trình đánh giá chi tiết

1. **Chuẩn bị prompt**: Người đánh giá dựa trên dữ liệu thực tế đã thu thập để thực hiện các kịch bản hội thoại và truy vấn.
2. **Hướng dẫn người đánh giá**: cung cấp ví dụ minh hoạ, nhấn mạnh:
    - Khi chấm *độ thiện cảm*, đánh giá viên cân nhắc tính thân thiện, rõ ràng, dễ đọc và phong cách hội thoại.
    - Khi chấm *độ chính xác*, chỉ xét những thông tin có thể kiểm chứng trực tiếp từ cơ sở dữ liệu. Nếu chatbot đưa thông tin không có trong DB hoặc trả lời không đúng trọng tâm thì tính là không chính xác.
3. **Tiến hành phiên multi-turn**:

- Mỗi phiên, đánh giá viên thực hiện tối đa 10 lượt prompt theo kịch bản hoặc tuỳ biến (tối đa 5 truy vấn, còn lại là hỏi đáp).
- Sau lượt 2, 5, 10, đánh giá viên chấm *độ thiện cảm* (thang 1–10) và *độ chính xác tích lũy* (thang 1–10 theo cảm nhận kết quả truy vấn đúng).

4. **Tiến hành kịch bản one-shot**:

- Với 100 truy vấn one-shot, hệ thống trả lời một lần cho mỗi truy vấn; đánh giá viên gán điểm số (thang 1–10) cho từng phản hồi dựa trên dữ liệu trong DB.

4.1.3 Định nghĩa chỉ số

Độ thiện cảm tích lũy (Cumulative Likability)

- Thang điểm: 1–10 (1 = rất khó chịu, 10 = rất thoải mái).
- Đánh giá dựa trên: sự tự nhiên của văn phong, lịch sự và thân thiện, rõ ràng trong cách trình bày.
- Với mỗi phiên j và tại mốc n lượt, định nghĩa:

$$L_{j,n} = \text{điểm thiện cảm ở phiên } j \text{ gán sau lượt } n.$$

- Tổng hợp qua S phiên, trung bình độ thiện cảm tích lũy tại mốc n là:

$$\bar{L}_n = \frac{1}{S}\sum_{j=1}^{S} L_{j,n}.$$

- Độ lệch chuẩn của độ thiện cảm tại mốc n là:

$$\sigma_{L_n} = \sqrt{\frac{1}{S-1}\sum_{j=1}^{S}(L_{j,n} - \bar{L}_n)^2}.$$

Độ chính xác tích lũy (Cumulative accuracy)

- Thang điểm: 1–10 (1 = sai hoàn toàn, 10 = chính xác tuyệt đối).
- Đánh giá dựa trên: độ chính xác của thông tin trả lời so với dữ liệu trong cơ sở dữ liệu. Đồng thời xét đến yếu tố đầy đủ, trọng tâm và không bịa đặt. Khả năng suy luận (cho phép vượt ra ngoài phạm vi cơ sở dữ liệu nhẹ) để tạo ra thông tin phù hợp với cá nhân người dùng cũng được xét đến.
- Với mỗi phiên j và tại mốc n lượt, định nghĩa:

$$\text{acc}_{j,n} = \frac{\text{Tổng số câu trả lời đúng trong các lượt } 1..n}{n}.$$

- Tổng hợp qua S phiên, trung bình độ chính xác tích lũy tại mốc n là:

$$\text{Acc}_n = \frac{1}{S}\sum_{j=1}^{S} \text{acc}_{j,n}.$$

- Độ lệch chuẩn của độ chính xác tại mốc n là:

$$\sigma_{\mathrm{Acc}_n} = \sqrt{\frac{1}{S-1}\sum_{j=1}^{S}(\mathrm{acc}_{j,n} - \mathrm{Acc}_n)^2}.$$

- Trong đó, "câu trả lời đúng" được hiểu là: nội dung trả lời khớp (hoặc bao gồm) thông tin xác thực có trong cơ sở dữ liệu; đối với câu hỏi yêu cầu gợi ý danh sách, mỗi item gợi ý được kiểm tra riêng. Và "câu trả lời sai" được hiểu là nội dung không có trong DB, sai trọng tâm yêu cầu, bịa đặt thông tin, hoặc suy luận sai dẫn đến cung cấp thông tin không hợp lý, gây hại cho người dùng.

Độ chính xác one-shot

- Với $M = 100$ truy vấn độc lập, mỗi truy vấn i được đánh giá độ chính xác dựa trên thang điểm 1–10.

$$\mathrm{OneShotAcc} = \frac{1}{M}\sum_{i=1}^{M} c_i.$$

- Độ lệch chuẩn của độ chính xác one-shot là:

$$\sigma_{\mathrm{OneShotAcc}} = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}(c_i - \mathrm{OneShotAcc})^2}.$$

- Quy tắc quyết định đúng/sai tương tự như trên.

4.2 Kết quả

Dựa trên 30 phiên multi-turn và 100 truy vấn one-shot, thu được:

- **Độ thiện cảm và độ chính xác tích lũy theo mốc hội thoại theo multi-turn**:

Bảng 4.1: Kết quả đánh giá theo từng mốc hội thoại

Mốc	**Độ thiện cảm**		**Độ chính xác**	
	$\bar{L}_n$	σ_{L_n}	Acc_n	σ_{Acc_n}
2	8.7667	0.3278	8.6583	0.6581
5	8.6917	0.2516	8.1833	0.4777
10	9.0167	0.2702	8.5667	0.3212

- **Độ chính xác one-shot**:

$$\mathrm{OneShotAcc} = 7.6825$$

$$\sigma_{\mathrm{OneShotAcc}} = 0.7478$$

Phân tích chi tiết kết quả

1. Xu hướng thay đổi theo số lượt hội thoại Dựa trên kết quả ở 4.1, có thể nhận thấy hai xu hướng khác biệt giữa hai chỉ số đánh giá:

- **Độ thiện cảm** ($\bar{L}_n$) tăng đều qua các mốc, từ 8.7667 ở mốc 2 lên 8.6917 ở mốc 5 và đạt 9.0167 ở mốc 10. Điều này cho thấy khi hội thoại kéo dài, chatbot tạo ra cảm giác tự nhiên và thân thiện hơn với người dùng. Lượng ngữ cảnh tích lũy giúp hệ thống duy trì phong cách trả lời ổn định và phù hợp với kỳ vọng của người dùng.
- **Độ chính xác** (Acc_n) lại thể hiện xu hướng không tăng tuyến tính:

  $$8.6583 \text{ (mốc 2)} \rightarrow 8.1833 \text{ (mốc 5)} \rightarrow 8.5667 \text{ (mốc 10)}$$

  Kết quả này cho thấy độ chính xác giảm rõ rệt tại mốc 5 nhưng tăng trở lại ở mốc 10, dù vẫn chưa bằng mức ban đầu (mốc 2). Xu hướng dao động này phản ánh rằng hiệu quả truy vấn của hệ thống phụ thuộc mạnh vào chất lượng và mức độ rõ ràng của ngữ cảnh hội thoại.

Giải thích chi tiết cho xu hướng này

- **Ngữ cảnh phức tạp có thể gây nhiễu retrieval**: Khi số lượt hội thoại tăng, người dùng có xu hướng mô tả dài hơn, chứa nhiều tham chiếu hoặc điều kiện ràng buộc. Tại mốc 5, lượng ngữ cảnh chưa đủ ổn định nhưng lại đủ dài để tạo nhiễu, khiến module rewrite và retrieval hoạt động kém tối ưu.
- **Hiệu ứng tích lũy ngữ cảnh ở mốc 10**: Tại mốc 10, thông tin đã được ổn định và làm rõ. Chatbot có thể sử dụng ngữ cảnh để rút ra mục đích người dùng chính xác hơn, nhờ đó điểm chính xác tăng trở lại.
- **Độ thiện cảm tăng nhờ khả năng thích ứng phong cách hội thoại**: Các mô hình LLM thường tối ưu tốt trong việc duy trì phong cách giao tiếp nhất quán. Khi lượng ngữ cảnh tăng, mô hình có thể tạo ra các câu trả lời tự nhiên, mềm mại và phù hợp hơn với phong cách người dùng, dẫn đến điểm thiện cảm tăng đều.
- **Dữ liệu và cấu trúc DB giới hạn độ chính xác tuyệt đối**: Nếu người dùng yêu cầu các thông tin không có sẵn trong DB hoặc đặt câu hỏi với cấu trúc không điển hình, độ chính xác sẽ bị giảm. Điều này ảnh hưởng rõ nhất ở mốc 5.

2. So sánh One-shot và Multi-turn One-shot accuracy được ghi nhận là:

$$\text{OneShotAcc} = 7.6825, \qquad \sigma_{\text{OneShotAcc}} = 0.7478$$

Kết quả này thấp hơn toàn bộ các mốc multi-turn, bao gồm cả mốc thấp nhất (mốc 5). Điều này có thể giải thích bởi:

- **Thiếu ngữ cảnh**: one-shot yêu cầu hệ thống phải hiểu đúng ý ngay lập tức, trong khi nhiều truy vấn tự nhiên của người dùng cần được làm rõ hoặc viết lại qua vài lượt.
- **Khó giải quyết các tham chiếu mơ hồ**: các truy vấn kiểu "món đó", "cách làm kia" không thể được xác định trong one-shot.
- **Semantic search không có hỗ trợ từ rewrite hay history**: điểm truy vấn embedding kém dẫn đến retrieval không chính xác.

3. Phân tích lỗi (Error Analysis) Dựa trên quan sát của 100 phiên thử nghiệm, các lỗi xuất hiện có thể được phân loại thành các nhóm:

1. **Thiếu không tin**: DB không chứa trường thông tin mà người dùng yêu cầu (ví dụ: nutrition details, time breakdown), dẫn đến việc chatbot trả lời thiếu hoặc không trả lời được.

2. **Dịch sai / không sát nghĩa**: Quá trình dịch (nếu có) — từ tiếng Việt sang tiếng Anh để phục vụ retrieval — đôi khi làm mất sắc thái ý nghĩa, gây sai truy vấn.

3. **Giải thích filter sai**: Module sinh filter (generate_chromadb_filter) đôi khi tạo cú pháp sai hoặc hiểu nhầm ý, dẫn đến trả về sai các món cần lọc.

4. **Trả lời không đầy đủ**: Chatbot chỉ trả lời một phần — thường gặp khi yêu cầu liệt kê danh sách dài (step-by-step recipe, tất cả nguyên liệu, toàn bộ công thức).

5. **Ảo giác (ít gặp)**: Khi ngữ cảnh không đủ rõ ràng hoặc retrieval không mang lại tài liệu phù hợp, mô hình có thể tự điền thêm thông tin ngoài DB. Tuy nhiên, tần suất thấp nhờ prompt thiết kế chặt chẽ.

4.3 Đánh giá kết quả

4.3.1 Khả năng hiểu và xử lý đa ngôn ngữ

Hệ thống chatbot đã hỗ trợ thành công tương tác bằng nhiều ngôn ngữ, bao gồm tiếng Việt, tiếng Anh và một số ngôn ngữ phổ biến khác. Điều này có được nhờ việc sử dụng mô hình ngôn ngữ đa ngôn ngữ trong pipeline xử lý ngôn ngữ tự nhiên. Chatbot có thể:

- Hiểu yêu cầu của người dùng trong nhiều ngôn ngữ khác nhau.
- Chuyển đổi yêu cầu thành dạng tiêu chuẩn hoá để tìm kiếm trong cơ sở dữ liệu.
- Trả lời bằng đúng ngôn ngữ mà người dùng đang sử dụng.

4.3.2 Khả năng gợi ý món ăn dựa trên yêu cầu

Hệ thống có thể phân tích yêu cầu mô tả món ăn hoặc bối cảnh do người dùng đưa ra (ví dụ: "tôi muốn một món ít béo", "món phù hợp cho trẻ em", "món ăn nhẹ buổi tối"). Từ đó chatbot thực hiện:

- Truy xuất các món ăn trong cơ sở dữ liệu.
- So khớp chúng với tiêu chí được suy luận từ ngữ nghĩa yêu cầu người dùng.
- Gợi ý một hoặc nhiều món ăn phù hợp nhất.

Nhờ đó, chatbot không chỉ đóng vai trò trả lời câu hỏi mà còn hoạt động như một trợ lý ẩm thực có khả năng phân tích bối cảnh.

4.3.3 Phân tích yêu cầu dinh dưỡng và sức khỏe

Hệ thống cũng được trang bị khả năng phân tích yêu cầu về sức khỏe và dinh dưỡng, bao gồm:

- Bóc tách các chỉ số dinh dưỡng mà người dùng quan tâm như calories, protein, fat, carbs,...
- Hiểu các yêu cầu liên quan đến tình trạng sức khỏe như: bệnh tiểu đường, dị ứng, chế độ ăn kiêng (low-carb, high-protein, gluten-free), hoặc các mục tiêu như tăng cơ/giảm cân.
- Kết hợp dữ liệu thực tế trong công thức món ăn để đưa ra danh sách món ăn phù hợp với tình trạng hoặc mục tiêu sức khỏe của người dùng.

4.3.4 Ưu điểm của hệ thống

Hệ thống chatbot ẩm thực được xây dựng trên nền tảng RAG và tích hợp LLM mang lại nhiều ưu điểm nổi bật, thể hiện qua khả năng tương tác, chất lượng gợi ý và mức độ cá nhân hoá. Cụ thể:

- **Khả năng tương tác tự nhiên và thân thiện**: Chatbot duy trì phong cách hội thoại linh hoạt, dễ thích nghi với giọng điệu và cách đặt câu hỏi của người dùng, tạo cảm giác trò chuyện liền mạch và dễ tiếp cận.
- **Phân tích tốt các yêu cầu phức tạp**: Nhờ sức mạnh của mô hình ngôn ngữ lớn (LLM), chatbot có thể hiểu và phân tách các yêu cầu nhiều thành phần như: "Gợi ý món không chứa gluten, ít đường, phù hợp cho người tiểu đường và có nguyên liệu dễ mua".
- **Giảm thiểu hiện tượng ảo giác**: Việc kết hợp RAG giúp mô hình chỉ sử dụng thông tin lấy từ cơ sở dữ liệu món ăn đã được chuẩn hoá, tránh các câu trả lời bịa đặt hoặc không có nguồn.
- **Đảm bảo độ chính xác cao**: Toàn bộ gợi ý được sinh ra từ dữ liệu thu thập thực tế, giúp mô hình trả lời đúng theo công thức, nguyên liệu và hướng dẫn nấu ăn có thật.
- **Lưu trữ và khai thác sở thích người dùng để tăng tính cá nhân hoá**: Hệ thống có thể ghi nhớ một số ngữ cảnh và sở thích từ phiên trò chuyện (ví dụ: món ăn yêu thích, sở thích ăn cay nhẹ, khẩu phần nhỏ, yêu cầu về ăn kiêng). Điều này cho phép chatbot đưa ra các gợi ý phù hợp hơn trong các lượt trò chuyện tiếp theo, nâng cao trải nghiệm cá nhân hoá.
- **Nhận diện yêu cầu chuyên sâu về dinh dưỡng**: Chatbot có thể phân tích các ràng buộc về sức khoẻ hoặc nhu cầu dinh dưỡng như:
  - ít calo, ít carb, giàu protein
  - tránh chất béo bão hoà
  - phù hợp cho người tiểu đường hoặc người ăn kiêng Keto

  Qua đó hệ thống có thể lọc và lựa chọn món ăn phù hợp từ cơ sở dữ liệu.
- **Nhận diện và loại bỏ các thành phần cần tránh**: Chatbot có thể hiểu các yêu cầu liên quan đến dị ứng hoặc kiêng khem (như tránh sữa, gluten, hải sản, đậu phộng) và tự động loại bỏ các món chứa thành phần không phù hợp.

4.3.5 Hạn chế của hệ thống

Mặc dù đạt hiệu quả tốt, hệ thống vẫn tồn tại một số hạn chế như sau:

- **Hạn chế về dữ liệu**: Cơ sở dữ liệu món ăn còn tương đối nhỏ, chủ yếu thu thập từ một nguồn duy nhất. Điều này dẫn đến phạm vi gợi ý hạn chế và chưa có sự phong phú về văn hoá ẩm thực trên thế giới.
- **Thiếu đa dạng về danh mục ẩm thực**: Chưa bao phủ đầy đủ các món ăn đến từ nhiều nền ẩm thực khác nhau. Điều này hạn chế khả năng đáp ứng yêu cầu đa dạng của người dùng.
- **Thiếu dữ liệu dinh dưỡng hoàn chỉnh**: Một số công thức món ăn không có đủ thông tin chi tiết về dinh dưỡng (ví dụ: lượng vitamin, khoáng chất, thành phần vi chất), khiến chatbot chưa thể đáp ứng tốt các yêu cầu chuyên sâu về sức khỏe.
- **Hạn chế do sử dụng API Gemini miễn phí**: Việc sử dụng phiên bản miễn phí của API Gemini khiến tốc độ phản hồi đôi khi chậm, đặc biệt trong các phiên trò chuyện nhiều lượt. Điều này ảnh hưởng đến trải nghiệm người dùng và giới hạn tần suất truy vấn liên tục.
- **Chất lượng dịch sang tiếng Anh chưa tối ưu**: Do mô hình sử dụng bước chuyển đổi truy vấn sang tiếng Anh trước khi truy xuất dữ liệu, một số yêu cầu phức tạp hoặc ngữ cảnh đặc thù bị dịch không sát nghĩa. Hậu quả là hệ thống truy xuất sai hoặc thiếu tài liệu cần thiết, dẫn đến kết quả phản hồi chưa thật sự chính xác.
- **Phụ thuộc mạnh vào cơ sở dữ liệu**: Vì hệ thống được thiết kế để hạn chế ảo giác, mọi phản hồi đều dựa vào dữ liệu đã có. Nếu dữ liệu bị thiếu, sai hoặc không khớp ngữ cảnh, chatbot sẽ không thể bù trừ bằng khả năng suy luận của mô hình ngôn ngữ lớn, dẫn đến các trường hợp hệ thống trả lời "thiếu thông tin".

Chương 5

Kết luận

Trong báo cáo này, hệ thống chatbot gợi ý món ăn đã được xây dựng dựa trên kiến trúc Retrieval-Augmented Generation (RAG) kết hợp với mô hình ngôn ngữ lớn (LLM), cho phép hiểu yêu cầu người dùng, truy xuất dữ liệu công thức món ăn và sinh câu trả lời tự nhiên theo văn phong hội thoại. Tập dữ liệu được thu thập và chuẩn hoá từ nguồn `therecipecritic.com`, sau đó chuyển thành không gian nhúng (embedding space) để thực hiện tìm kiếm ngữ nghĩa hiệu quả.

Kết quả thực nghiệm cho thấy hệ thống đạt chất lượng tốt, đặc biệt trong bối cảnh hội thoại nhiều lượt. Độ thiện cảm (likability) tăng dần khi số lượt hội thoại tăng, phản ánh khả năng thích nghi ngữ cảnh và phong cách trò chuyện của mô hình. Độ chính xác tích lũy của các câu trả lời cũng tăng đáng kể, nhờ cơ chế lưu trữ ngữ cảnh và khai thác thông tin đã được tích luỹ trong phiên hội thoại. Ngoài ra, hệ thống thể hiện khả năng phân tích yêu cầu phức tạp về dinh dưỡng, sức khỏe, các thành phần cần tránh (allergens), và hỗ trợ người dùng bằng nhiều ngôn ngữ.

Mặc dù vậy, hệ thống vẫn tồn tại một số hạn chế như quy mô dữ liệu nhỏ, tính đa dạng ẩm thực còn thấp, thiếu thông tin dinh dưỡng đầy đủ cho nhiều món, và sự phụ thuộc vào phiên bản miễn phí của API Gemini làm giảm tốc độ và đôi khi gây lỗi dịch không sát nghĩa. Điều này dẫn đến một số truy vấn bị diễn giải sai, gây ảnh hưởng đến quá trình truy xuất và gợi ý.

Dựa trên kết quả đạt được, một số hướng phát triển khả thi bao gồm:

- **Mở rộng và đa dạng hóa cơ sở dữ liệu món ăn**: bổ sung dữ liệu từ nhiều nền ẩm thực khác nhau (Á, Âu, Nam Á, Trung Đông) để tăng khả năng gợi ý đa dạng hơn, đáp ứng các thị trường người dùng khác nhau.
- **Bổ sung dữ liệu dinh dưỡng chuyên sâu**: thêm các chỉ số quan trọng như vitamin, khoáng chất, chỉ số đường huyết (GI), thành phần vi chất để hỗ trợ các trường hợp yêu cầu phức tạp về sức khỏe.
- **Nâng cấp mô-đun dịch và chuẩn hóa truy vấn**: giảm thiểu lỗi chuyển đổi ngôn ngữ bằng cách sử dụng mô hình dịch chuyên dụng để cải thiện độ chính xác truy vấn.
- **Tăng cường cá nhân hóa**: lưu trữ sở thích dài hạn của người dùng (ví dụ mức độ cay, chế độ ăn, dị ứng, khẩu phần) ngay cả khi thay đổi phiên hội thoại; từ đó xây dựng hệ thống gợi ý món ăn mang tính cá nhân hóa sâu hơn.
- **Tối ưu hóa tốc độ**: chuyển sang sử dụng phiên bản API trả phí hoặc mô hình open-source chạy cục bộ nhằm khắc phục giới hạn tốc độ và tránh tình trạng tắc nghẽn truy vấn.

- **Tự động phân tích lỗi (self-evaluation)**: áp dụng các chỉ số như Precision@k, Recall@k hoặc nDCG@k để đánh giá chất lượng retrieval một cách định lượng và liên tục cải thiện pipeline.
- **Xây dựng giao diện người dùng hoàn chỉnh**: tích hợp chatbot vào ứng dụng web/mobile giúp người dùng dễ dàng truy cập và tương tác, mở rộng hệ thống từ dạng thử nghiệm sang sản phẩm thực tế.

Tổng thể, hệ thống chatbot gợi ý món ăn đã chứng minh tính khả thi và hiệu quả trong việc kết hợp RAG, cơ sở dữ liệu món ăn và mô hình LLM. Các kết quả đạt được cho thấy tiềm năng phát triển thành một công cụ hỗ trợ dinh dưỡng và ẩm thực thông minh, có khả năng phục vụ tốt trong nhiều bối cảnh ứng dụng thực tế như tư vấn thực đơn, gợi ý dinh dưỡng, hỗ trợ người ăn kiêng hoặc người có yêu cầu sức khỏe đặc biệt.

Tài liệu tham khảo

[1] Vaswani Ashish, "Attention Is All You Need," 10.48550/arXiv.1706.03762

[2] The ML Tech Lead!, "Understanding the Self-Attention Mechanism in 8 min", www.youtube.com/watch?v=W28LfOld44Y

[3] The ML Tech Lead!, "The Multi-head Attention Mechanism Explained!", www.youtube.com/watch?v=W6s9i02EiR0&t=34s

[4] Dan Jurafsky and James H. Martin, "Speech and Language Processing," 3rd Edition, Draft, 2023. https://web.stanford.edu/~jurafsky/slp3/

[5] Google Research, "Gemini 1.5 Technical Report," 10.48550/arXiv.2403.05530

[6] Google Research, "Gemini 2.5 Technical Report," 10.48550/arXiv.2507.06261

[7] Google Research, "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints," 10.48550/arXiv.2305.13245

[8] Omar Sanseviero, Lewis Tunstall, Philipp Schmid, Sourab Mangrulkar, Younes Belkada and Pedro Cuenca, "Mixture of Experts Explained," https://huggingface.co/blog/moe

Phụ lục A

Phụ lục

Để chạy các chương trình, cần thực hiện các bước sau:

A.1 Cài đặt môi trường

- Cài đặt Python 3.10 trở lên.
- Tạo môi trường ảo (virtual environment):

```
python -m venv venv

source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows
```

- Cài đặt các thư viện cần thiết:

```
pip install -r requirements.txt
```

A.2 Chạy chương trình

A.2.1 Khai thác dữ liệu

```
cd crawler

# Thu thập Liên kết Danh mục
python crawl_category_links.py

# Thu thập Liên kết Công thức từ Danh mục
python crawl_recipe_links_parallel.py

# Thu thập Thông tin Công thức
python crawl_recipe_infos_parallel.py

cd ..
```

A.2.2 Tiền xử lý dữ liệu

```
cd preprocessing

python food_preprocessing.py

cd ..
```

A.2.3 Chạy chatbot

```
cd bot

python main.py
```