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: MAT1207E – 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.

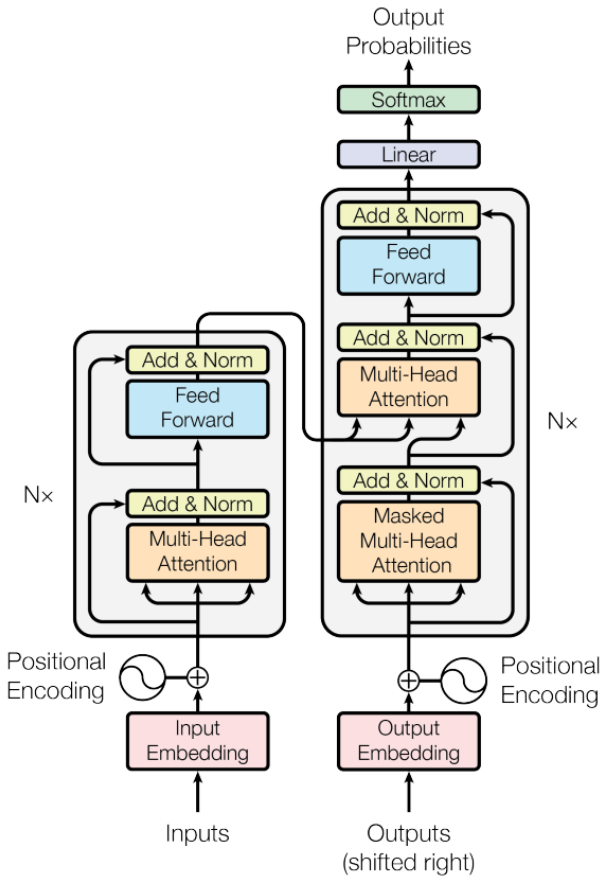
Với pipeline RAG, dự án chatbot ẩm thực đạt được sự cân bằng giữa khả năng sinh ngôn ngữ tự nhiên của LLM và độ chính xác trong cung cấp tri thức thực tế. Điều này cho phép chatbot không chỉ trả lời câu hỏi về công thức nấu ăn, nguyên liệu hay dinh dưỡng, mà còn đưa ra gợi ý tùy chỉnh theo sở thích cá 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 $E = [e_1, \ldots, e_n]$.



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

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:

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

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

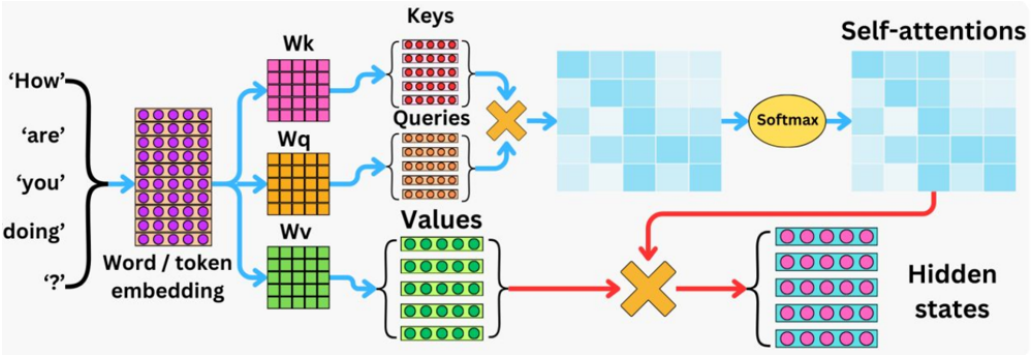
$$\text{score}(i, j) = \frac{q_i \cdot k_j}{\sqrt{d_k}},$$

sau đó áp dụng softmax để có trọng số attention:

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

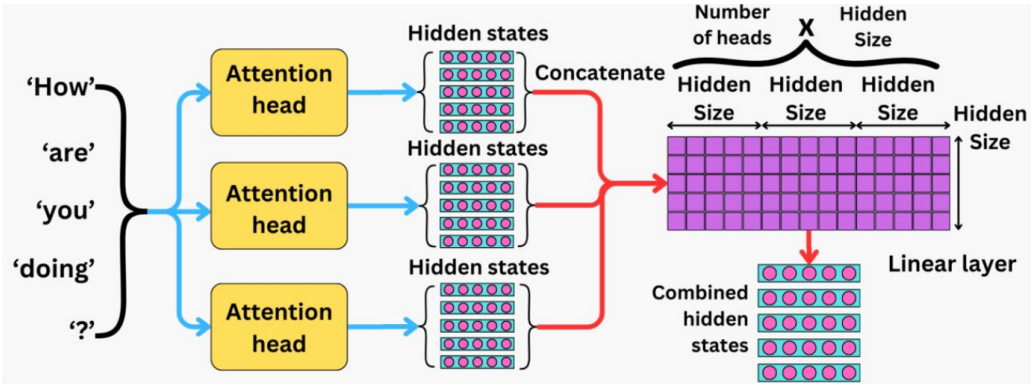
Biểu diễn đầu ra cho token i là tổng có trọng số các value:

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



Hình 2.2: Cơ chế Self-Attention.[2]

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 2.3: Cơ chế Multi-Head Attention.[3]

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

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

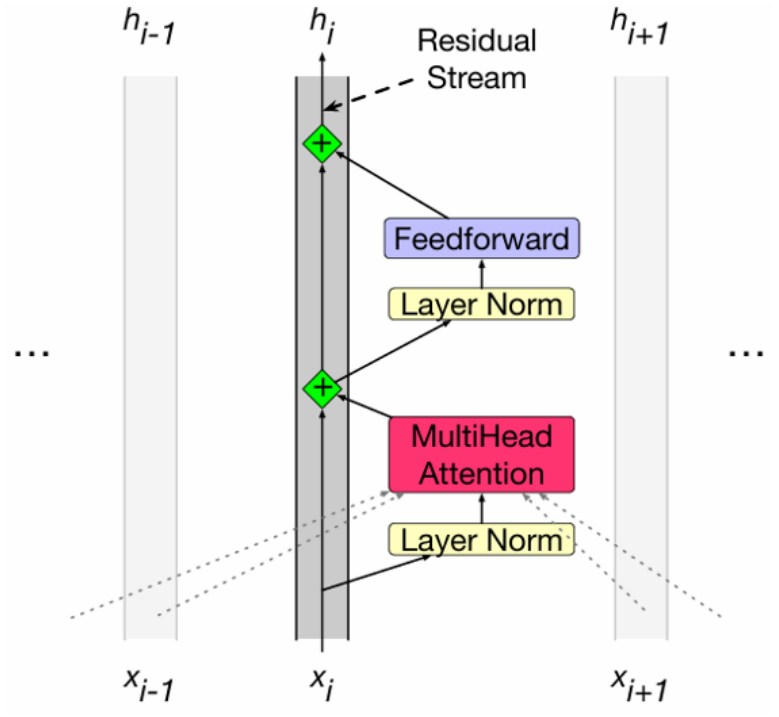
Các lớp này tạo ra biểu diễn phi tuyến mạnh hơn cho từng token.

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:

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

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).



Hình 2.4: Cơ chế Residual Connection.[4]

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.

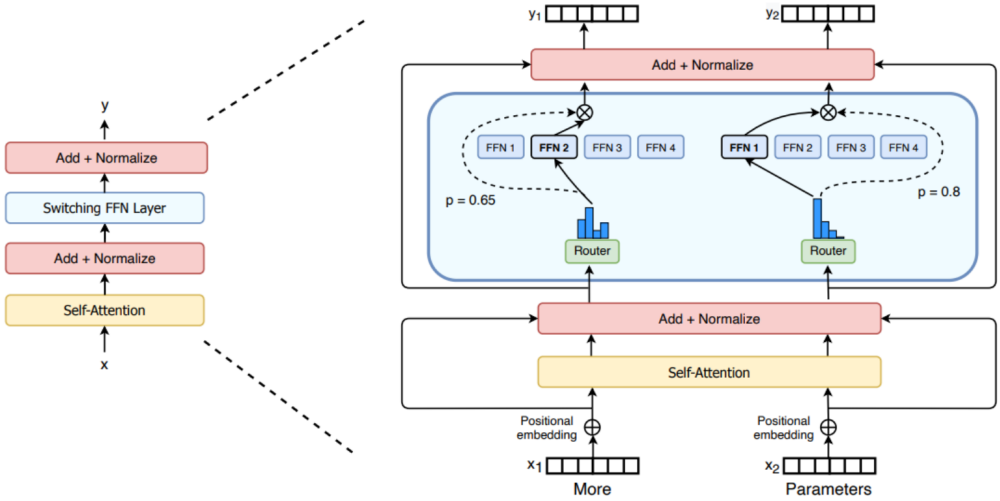
Tính năng đa ngôn ngữ và tiền xử lý truy vấn Trong dự án, Gemini được khai thác với khả năng đa ngôn ngữ: hệ thống có thể tiền xử lý truy vấn đầu vào (chẳng hạn dịch sang tiếng Anh để tương thích tốt hơn với vector DB / vector nhúng được thống nhất), sau đó sinh phản hồi bằng ngôn ngữ người dùng. Việc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

Khái niệm nhúng từ (Text Embedding) Nhúng từ (text embedding) là quá trình học một hàm ánh xạ liên tục:

$$f_\theta : \mathcal{T} \to \mathbb{R}^d$$

trong đó:

- $\mathcal{T}$ là không gian các chuỗi văn bản rời rạc (discrete text space),
- $\mathbb{R}^d$ là không gian vector liên tục d-chiều,
- θ là tập tham số của mô hình nhúng đã được huấn luyện trước.

Mục tiêu của nhúng từ không phải là biểu diễn cú pháp (syntax), mà là học một biểu diễn ngữ nghĩa (semantic representation) sao cho các văn bản có nội dung và ý nghĩa tương tự được ánh xạ đến các vector nằm gần nhau trong không gian hình học.

Góc nhìn hình học của không gian nhúng Trong không gian nhúng, mỗi đoạn văn bản được xem như một điểm trong $\mathbb{R}^d$. Khoảng cách hoặc góc giữa các vector phản ánh mức độ tương đồng ngữ nghĩa giữa các văn bản tương ứng. Điều này cho phép chuyển bài toán xử lý ngôn ngữ tự nhiên từ:

so khớp chuỗi ký tự

sang:

tìm kiếm láng giềng gần nhất trong không gian metric.

Giả định quan trọng của phương pháp này là các văn bản tự nhiên không phân bố ngẫu nhiên trong $\mathbb{R}^d$ mà nằm trên các manifold ngữ nghĩa có chiều thấp hơn, nơi mà các phép đo khoảng cách như cosine similarity có ý nghĩa.

Cấu trúc dữ liệu trong Vector Database Mỗi mục trong vector database được thiết kế để tách biệt rõ ràng ba không gian thông tin:

- `ids`: khóa định danh duy nhất, dùng để truy vết và trích dẫn nguồn.
- `embeddings`: vector nhúng $\in \mathbb{R}^d$, chỉ dùng cho truy xuất ngữ nghĩa.
- `documents`: nội dung văn bản thô hoặc đã được chuẩn hóa, dùng để đưa vào prompt của LLM.
- `metadatas`: tập thuộc tính có cấu trúc (thời gian nấu, dinh dưỡng, khẩu phần, nguồn, ...) dùng cho lọc logic.

Thiết kế này cho phép:

- thay đổi chiến lược truy xuất mà không ảnh hưởng đến mô hình sinh,
- mở rộng metadata mà không cần tái nhúng toàn bộ dữ liệu,
- kết hợp tìm kiếm ngữ nghĩa với các ràng buộc logic một cách linh hoạt.

Ưu điểm của Vector Database trong hệ RAG Vector database đóng vai trò là bộ nhớ tri thức ngoài (external knowledge store) cho mô hình ngôn ngữ lớn, mang lại các lợi ích sau:

- **Tìm kiếm ngữ nghĩa**: không phụ thuộc vào từ khóa chính xác, giúp hệ thống hiểu các truy vấn diễn đạt tự nhiên.
- **Khả năng mở rộng**: có thể lưu trữ từ hàng chục nghìn đến hàng triệu vector, kết hợp với các thuật toán ANN để truy vấn hiệu quả.
- **Dễ cập nhật tri thức**: dữ liệu có thể được thêm, sửa hoặc xoá mà không cần huấn luyện lại mô hình LLM.
- **Giảm hiện tượng ảo giác**: LLM chỉ sinh câu trả lời dựa trên tri thức được truy xuất từ database.

2.2.3 Độ tương đồng cosine và tìm kiếm láng giềng

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

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

Cosine similarity đo góc giữa hai vector, thay vì độ lớn tuyệt đối, do đó bất biến với phép co giãn (scaling) của vector.

Lý do lựa chọn cosine similarity Cosine similarity đặc biệt phù hợp với embedding từ các mô hình ngôn ngữ hiện đại vì:

- embedding thường được chuẩn hóa theo chuẩn L_2,
- độ lớn vector ít mang ý nghĩa ngữ nghĩa,
- dot-product và cosine trở nên gần tương đương khi vector đã chuẩn hóa.

Trong thực nghiệm, giá trị cosine similarity giữa các văn bản thường tập trung trong khoảng $[0.2, 0.9]$, thay vì toàn miền $[-1, 1]$.

Ngưỡng tương đồng và chiến lược chọn top-k Trong thực tế triển khai, hệ thống thường:

- đặt một ngưỡng cosine tối thiểu (ví dụ 0.65–0.75) để loại bỏ các văn bản không liên quan,
- sắp xếp các vector còn lại theo cosine giảm dần,
- chọn top-k văn bản (với k thường trong khoảng 3–10) để đưa vào prompt.

Giá trị ngưỡng và k không mang tính cố định mà cần được hiệu chỉnh dựa trên đánh giá thực nghiệm và đặc thù của miền dữ liệu.

Approximate Nearest Neighbor (ANN) Bài toán tìm kiếm láng giềng gần nhất chính xác (Exact Nearest Neighbor Search) yêu cầu tính khoảng cách giữa vector truy vấn và toàn bộ N vector trong database, với độ phức tạp thời gian:

$$O(N \cdot d)$$

trong đó d là số chiều của embedding. Khi N lớn (từ hàng trăm nghìn đến hàng triệu vector), cách tiếp cận này trở nên không khả thi về mặt độ trễ.

Do đó, các vector database hiện đại sử dụng các thuật toán tìm kiếm láng giềng gần đúng (Approximate Nearest Neighbor – ANN), với mục tiêu:

- giảm độ phức tạp truy vấn xuống gần $O(\log N)$ hoặc $O(N^\alpha)$ với $\alpha < 1$,
- đánh đổi một phần độ chính xác để đạt được tốc độ truy vấn cao và khả năng mở rộng.

HNSW (Hierarchical Navigable Small World) HNSW xây dựng một đồ thị đa tầng (multi-layer graph) dựa trên ý tưởng *small-world network*. Mỗi vector được biểu diễn như một nút, các cạnh nối tới những vector gần nhất theo khoảng cách metric.

Cấu trúc HNSW gồm:

- nhiều tầng đồ thị, trong đó tầng cao có ít nút hơn và liên kết thưa,
- tầng thấp nhất chứa toàn bộ vector và liên kết dày đặc hơn.

Quá trình truy vấn diễn ra theo kiểu *top-down greedy search*:

1. bắt đầu từ tầng cao nhất với một nút khởi tạo,
2. di chuyển dần xuống các tầng thấp hơn,
3. tại mỗi tầng, chỉ khám phá một tập con nhỏ các láng giềng tiềm năng.

HNSW có ưu điểm:

- độ chính xác (recall) rất cao,
- độ trễ thấp,
- không cần huấn luyện offline.

Nhược điểm chính là:

- tiêu tốn nhiều bộ nhớ do lưu trữ đồ thị,
- chi phí xây dựng index cao khi số lượng vector lớn.

IVF (Inverted File Index) IVF dựa trên ý tưởng *phân cụm không gian vector*. Trước tiên, toàn bộ tập vector được phân cụm thành K centroid (thường bằng k-means).

Mỗi vector được gán vào cụm gần nhất, và database chỉ lưu:

- danh sách các vector thuộc mỗi cụm,
- vector centroid đại diện cho cụm.

Khi truy vấn:

1. vector truy vấn được so sánh với các centroid,
2. chỉ $n \ll K$ cụm gần nhất được chọn,
3. tìm kiếm chi tiết chỉ diễn ra trong các cụm này.

Ưu điểm của IVF:

- giảm đáng kể số lượng vector cần so sánh,
- chi phí bộ nhớ thấp hơn HNSW.

Nhược điểm:

- phụ thuộc mạnh vào chất lượng phân cụm,
- recall giảm nếu vector truy vấn nằm gần ranh giới các cụm.

Product Quantization (PQ) Product Quantization là kỹ thuật *nén vector* nhằm giảm chi phí bộ nhớ và tăng tốc tính khoảng cách.

Ý tưởng chính của PQ là:

- chia mỗi vector d-chiều thành m vector con,
- mỗi vector con được lượng tử hóa bằng một codebook nhỏ,
- toàn bộ vector được biểu diễn bằng các chỉ số codebook thay vì giá trị thực.

Khoảng cách giữa hai vector được xấp xỉ bằng cách tra bảng khoảng cách đã tính trước giữa các codebook.
Ưu điểm của PQ:

- giảm mạnh bộ nhớ lưu trữ,
- phù hợp với hệ thống quy mô rất lớn.

Nhược điểm:

- độ chính xác thấp hơn do sai số lượng tử hóa,
- thường được dùng kết hợp với IVF (IVF+PQ) để cân bằng tốc độ và độ chính xác.

So sánh và lựa chọn trong thực tế Trong bối cảnh hệ Retrieval-Augmented Generation:

- HNSW thường được ưu tiên khi yêu cầu độ chính xác cao và số lượng vector ở mức vừa phải,
- IVF và IVF+PQ phù hợp hơn với hệ thống có quy mô rất lớn và hạn chế bộ nhớ,
- việc lựa chọn thuật toán ANN là một quyết định kiến trúc quan trọng, ảnh hưởng trực tiếp đến chất lượng truy xuất và độ trễ của toàn hệ thống.

ANN đánh đổi một phần độ chính xác để đạt được độ trễ thấp và khả năng mở rộng cao. Trong bối cảnh hệ RAG, việc truy xuất được tập văn bản "đủ liên quan" quan trọng hơn so với việc tìm đúng láng giềng gần nhất tuyệt đối.

Vai trò của Vector Database trong kiến trúc RAG Trong kiến trúc Retrieval-Augmented Generation, vector database đóng vai trò là cầu nối giữa:

- khả năng biểu diễn ngữ nghĩa của embedding,
- và khả năng sinh ngôn ngữ của mô hình LLM.

LLM không trực tiếp "nhớ" tri thức chi tiết, mà dựa vào vector database để truy xuất thông tin xác thực. Điều này giúp hệ thống trở nên an toàn hơn, dễ kiểm soát hơn và phù hợp với các bài toán yêu cầu độ chính xác cao như tư vấn ẩm thực và dinh dưỡng.

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

Nguyên lý tổng quát Retrieval-Augmented Generation (RAG) là một kiến trúc lai (hybrid architecture) kết hợp giữa:

- khả năng hiểu và sinh ngôn ngữ tự nhiên của mô hình ngôn ngữ lớn (LLM),
- và khả năng truy xuất tri thức chính xác, có kiểm soát từ cơ sở dữ liệu bên ngoài.

Thay vì để LLM sinh câu trả lời hoàn toàn dựa trên tri thức nội tại được mã hoá trong tham số mô hình, RAG đưa vào một bước truy xuất tri thức trung gian. Khi người dùng đưa ra một truy vấn, hệ thống thực hiện:

1. Truy xuất các tài liệu liên quan nhất từ cơ sở dữ liệu vector dựa trên ngữ nghĩa.
2. Kết hợp các tài liệu này vào prompt để LLM sinh câu trả lời dựa trên thông tin có thật.

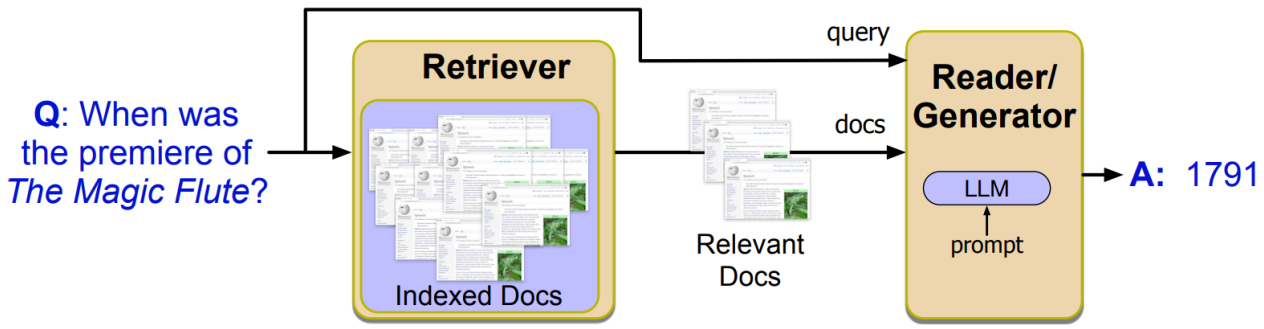
Cách tiếp cận này đặc biệt phù hợp với các bài toán yêu cầu thông tin cập nhật, chính xác và có thể kiểm chứng, đồng thời giúp giảm thiểu hiện tượng mô hình sinh thông tin sai lệch (hallucination).

Phân rã kiến trúc tổng thể Về mặt kiến trúc, một hệ thống RAG hoàn chỉnh được chia thành hai mô-đun chính, hoạt động độc lập nhưng liên kết chặt chẽ:

1. **Retriever**: chịu trách nhiệm truy xuất tri thức liên quan.
2. **Generator**: chịu trách nhiệm sinh câu trả lời dựa trên ngữ cảnh đã truy xuất.

Cách phân tách này giúp hệ thống:

- dễ bảo trì và mở rộng,
- cho phép cải tiến từng mô-đun mà không ảnh hưởng đến mô-đun còn lại,
- tách biệt rõ ràng giữa truy xuất tri thức và suy luận ngôn ngữ.



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

Chi tiết mô-đun Retriever Trong thực tế, Retriever không phải là một khối đơn giản mà bao gồm nhiều thành phần con:

- **Query Encoder**: chuyển truy vấn người dùng thành vector embedding.
- **Vector Search Engine**: thực hiện tìm kiếm láng giềng gần đúng (ANN) trên vector database.
- **Metadata Filtering**: áp dụng các ràng buộc logic (nguyên liệu, thời gian nấu, chế độ ăn, ...).
- **Re-ranking Module**: sắp xếp lại các kết quả truy xuất bằng mô hình đánh giá mức độ liên quan chi tiết hơn.

Chiến lược phổ biến là truy xuất hai giai đoạn:

- Giai đoạn 1: dùng embedding để lấy top-k_1 ứng viên (ví dụ $k_1 = 50$–100).
- Giai đoạn 2: dùng mô hình re-ranking để chọn top-k_2 tài liệu tốt nhất (ví dụ $k_2 = 5$–10).

Chi tiết mô-đun Generator Mô-đun Generator sử dụng LLM để sinh câu trả lời dựa trên ngữ cảnh đã được truy xuất. LLM không được phép suy luận vượt quá thông tin cung cấp trong context, nhằm đảm bảo tính chính xác và nhất quán.

Vai trò chính của Generator gồm:

- tổng hợp thông tin từ nhiều đoạn context,
- suy luận và diễn đạt lại thông tin theo ngôn ngữ tự nhiên,
- đảm bảo câu trả lời phù hợp với mục tiêu hội thoại.

Chiến lược Prompt Injection Prompt injection là bước then chốt quyết định chất lượng đầu ra của RAG. Một chiến lược chuẩn thường gồm:

1. Chọn top-k đoạn context có độ liên quan cao nhất.
2. Tóm tắt hoặc rút gọn các đoạn dài để tiết kiệm ngân sách token.
3. Chèn metadata quan trọng (nguồn, thời gian, dinh dưỡng, ...).
4. Cấu trúc prompt theo dạng:

$$P = [\text{System Instruction}] + [\text{Retrieved Context}] + [\text{User Query}]$$

System Instruction thường được thiết kế để:

- buộc mô hình chỉ sử dụng thông tin trong context,
- từ chối trả lời nếu thiếu dữ liệu,
- hạn chế suy diễn ngoài phạm vi tri thức được cung cấp.

Các biến thể kiến trúc RAG Tuỳ theo độ phức tạp của bài toán, RAG có thể được triển khai dưới nhiều biến thể:

- **Naive RAG**: truy xuất một lần và sinh câu trả lời trực tiếp.
- **RAG với Re-ranking**: bổ sung tầng đánh giá lại kết quả truy xuất.
- **Multi-hop RAG**: thực hiện nhiều lượt truy xuất nối tiếp để giải các câu hỏi suy luận nhiều bước.
- **Modular RAG**: tách riêng các mô-đun truy xuất, suy luận, kiểm chứng và tổng hợp.

Kiểm soát hallucination và các dạng lỗi Mặc dù RAG giúp giảm đáng kể hallucination, hệ thống vẫn có thể gặp các failure modes sau:

- **Retrieval failure**: truy xuất thiếu hoặc sai ngữ cảnh.
- **Context conflict**: các đoạn truy xuất chứa thông tin mâu thuẫn.
- **Over-generation**: mô hình suy diễn vượt quá dữ liệu cung cấp.

Các biện pháp kiểm soát thường bao gồm:

- đặt ngưỡng cosine similarity tối thiểu,
- yêu cầu LLM trích dẫn nguồn từ metadata,
- từ chối trả lời nếu không có context phù hợp.

2.2.5 Quy trình tổng quát của hệ thống RAG

Pipeline thuật toán của hệ thống được thiết kế xoay quanh kiến trúc RAG, bao gồm các bước sau:

1. Nhận và chuẩn hoá truy vấn người dùng

- Tiếp nhận câu hỏi từ người dùng.
- Phát hiện ngôn ngữ đầu vào.
- Chuẩn hoá văn bản và loại bỏ nhiễu.
- Dịch truy vấn sang ngôn ngữ tương thích với dữ liệu embedding nếu cần.

2. Sinh embedding cho truy vấn

- Truy vấn được ánh xạ sang không gian vector:

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

- Vector $q \in \mathbb{R}^d$ biểu diễn ngữ nghĩa của truy vấn.

3. Truy vấn ngữ nghĩa trên Vector Database

- Thực hiện tìm kiếm top-k văn bản tương đồng nhất:

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

- Độ tương đồng được đo bằng cosine similarity:

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

- Kết quả trả về bao gồm văn bản và metadata tương ứng.

4. Tiêm ngữ cảnh (Context Injection)

- Tổng hợp các đoạn truy xuất:

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

- Chèn context vào prompt:

$$P = [\text{System}] + [C] + [\text{User Query}]$$

5. Sinh phản hồi

- Prompt hoàn chỉnh được đưa vào LLM.
- LLM sinh câu trả lời dựa trên context.
- Kết quả được hậu xử lý và dịch ngược 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** được thiết kế như một pipeline crawler nhiều tầng, nhằm đảm bảo:

- độ bao phủ dữ liệu lớn,
- khả năng mở rộng,
- tính ổn định khi thu thập trên website động,
- và hạn chế tối đa nguy cơ bị chặn bởi cơ chế chống bot.

Toàn bộ pipeline thu thập dữ liệu được chia thành ba giai đoạn chính:

1. Thu thập các liên kết danh mục (category links).
2. Thu thập các liên kết công thức chi tiết (recipe links).
3. Thu thập nội dung chi tiết của từng công thức.

Công cụ và công nghệ sử dụng Hệ thống crawler sử dụng **Selenium WebDriver** làm công cụ chính để tương tác với website, do các lý do sau:

- Website mục tiêu sử dụng nội dung sinh động (dynamic content) dựa trên JavaScript.
- Nhiều thành phần (pagination, lazy loading, comment) không thể truy xuất bằng HTTP request thuần.

Để giảm nguy cơ bị phát hiện là bot, hệ thống áp dụng các kỹ thuật **undetected driver**, kết hợp cấu hình trình duyệt nâng cao và chiến lược hành vi giống người dùng thật.

3.2.1 Selenium WebDriver trong hệ thống thu thập dữ liệu

Tổng quan về Selenium WebDriver Selenium WebDriver là một framework tự động hoá trình duyệt cho phép điều khiển trực tiếp các trình duyệt web phổ biến (Chrome, Firefox, Edge, ...) thông qua giao diện lập trình. Không giống các thư viện thu thập dữ liệu dựa trên HTTP request thuần, Selenium vận hành bằng cách:

- khởi tạo một phiên trình duyệt thật,
- tải và thực thi đầy đủ mã HTML, CSS và JavaScript,
- cho phép truy cập vào cây DOM sau khi nội dung động được render.

Do đó, Selenium đặc biệt phù hợp với các website hiện đại sử dụng kiến trúc frontend động (Single Page Application hoặc heavy JavaScript rendering).

Lý do lựa chọn Selenium Trong hệ thống **FoodChatbot**, Selenium được lựa chọn thay vì các giải pháp crawler truyền thống vì các lý do kỹ thuật sau:

- Website mục tiêu sử dụng JavaScript để tải nội dung chính (lazy loading, pagination động).
- Nhiều thành phần quan trọng (bình luận, đánh giá, danh sách nguyên liệu) chỉ xuất hiện sau khi thực thi script phía client.
- Một số dữ liệu được nạp theo hành vi người dùng (scroll, click), không thể truy xuất trực tiếp qua API công khai.

Việc sử dụng Selenium cho phép hệ thống quan sát website gần giống hành vi của người dùng thật, từ đó đảm bảo độ đầy đủ và chính xác của dữ liệu thu thập.

Mô hình hoạt động của Selenium trong crawler Trong pipeline thu thập dữ liệu, mỗi instance Selenium được xem như một *worker độc lập*, bao gồm:

- một tiến trình trình duyệt riêng,
- một phiên WebDriver riêng,
- một ngữ cảnh DOM tách biệt.

Cách tổ chức này giúp:

- tránh xung đột trạng thái giữa các luồng thu thập,
- tăng khả năng song song hoá,
- dễ dàng khởi động lại khi gặp lỗi.

Quản lý trạng thái và đồng bộ hoá Selenium hoạt động theo mô hình bất đồng bộ với thời gian tải trang không xác định. Do đó, hệ thống áp dụng các kỹ thuật đồng bộ hoá sau:

- **Explicit wait**: chờ các phần tử DOM cụ thể xuất hiện trước khi trích xuất dữ liệu.
- **Implicit timeout**: giới hạn thời gian chờ để tránh treo tiến trình.
- **Scroll simulation**: cuộn trang để kích hoạt lazy loading.

Các kỹ thuật này giúp giảm lỗi trích xuất do DOM chưa hoàn tất render.

Hạn chế của Selenium Mặc dù mạnh mẽ, Selenium cũng tồn tại một số hạn chế:

- tiêu tốn nhiều tài nguyên CPU và RAM,
- độ trễ cao hơn so với crawler dựa trên HTTP request,
- khó mở rộng lên quy mô rất lớn nếu không có chiến lược quản lý tiến trình hợp lý.

Do đó, Selenium trong hệ thống này được sử dụng một cách có chọn lọc, tập trung vào các giai đoạn yêu cầu tương tác với nội dung động, thay vì áp dụng cho toàn bộ pipeline.

Vai trò của Selenium trong kiến trúc tổng thể Trong kiến trúc thu thập dữ liệu của **FoodChatbot**, Selenium đóng vai trò:

- lớp tương tác frontend,
- cầu nối giữa dữ liệu động và pipeline xử lý phía sau,
- nền tảng để triển khai các kỹ thuật chống phát hiện bot.

Dữ liệu thu thập từ Selenium sau đó được chuyển sang các giai đoạn tiền xử lý, chuẩn hoá và nhúng vector để phục vụ hệ thống Retrieval-Augmented Generation.

3.2.2 Kỹ thuật Undetected WebDriver và chống phát hiện bot

Bối cảnh và vấn đề Các website hiện đại thường triển khai nhiều cơ chế nhằm phát hiện và ngăn chặn hành vi thu thập dữ liệu tự động (bot), đặc biệt là khi sử dụng Selenium WebDriver mặc định. Các cơ chế này có thể dựa trên:

- đặc điểm trình duyệt (browser fingerprint),
- hành vi người dùng bất thường,
- tần suất truy cập cao,
- hoặc các thuộc tính đặc thù của WebDriver.

Nếu không có biện pháp phòng tránh, crawler rất dễ bị:

- chặn truy cập (HTTP 403),
- hiển thị CAPTCHA,
- hoặc trả về nội dung giả/thiếu dữ liệu.

Nguyên lý phát hiện Selenium của website Trong thực tế, nhiều website sử dụng các kỹ thuật phổ biến để phát hiện Selenium, bao gồm:

- Kiểm tra biến JavaScript `navigator.webdriver`.
- Phân tích fingerprint trình duyệt (user-agent, canvas, WebGL, timezone).
- So sánh hành vi người dùng (scroll, click, delay) với hành vi tự nhiên.
- Phát hiện các đặc điểm đặc trưng của ChromeDriver mặc định.

Các dấu hiệu này đủ để phân biệt giữa người dùng thật và trình duyệt tự động.

Undetected ChromeDriver Để khắc phục vấn đề trên, hệ thống sử dụng **undetected ChromeDriver**, một biến thể của ChromeDriver được thiết kế nhằm:

- che giấu các đặc trưng của WebDriver,
- mô phỏng hành vi trình duyệt người dùng thật,
- giảm khả năng bị phát hiện bởi các script anti-bot.

Undetected ChromeDriver hoạt động bằng cách:

- vá (patch) các thuộc tính JavaScript nhạy cảm,
- tuỳ chỉnh quá trình khởi tạo Chrome,
- loại bỏ hoặc thay đổi các flag đặc trưng của automation.

Các kỹ thuật anti-detection được áp dụng Trong hệ thống thu thập dữ liệu của **FoodChatbot**, các kỹ thuật sau được áp dụng kết hợp:

- **Ẩn dấu hiệu WebDriver**:
  - Vô hiệu hoá hoặc giả lập `navigator.webdriver`.
  - Điều chỉnh các thuộc tính trình duyệt mặc định.
- **Giả lập User-Agent**:
  - Sử dụng User-Agent hợp lệ của trình duyệt Chrome phổ biến.
  - Tránh các chuỗi User-Agent đặc trưng của automation.
- **Hành vi người dùng giả lập**:
  - Scroll trang theo từng bước nhỏ.
  - Thêm độ trễ ngẫu nhiên giữa các thao tác.
  - Tránh các hành vi truy cập liên tục không nghỉ.
- **Quản lý tần suất truy cập**:
  - Giới hạn số request trên mỗi phiên trình duyệt.
  - Phân tán truy cập theo thời gian.

Kết hợp với kiến trúc song song Kỹ thuật undetected được thiết kế để tương thích với kiến trúc đa tiến trình và đa luồng của hệ thống:

- Mỗi tiến trình có một instance WebDriver độc lập.
- Tránh tái sử dụng fingerprint giữa các worker.
- Giảm nguy cơ bị phát hiện do hành vi đồng loạt.

Đánh đổi và giới hạn Mặc dù undetected WebDriver giúp tăng tỷ lệ thu thập thành công, vẫn tồn tại một số đánh đổi:

- chi phí tài nguyên cao hơn (CPU, RAM),
- thời gian thu thập dài hơn do cần delay ngẫu nhiên,
- không thể đảm bảo tránh phát hiện 100% nếu website nâng cấp cơ chế anti-bot.

Do đó, hệ thống được thiết kế theo hướng *giảm thiểu rủi ro bị phát hiện* thay vì cố gắng vượt qua mọi cơ chế bảo vệ.

Vai trò trong pipeline thu thập dữ liệu Kỹ thuật undetected đóng vai trò then chốt trong pipeline thu thập dữ liệu:

- đảm bảo dữ liệu thu thập đầy đủ và ổn định,
- giảm tỷ lệ lỗi và gián đoạn tiến trình,
- tạo nền tảng dữ liệu đáng tin cậy cho các bước tiền xử lý và xây dựng vector database phía sau.

3.2.3 Chi tiết các giai đoạn thu thập dữ liệu

Giai đoạn 1: Thu thập danh mục món ăn

Mục tiêu của giai đoạn này là xây dựng danh sách đầy đủ các danh mục món ăn làm đầu vào cho các bước thu thập tiếp theo.

- Selenium được sử dụng để tải trang chủ và các menu động.
- Trình duyệt được cấu hình với:
  - User-Agent ngẫu nhiên,

  - vô hiệu hoá các dấu hiệu tự động hoá (AutomationControlled),
  - mô phỏng kích thước màn hình và hành vi người dùng thật.
- Tất cả các liên kết danh mục được trích xuất từ DOM sau khi trang tải hoàn tất.
- Các liên kết không liên quan hoặc nằm trong danh sách cấm (blacklist) được loại bỏ.
- Kết quả được lưu vào các file trung gian (temporary files) trong thư mục dữ liệu để phục vụ cho bước tiếp theo.

Giai đoạn 2: Thu thập liên kết công thức chi tiết

Giai đoạn này chịu trách nhiệm mở rộng từ danh mục sang các trang công thức cụ thể, với yêu cầu xử lý khối lượng lớn URL.

- Mỗi danh mục được xử lý song song bằng cơ chế **đa tiến trình (multiprocessing)**.
- Mỗi tiến trình Selenium độc lập với profile trình duyệt riêng nhằm:
  - tránh chia sẻ trạng thái,
  - giảm nguy cơ bị phát hiện hành vi bất thường.
- Hệ thống hỗ trợ **resume crawling**, cho phép:
  - tiếp tục từ trang phân trang cuối cùng đã xử lý,
  - hoặc từ URL cuối cùng trước khi bị gián đoạn.
- Tất cả liên kết công thức trên mỗi trang danh mục được trích xuất và chuẩn hoá.
- Quá trình thu thập dừng khi:
  - không còn trang mới,
  - hoặc số lỗi liên tiếp vượt quá ngưỡng cho phép.
- Các liên kết được lưu theo từng danh mục hoặc từng trang để giảm thiểu rủi ro mất dữ liệu.

Giai đoạn 3: Thu thập chi tiết từng công thức

Đây là giai đoạn quan trọng nhất, nơi dữ liệu thô được trích xuất và chuẩn hoá thành cấu trúc dữ liệu phục vụ hệ RAG.

- Mỗi URL công thức được xử lý bởi một worker riêng.
- Hệ thống sử dụng **đa luồng (multithreading)** trong mỗi tiến trình để:
  - tăng throughput,
  - tận dụng thời gian chờ I/O của Selenium.
- Các thành phần dữ liệu được trích xuất bao gồm:
  - mô tả tổng quan món ăn,
  - danh sách nguyên liệu,
  - các bước chế biến,
  - thời gian nấu,
  - thông tin dinh dưỡng (nếu có),
  - bình luận và đánh giá từ người dùng.
- Dữ liệu được chuẩn hoá và lưu dưới dạng file `JSON`, với mỗi công thức là một file độc lập.

Cơ chế an toàn và chống phát hiện Để đảm bảo quá trình thu thập ổn định và bền vững, hệ thống áp dụng nhiều biện pháp an toàn:

- Bỏ qua các URL đã thu thập trước đó (duplicate check).
- Tự động khởi động lại trình duyệt khi gặp lỗi không phục hồi.
- Chèn thời gian chờ ngẫu nhiên giữa các thao tác.
- Mô phỏng hành vi người dùng như cuộn trang và chờ tải nội dung.

Tối ưu hoá hiệu năng và độ tin cậy

- Kết hợp đa tiến trình (cho danh mục) và đa luồng (cho công thức).
- Hỗ trợ tạm dừng và tiếp tục thu thập mà không mất tiến trình.
- Ghi nhật ký chi tiết cho từng giai đoạn:
  - số trang đã xử lý,
  - số công thức thu thập được,
  - lỗi phát sinh.
- Sử dụng cơ chế ghi file an toàn (atomic write) để tránh dữ liệu bị hỏng khi hệ thống dừng đột ngột.

Toàn bộ dữ liệu sau khi thu thập được sử dụng làm đầu vào cho giai đoạn tiền xử lý, nhúng vector và xây dựng cơ sở dữ liệu phục vụ kiến trúc Retrieval-Augmented Generation.

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

Trong kiến trúc Retrieval-Augmented Generation (RAG) của hệ thống **FoodChatbot**, vector database đóng vai trò là **bộ nhớ ngữ nghĩa ngoài (external semantic memory)**, chịu trách nhiệm lưu trữ, tổ chức và truy xuất tri thức theo không gian vector.

Thay vì để mô hình sinh ngôn ngữ (LLM) dựa hoàn toàn vào tri thức nội tại, ChromaDB cho phép hệ thống truy xuất các đoạn thông tin có liên quan một cách có kiểm soát, từ đó cung cấp ngữ cảnh chính xác cho quá trình sinh câu trả lời và giảm thiểu hiện tượng sinh thông tin sai lệch (hallucination).

3.4.1 Vai trò của ChromaDB trong hệ thống FoodChatbot

Trong hệ thống FoodChatbot, ChromaDB đảm nhiệm các vai trò chính sau:

- Lưu trữ embedding biểu diễn ngữ nghĩa của các công thức nấu ăn.
- Thực hiện truy vấn tương đồng ngữ nghĩa (semantic similarity search).
- Cung cấp ngữ cảnh đầu vào đáng tin cậy cho mô hình LLM.
- Tách biệt hoàn toàn tri thức (data) khỏi mô hình sinh ngôn ngữ (model).

Nhờ kiến trúc này, việc cập nhật hoặc mở rộng tri thức chỉ yêu cầu cập nhật dữ liệu trong ChromaDB mà không cần huấn luyện lại mô hình LLM.

3.4.2 Các thành phần kiến trúc chính của ChromaDB

ChromaDB được thiết kế theo kiến trúc *dataflow-driven*, tối ưu cho luồng xử lý dữ liệu từ văn bản sang vector và từ vector sang ngữ cảnh. Các thành phần cốt lõi bao gồm:

- **Embedding Layer**: chuyển đổi văn bản thành vector ngữ nghĩa.
- **Vector Store**: lưu trữ các vector embedding.
- **Indexing Engine (ANN)**: xây dựng chỉ mục phục vụ truy vấn nhanh.
- **Metadata Store**: lưu trữ thông tin cấu trúc đi kèm mỗi vector.
- **Query Engine**: điều phối toàn bộ quá trình truy vấn.

Các thành phần này được tách biệt rõ ràng nhằm đảm bảo khả năng mở rộng, bảo trì và thay thế từng phần mà không ảnh hưởng toàn bộ hệ thống.

3.4.3 Mô hình embedding và không gian vector

Trong hệ thống, ChromaDB sử dụng mô hình nhúng từ:

`all-MiniLM-L6-v2`

Mô hình này có các đặc điểm:

- Kiến trúc Transformer nhẹ, hiệu quả.
- Vector đầu ra có kích thước cố định $d = 384$.
- Tối ưu cho các tác vụ semantic search.

Với mỗi đoạn văn bản x_i, embedding được sinh:

$$d_i = \text{Embed}(x_i), \quad d_i \in \mathbb{R}^{384}$$

Tất cả embedding của dữ liệu và truy vấn đều nằm trong cùng một không gian vector, đảm bảo khả năng so sánh ngữ nghĩa.

3.4.4 Chiến lược chia tài liệu (Document Chunking Strategy)

Trong hệ thống **FoodChatbot**, chiến lược chia tài liệu được thiết kế theo nguyên tắc:

Mỗi món ăn tương ứng với đúng một document chunk

Tức là toàn bộ thông tin liên quan đến một công thức nấu ăn — bao gồm mô tả, nguyên liệu, các bước chế biến, thời gian nấu và thông tin dinh dưỡng — được gộp thành một đơn vị văn bản duy nhất trước khi đưa vào ChromaDB.

Cấu trúc nội dung của một chunk

Mỗi chunk đại diện cho một món ăn có cấu trúc văn bản chuẩn hoá như sau:

- **Tên và mô tả món ăn**
- **Danh sách nguyên liệu**
- **Các bước chế biến**
- **Thời gian chuẩn bị và nấu**
- **Thông tin dinh dưỡng (nếu có)**

Nội dung này được nối lại thành một chuỗi văn bản liền mạch, đảm bảo giữ được ngữ cảnh tổng thể của món ăn khi sinh embedding.

Lý do lựa chọn chiến lược một món ăn – một chunk

Việc không chia nhỏ công thức thành nhiều chunk con được đưa ra dựa trên các cân nhắc sau:

- **Tính toàn vẹn ngữ nghĩa**: Một công thức nấu ăn là một đơn vị tri thức hoàn chỉnh. Việc chia nhỏ có thể làm mất mối liên hệ giữa nguyên liệu và các bước chế biến.
- **Đặc thù truy vấn của người dùng**: Người dùng thường đặt câu hỏi xoay quanh *toàn bộ món ăn* (ví dụ: "cách nấu", "nguyên liệu", "thời gian nấu"), thay vì từng phần nhỏ riêng lẻ.
- **Giảm độ phức tạp hệ thống**: Chiến lược này đơn giản hoá việc quản lý ID, metadata và ánh xạ ngược từ vector sang dữ liệu gốc.
- **Độ dài văn bản phù hợp**: Độ dài trung bình của một công thức nằm trong giới hạn mà mô hình embedding `all-MiniLM-L6-v2` có thể xử lý hiệu quả mà không làm suy giảm chất lượng biểu diễn.

Ảnh hưởng đến không gian vector

Với chiến lược này, mỗi vector embedding:

$$d_i = \text{Embed}(x_i)$$

biểu diễn **ngữ nghĩa tổng hợp của toàn bộ món ăn**, thay vì một thành phần cục bộ.

Điều này khiến truy vấn trong không gian vector mang tính:

- tổng quát hơn,
- phù hợp với các câu hỏi tự nhiên của người dùng,
- ưu tiên truy xuất đúng món ăn thay vì từng đoạn nhỏ rời rạc.

Tác động đến pipeline truy vấn

Trong pha truy vấn, ChromaDB trả về top-k món ăn có embedding gần nhất với truy vấn người dùng. Mỗi kết quả trả về tương ứng với:

- một công thức hoàn chỉnh,
- đầy đủ thông tin để cung cấp ngữ cảnh cho mô hình LLM,
- không cần bước hợp nhất nhiều chunk con.

Thiết kế này giúp giảm độ phức tạp hậu xử lý và đảm bảo tính nhất quán của ngữ cảnh đầu vào cho mô hình sinh ngôn ngữ.

Giới hạn của chiến lược

Chiến lược một chunk cho mỗi món ăn cũng có một số giới hạn nhất định:

- Độ chi tiết truy xuất thấp hơn so với chia nhỏ theo bước nấu.
- Không phù hợp cho các truy vấn cực kỳ cụ thể (ví dụ: chỉ hỏi một bước chế biến).

Tuy nhiên, trong phạm vi mục tiêu của hệ thống FoodChatbot, các giới hạn này là chấp nhận được và được đánh đổi để lấy tính đơn giản và ổn định của hệ thống.

3.4.5 Kiến trúc hoạt động chi tiết của ChromaDB

Toàn bộ vòng đời dữ liệu trong ChromaDB được chia thành hai pha chính:

- **Pha xây dựng dữ liệu (Ingestion / Indexing Time)**
- **Pha truy vấn (Query Time)**

Hai pha này sử dụng chung hạ tầng lưu trữ và chỉ mục, nhưng có luồng xử lý và mục tiêu tối ưu khác nhau.

Pha 1: Ingestion Pipeline

Bước 1: Nhận dữ liệu văn bản Dữ liệu đầu vào là các document chunk đã được tiền xử lý và chuẩn hoá.

Bước 2: Sinh embedding Mỗi chunk x_i được ánh xạ sang vector:

$$d_i = \text{Embed}(x_i)$$

Bước 3: Gán định danh và metadata Mỗi embedding được gán một ID duy nhất, liên kết với document gốc và metadata.

Bước 4: Lưu trữ và lập chỉ mục Embedding được đưa vào vector store và indexing engine để xây dựng chỉ mục ANN. ChromaDB hỗ trợ cập nhật incremental, không cần tái xây dựng toàn bộ chỉ mục.

Pha 2: Query Pipeline

Bước 1: Sinh embedding truy vấn

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

Bước 2: Truy vấn ANN Indexing engine tìm top-k vector gần nhất với q:

$$\{d_{i_1}, d_{i_2}, \ldots, d_{i_k}\}$$

Bước 3: Tính độ tương đồng Độ tương đồng được tính bằng cosine similarity:

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

Bước 4: Ánh xạ ngược sang document Các vector top-k được ánh xạ ngược sang document và metadata tương ứng. Tại đây, vector chỉ đóng vai trò **chỉ mục ngữ nghĩa**, còn nội dung thực tế được lấy từ document store.

3.4.6 Cấu trúc dữ liệu trong ChromaDB

```
{
    "id": "recipe_001_chunk_01",
    "embedding": [...],
    "document": "Summary: ... | Ingredients: ... |
                 Instructions: ... | Prep: ...min",
    "metadata": {
        "url": "...",
        "recipe_name": "...",
        "prep_time": 0,
        "cook_time": 0,
        "servings": 0,
        "nutr_val_{nutrient}": 0,
        "nutr_unit_{nutrient}": "...",
        "source": "..."
    }
}
```

Metadata cho phép kết hợp truy vấn vector với truy vấn có điều kiện.

3.4.7 Ý nghĩa kiến trúc đối với hệ thống FoodChatbot

Nhờ kiến trúc trên, ChromaDB:

- đóng vai trò bộ nhớ ngữ nghĩa ngoài cho LLM,
- cung cấp tri thức chính xác, có kiểm soát,
- đảm bảo tính linh hoạt và độ tin cậy của hệ thống RAG.

3.5 Mở rộng truy vấn RAG cho dữ liệu định lượng và ràng buộc số liệu

Mặc dù truy vấn ngữ nghĩa dựa trên embedding mang lại khả năng tìm kiếm linh hoạt và gần với cách con người diễn đạt, phương pháp này vẫn tồn tại những hạn chế nghiêm trọng trong các trường hợp truy vấn có chứa **ràng buộc định lượng chính xác**. Đặc biệt trong lĩnh vực ẩm thực và dinh dưỡng, người dùng thường đưa ra các yêu cầu dưới dạng số liệu cụ thể như lượng calo, hàm lượng protein, chất béo hoặc thời gian nấu.

Trong các tình huống này, việc chỉ dựa vào độ tương đồng cosine giữa embedding truy vấn và embedding tài liệu là **không đủ** để đảm bảo tính chính xác của kết quả truy xuất.

Hạn chế của truy vấn thuần ngữ nghĩa

Embedding ngữ nghĩa có đặc điểm:

- mã hoá ý nghĩa tổng quát của văn bản,
- không đảm bảo bảo toàn quan hệ thứ tự hay bất đẳng thức số học,
- không phù hợp để so sánh chính xác các giá trị định lượng.

Ví dụ, hai công thức có mô tả ngữ nghĩa rất giống nhau nhưng:

- một món có 350 kcal,
- món còn lại có 700 kcal,

vẫn có thể nằm gần nhau trong không gian vector. Điều này dẫn đến việc hệ thống có thể trả về kết quả *ngữ nghĩa đúng* nhưng *số liệu sai* so với yêu cầu người dùng.

Nhu cầu truy vấn lai: ngữ nghĩa + cấu trúc

Để khắc phục hạn chế trên, hệ thống FoodChatbot được thiết kế theo hướng **truy vấn lai (hybrid retrieval)**, kết hợp:

- truy vấn ngữ nghĩa trên không gian embedding,
- truy vấn có điều kiện trên metadata dạng số.

Trong kiến trúc này, truy vấn người dùng được phân tách thành hai thành phần:

- phần biểu diễn ý định và chủ đề (semantic intent),
- phần ràng buộc định lượng và điều kiện lọc (structured constraints).

Vai trò của LLM trong trích xuất truy vấn có cấu trúc

Mô hình ngôn ngữ lớn (LLM) được sử dụng như một tầng phân tích trung gian, có nhiệm vụ:

- hiểu truy vấn tự nhiên của người dùng,
- phát hiện các điều kiện mang tính định lượng,
- chuyển đổi các điều kiện này thành biểu diễn truy vấn có cấu trúc.

Thay vì để LLM trả lời trực tiếp, hệ thống yêu cầu LLM sinh ra một biểu diễn truy vấn trung gian, phản ánh đầy đủ:

- nội dung cần tìm kiếm về mặt ngữ nghĩa,
- các ràng buộc số liệu cần được thoả mãn tuyệt đối.

Biểu diễn trung gian này được thiết kế để có thể ánh xạ trực tiếp sang các tham số truy vấn của ChromaDB.

Tương tác giữa LLM và ChromaDB trong truy vấn lai

Trong pipeline truy vấn mở rộng, luồng xử lý được thực hiện như sau:

1. LLM phân tích truy vấn và tách các yêu cầu ngữ nghĩa khỏi các điều kiện định lượng.
2. Phần ngữ nghĩa được dùng để sinh embedding truy vấn.
3. Phần điều kiện định lượng được ánh xạ sang các tham số lọc metadata của ChromaDB.
4. ChromaDB thực hiện tìm kiếm cosine similarity trên tập dữ liệu đã được lọc theo các điều kiện số liệu.

Cách tiếp cận này đảm bảo rằng:

- mọi kết quả trả về đều thoả mãn các ràng buộc số học,
- việc xếp hạng kết quả vẫn dựa trên mức độ tương đồng ngữ nghĩa.

Lợi ích kiến trúc của cách tiếp cận

Việc sử dụng LLM để trích xuất truy vấn có cấu trúc mang lại nhiều lợi ích quan trọng:

- **Tính chính xác**: các điều kiện số liệu được xử lý bằng so sánh logic thay vì suy diễn ngữ nghĩa.
- **Tính linh hoạt**: người dùng không cần học cú pháp truy vấn cứng.
- **Khả năng mở rộng**: dễ dàng bổ sung thêm các loại ràng buộc mới trong tương lai.
- **Tách biệt trách nhiệm**: LLM chỉ đảm nhiệm hiểu ngôn ngữ, trong khi ChromaDB chịu trách nhiệm truy vấn dữ liệu.

Ý nghĩa đối với hệ thống FoodChatbot

Trong bối cảnh FoodChatbot, cơ chế truy vấn lai giúp hệ thống:

- đáp ứng chính xác các yêu cầu dinh dưỡng cá nhân hoá,
- tránh các câu trả lời sai lệch do embedding không phản ánh đúng số liệu,
- nâng cao độ tin cậy khi sử dụng trong các kịch bản liên quan đến sức khoẻ.

Cách thiết kế này cho phép hệ thống kết hợp ưu điểm của truy vấn ngữ nghĩa và truy vấn cấu trúc, tạo thành một pipeline Retrieval-Augmented Generation vừa linh hoạt về ngôn ngữ, vừa chặt chẽ về mặt dữ liệu.

3.6 Chi tiết từng giai đoạn trong pipeline RAG

Quy trình Retrieval-Augmented Generation trong FoodChatbot được thiết kế theo mô hình pipeline tuyến tính, trong đó mỗi bước đảm nhiệm một vai trò cụ thể và có thể kiểm soát được. Việc giữ nguyên thứ tự và chức năng của từng bước giúp hệ thống ổn định, dễ mở rộng và dễ phân tích lỗi.

Giai đoạn 1: Tiếp nhận và chuẩn hoá truy vấn đa ngôn ngữ Việc tiếp nhận truy vấn Q ở dạng đa ngôn ngữ phản ánh kịch bản sử dụng thực tế của người dùng. Tuy nhiên, để đảm bảo tính nhất quán trong không gian embedding và trong dữ liệu lưu trữ, truy vấn cần được chuẩn hoá về một ngôn ngữ chung (trong hệ thống này là tiếng Anh).

Mô hình Gemini được sử dụng ở giai đoạn này không nhằm sinh câu trả lời, mà đóng vai trò:

- nhận diện ngôn ngữ đầu vào,
- dịch chính xác về ngôn ngữ chuẩn,
- giữ nguyên ý nghĩa ngữ cảnh của truy vấn.

Điều này giúp tránh việc embedding các truy vấn cùng ý nghĩa nhưng khác ngôn ngữ vào các vùng vector khác nhau.

Giai đoạn 2: Trích xuất sở thích và ngữ cảnh người dùng Tập S đại diện cho các yếu tố mang tính định hướng trả lời, bao gồm:

- khẩu vị,
- nguyên liệu ưa thích hoặc cần tránh,
- yêu cầu dinh dưỡng hoặc tình trạng sức khoẻ.

Các thông tin này không được dùng trực tiếp cho truy vấn vector database, mà được sử dụng ở giai đoạn sinh câu trả lời để:

- cá nhân hoá phản hồi,
- điều chỉnh cách diễn đạt của LLM,
- bổ sung các lưu ý phù hợp cho người dùng.

Việc tách riêng S giúp hệ thống không làm nhiễu không gian truy vấn embedding.

Giai đoạn 3: Trích xuất bộ lọc thông số dinh dưỡng Tập F chứa các ràng buộc mang tính định lượng, ví dụ như giới hạn calo, chất béo hoặc thời gian nấu. Đây là bước then chốt để chuyển từ truy vấn ngữ nghĩa thuần sang truy vấn lai.

Các ràng buộc này được:

- trích xuất bởi Gemini,
- chuẩn hoá về dạng điều kiện logic,
- ánh xạ trực tiếp sang metadata trong ChromaDB.

Việc xử lý các ràng buộc số liệu bên ngoài không gian embedding giúp đảm bảo tính chính xác tuyệt đối của kết quả truy xuất.

Giai đoạn 4: Sinh embedding truy vấn Vector V được sinh từ truy vấn đã chuẩn hoá Q' đại diện cho ý định ngữ nghĩa tổng quát của người dùng. Embedding này:

- không chứa thông tin số liệu cụ thể,
- được dùng để so sánh cosine similarity,
- quyết định thứ tự xếp hạng kết quả.

Cách tiếp cận này giúp hệ thống ưu tiên đúng chủ đề, trong khi vẫn giữ được khả năng lọc chính xác theo điều kiện.

Giai đoạn 5: Truy vấn ChromaDB theo cơ chế lai Bước truy vấn $D = \text{Retrieve}(V, F, k)$ được thực hiện theo hai tầng:

- lọc dữ liệu dựa trên metadata theo F,
- xếp hạng ngữ nghĩa dựa trên cosine similarity giữa V và embedding tài liệu.

Thiết kế này giúp:

- giảm không gian tìm kiếm,
- tăng độ chính xác của kết quả,
- đảm bảo mọi tài liệu trả về đều thoả mãn các ràng buộc số liệu.

Giai đoạn 6: Ghép ngữ cảnh và xây dựng prompt Tập tài liệu D được ghép thành ngữ cảnh có cấu trúc trước khi đưa vào prompt. Đây là bước quan trọng nhằm:

- cung cấp tri thức có kiểm soát cho LLM,
- hạn chế hiện tượng hallucination,
- đảm bảo phản hồi dựa trên dữ liệu thực.

Prompt được xây dựng theo hướng tách bạch giữa hướng dẫn hệ thống, ngữ cảnh truy xuất và câu hỏi người dùng.

Giai đoạn 7: Sinh phản hồi và hậu xử lý Ở bước này, Gemini thực hiện sinh phản hồi dựa trên:

- ngữ cảnh đã truy xuất,
- sở thích người dùng S,
- hướng dẫn hội thoại của hệ thống.

Việc dịch ngược phản hồi về ngôn ngữ gốc của người dùng đảm bảo trải nghiệm tự nhiên, trong khi vẫn giữ được tính nhất quán của pipeline nội bộ.

Nhận xét tổng quát về kiến trúc pipeline

Pipeline RAG của FoodChatbot có các đặc điểm nổi bật:

- Luồng xử lý rõ ràng, không vòng lặp ngầm.
- LLM được sử dụng như bộ phân tích và sinh ngôn ngữ, không can thiệp trực tiếp vào truy vấn dữ liệu.
- Truy vấn ngữ nghĩa và truy vấn định lượng được xử lý ở hai không gian riêng biệt nhưng phối hợp chặt chẽ.

Thiết kế này giúp hệ thống vừa linh hoạt về ngôn ngữ, vừa đảm bảo tính chính xác và khả năng kiểm soát trong các kịch bản thực tế.

3.7 Kiến trúc tổng thể Backend – Frontend

Hệ thống FoodChatbot được triển khai theo mô hình kiến trúc client–server, trong đó backend và frontend được tách biệt rõ ràng về trách nhiệm. Cách tiếp cận này giúp hệ thống dễ bảo trì, dễ mở rộng và phù hợp với các kịch bản triển khai thực tế.

- **Frontend** đảm nhiệm toàn bộ giao diện người dùng và tương tác.
- **Backend** đóng vai trò trung gian, xử lý logic nghiệp vụ và điều phối pipeline RAG.
- **Chatbot Core** chịu trách nhiệm hiểu truy vấn, truy xuất tri thức và sinh phản hồi.

Luồng dữ liệu được thiết kế theo hướng một chiều, từ người dùng đến backend, sau đó đi vào pipeline xử lý và trả kết quả ngược lại, giúp giảm độ phức tạp và tránh các trạng thái không nhất quán.

3.7.1 Thiết kế Backend dựa trên Flask

Flask được lựa chọn làm nền tảng backend do các đặc điểm:

- nhẹ, dễ triển khai và phù hợp với ứng dụng AI quy mô vừa,
- hỗ trợ tốt REST API,
- dễ tích hợp với các module Python xử lý NLP và RAG.

Backend không thực hiện bất kỳ xử lý ngôn ngữ hoặc truy vấn tri thức phức tạp nào một cách trực tiếp, mà chỉ đóng vai trò *orchestrator* cho pipeline chatbot.

Tách biệt tầng Web Server và Chatbot Logic

Trong kiến trúc hiện tại:

- Flask chịu trách nhiệm tiếp nhận request và trả response.
- Lớp `Chatbot` chịu trách nhiệm toàn bộ logic xử lý RAG.

Sự phân tách này đảm bảo:

- backend có thể thay thế hoặc mở rộng mà không ảnh hưởng đến chatbot,
- chatbot có thể tái sử dụng trong các giao diện khác (CLI, mobile app).

3.7.2 Quản lý hội thoại và ngữ cảnh

Mỗi phiên trò chuyện của người dùng được quản lý thông qua session của Flask. Cơ chế này cho phép:

- duy trì lịch sử hội thoại,
- giữ ngữ cảnh xuyên suốt nhiều lượt hỏi–đáp,
- hỗ trợ reset hội thoại khi cần.

Việc quản lý ngữ cảnh ở tầng backend giúp LLM không cần lưu trạng thái dài hạn, giảm chi phí tính toán và tăng khả năng kiểm soát.

3.7.3 Thiết kế API và giao tiếp Frontend

Các API được thiết kế theo phong cách RESTful, sử dụng JSON làm định dạng trao đổi dữ liệu. Thiết kế này mang lại:

- tính đơn giản,
- khả năng mở rộng sang các client khác,
- dễ dàng kiểm thử và debug.

Đặc biệt, endpoint `/api/chat` được thiết kế như một điểm vào duy nhất (single entry point) cho toàn bộ pipeline xử lý chatbot, giúp đơn giản hoá frontend và tập trung logic xử lý ở backend.

3.7.4 Thiết kế Frontend và trải nghiệm người dùng

Giao diện frontend được xây dựng theo hướng tối giản nhưng trực quan, phù hợp với mô hình chatbot hội thoại.

Các quyết định thiết kế chính bao gồm:

- hiển thị hội thoại theo dạng bong bóng để phân biệt người dùng và chatbot,
- hỗ trợ đa ngôn ngữ ngay từ đầu vào,
- hiển thị nguồn tham chiếu để tăng độ tin cậy của câu trả lời.

Việc bổ sung chức năng reset hội thoại giúp người dùng dễ dàng bắt đầu một phiên làm việc mới mà không bị ảnh hưởng bởi ngữ cảnh cũ.

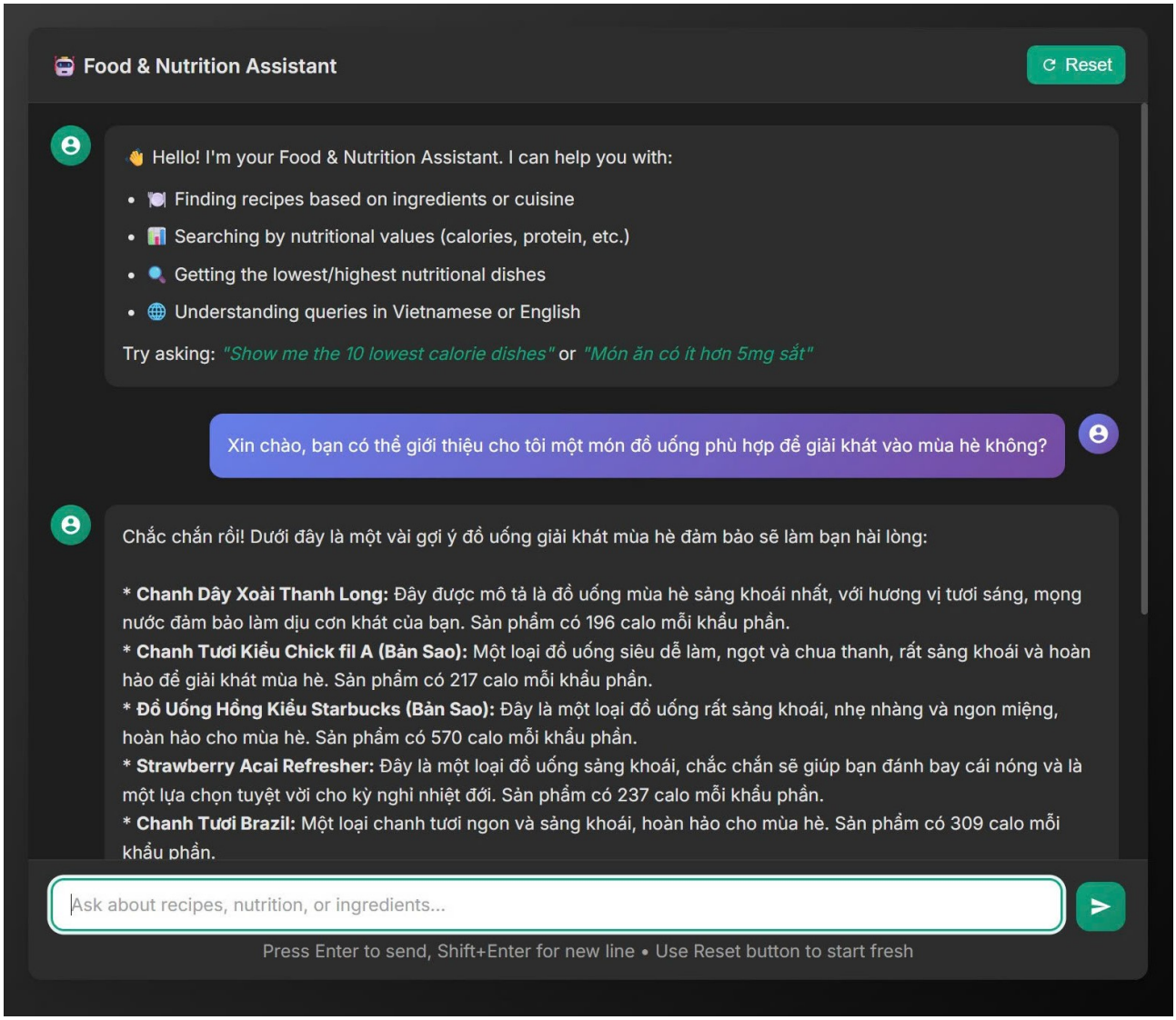
3.7.5 Đánh giá kiến trúc triển khai

Kiến trúc triển khai hiện tại đáp ứng tốt các yêu cầu của hệ thống FoodChatbot:

- dễ triển khai trên môi trường local và server,
- phù hợp cho thử nghiệm và mở rộng,
- đủ linh hoạt để tích hợp thêm các mô hình hoặc dịch vụ bên ngoài.

Tuy nhiên, trong các kịch bản mở rộng quy mô lớn, hệ thống có thể được cải tiến bằng cách:

- tách backend thành các service độc lập,
- bổ sung cơ chế xác thực và phân quyền,
- triển khai caching cho các truy vấn phổ biến.



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:

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

- 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.

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

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

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

- 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**	
	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**:

$$\text{OneShotAcc} = 7.6825$$

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

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
```