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: MAT1207E
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: [Ngày nộp] (ví dụ: 30/06/2025)
Báo cáo PDF: Liên kết tới báo cáo PDF trong kho GitHub
Slide thuyết trình: Liên kết tới slide thuyết trình trong kho GitHub
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 embedding và lưu trữ trong vector database 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 từ nhiều nguồn.
- Mã hóa tri thức bằng các vector embedding 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 & Triển khai

2.1 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.

2.1.1 Cách 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 (crawler). 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 embedding**: Văn bản sau khi được chuẩn hóa được đưa vào mô hình embedding 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 embedding 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 embedding cho truy vấn, sau đó truy xuất các vector gần nhất trong không gian embedding. 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.

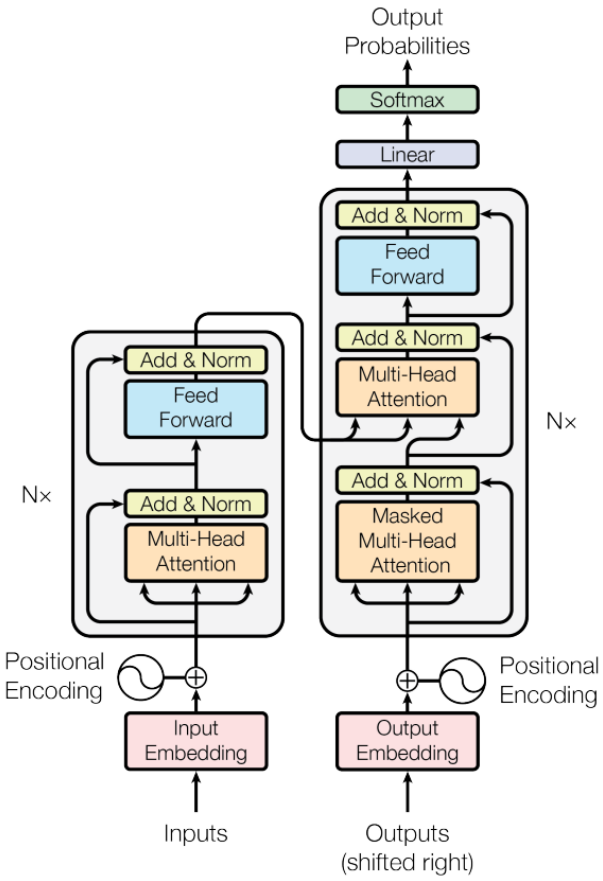
Với pipeline RAG, dự án chatbot ẩm thực đạt được sự cân bằng giữa khả năng sinh ngôn ngữ tự nhiên của LLM và độ chính xác trong cung cấp tri thức thực tế. Điều này cho phép chatbot không chỉ trả lời câu hỏi về công thức nấu ăn, nguyên liệu hay dinh dưỡng, mà còn đưa ra gợi ý tùy chỉnh theo sở thích cá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

2.1.2 Cơ sở lý thuyết

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:

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

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

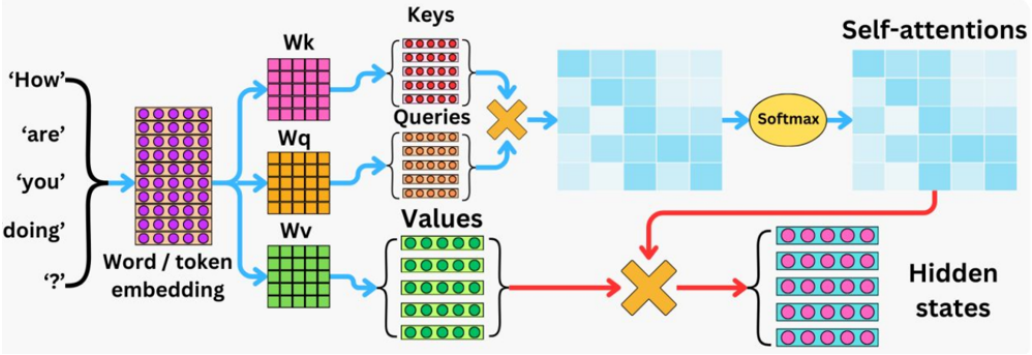
$$\text{score}(i, j) = \frac{q_i \cdot k_j}{\sqrt{d_k}},$$

sau đó áp dụng softmax để có trọng số attention:

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

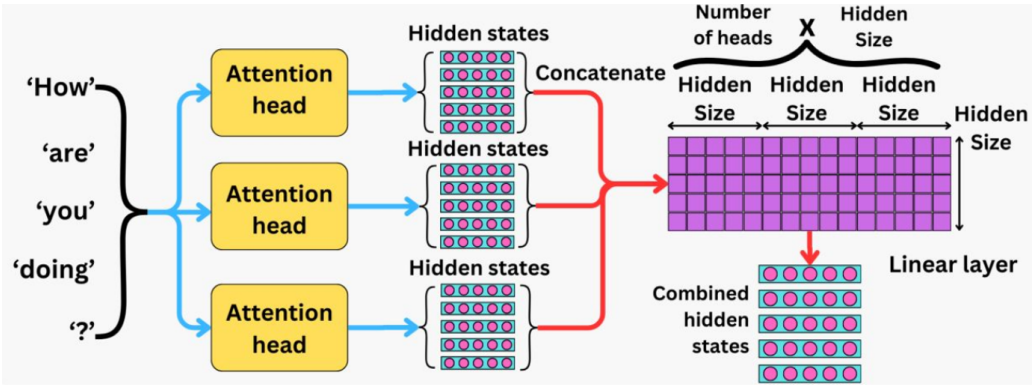
Biểu diễn đầu ra cho token i là tổng có trọng số các value:

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



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:

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

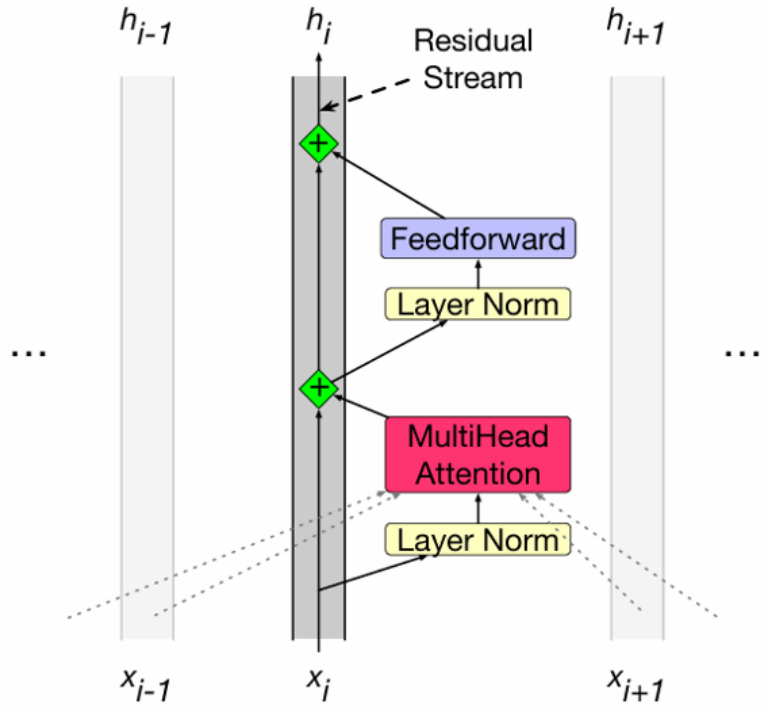
Các lớp này tạo ra biểu diễn phi tuyến mạnh hơn cho từng token.

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 $\mathrm{SubLayer}(x)$. Residual connection được định nghĩa:

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

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 / embedding được thống nhất), sau đó sinh phản hồi bằng ngôn ngữ người dùng. Việc này tối ưu hóa độ tương đồng ngữ nghĩa khi embedding 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 (FLOPs) 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.

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:

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

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

Vector Database và Embedding

Ý tưởng embedding Embedding 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. ChromaDB cung cấp pipeline để sinh embedding cho từng document và lưu trữ chúng cùng metadata.

Cách cấu trúc 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 để truy vấn hiệu quả.
- **Dễ cập nhật tri thức**: chỉ cần thêm/sửa document mà không phải fine-tune LLM.

Cosine Similarity và các chỉ số đánh giá tương đồng

Định nghĩa cosine similarity[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 document có cosine cao nhất so với embedding của truy vấn.

Thuộc tính và ngưỡng

- Giá trị nằm trong $[-1, 1]$, nhưng với embedding từ LLM/public models thường nằm trong $[0, 1]$ do cấu trúc embedding.
- Thực tiễn: đặt ngưỡng lọc (ví dụ 0.65–0.75) để loại bỏ document 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 document (k thường trong khoảng 3–10).

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 kho dữ liệu ẩm thực.
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.

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 sentence ngắn (summary) 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]`.

2.1.3 Thuật toán 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 Gemini và khả năng truy xuất tri thức theo ngữ nghĩa từ ChromaDB. 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 món ăn, nguyên liệu hoặc gợi ý nấu nướng.
- Truy vấn được đưa vào mô-đun sử dụng Gemini để đảm nhiệm việc:
  - Phát hiện ngôn ngữ đầu vào.
  - Dịch truy vấn sang tiếng Anh nếu cần (để tương thích với dữ liệu embedding trong DB).
  - Chuẩn hóa và loại bỏ nhiễu trong câu hỏi.

2. Sinh embedding cho truy vấn

- Truy vấn tiếng Anh sẽ được đưa qua API embedding của ChromaDB:

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

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

3. Truy vấn ngữ nghĩa (Semantic Retrieval)

- ChromaDB lưu các embedding của tài liệu ẩm thực.
- Hệ thống thực hiện tìm kiếm top-k đoạn gần nhất:

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

- Độ tương đồng được tính bằng cosine similarity:

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

- Các đoạn văn liên quan nhất được trả về kèm metadata: 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 Gemini không sinh thông tin sai (hallucination).

5. Sinh phản hồi (Answer Generation)

- Prompt hoàn chỉnh được gửi vào mô hình Gemini.
- Gemini sinh câu trả lời bằng ngôn ngữ tự nhiên tiếng Anh, sau khi:
  - Kết hợp tri thức từ context đã truy xuất.
  - Giữ đúng ngôn ngữ đầu vào của người dùng.
  - 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.

6. Pipeline tổng thể

1. Nhận câu hỏi Q.
2. Chuẩn hoá và dịch về tiếng Anh:
$$Q' = \text{Normalize}(Q)$$
3. Sinh embedding truy vấn:
$$v = \text{Embed}(Q')$$
4. Truy xuất top-k đoạn liên quan:
$$D = \text{Retrieve}(v, k)$$
5. Ghép ngữ cảnh:
$$P = \text{ConstructPrompt}(D, Q)$$
6. Gửi prompt vào Gemini:
$$A = \text{Gemini.generate}(P)$$
7. Nếu ngôn ngữ đầu vào không phải tiếng Anh, dịch phản hồi về ngôn ngữ gốc.
$$A' = \text{Translate}(A, \text{lang}(Q))$$
8. Trả về phản hồi A' cho người dùng.

2.1.4 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 embedding.

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, thường do tác giả cung cấp.
- **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.

Lưu trữ và 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 embedding và indexing.
- **Lưu trữ dữ liệu cuối:** Dữ liệu đã xử lý được hợp nhất thành một tệp JSON duy nhất, sẵn sàng cho các bước embedding, 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.

2.2 Triển khai

Phần này mô tả chi tiết toàn bộ quá trình triển khai hệ thống FoodChatbot, trong đó quy trình thu thập dữ liệu (crawling) và tiền xử lý (preprocessing) được đưa lên đầu tiên, vì đây là nền tảng tạo nên kho tri thức cho hệ thống RAG.

2.2.1 Thu thập dữ liệu

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

Thu thập danh mục

- **Công nghệ:** Selenium kết hợp trình duyệt tự động (Chrome/undetected driver) để tương tác với trang web động.
- 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.

2.2.2 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 Python để làm sạch, chuẩn hóa và cấu trúc dữ liệu trước khi lưu trữ hoặc đưa vào pipeline RAG.

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": "..."}
    ]
}
```

2.2.3 Pipeline tổng thể của hệ thống

Sau khi dữ liệu đã được crawl và chuẩn hoá, hệ thống FoodChatbot vận hành theo pipeline:

1. Nhận truy vấn đầu vào (đa ngôn ngữ).
2. Chuẩn hoá truy vấn bằng Gemini (dịch sang tiếng Anh nếu cần).
3. Sinh embedding truy vấn.
4. Thực hiện semantic search trong ChromaDB.
5. Chọn top-k đoạn liên quan nhất.
6. Ghép prompt (retrieved context + user query).
7. LLM sinh phản hồi theo ngôn ngữ người dùng.
8. Gửi kết quả + nguồn trích dẫn về frontend.

2.2.4 Retrieval-Augmented Generation (RAG)

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).

Quy trình RAG chi tiết

1. Người dùng nhập truy vấn.
2. Gemini chuẩn hóa truy vấn về tiếng Anh.
3. Tạo embedding truy vấn.
4. Truy xuất top-k đoạn từ ChromaDB.
5. Xây dựng prompt đầy đủ.
6. Gemini sinh phản hồi cuối.

2.2.5 Vector Database: ChromaDB

ChromaDB được sử dụng để lưu trữ:

- Embedding của các đoạn tài liệu.
- Doc-text (nội dung).
- 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, Cook: ...min",
    "metadata": {
        "url": "...",
        "prep_time": 0,
        "cook_time": 0,
        "servings": 0,
        "nutr_val_{nutrient}": 0,
        "nutr_unit_{nutrient}": "...",
    }
}
```

2.2.6 Tích hợp Large Language Model (Gemini)

Gemini thực hiện hai nhiệm vụ chính:

1. Normalization: chuẩn hóa truy vấn và dịch về tiếng Anh.
2. Final Answer Generation: sinh câu trả lời cuối.

2.2.7 Triển khai Backend với Flask

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:
   (a) Chuẩn hóa và dịch truy vấn sang tiếng Anh bằng mô hình ngôn ngữ.
   (b) Sinh embedding và truy xuất các đoạn tài liệu liên quan từ cơ sở dữ liệu vector.
   (c) Chuẩn hóa và chèn ngữ cảnh vào prompt theo cơ chế context injection.

(d) Mô hình ngôn ngữ sinh ra phản hồi cuối dựa trên ngữ cảnh được cung cấp.

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.

Chương 3

Kết quả & Phân tích

3.1 Kết quả & Thảo luận

[Trình bày kết quả chính, các chỉ số đánh giá và phân tích. Sử dụng bảng, hình hoặc biểu đồ nếu cần.]

Chương 4

Kết luận

4.1 Kết luận & Hướng phát triển

[Tóm tắt đóng góp và đề xuất cải tiến hoặc hướng phát triển tiếp theo.]

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

Phụ lục A

Phụ lục