Giáo trình Retrieval-Augmented Generation (RAG)

Giới thiệu

Giáo trình này cung cấp một cái nhìn tổng quan toàn diện về Retrieval-Augmented Generation (RAG), một kỹ thuật tiên tiến kết hợp khả năng truy xuất thông tin từ các nguồn dữ liệu bên ngoài với sức mạnh tạo nội dung của các mô hình ngôn ngữ lớn (LLM). RAG giúp LLM cung cấp các phản hồi chính xác, đáng tin cậy và có căn cứ hơn bằng cách bổ sung kiến thức thực tế từ các nguồn bên ngoài, khắc phục hạn chế về kiến thức tĩnh của LLM.

Giáo trình sẽ đi sâu vào các thành phần chính của RAG, bao gồm quá trình truy xuất (Retrieval) và quá trình tạo sinh (Generation), cùng với các kỹ thuật tối ưu hóa và những thách thức thường gặp. Ngoài ra, giáo trình còn bao gồm một dự án thực hành để học viên có thể áp dụng kiến thức đã học vào thực tế.

Mục lục

1. Giới thiệu về RAG và các thành phần chính

1.1. RAG là gì?

Retrieval-Augmented Generation (RAG) là một kỹ thuật tiên tiến trong lĩnh vực Trí tuệ nhân tạo (AI) và Xử lý ngôn ngữ tự nhiên (NLP) nhằm nâng cao khả năng của các mô hình ngôn ngữ lớn (Large Language Models - LLM) bằng cách tích hợp khả năng truy xuất thông tin từ các nguồn dữ liệu bên ngoài. Thay vì chỉ dựa vào kiến thức đã được huấn luyện trong quá trình đào tạo, RAG cho phép LLM tìm kiếm và sử dụng thông tin liên quan từ một kho dữ liệu (knowledge base) để tạo ra các phản hồi chính xác, cập nhật và có căn cứ hơn [1, 2].

Các mô hình LLM truyền thống, mặc dù có khả năng tạo văn bản ấn tượng, thường gặp phải các vấn đề như:

- **"Hallucination" (Ảo giác):** Tạo ra thông tin sai lệch hoặc không có thật.

- **Kiến thức lỗi thời:** Kiến thức của mô hình bị giới hạn bởi dữ liệu huấn luyện, không thể cập nhật thông tin mới nhất.
- **Thiếu khả năng giải thích:** Khó khăn trong việc truy xuất nguồn gốc của thông tin được tạo ra.

RAG ra đời để giải quyết những hạn chế này bằng cách cung cấp cho LLM một cơ chế để "tra cứu" thông tin bên ngoài trước khi tạo ra câu trả lời. Điều này giúp tăng cường độ chính xác, độ tin cậy và khả năng giải thích của các phản hồi do LLM tạo ra [3].

1.2. Tại sao RAG lại quan trọng?

RAG mang lại nhiều lợi ích đáng kể, đặc biệt trong các ứng dụng yêu cầu độ chính xác cao và thông tin cập nhật:

- **Tăng cường độ chính xác và tin cậy:** Bằng cách truy xuất thông tin từ các nguồn đáng tin cậy, RAG giảm thiểu hiện tượng "hallucination" và đảm bảo các phản hồi dựa trên dữ liệu thực tế.
- **Cập nhật kiến thức liên tục:** Mô hình có thể truy cập thông tin mới nhất mà không cần phải huấn luyện lại toàn bộ LLM, giúp hệ thống luôn được cập nhật với dữ liệu mới.
- **Khả năng giải thích (Explainability):** RAG cho phép người dùng hoặc hệ thống kiểm tra các nguồn thông tin đã được sử dụng để tạo ra câu trả lời, tăng tính minh bạch và khả năng giải thích.
- **Giảm chi phí huấn luyện:** Không cần huấn luyện lại LLM với dữ liệu mới, tiết kiệm tài nguyên tính toán và thời gian.
- **Xử lý thông tin chuyên biệt:** Cho phép LLM trả lời các câu hỏi về các lĩnh vực chuyên môn hoặc dữ liệu nội bộ của tổ chức mà LLM chưa từng được huấn luyện.

1.3. Các thành phần chính của hệ thống RAG

Một hệ thống RAG cơ bản thường bao gồm hai thành phần chính hoạt động song song:

1.3.1. Retrieval Module (Mô-đun Truy xuất)

Mô-đun Retrieval chịu trách nhiệm tìm kiếm và truy xuất các đoạn văn bản hoặc tài liệu liên quan từ một kho dữ liệu lớn (knowledge base) dựa trên truy vấn của người dùng. Quá trình này thường bao gồm các bước sau:

1. **Chuyển đổi truy vấn:** Truy vấn của người dùng được chuyển đổi thành một biểu diễn vector (embedding) bằng cách sử dụng một mô hình nhúng (embedding model).
2. **Tìm kiếm tương đồng:** Biểu diễn vector của truy vấn được sử dụng để tìm kiếm các đoạn văn bản có biểu diễn vector tương tự trong kho dữ liệu. Kho dữ liệu này thường được lưu trữ dưới dạng cơ sở dữ liệu vector (vector database) để tăng tốc độ tìm kiếm.
3. **Truy xuất tài liệu:** Các tài liệu hoặc đoạn văn bản có độ tương đồng cao nhất với truy vấn sẽ được truy xuất và chuyển đến mô-đun Generation [4].

1.3.2. Generation Module (Mô-đun Tạo sinh)

Mô-đun Generation, thường là một mô hình ngôn ngữ lớn (LLM), nhận đầu vào là truy vấn ban đầu của người dùng và các đoạn văn bản liên quan được truy xuất từ mô-đun Retrieval. Sau đó, LLM sẽ sử dụng cả hai thông tin này để tạo ra một phản hồi mạch lạc, chính xác và phù hợp với ngữ cảnh [5].

Quá trình này cho phép LLM:

- **Tổng hợp thông tin:** Kết hợp kiến thức nội tại của mô hình với thông tin mới được truy xuất.
- **Tạo văn bản tự nhiên:** Tạo ra câu trả lời bằng ngôn ngữ tự nhiên, dễ hiểu cho người dùng.
- **Cung cấp ngữ cảnh:** Đảm bảo câu trả lời có liên quan trực tiếp đến truy vấn và các tài liệu đã truy xuất.

1.4. Quy trình hoạt động của RAG

Quy trình hoạt động của một hệ thống RAG có thể được tóm tắt như sau:

1. **Người dùng gửi truy vấn:** Người dùng đặt câu hỏi hoặc đưa ra yêu cầu.
2. **Truy vấn được nhúng:** Truy vấn được chuyển đổi thành một vector biểu diễn (query embedding).
3. **Tìm kiếm trong kho kiến thức:** Query embedding được sử dụng để tìm kiếm các đoạn văn bản liên quan nhất trong kho kiến thức (đã được nhúng sẵn).
4. **Truy xuất tài liệu:** Các đoạn văn bản (chunks) có liên quan được truy xuất.
5. **Tạo ngữ cảnh (Context Augmentation):** Truy vấn ban đầu của người dùng và các đoạn văn bản được truy xuất được kết hợp lại để tạo thành một ngữ cảnh đầu vào phong phú cho LLM.
6. **Tạo phản hồi:** LLM nhận ngữ cảnh đã được tăng cường và tạo ra câu trả lời cuối cùng cho người dùng.

1.5. Kết luận

RAG đại diện cho một bước tiến quan trọng trong việc phát triển các hệ thống AI đàm thoại và thông tin. Bằng cách kết hợp sức mạnh của LLM với khả năng truy xuất thông tin hiệu quả, RAG mở ra nhiều ứng dụng mới và cải thiện đáng kể hiệu suất của các hệ thống AI hiện có. Các chương tiếp theo sẽ đi sâu vào từng thành phần của RAG, các kỹ thuật triển khai và những thách thức cần vượt qua.

Tài liệu tham khảo

[1] AWS. (n.d.). *RAG là gì? - Giải thích về AI tạo có kết hợp truy xuất thông tin ngoài*. Retrieved from https://aws.amazon.com/vi/what-is/retrieval-augmented-generation/ [2] Viblo.asia. (n.d.). *Tìm hiểu về Retrieval Augmented Generation (RAG)*. Retrieved from https://viblo.asia/p/chatgpt-series-5-tim-hieu-ve-retrieval-augmented-generation-rag-Ny0VGRd7LPA [3] 200Lab. (2025, March 11). *RAG (Retrieval-Augmented Generation) là gì? Giải thích dễ hiểu cho người mới bắt đầu*. Retrieved from https://200lab.io/blog/rag-la-gi/ [4] Techbase.vn. (2024, November 27). *Retrieval Augmented Generation (RAG) và 10 Thư Viện Mã Nguồn Mở Để Xây Dựng RAG*. Retrieved from https://www.techbasevn.com/blog-technical/retrieval-augmented-generation-rag-va-10-thu-vien-ma-

nguon-mo-de-xay-dung-rag.html [5] FPT.AI. (2025, January 23). *Retrieval-Augmented Generation Cải Thiện Phản Hồi Cho LLMs*. Retrieved from https://fpt.ai/vi/bai-viet/retrieval-augmented-generation/

2. Retrieval (Truy xuất)

2.1. Giới thiệu về Retrieval Module

Mô-đun Retrieval là trái tim của hệ thống RAG, chịu trách nhiệm tìm kiếm và truy xuất các thông tin liên quan từ một kho dữ liệu lớn (knowledge base) để cung cấp ngữ cảnh cho mô hình ngôn ngữ lớn (LLM). Mục tiêu chính của mô-đun này là đảm bảo rằng các tài liệu được truy xuất có chất lượng cao, phù hợp và đa dạng, giúp LLM tạo ra các phản hồi chính xác và toàn diện [1].

Quá trình truy xuất thường bao gồm việc chuyển đổi truy vấn của người dùng và các tài liệu trong kho kiến thức thành các biểu diễn vector (embeddings), sau đó sử dụng các kỹ thuật tìm kiếm tương đồng để xác định các tài liệu phù hợp nhất. Hiệu suất của mô-đun Retrieval ảnh hưởng trực tiếp đến chất lượng đầu ra của toàn bộ hệ thống RAG.

2.2. Các thành phần chính của Retrieval Module

2.2.1. Kho kiến thức (Knowledge Base)

Kho kiến thức là nơi lưu trữ tất cả các tài liệu mà hệ thống RAG có thể truy xuất. Đây có thể là:

- **Cơ sở dữ liệu văn bản:** Các tài liệu dạng văn bản như bài viết, sách, báo cáo, tài liệu kỹ thuật, v.v.
- **Cơ sở dữ liệu có cấu trúc:** Dữ liệu từ các bảng, cơ sở dữ liệu SQL, NoSQL.
- **Dữ liệu đa phương tiện:** Hình ảnh, video, âm thanh (sau khi được chuyển đổi thành văn bản hoặc biểu diễn có thể tìm kiếm).

Để tối ưu hóa quá trình truy xuất, các tài liệu trong kho kiến thức thường được chia nhỏ thành các đoạn (chunks) nhỏ hơn và được chuyển đổi thành các vector nhúng (embeddings) trước khi lưu trữ. Kích thước của các chunk là một yếu tố quan trọng cần cân nhắc, vì chunk quá lớn có thể chứa nhiều thông tin không liên quan, trong khi chunk quá nhỏ có thể làm mất ngữ cảnh [2].

2.2.2. Mô hình nhúng (Embedding Model)

Mô hình nhúng (embedding model) là một thành phần quan trọng có nhiệm vụ chuyển đổi văn bản (truy vấn của người dùng và các đoạn tài liệu trong kho kiến thức) thành các vector số (embeddings) trong một không gian đa chiều. Các vector này được thiết kế sao cho các văn bản có ý nghĩa tương tự sẽ có các vector gần nhau trong không gian nhúng [3].

Các loại embedding model phổ biến:

- **Word Embeddings:** Word2Vec, GloVe, FastText (biểu diễn từng từ).
- **Sentence Embeddings:** Sentence-BERT, Universal Sentence Encoder (biểu diễn cả câu hoặc đoạn văn).

- **Contextual Embeddings:** Các mô hình dựa trên Transformer như BERT, RoBERTa, OpenAI Embeddings (tạo embeddings dựa trên ngữ cảnh của từ trong câu).

Việc lựa chọn mô hình nhúng phù hợp là rất quan trọng, vì nó ảnh hưởng trực tiếp đến khả năng tìm kiếm các tài liệu liên quan của hệ thống.

2.2.3. Cơ sở dữ liệu Vector (Vector Database)

Cơ sở dữ liệu vector (vector database) là một loại cơ sở dữ liệu chuyên dụng được thiết kế để lưu trữ và tìm kiếm hiệu quả các vector nhúng. Thay vì tìm kiếm dựa trên từ khóa truyền thống, vector database cho phép tìm kiếm dựa trên độ tương đồng ngữ nghĩa giữa các vector [4].

Các thuật toán tìm kiếm tương đồng phổ biến bao gồm:

- **Approximate Nearest Neighbor (ANN):** Các thuật toán như HNSW, FAISS, Annoy được sử dụng để tìm kiếm các vector gần nhất một cách hiệu quả trong các tập dữ liệu lớn.
- **Cosine Similarity:** Một độ đo phổ biến để tính toán độ tương đồng giữa hai vector.

Các vector database phổ biến: Pinecone, Weaviate, Milvus, Chroma, Faiss (thư viện tìm kiếm vector).

2.3. Quy trình Retrieval

Quá trình truy xuất trong RAG diễn ra theo các bước sau:

1. **Tiền xử lý và nhúng tài liệu (Offline):**
    - Các tài liệu trong kho kiến thức được thu thập và làm sạch.
    - Tài liệu được chia thành các đoạn (chunks) nhỏ hơn.
    - Mỗi chunk được chuyển đổi thành một vector nhúng bằng embedding model.
    - Các vector nhúng này cùng với ID của chunk được lưu trữ trong vector database.
2. **Xử lý truy vấn (Online):**
    - Khi người dùng gửi một truy vấn, truy vấn đó cũng được chuyển đổi thành một vector nhúng bằng cùng embedding model đã sử dụng cho tài liệu.
    - Vector truy vấn được gửi đến vector database để tìm kiếm các vector tài liệu có độ tương đồng cao nhất.
    - Vector database trả về các ID của các chunk tài liệu liên quan nhất.
    - Các chunk tài liệu gốc tương ứng với các ID này được truy xuất từ kho kiến thức và chuyển đến mô-đun Generation [5].

2.4. Các chiến lược Chunking

Việc chia tài liệu thành các chunk là một bước quan trọng trong Retrieval, ảnh hưởng đến chất lượng của các tài liệu được truy xuất. Các chiến lược chunking bao gồm:

- **Fixed-size Chunking:** Chia tài liệu thành các chunk có kích thước cố định (ví dụ: 256, 512 token) với một phần chồng lấp (overlap) nhất định để duy trì ngữ cảnh.
- **Semantic Chunking:** Chia tài liệu dựa trên cấu trúc ngữ nghĩa, ví dụ: theo đoạn văn, tiêu đề, hoặc các đoạn có ý nghĩa hoàn chỉnh.
- **Recursive Chunking:** Chia tài liệu theo nhiều cấp độ, bắt đầu từ các đơn vị lớn hơn (ví dụ: chương, phần) sau đó chia nhỏ hơn (đoạn, câu) [6].

2.5. Các kỹ thuật Retrieval nâng cao

- **Re-ranking:** Sau khi truy xuất một tập hợp các tài liệu ban đầu, một mô hình re-ranking có thể được sử dụng để sắp xếp lại các tài liệu này dựa trên mức độ liên quan chính xác hơn, giúp chọn ra những tài liệu tốt nhất để đưa vào LLM [7].
- **Hybrid Search:** Kết hợp tìm kiếm từ khóa truyền thống (ví dụ: BM25) với tìm kiếm vector để tận dụng cả độ chính xác của từ khóa và khả năng hiểu ngữ nghĩa.
- **Multi-hop Retrieval:** Đối với các câu hỏi phức tạp yêu cầu thông tin từ nhiều nguồn hoặc nhiều bước suy luận, hệ thống có thể thực hiện nhiều vòng truy xuất để thu thập đầy đủ thông tin cần thiết.
- **Query Expansion/Rewriting:** Mở rộng hoặc viết lại truy vấn của người dùng để cải thiện kết quả tìm kiếm, ví dụ: thêm từ đồng nghĩa, hoặc tạo ra các truy vấn con.

2.6. Thách thức trong Retrieval

- **Chất lượng Embedding:** Mô hình nhúng không đủ tốt có thể dẫn đến việc truy xuất các tài liệu không liên quan.
- **Kích thước Chunk:** Chọn kích thước chunk phù hợp là một thách thức, ảnh hưởng đến ngữ cảnh và độ nhiễu.
- **Tìm kiếm trên dữ liệu lớn:** Đảm bảo tốc độ và hiệu quả khi tìm kiếm trên các kho kiến thức khổng lồ.
- **Xử lý thông tin lỗi thời/không chính xác:** Đảm bảo rằng kho kiến thức được cập nhật và không chứa thông tin sai lệch.
- **Tính mơ hồ của truy vấn:** Một truy vấn có thể có nhiều cách hiểu, dẫn đến việc truy xuất các tài liệu không mong muốn.

2.7. Kết luận

Mô-đun Retrieval là yếu tố then chốt quyết định chất lượng của hệ thống RAG. Bằng cách thiết kế một quy trình truy xuất hiệu quả, chúng ta có thể đảm bảo rằng LLM nhận được thông tin chính xác và phù hợp, từ đó tạo ra các phản hồi chất lượng cao. Chương tiếp theo sẽ tập trung vào mô-đun Generation và cách nó sử dụng thông tin được truy xuất để tạo ra câu trả lời.

Tài liệu tham khảo

[1] AWS. (n.d.). *RAG là gì? - Giải thích về AI tạo có kết hợp truy xuất thông tin ngoài*. Retrieved from https://aws.amazon.com/vi/what-is/retrieval-augmented-generation/ [2] Pinecone. (n.d.). *What is a vector database?*. Retrieved from https://www.pinecone.io/learn/vector-database/ [3] Hugging Face. (n.d.). *Sentence Transformers*. Retrieved from https://www.sbert.net/ [4] Weaviate. (n.d.). *What is a vector database?*. Retrieved from https://weaviate.io/blog/what-is-a-vector-database [5] LlamaIndex. (n.d.). *RAG Fundamentals*. Retrieved from https://docs.llamaindex.ai/en/stable/getting_started/concepts/rag_fundamentals.html [6] LangChain. (n.d.). *Text Splitters*. Retrieved from https://python.langchain.com/docs/modules/data_connection/document_loaders/how_to/text_splitter [7] Cohere. (n.d.). *Rerank*. Retrieved from https://cohere.com/rerank

3. Generation (Tạo sinh)

3.1. Giới thiệu về Generation Module

Mô-đun Generation là thành phần thứ hai và cũng là cuối cùng trong quy trình RAG, chịu trách nhiệm tạo ra phản hồi cuối cùng cho người dùng. Sau khi mô-đun Retrieval đã truy xuất các đoạn văn bản liên quan từ kho kiến thức, mô-đun Generation (thường là một mô hình ngôn ngữ lớn - LLM) sẽ sử dụng thông tin này cùng với truy vấn ban đầu của người dùng để tổng hợp và tạo ra một câu trả lời mạch lạc, chính xác và tự nhiên [1, 2].

Mục tiêu của mô-đun Generation không chỉ là tạo ra văn bản trôi chảy mà còn phải đảm bảo rằng văn bản đó có căn cứ, phù hợp với ngữ cảnh và giải quyết được truy vấn của người dùng một cách hiệu quả. Điều này đòi hỏi LLM phải có khả năng hiểu sâu sắc cả truy vấn và các tài liệu được cung cấp.

3.2. Các thành phần chính của Generation Module

3.2.1. Mô hình ngôn ngữ lớn (Large Language Model - LLM)

LLM là cốt lõi của mô-đun Generation. Đây là các mô hình học sâu được huấn luyện trên lượng lớn dữ liệu văn bản, cho phép chúng hiểu, tạo và tương tác với ngôn ngữ tự nhiên. Trong ngữ cảnh RAG, LLM đóng vai trò là bộ não tổng hợp, kết hợp kiến thức nội tại của nó với thông tin bên ngoài được cung cấp bởi mô-đun Retrieval [3].

Các loại LLM phổ biến được sử dụng trong RAG bao gồm:

- **Encoder-Decoder Models:** Ví dụ như T5, BART, có khả năng mã hóa (encode) đầu vào và giải mã (decode) đầu ra.
- **Decoder-only Models:** Ví dụ như GPT-3, GPT-4, Llama, được thiết kế để tạo ra văn bản một cách tự do dựa trên đầu vào đã cho.

3.2.2. Kỹ thuật Prompt Engineering

Prompt Engineering là nghệ thuật và khoa học thiết kế các câu lệnh (prompts) hiệu quả để hướng dẫn LLM tạo ra đầu ra mong muốn. Trong RAG, prompt được xây dựng bằng cách kết hợp truy vấn của người dùng với các đoạn văn bản được truy xuất. Cấu trúc của prompt có thể ảnh hưởng đáng kể đến chất lượng của phản hồi được tạo ra [4].

Một prompt điển hình trong RAG có thể có dạng:

```
Bạn là một trợ lý thông minh. Dựa trên thông tin sau đây, hãy trả lời câu hỏi của người dùng:

Thông tin được truy xuất:
[Đoạn văn bản 1]
[Đoạn văn bản 2]
...

Câu hỏi của người dùng: [Truy vấn của người dùng]

Trả lời:
```

3.2.3. Kỹ thuật tạo sinh văn bản (Text Generation Techniques)

Sau khi nhận được prompt, LLM sử dụng các kỹ thuật tạo sinh văn bản để tạo ra phản hồi. Các kỹ thuật này kiểm soát cách LLM chọn từ tiếp theo trong chuỗi, ảnh hưởng đến tính trôi chảy, đa dạng và chất lượng của văn bản được tạo ra [5].

Các kỹ thuật phổ biến bao gồm:

- **Greedy Search:** Chọn từ có xác suất cao nhất ở mỗi bước.
- **Beam Search:** Duy trì một số lượng nhất định các chuỗi có khả năng cao nhất ở mỗi bước, sau đó chọn chuỗi tốt nhất.
- **Sampling (Top-K, Nucleus Sampling):** Giới hạn việc lựa chọn từ tiếp theo trong một tập hợp các từ có xác suất cao nhất, giúp tăng tính đa dạng và sáng tạo của văn bản.
- **Temperature:** Một tham số điều chỉnh mức độ ngẫu nhiên của đầu ra. Nhiệt độ cao hơn tạo ra văn bản đa dạng hơn nhưng có thể kém chính xác hơn.

3.3. Quy trình Generation

Quá trình tạo sinh trong RAG diễn ra như sau:

1. **Nhận đầu vào:** LLM nhận truy vấn ban đầu của người dùng và các đoạn văn bản liên quan được truy xuất từ mô-đun Retrieval.
2. **Xây dựng ngữ cảnh:** Các đoạn văn bản được truy xuất được sắp xếp và định dạng để tạo thành một ngữ cảnh rõ ràng cho LLM. Điều này có thể bao gồm việc thêm các tiêu đề, đánh số đoạn, hoặc các chỉ dẫn khác để LLM dễ dàng tham chiếu.
3. **Tạo phản hồi:** LLM sử dụng ngữ cảnh đã xây dựng và kiến thức nội tại của nó để tạo ra câu trả lời. LLM sẽ cố gắng tổng hợp thông tin từ các đoạn văn bản được cung cấp, kết hợp với khả năng hiểu và tạo ngôn ngữ của nó để đưa ra một phản hồi toàn diện và chính xác.

4. **Hậu xử lý (Post-processing):** Phản hồi được tạo ra có thể trải qua một số bước hậu xử lý như kiểm tra ngữ pháp, chính tả, hoặc định dạng lại để đảm bảo chất lượng cuối cùng trước khi hiển thị cho người dùng [6].

3.4. Các chiến lược kết hợp Retrieval và Generation

Có nhiều cách để kết hợp mô-đun Retrieval và Generation, tùy thuộc vào kiến trúc và mục tiêu của hệ thống RAG:

- **Naive RAG:** Đây là cách tiếp cận cơ bản nhất, trong đó các tài liệu được truy xuất được nối trực tiếp vào prompt của LLM.
- **Iterative RAG:** Hệ thống có thể thực hiện nhiều vòng truy xuất và tạo sinh, trong đó phản hồi từ một vòng có thể được sử dụng để tinh chỉnh truy vấn cho vòng tiếp theo.
- **Adaptive RAG:** Hệ thống tự động quyết định khi nào cần truy xuất thông tin và khi nào có thể dựa vào kiến thức nội tại của LLM, hoặc điều chỉnh số lượng tài liệu cần truy xuất dựa trên độ phức tạp của truy vấn.
- **Self-RAG:** LLM tự đánh giá chất lượng của các tài liệu được truy xuất và quyết định liệu có cần truy xuất thêm hay không, hoặc điều chỉnh cách tạo sinh dựa trên độ tin cậy của thông tin [7].

3.5. Thách thức trong Generation

- **Độ dài ngữ cảnh (Context Window Limitation):** LLM có giới hạn về số lượng token mà nó có thể xử lý trong một lần. Nếu các tài liệu được truy xuất quá dài, chúng có thể vượt quá giới hạn này, dẫn đến việc mất thông tin hoặc giảm hiệu suất.
- **Tính nhất quán và chính xác:** Đảm bảo rằng LLM không tạo ra thông tin mâu thuẫn với các tài liệu được truy xuất hoặc thông tin sai lệch.
- **Tính trôi chảy và tự nhiên:** Mặc dù LLM có khả năng tạo văn bản tốt, việc đảm bảo phản hồi luôn trôi chảy, tự nhiên và phù hợp với giọng điệu mong muốn vẫn là một thách thức.
- **Xử lý thông tin không liên quan:** Nếu mô-đun Retrieval trả về các tài liệu không liên quan, LLM có thể bị nhiễu và tạo ra phản hồi kém chất lượng.
- **Đánh giá chất lượng đầu ra:** Việc đánh giá tự động chất lượng của phản hồi được tạo ra bởi RAG là một vấn đề phức tạp, đòi hỏi các chỉ số và phương pháp đánh giá tinh vi.

3.6. Kết luận

Mô-đun Generation là nơi thông tin được truy xuất được biến thành phản hồi có ý nghĩa cho người dùng. Bằng cách tận dụng sức mạnh của các LLM và các kỹ thuật tạo sinh văn bản tiên tiến, RAG cho phép chúng ta xây dựng các hệ thống AI có khả năng trả lời câu hỏi, tóm tắt thông tin và tạo ra nội dung một cách thông minh và có căn cứ. Chương tiếp theo sẽ đi sâu vào các kỹ thuật tối ưu hóa và những thách thức tổng thể trong việc triển khai hệ thống RAG.

Tài liệu tham khảo

[1] AWS. (n.d.). *RAG là gì? - Giải thích về AI tạo có kết hợp truy xuất thông tin ngoài*. Retrieved from https://aws.amazon.com/vi/what-is/retrieval-augmented-generation/ [2] 200Lab. (2025, March 11). *RAG (Retrieval-Augmented Generation) là gì? Giải thích dễ hiểu cho người mới bắt đầu*. Retrieved from https://200lab.io/blog/rag-la-gi/ [3] IBM. (n.d.). *What are Large Language Models (LLMs)?*. Retrieved from https://www.ibm.com/topics/large-language-models [4] OpenAI. (n.d.). *Prompt engineering*. Retrieved from https://platform.openai.com/docs/guides/prompt-engineering [5] Hugging Face. (n.d.). *Text Generation Strategies*. Retrieved from https://huggingface.co/docs/transformers/generation_strategies [6] LlamaIndex. (n.d.). *RAG Fundamentals*. Retrieved from https://docs.llamaindex.ai/en/stable/getting_started/concepts/rag_fundamentals.html [7] Google AI Blog. (2024, January 11). *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*. Retrieved from https://ai.googleblog.com/2024/01/self-rag-learning-to-retrieve-generate.html

4. Tối ưu hóa và các thách thức trong RAG

4.1. Tối ưu hóa hiệu suất RAG

Để xây dựng một hệ thống RAG hiệu quả, việc tối ưu hóa là rất quan trọng. Các chiến lược tối ưu hóa có thể tập trung vào cả mô-đun Retrieval và Generation, cũng như sự tương tác giữa chúng.

4.1.1. Tối ưu hóa Retrieval

- **Chất lượng Embedding:** Sử dụng các mô hình nhúng (embedding models) tiên tiến và phù hợp với miền dữ liệu cụ thể. Huấn luyện hoặc tinh chỉnh (fine-tune) các mô hình nhúng trên dữ liệu riêng có thể cải thiện đáng kể chất lượng truy xuất [1].
- **Chiến lược Chunking thông minh:** Thay vì chỉ chia chunk theo kích thước cố định, hãy áp dụng các chiến lược chunking ngữ nghĩa hoặc đệ quy để đảm bảo mỗi chunk chứa thông tin đầy đủ và có ngữ cảnh. Điều này giúp LLM dễ dàng hiểu và sử dụng thông tin được truy xuất [2].
- **Cải thiện tìm kiếm tương đồng:** Sử dụng các thuật toán tìm kiếm tương đồng hiệu quả trong vector database. Áp dụng các kỹ thuật như re-ranking để tinh chỉnh kết quả truy xuất ban đầu, đảm bảo các tài liệu liên quan nhất được ưu tiên [3].
- **Quản lý kho kiến thức:** Thường xuyên cập nhật và làm sạch kho kiến thức để đảm bảo thông tin luôn mới và chính xác. Loại bỏ các tài liệu trùng lặp hoặc không còn phù hợp.
- **Tối ưu hóa truy vấn:** Áp dụng các kỹ thuật mở rộng hoặc viết lại truy vấn (query expansion/rewriting) để cải thiện khả năng tìm kiếm. Ví dụ, thêm các từ đồng nghĩa hoặc tạo các truy vấn con cho các câu hỏi phức tạp.

4.1.2. Tối ưu hóa Generation

- **Lựa chọn LLM phù hợp:** Chọn LLM có kích thước và khả năng phù hợp với yêu cầu của ứng dụng. Các LLM nhỏ hơn có thể nhanh hơn và tiết kiệm chi phí hơn cho các tác vụ đơn giản, trong khi các LLM lớn hơn có thể cần thiết cho các tác vụ phức tạp hơn.

- **Prompt Engineering nâng cao:** Thiết kế các prompt rõ ràng, cụ thể và có cấu trúc tốt. Sử dụng các kỹ thuật như few-shot prompting (cung cấp ví dụ trong prompt) hoặc chain-of-thought prompting (hướng dẫn LLM suy luận từng bước) để cải thiện chất lượng phản hồi [4].
- **Kiểm soát đầu ra:** Sử dụng các kỹ thuật tạo sinh văn bản như temperature, top-k, nucleus sampling để kiểm soát tính đa dạng và sáng tạo của đầu ra. Áp dụng các ràng buộc (constraints) nếu cần để đảm bảo phản hồi tuân thủ các quy tắc nhất định.
- **Tinh chỉnh LLM (Fine-tuning):** Trong một số trường hợp, việc tinh chỉnh LLM trên dữ liệu cụ thể của miền có thể cải thiện đáng kể hiệu suất tạo sinh, đặc biệt là về giọng điệu, phong cách và độ chính xác của thông tin [5].

4.1.3. Tối ưu hóa toàn hệ thống

- **Phản hồi vòng lặp (Feedback Loop):** Triển khai một cơ chế phản hồi để thu thập đánh giá từ người dùng về chất lượng phản hồi của RAG. Dữ liệu này có thể được sử dụng để cải thiện cả mô-đun Retrieval và Generation.
- **A/B Testing:** Thực hiện A/B testing để so sánh hiệu suất của các phiên bản RAG khác nhau (ví dụ: các chiến lược chunking khác nhau, các mô hình nhúng khác nhau) và xác định cấu hình tối ưu.
- **Giám sát và phân tích lỗi:** Liên tục giám sát hiệu suất của hệ thống RAG trong môi trường thực tế và phân tích các trường hợp lỗi để xác định nguyên nhân gốc rễ và cải thiện hệ thống.

4.2. Các thách thức trong triển khai RAG

Mặc dù RAG mang lại nhiều lợi ích, việc triển khai một hệ thống RAG hiệu quả vẫn đối mặt với một số thách thức đáng kể:

4.2.1. Chất lượng dữ liệu và kho kiến thức

- **Dữ liệu không sạch:** Dữ liệu trong kho kiến thức có thể chứa lỗi, thông tin không chính xác, hoặc định dạng không nhất quán, ảnh hưởng đến chất lượng truy xuất.
- **Tính đầy đủ của kiến thức:** Kho kiến thức có thể không bao phủ đủ tất cả các chủ đề hoặc câu hỏi mà người dùng có thể đặt ra.
- **Cập nhật dữ liệu:** Việc duy trì và cập nhật kho kiến thức liên tục với thông tin mới nhất là một thách thức, đặc biệt với các lĩnh vực có tốc độ thay đổi nhanh.

4.2.2. Vấn đề về ngữ cảnh và tính mơ hồ

- **Giới hạn độ dài ngữ cảnh của LLM:** Các LLM có giới hạn về số lượng token mà chúng có thể xử lý. Nếu các tài liệu được truy xuất quá dài, chúng có thể bị cắt bớt, dẫn đến mất thông tin quan trọng.
- **"Lost in the Middle":** Một số nghiên cứu chỉ ra rằng LLM có xu hướng bỏ qua thông tin quan trọng nếu nó nằm ở giữa một đoạn văn bản dài, thay vì ở đầu hoặc cuối [6].
- **Tính mơ hồ của truy vấn và tài liệu:** Một truy vấn hoặc một đoạn tài liệu có thể có nhiều cách hiểu, dẫn đến việc LLM tạo ra phản hồi không chính xác hoặc không phù hợp.

4.2.3. Đánh giá và đo lường hiệu suất

- **Thiếu các chỉ số đánh giá toàn diện:** Việc đánh giá chất lượng của hệ thống RAG là phức tạp, đòi hỏi các chỉ số không chỉ đo lường độ chính xác mà còn cả tính trôi chảy, sự liên quan và khả năng giải thích của phản hồi.
- **Đánh giá thủ công tốn kém:** Việc đánh giá thủ công bởi con người là tốn kém và không thể mở rộng cho các hệ thống lớn.
- **Đánh giá tự động khó khăn:** Các phương pháp đánh giá tự động hiện tại (ví dụ: ROUGE, BLEU) có thể không phản ánh đầy đủ chất lượng ngữ nghĩa của phản hồi.

4.2.4. Chi phí và tài nguyên

- **Chi phí tính toán:** Việc chạy các mô hình nhúng và LLM, đặc biệt là các mô hình lớn, đòi hỏi tài nguyên tính toán đáng kể.
- **Chi phí lưu trữ:** Lưu trữ các vector nhúng trong vector database có thể tốn kém, đặc biệt với các kho kiến thức lớn.
- **Chi phí phát triển và bảo trì:** Xây dựng và duy trì một hệ thống RAG đòi hỏi kiến thức chuyên sâu về NLP, học máy và kỹ thuật phần mềm.

4.2.5. Các thách thức khác

- **Độ trễ (Latency):** Quá trình truy xuất và tạo sinh có thể gây ra độ trễ, ảnh hưởng đến trải nghiệm người dùng trong các ứng dụng thời gian thực.
- **Tính công bằng và thiên vị:** Nếu dữ liệu trong kho kiến thức hoặc dữ liệu huấn luyện LLM có thiên vị, hệ thống RAG có thể tạo ra các phản hồi thiên vị hoặc không công bằng.
- **Bảo mật và quyền riêng tư:** Xử lý thông tin nhạy cảm trong kho kiến thức đòi hỏi các biện pháp bảo mật và quyền riêng tư nghiêm ngặt.

4.3. Kết luận

Tối ưu hóa và giải quyết các thách thức trong RAG là một quá trình liên tục. Bằng cách hiểu rõ các yếu tố ảnh hưởng đến hiệu suất và các vấn đề tiềm ẩn, chúng ta có thể thiết kế và triển khai các hệ thống RAG mạnh mẽ, đáng tin cậy và hiệu quả hơn. Chương tiếp theo sẽ hướng dẫn bạn xây dựng một dự án thực hành RAG để áp dụng các kiến thức đã học.

Tài liệu tham khảo

[1] Cohere. (n.d.). *Embeddings*. Retrieved from https://cohere.com/embeddings [2] LangChain. (n.d.). *Text Splitters*. Retrieved from https://python.langchain.com/docs/modules/data_connection/document_loaders/how_to/text_splitter [3] Cohere. (n.d.). *Rerank*. Retrieved from https://cohere.com/rerank [4] OpenAI. (n.d.). *Prompt engineering*. Retrieved from https://platform.openai.com/docs/guides/prompt-engineering [5] Hugging Face. (n.d.). *Fine-tuning a pre-trained model*. Retrieved from https://huggingface.co/docs/transformers/training#fine-tuning-a-pre-trained-model [6] Google AI Blog.

(2023, July 26). *Lost in the middle: How language models use long contexts*. Retrieved from https://ai.googleblog.com/2023/07/lost-in-middle-how-language-models-use.html

5. Dự án thực hành RAG

5.1. Giới thiệu

Dự án thực hành này nhằm mục đích cung cấp cho học viên trải nghiệm thực tế trong việc xây dựng một hệ thống Retrieval-Augmented Generation (RAG) đơn giản. Bằng cách thực hiện dự án này, bạn sẽ củng cố kiến thức về các thành phần chính của RAG (Retrieval và Generation), các kỹ thuật tiền xử lý dữ liệu, tạo embeddings, tìm kiếm vector và tích hợp với LLM.

5.2. Mục tiêu dự án

- Hiểu rõ quy trình end-to-end của một hệ thống RAG.
- Áp dụng các thư viện và công cụ phổ biến để xây dựng RAG.
- Đánh giá chất lượng của các tài liệu được truy xuất và phản hồi được tạo ra.
- Nắm vững cách LLM sử dụng ngữ cảnh được cung cấp để tạo ra câu trả lời.

5.3. Kịch bản dự án: Xây dựng Chatbot hỏi đáp về tài liệu

Chúng ta sẽ xây dựng một chatbot có khả năng trả lời các câu hỏi dựa trên một tập hợp các tài liệu cụ thể (ví dụ: tài liệu hướng dẫn sử dụng sản phẩm, các bài viết blog về một chủ đề nhất định, hoặc các tài liệu nội bộ của công ty). Chatbot sẽ không chỉ dựa vào kiến thức của LLM mà còn truy xuất thông tin từ các tài liệu đã cho để đưa ra câu trả lời chính xác và có căn cứ.

5.4. Các bước thực hiện dự án

5.4.1. Chuẩn bị môi trường

1. **Cài đặt Python và pip:** Đảm bảo bạn đã cài đặt Python 3.8+ và pip.
2. **Cài đặt các thư viện cần thiết:** `bash pip install langchain pip install openai pip install chromadb pip install pypdf pip install tiktoken` *Lưu ý: Bạn cần có OpenAI API Key để sử dụng các mô hình của OpenAI. Thay thế* `your_openai_api_key` *bằng khóa API thực tế của bạn.*

5.4.2. Chuẩn bị dữ liệu

1. **Thu thập tài liệu:** Chọn một tập hợp các tài liệu dạng văn bản (ví dụ: vài file `.txt` hoặc `.pdf` về một chủ đề bạn quan tâm). Đặt các tài liệu này vào một thư mục có tên `data` trong thư mục dự án của bạn.
2. **Tải tài liệu:** Sử dụng `PyPDFLoader` (nếu là PDF) hoặc `DirectoryLoader` để tải các tài liệu. ```python from langchain.document_loaders import PyPDFLoader, DirectoryLoader from langchain.text_splitter import RecursiveCharacterTextSplitter

Tải tài liệu từ thư mục 'data'

loader = DirectoryLoader('./data', glob="*/.pdf", loader_cls=PyPDFLoader) documents = loader.load()

Chia tài liệu thành các chunk

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200) docs = text_splitter.split_documents(documents) print(f"Số lượng chunks: {len(docs)}") ```

5.4.3. Tạo Embeddings và Vector Store

1. **Khởi tạo Embedding Model:** Sử dụng `OpenAIEmbeddings` để tạo vector nhúng cho các chunk.
2. **Khởi tạo Vector Store:** Sử dụng `Chroma` làm vector database để lưu trữ các embeddings và thực hiện tìm kiếm tương đồng. ```python from langchain.embeddings.openai import OpenAIEmbeddings from langchain.vectorstores import Chroma import os

Thiết lập OpenAI API Key

os.environ["OPENAI_API_KEY"] = "your_openai_api_key"

Khởi tạo embedding model

embeddings = OpenAIEmbeddings()

Tạo vector store từ các chunks

persist_directory = 'db' vectordb = Chroma.from_documents(documents=docs, embedding=embeddings, persist_directory=persist_directory)

Lưu vector store vào đĩa

vectordb.persist() vectordb = None # Xóa khỏi bộ nhớ

Tải lại vector store (khi cần)

vectordb = Chroma(persist_directory=persist_directory, embedding_function=embeddings) ```

5.4.4. Xây dựng Retrieval Chain

1. **Khởi tạo Retriever:** Tạo một retriever từ vector store để tìm kiếm các tài liệu liên quan. `python retriever = vectordb.as_retriever()`
2. **Kiểm tra Retriever (Tùy chọn):** Thử truy vấn để xem các tài liệu được truy xuất. `python query = "Câu hỏi của bạn về tài liệu" docs_retrieved = retriever.get_relevant_documents(query) print(f"Số lượng tài liệu được truy xuất: {len(docs_retrieved)}") for doc in docs_retrieved: print(doc.page_content[:200]) # In 200 ký tự đầu tiên của mỗi tài liệu`

5.4.5. Tích hợp với LLM và tạo phản hồi

1. **Khởi tạo LLM:** Sử dụng `ChatOpenAI` làm mô hình tạo sinh.
2. **Xây dựng RetrievalQA Chain:** Kết hợp retriever và LLM để tạo thành một chuỗi hỏi đáp. ```python from langchain.chat_models import ChatOpenAI from langchain.chains import RetrievalQA

Khởi tạo LLM

llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)

Xây dựng RetrievalQA chain

qa_chain = RetrievalQA.from_chain_type(llm=llm, chain_type="stuff", retriever=retriever, return_source_documents=True) `3. **Đặt câu hỏi và nhận câu trả lời:**` python question = "Câu hỏi của bạn về tài liệu" result = qa_chain({"query": question}) print("Câu trả lời:", result["result"]) print("Tài liệu nguồn:") for doc in result["source_documents"]: print(doc.metadata) # In metadata của tài liệu nguồn ```

5.5. Đánh giá và cải tiến

- **Đánh giá thủ công:** Đặt nhiều câu hỏi khác nhau và kiểm tra xem câu trả lời có chính xác, đầy đủ và có căn cứ từ các tài liệu nguồn hay không.
- **Kiểm tra các trường hợp lỗi:** Đặt các câu hỏi nằm ngoài phạm vi tài liệu hoặc các câu hỏi mơ hồ để xem hệ thống xử lý như thế nào.

- **Cải tiến:**
    - **Thay đổi kích thước chunk và overlap:** Thử nghiệm các giá trị khác nhau để tìm ra cấu hình tối ưu.
    - **Sử dụng các embedding model khác:** Khám phá các mô hình nhúng khác (ví dụ: từ Hugging Face) để xem có cải thiện hiệu suất không.
    - **Sử dụng các LLM khác:** Thử nghiệm với các LLM khác nhau (ví dụ: Llama 2, Mistral) nếu bạn có tài nguyên.
    - **Kỹ thuật Re-ranking:** Tích hợp một mô hình re-ranking để cải thiện chất lượng của các tài liệu được truy xuất.
    - **Prompt Engineering nâng cao:** Tinh chỉnh prompt để hướng dẫn LLM tạo ra phản hồi tốt hơn.

5.6. Kết luận dự án

Dự án này cung cấp một nền tảng vững chắc để bạn bắt đầu với RAG. Từ đây, bạn có thể mở rộng dự án bằng cách tích hợp thêm các tính năng phức tạp hơn, xử lý các loại dữ liệu đa dạng hơn, hoặc triển khai hệ thống trong môi trường sản phẩm.