RAG - Retrieval-Augmented Generation

☆ RAG là gì?

RAG (Retrieval-Augmented Generation) là hướng tiếp cận kết hợp sức mạnh của truy xuất thông tin với khả năng sinh nội dung của Large Language Models (LLMs). Thay vì chỉ dựa vào kiến thức nội tại của mô hình, RAG lấy thông tin liên quan từ nguồn dữ liệu bên ngoài rồi đưa vào cho mô hình dưới dạng context, nhờ đó mô hình tạo ra câu trả lời chính xác, cập nhật và đúng sự thật hơn.

☆ Ý tưởng cốt lõi

RAG đi theo một ý tưởng đơn giản nhưng cực kỳ hiệu quả:

Retrieve
Tìm thông tin liên quan từ các nguồn bên ngoài.

→

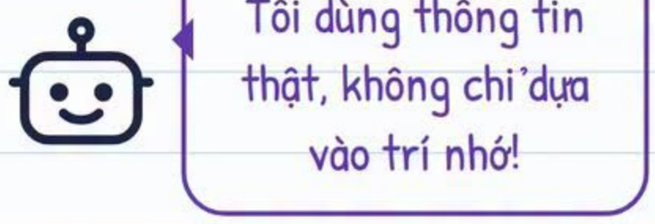
Augment
Bổ sung thông tin vừa tìm được vào câu hỏi của người dùng.

→

Generate
LLM sinh ra câu trả lời tốt hơn và có căn cứ.

☆ Nói một cách đơn giản

Thay vì bắt mô hình trả lời mọi thứ bằng trí nhớ của nó, ta đi tìm thông tin đúng ở bên ngoài trước, đưa thông tin đó cho mô hình, rồi mới yêu cầu nó trả lời. Nhờ vậy mô hình trả lời dựa trên dữ kiện thực tế, chứ không chỉ dựa vào những gì nó ghi nhớ.



☆ Vì sao RAG quan trọng?

- Cung cấp thông tin cập nhật.
- Giảm hallucination (bịa thông tin).
- Dùng được với dữ liệu riêng tư hoặc theo từng lĩnh vực.
- Tăng sự tin cậy và tính ổn định của LLM.
- Phù hợp cho doanh nghiệp, nghiên cứu và ứng dụng thực tế.

Lợi ích chính

- Câu trả lời chính xác hơn
- Bám sát dữ liệu thực tế
- Minh bạch, kiểm chứng được
- Linh hoạt với nhiều nguồn dữ liệu

RAG giúp LLM thông minh hơn bằng cách cho nó tiếp cận đúng thông tin vào đúng thời điểm.

RAG - Retrieval-Augmented Generation

☆ Vì sao chúng ta cần RAG?

LLM rất mạnh, nhưng vẫn tồn tại một số hạn chế quan trọng.

RAG giúp khắc phục chúng bằng cách bổ sung thông tin thật, đúng trọng tâm.

Hạn chế của LLM

① **Kiến thức lỗi thời** - LLM chỉ được huấn luyện trên dữ liệu đến một mốc thời gian nhất định nên có thể không biết các sự kiện gần đây.

② **Hallucination** - LLM có thể rất tự tin đưa ra thông tin sai lệch hoặc bịa đặt.

③ **Không truy cập được dữ liệu riêng tư hay dữ liệu chuyên ngành** - LLM không đọc được tài liệu, cơ sở dữ liệu hay kiến thức nội bộ của công ty bạn.

④ **Thiếu minh bạch về nguồn** - LLM không cho biết thông tin đến từ đâu, khiến bạn rất khó kiểm chứng.

☆ RAG giải quyết những vấn đề này thế nào

- Lấy thông tin cập nhật từ các nguồn bên ngoài.
- Đưa ra câu trả lời dựa trên dữ kiện, có căn cứ.
- Kết nối LLM với dữ liệu riêng tư và chuyên ngành của bạn.
- Trích dẫn nguồn, tăng tính minh bạch và sự tin tưởng.
- Giảm hallucination và cải thiện độ tin cậy.

RAG kết hợp khả năng sáng tạo của LLM với độ chính xác của dữ liệu thực tế.

☆ Điểm mấu chốt

RAG lấp khoảng trống giữa những gì LLM biết và những gì bạn cần, bằng cách mang đúng thông tin đến đúng thời điểm.

☆ Tóm lại:

LLM là bộ não, còn RAG cho nó trí nhớ, context và dữ kiện từ thế giới thực.

RAG - Retrieval-Augmented Generation

RAG hoạt động như thế nào

RAG đi theo một vòng lặp đơn giản để đưa ra câu trả lời chính xác và bám sát context.

Retrieve
↓
Augment
↓
Generate

① Câu hỏi → ② Retrieve → ③ Augment → ④ Generate → ⑤ Trả lời

① Câu hỏi	② Retrieve	③ Augment	④ Generate	⑤ Trả lời
Người dùng đặt câu hỏi.	Thông tin liên quan được truy xuất từ nguồn bên ngoài.	Thông tin đó được thêm vào câu hỏi dưới dạng context.	LLM dùng context để sinh câu trả lời có căn cứ.	Câu trả lời cuối cùng được gửi tới người dùng.

RAG giúp LLM thông minh hơn bằng cách kết hợp khả năng suy luận của nó với kiến thức từ thế giới thực.

Thành phần chính

- **User** - Người đặt câu hỏi.
- **Retriever** - Tìm thông tin liên quan nhất từ các nguồn bên ngoài.
- **Data Sources** - Tài liệu, cơ sở dữ liệu, website, API, PDF, v.v.
- **LLM** - Sinh câu trả lời cuối cùng dựa trên context được cung cấp.
- **Response** - Câu trả lời cuối cùng, dễ hiểu với người dùng.

Điểm quan trọng

- RAG **không** thay đổi bản thân LLM, nó chỉ cung cấp thông tin tốt hơn.
- Chất lượng câu trả lời phụ thuộc vào chất lượng thông tin truy xuất được.
- Nó giảm hallucination và giữ câu trả lời bám sát dữ kiện.

Takeaway

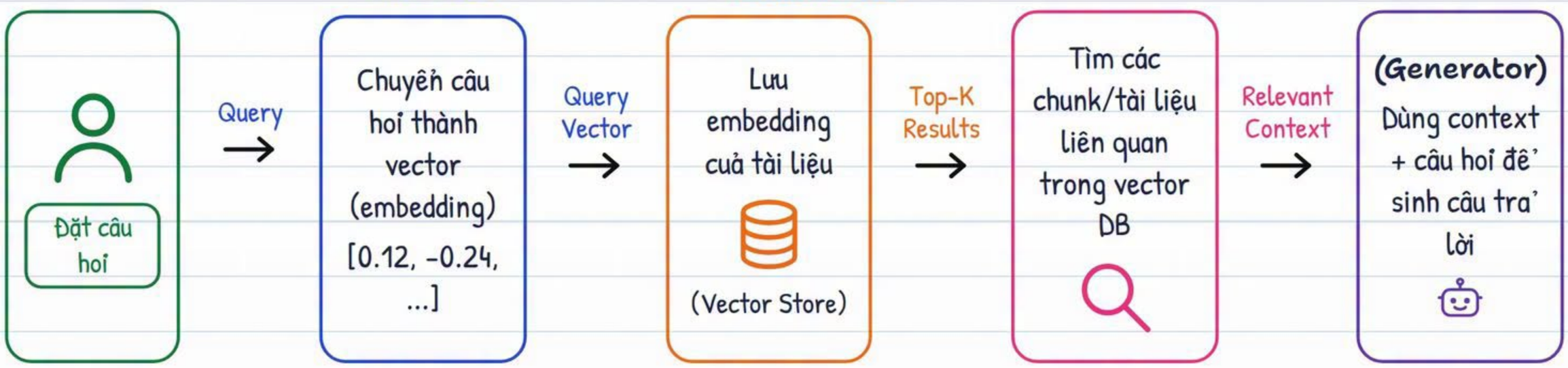
RAG giống như trang bị cho LLM một combo "search engine + bộ não"! Nó tìm đúng thông tin rồi mới suy nghĩ và trả lời tốt hơn.

RAG - Retrieval-Augmented Generation

☆ Kiến trúc RAG

RAG kết hợp một retriever và một generator.

Dưới đây là kiến trúc tổng quan của một hệ thống RAG.



☆ Kiến trúc RAG đảm bảo LLM nhận được đúng context từ nguồn bên ngoài **trước khi** sinh câu trả lời, nhờ đó câu trả lời chính xác, đáng tin cậy và đúng sự thật.

☆ Giải thích các thành phần chính

① User	• Người đặt câu hỏi.	Ví dụ: "RAG là gì?"
② Embedding Model	• Chuyển câu hỏi thành vector số nắm bắt được ý nghĩa của nó.	Ví dụ: [0.12, -0.24, 0.88, ...]
③ Vector Database	• Lưu embedding của các chunk tài liệu để tìm kiếm tương đồng nhanh.	Ví dụ: Pinecone, Chroma, Weaviate, FAISS
④ Retriever	• Tìm các chunk/tài liệu liên quan nhất bằng similarity search (Top-K).	Ví dụ: Trả về 5 chunk liên quan nhất.
⑤ LLM (Generator)	• Nhận câu hỏi + context truy xuất được và tạo ra câu trả lời có căn cứ.	Ví dụ: GPT-4, Llama 3, Gemini, Claude, v.v.
⑥ Response to User	• Câu trả lời cuối cùng được trả về cho người dùng.	Ví dụ: Câu trả lời rõ ràng, chính xác hiện ra cho người dùng.

Pro Tip

Chất lượng của một hệ thống RAG phụ thuộc vào 3 thứ:

Dữ liệu tốt + Truy xuất tốt + Sinh nội dung tốt = Câu trả lời tuyệt vời!

RAG - Retrieval-Augmented Generation

☆ Data Ingestion Pipeline của RAG

Trước khi có thể truy xuất thông tin, ta cần chuẩn bị dữ liệu.

Ingestion pipeline biến tài liệu thô thành kho kiến thức tìm kiếm được.

① Documents

Thu thập dữ liệu từ nhiều nguồn khác nhau.

(PDF, Docs, Web, TXT, API, v.v.)

② Cleaning

Loại bỏ nội dung thừa và chuẩn hoá văn bản.

- Xoá khoảng trắng thừa
- Loại bỏ nhiễu
- Sửa lỗi encoding
- Bỏ header/footer

③ Chunking

Chia văn bản đã làm sạch thành các chunk nhỏ.

Chunk 1

Chunk 2

...

Chunk N

(khoảng 300-500 token mỗi chunk)

④ Embeddings

Chuyển mỗi chunk thành vector bằng embedding model.

[0.12, -0.24, ...]

[-0.11, 0.78, ...]

...

[0.33, -0.91, ...]

⑤ Vector Database

Lưu vector vào vector database để tìm kiếm tương đồng nhanh.

(Pinecone, Chroma, FAISS, Weaviate, Qdrant, v.v.)

☆ Pipeline này thường chỉ chạy một lần (hoặc cập nhật định kỳ) khi có dữ liệu mới. Nhờ đó việc truy xuất lúc người dùng hỏi trở nên nhanh và hiệu quả.

☆ Giải thích từng bước

① **Documents:** Thu thập dữ liệu từ mọi nguồn có chứa thông tin hữu ích.

② **Cleaning:** Làm cho văn bản sạch và nhất quán để mô hình hiểu tốt hơn.

③ **Chunking:** Cắt văn bản dài thành các phần nhỏ, sao cho mỗi chunk vẫn đủ ngữ nghĩa và nằm trong giới hạn context của mô hình.

④ **Embeddings:** Biến mỗi chunk thành một vector (một dãy số) nắm bắt ý nghĩa của nó. [0.21, -0.45, ...]

⑤ **Vector DB:** Lưu các vector này vào cơ sở dữ liệu được thiết kế cho similarity search.

Điểm mấu chốt

Ingestion tốt = Truy xuất tốt = Câu trả lời tốt.

Dữ liệu càng được chuẩn bị kỹ, hệ thống RAG của bạn càng chạy hiệu quả!

Ghi nhớ: Garbage in → Garbage out.

Dữ liệu sạch, chia chunk hợp lý và đúng chủ đề sẽ cho câu trả lời chính xác và đáng tin cậy.

RAG - Retrieval-Augmented Generation

☆ Embeddings & Vector Databases

Embeddings và vector database là xương sống của RAG.
Chúng giúp hệ thống hiểu được ý nghĩa, chứ không chỉ so khớp từ khoá.

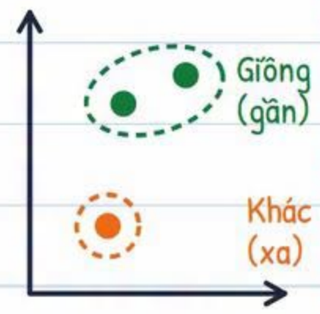
Embeddings giúp máy hiểu được ý nghĩa giống như con người!

Embeddings là gì?

- Embeddings là **biểu diễn dạng số** của văn bản.
- Chúng nắm bắt ý nghĩa ngữ nghĩa của từ, câu hoặc cả tài liệu.
- Nội dung có ý nghĩa giống nhau sẽ có vector nằm gần nhau trong không gian vector.

Ví dụ

Văn bản	Embeddings (Vectors)
"Tôi thích ăn pizza."	→ [0.21, -0.34, 0.72, ...]
"Pizza là món tôi thích nhất."	→ [0.19, -0.32, 0.70, ...]
"Tôi thích chơi cầu lông."	→ [-0.45, 0.61, -0.22, ...]



Vector Databases

- Vector database lưu embeddings (vector) của các chunk tài liệu.
- Nó được tối ưu cho similarity search (thay vì so khớp chính xác).
- Nó trả về những vector **giống nhất** với vector của câu hỏi.

Vector Database phổ biến

- Pinecone
- Weaviate
- FAISS
- Chroma
- Qdrant

Similarity Search hoạt động thế nào?

① Query (câu hỏi): Nên đi đâu chơi ở Việt Nam? →
② Chuyển thành Embedding: [0.25, -0.31, 0.68, ...] →
③ Tìm trong Vector DB →
④ Top-K Results:
1. Chunk #3
2. Chunk #8
3. Chunk #12 →
⑤ Trả về chunk liên quan

Embeddings + Vector DB giúp RAG tìm được đúng thông tin một cách nhanh chóng và chính xác, kể cả với lượng dữ liệu rất lớn.

Điểm mấu chốt

- Embeddings biến văn bản thành số nắm bắt được ý nghĩa.
- Vector database lưu chúng để similarity search nhanh.
- Nhờ đó RAG tìm được thông tin liên quan nhất cho mọi câu hỏi.

Ghi nhớ!

Vector = Ý nghĩa
Similarity Search = Độ liên quan
Độ liên quan = Câu trả lời tốt hơn

RAG - Retrieval-Augmented Generation

☆ Quy trình truy xuất

Quá trình truy xuất tìm ra thông tin liên quan nhất trong vector database cho một query cụ thể.

Mục tiêu: Lấy đúng thông tin để giúp LLM trả lời tốt hơn!

① Query

Người dùng đặt câu hỏi.

→

② Embed Query

Chuyển câu hỏi thành vector embedding.

[0.21, -0.34, 0.72, ...]

→

③ Similarity Search

Tìm các vector tương đồng trong vector database.

→

④ Top-K Results

Lấy ra K chunk giống nhất.

- Chunk #7
- Chunk #12
- Chunk #3
- ...

→

⑤ Rerank

(tuỳ chọn)

Sắp xếp lại kết quả để tăng độ liên quan.

① ——
② ——
③ ——

→

⑥ Retrieved Context

Đưa các chunk tốt nhất cho LLM làm context.

Truy xuất đảm bảo LLM nhận được thông tin liên quan và hữu ích nhất, thay vì phải đoán mò từ kiến thức nội tại của nó.

☆ Điểm chính

- Query được chuyển thành vector (embedding).
- Vector DB tìm các vector tương đồng.
- Top-K results là những chunk liên quan nhất.
- Reranking (tuỳ chọn) giúp tăng độ liên quan.
- Các chunk này được dùng làm context để sinh câu trả lời.

Vì sao dùng Top-K?

Ta không đưa tất cả mọi thứ cho LLM (quá nhiều context = tốn kém và chậm). Top-K giữ lại những phần hữu ích nhất mà vẫn hiệu quả.

☆ Ví dụ

User Query

Paracetamol có tác dụng phụ gì?

→

Embed Query

[0.14, -0.26, 0.68, ...]

→

Search Vector DB

→

Top-3 Results

1. Chunk #15
2. Chunk #8
3. Chunk #2

→

Retrieved Context

(Các chunk này được gửi cho LLM)

Takeaway

Truy xuất tốt = Context tốt = Câu trả lời tốt.

Truy xuất chính là cầu nối giữa dữ liệu của bạn và LLM.

RAG - Retrieval-Augmented Generation

☆ Quy trình sinh câu trả lời

Sau khi truy xuất, LLM dùng context truy xuất được + câu hoi của người dùng đê'sinh ra câu tra'lời có căn cứ và chính xác.

Mục tiêu: Giúp LLM tạo ra câu tra'lời dựa trên dữ kiện và đúng trọng tâm.

① User Query

Người dùng đặt câu hoi.

→

② Retrieved Context

Top-K chunk liên quan được truy xuất.

...

→

③ Build Augmented Prompt

Gộp câu hoi với context truy xuất được thành một prompt duy nhất.

Prompt = Query + Context

→

④ LLM (Generate)

LLM đọc augmented prompt và sinh ra câu tra'lời.

→

⑤ Câu trả lời cuối

Câu tra'lời được tra'vê cho người dùng.

☆ Context truy xuất được sẽ đăn dắt LLM tạo ra câu tra'lời cụ thê', chính xác và có thông tin thật hậu thuẫn.

☆ Bên trong LLM diễn ra gì?

- LLM đọc augmented prompt.
- Nó hiểu ý định của người dùng và context được cung cấp.
- Nó dùng kha'năng suy luận + các dữ kiện được cung cấp đê'tạo câu tra'lời phù hợp.
- Nó tránh bịa ra thông tin không có trong context.
- Kết qua'là câu tra'lời rõ ràng, hữu ích và có căn cứ.

Best Practices khi Generate

- ✓ Dùng prompt rõ ràng, ngắn gọn.
- ✓ Chi'đưa vào context thực sự liên quan.
- ✓ Giới hạn kích thước context ở mức cần thiết.
- ✓ Khuyên khích mô hình nói "Tôi không biết" nếu câu tra'lời không có trong context.

Ví dụ: Augmented Prompt

User Query:

Paracetamol có tác dụng phụ gì?

Retrieved Context (Top-2 Chunks):

- Tác dụng phụ thường gặp: buồn nôn, nôn, phát ban.
- Liều cao có thê'gây tôn thương gan.

↓

Augmented Prompt:

Q: Paracetamol có tác dụng phụ gì?

Context:

1. Tác dụng phụ thường gặp: buồn nôn, nôn, phát ban.
2. Liều cao có thê'gây tôn thương gan.

Ghi nhớ!

Kiến thức cua LLM rất rộng, nhưng không phai lúc nào cũng cập nhật hoặc chính xác. RAG bô'sung kiến thức thời gian thực, theo từng lĩnh vực đê'câu tra'lời trở nên đáng tin cậy.

Điểm mấu chốt

Generation trong RAG = **Trí thông minh của LLM** + **Kiến thức truy xuất được**.

Sự kết hợp này cho ra câu tra'lời chính xác, đúng trọng tâm và dựa trên dữ kiện!

RAG - Retrieval-Augmented Generation

☆ RAG trong thực tế

Cùng xem một hệ thống RAG hoạt động thế nào trong một tình huống thực tế.

Vì sao dùng RAG?

- ✓ Hiểu được tài liệu nội bộ
- ✓ Luôn dùng thông tin mới nhất
- ✓ Giảm hallucination
- ✓ Cho câu trả lời chính xác, đáng tin

Use Case: Hỏi đáp trên tài liệu công ty

Hãy tưởng tượng bạn có rất nhiều tài liệu nội bộ (quy định, hướng dẫn nhân sự, báo cáo, FAQ, v.v.) và bạn muốn đặt câu hỏi về chúng.

Luồng hoạt động đầu-cuối

① User	② Retrieve	③ Relevant Chunks	④ Augment	⑤ LLM (Generate)	⑥ Answer	⑦ User
Người dùng hỏi: "Chính sách nghỉ phép của công ty là gì?"	Hệ thống tìm trong vector DB và lấy ra các chunk liên quan. (Top-K Results)	Những phần thông tin liên quan nhất được trả về. • Chunk #1 • Chunk #4 • Chunk #9	Gộp câu hỏi + các chunk thành một augmented prompt. Prompt = Query + Retrieved Context	LLM đọc prompt và sinh ra câu trả lời.	Câu trả lời chính xác, có căn cứ được gửi tới người dùng.	Người dùng nhận câu trả lời và hài lòng!

Ví dụ chi tiết

① User Query:	"Chính sách nghỉ phép của công ty mình thế nào?"
② Retrieval:	Hệ thống lấy Top-3 chunk liên quan nhất từ kho tài liệu.
③ Chunks:	• Chunk 1: "Nhân viên được hưởng 20 ngày phép có lương mỗi năm..." • Chunk 4: "Nghỉ ốm cần có giấy chứng nhận của bác sĩ..." • Chunk 9: "Nghỉ việc riêng dưới 3 ngày không cần duyệt trước..."
④ Augment:	Gộp câu hỏi + các chunk → Một prompt duy nhất
⑤ LLM sinh:	LLM đọc prompt → hiểu → trả lời.
⑥ Kết quả:	"Nhân viên được hưởng 20 ngày phép có lương mỗi năm. Nghỉ ốm cần giấy chứng nhận của bác sĩ. Nghỉ việc riêng dưới 3 ngày không cần duyệt trước."

Lợi ích trong tình huống này

- ☆ Câu trả lời đến từ chính tài liệu công ty của bạn.
- ☆ Luôn cập nhật (thêm tài liệu mới → cập nhật kiến thức).
- ☆ Không hallucination (LLM chỉ dùng context được cấp).
- ☆ Nhân viên nhận được câu trả lời chính xác ngay lập tức.
- ☆ Tiết kiệm thời gian cho HR và các phòng ban.

Ứng dụng thực tế của RAG

- ✓ Kho kiến thức / tài liệu nội bộ công ty
- ✓ Chăm sóc khách hàng dựa trên tài liệu sản phẩm
- ✓ Hỏi đáp trên văn bản pháp lý
- ✓ Tra cứu bài báo khoa học, tài liệu học thuật
- ✓ Ghi chú cá nhân và trợ lý học tập
- ✓ Chatbot FAQ / hệ thống Helpdesk

Điểm mấu chốt

RAG mang dữ liệu của bạn đến với mô hình, chứ không phải ngược lại. Nó kết hợp điểm mạnh của cả hai: Kiến thức của LLM + Độ chính xác từ dữ liệu của bạn.

Ghi nhớ!

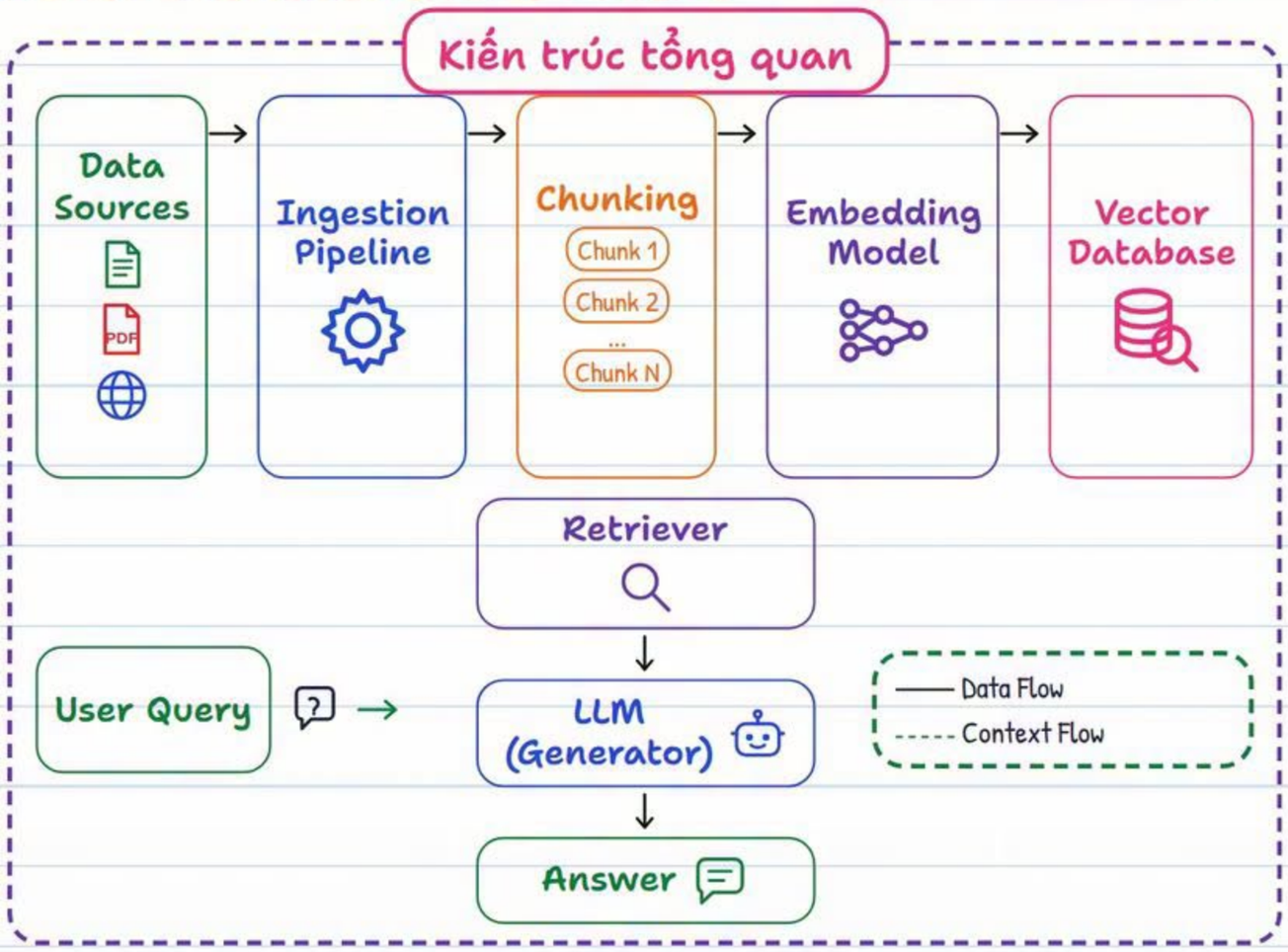
Dữ liệu tốt hơn → Truy xuất tốt hơn
Truy xuất tốt hơn → Context tốt hơn
Context tốt hơn → Câu trả lời tốt hơn

RAG - Retrieval-Augmented Generation

Các thành phần của hệ thống RAG

Một hệ thống RAG điển hình gồm những thành phần chính sau:

1. **Data Source:** Nơi chứa tài liệu thô của bạn (PDF, website, cơ sở dữ liệu, v.v.)
2. **Ingestion Pipeline:** Nạp, làm sạch và chuẩn bị dữ liệu.
3. **Chunking:** Chia văn bản thành các phần nhỏ hơn.
4. **Embedding Model:** Chuyển các chunk văn bản thành vector.
5. **Vector Database:** Lưu trữ và đánh chỉ mục các vector.
6. **Retriever:** Tìm các chunk liên quan nhất cho một câu hỏi.
7. **LLM (Generator):** Sinh câu trả lời cuối cùng dựa trên context truy xuất được.



RAG kết nối dữ liệu của bạn (dữ kiện) với sức mạnh của LLM (suy luận và ngôn ngữ) để cho ra câu trả lời chính xác, cập nhật và đáng tin cậy.

Ưu điểm của RAG

- ✓ Dùng thông tin thật, cập nhật.
- ✓ Giảm hallucination.
- ✓ Câu trả lời có nguồn dẫn chứng.
- ✓ Làm việc với dữ liệu riêng tư/nội bộ.
- ✓ Dễ mở rộng và bảo trì.

Hạn chế của RAG

- ✗ Chất lượng phụ thuộc vào chất lượng dữ liệu.
- ✗ Nếu truy xuất sai, câu trả lời có thể thiếu sót.
- ✗ Cần chiến lược chunking và embedding tốt.
- ✗ Phát sinh thêm chi phí và độ trễ (truy xuất + sinh nội dung).

Best Practices

- Giữ dữ liệu sạch và có cấu trúc rõ ràng.
- Chọn kích thước chunk hợp lý (không quá nhỏ, không quá lớn).
- Chọn đúng embedding model.
- Luôn truy xuất Top-K chunk liên quan.
- Cung cấp đủ context nhưng tránh nhiễu.
- Đánh giá và giám sát hệ thống thường xuyên.

Lưu ý quan trọng

RAG **không** chỉ là cắm thêm một vector database. Cốt lõi là tìm đúng thông tin và dùng nó một cách khôn ngoan.

Điểm mấu chốt

RAG = Retrieval (Tìm) + Augmentation (Bổ sung) + Generation (Trả lời).
Tìm đúng context, cho ra đúng câu trả lời!

Pro Tip: RAG là tương lai của việc xây dựng các ứng dụng AI đáng tin cậy - biết rõ dữ liệu của bạn và trả lời một cách thông minh.