**1. LLM Fundamentals**

* Khái niệm LLM, ứng dụng chính.

LLM: Large Language Model

Text-base -> Ability để train to a LLM

Applications:

* Programming
* Summarizing
* Text Generating
* QnA
* Etc.
* Tokenization:

Tách 1 text thành những từ “token” độc lập với nhau.

VD: **What is the tallest building** -> *What is the tall est building*

Có rất nhiều cách tokenize khác nhau, không có một quy chuẩn cố định

Embedding: Vectors:

Turn the token into numbers that represent the word. Let the computer make it easier to understand the corresponding position in the sentence and how the words relate to each other.

* Attention:

Các vector của token “giao tiếp” với nhau, tính toán để đưa ra value mới

VD: Từ “model” có 2 context:

Model trong “**Machine learning model**” và “**fashion model**”. Khi này, bước attention sẽ có trách nhiệm cho việc phát hiện xem các ndung liên quan đến ý nghĩa của từ “model” đó, và cho biết từ model đó nên sẽ được dùng trong trường hợp nào.

A black screen with white text and blue text

AI-generated content may be incorrect.

Tiếp tục cho ví dụ trên, model trong **machine learning model** và trong **fashion** **model** mặc dù giống nhau chữ “model”, nhưng qua bước Attention, khi liên kết các từ trong context, các vector của những token liên quan có giá trị không giống nhau khiến cho token “**model**” của 2 từ đó có 2 giá trị vector khác nhau.

* Transformers:

Bước 1: Chuyển từ (word) thành số (Word embedding)

Main Idea: Dùng 1 mạng NN đơn giản có 1 input cho từng từ và ký hiệu trong bộ vocab mình muốn dùng

A diagram of a mathematical equation

AI-generated content may be incorrect.

VD như mình có 1 bộ từ vựng chưa các input để tạo các câu đơn giản như “Let’s go” hoặc “to go”, và 1 input cho ký hiệu <EOS> aka End of Sentence (End of Sequence). Mỗi input này là 1 token. Các input này sẽ kết nối với hàm kích hoạt, vd như case này có 2 hàm kích hoạt.

Giả sử ta muốn đổi từ Let’s, mình sẽ đặt Let’s = 1 và các giá trị còn lại đều là 0. Let’s khi đi qua trọng số 1.87 (trọng số lấy từ back propagation), đi qua hàm kích hoạt đầu tiên sẽ ra 1.87, hàm kích hoạt s2 sẽ ra 0.09 (giả sử hàm kích hoạt có phương trình f(x) = x). Còn các giá trị còn lại khi có giá trị = 0, khi đi qua hàm kích hoạt đều bằng 0. Từ đó, khi các giá trị đầu ra 1.87 và 0.09 là những con số đại diện cho từ “Let’s”. Cấu trúc và trọng số đều giữ nguyên, chỉ đổi giá trị input cho từng trường hợp. VD muốn biết từ go chuyển về dạng số là gì thì đặt go = 1.

Step 2: Positional Encoding: mã hóa để set thứ tự câu từ.

VD: “Tui ăn pizza” và “Pizza ăn tui” mặc dù có cùng số từ lẫn cùng các từ với nhau nhưng khác ý nghĩa -> Kiểm tra word order.

**Lưu ý**: Có nhiều các thực hiện Positional Encoding, nhưng trong ví dụ sẽ sử dụng cách thông dụng nhất.

VD như câu “**tui ăn pizza**”. Ta có bộ từ vựng mới và tạo ra được 4 embedding cho mỗi từ (vd như 1 câu có bao nhiêu từ (chính xác hơn là token) sẽ có bấy nhiêu word embedding, tính luôn symbol cho mỗi từ).

A diagram of a graph

AI-generated content may be incorrect.

Trong trường hợp đang ví dụ, giá trị đại diện cho thứ tự từ đến từ chuỗi luân phiên các đường sin và cosine. Mỗi đường cho ta biết giá trị của từng embedding của từ.

Vì Sine và Cosine liên tục lặp lại, sẽ có case 2 từ có cùng vị trí:

A diagram of a graph

AI-generated content may be incorrect.

VD như từ **ăn** với từ **pizza**, embedding đầu có cùng giá trị là -0.9, nhưng mỗi đường sẽ dài hơn cho embedding position cao hơn. Càng nhiều giá trị embedding, độ rộng của đường cong càng tăng, nên mặc dù có một số lúc có cùng giá trị, nó đều cho ra các giá trị riêng biệt. Sau khi tính ra các embedding, cộng 2 cái đó lại sẽ xác định được vị trí của từ.

* Self-attention (Cách transformer hiểu mối quan hệ của từ)

VD: “The **pizza** cam out of the **oven** and **it** tasted good!”. It nó có thể là oven, hoặc pizza -> Self-attention tính ra điểm giống nhau giữa các từ, kể cả chính nó, lần lượt cho từng từ trong câu. Tính xong, nó sẽ được dùng để xác định Transformer encode các từ như thế nào. VD nếu mình thấy nhiều câu về pizza và từ “it” nó có mối liên hệ gần với pizza hơn oven -> Similarity score cho pizza sẽ có tác động lớn hơn đối cách từ “**it**” được encode trong TF

* Cách LLM sinh câu trả lời (next-token prediction).
* Prompting cơ bản: zero-shot, few-shot.

**2. Hạn chế của LLM:**

* ChatGPT in the past, chỉ cập nhật được thông tin đến 2021, hỏi những câu sau 2021 là lỏ.
* **Bịa thông tin, đặc biệt là chủ đề ít phổ biến (info hallucination)**
* Không truy cập được dữ liệu cụ thể
* Chí phí cập nhật cao

**3. RAG (Retrieval Augmented Generation)**

* RAG là gì: LLM + tri thức bên ngoài (external knowledge)

Dễ hiểu: thay vì chỉ trả lời dựa vô “trí nhớ” aka dữ liệu đã nạp, thủ thư sẽ truy xuất thông tin liên quan từ web, sách, v.v., add thông tin vào dữ liệu và cung cấp câu trả lời chính xác

RAG có thể áp dụng cho các tình huống, bài toán nào

* Tạo ndung
  + Viết báo, blog có tài liệu tham chiếu để đảm bảo chính xác
* Virtual assistant, Chatbot chuyên dụng
  + CSKH, trả lời FAQ, hỗ trợ kỹ thuật
  + Etc
* Nghiên cứu
  + Cho LLM đọc paper, báo cáo, sách, tổng hợp nhanh cho người dùng
* Phân tích thị trước & phát triển sản phẩm
* Dịch vụ “recommendations
  + RAG dùng để bổ sung CSDL sản phẩm vào LLM

Bài toán:

* Legal tech -> đọc hợp đồng, luật -> tóm tắt, highlight điều khoản quan trọng
* Healthcare: bác sĩ hỏi -> RAG lấy guideline y khoa, nghiên cứu mới hỗ trợ chẩn đoán
* Enterprise search: tìm ttin trong file PDF, docx, slides…
* Các loại RAG:

Hiện tại tìm được 10:

Naïve RAG:

* Simplest approach
* Take documents, break into chunks, then when query comes in, the system the most relevant chunks and pases to a LM to generate answers

Advanced RAG:

Modular RAG

Corective RAG

Speculative RAG

Fusion RAG

Agentic RAG

Self RAG

Graph RAG

RadioRAG

* Mô hình triển khai RAG

How RAG works?

A diagram of a process

AI-generated content may be incorrect.

Câu hỏi: Làm sao để bước Retrieval luôn tốt? Qua một model (BERT model or sth, idk)

* Embeding là gì, vector database là gì ?
* **Embedding** : Chuyển các từ ngữ, âm thanh, hình ảnh, etc. về dạng số (hoặc dãy số)
* **Vector database**: Quản lý dữ liệu dạng **vector**. Tìm kiếm + so sánh dữ liệu dựa độ tương đồng.

Ex: How to make a phở bòa?

Trước tiên nó sẽ được chuyển thành vector số. Sau đó trong vector db sẽ so sánh vector câu hỏi với vector tài liệu khác để trả về kết quả phù hợp.

A diagram of information processing

AI-generated content may be incorrect.

Chunking:

Thông dụng nhất là fixed-size aka set một lượng token bất kỳ cho 1 chunk. Thường là lụm max (1024: llama-text-embed-v2, 8196: text-embedding-3-small). Mỗi model embed tokenize text khác nhau nên cần ước lượng chính xác.

* + Cứ start bằng fixed-size chunking, lỏ thì tính tiếp
  + If not using fixed size, use hybrid (semantic + token window)

Embedding: Maybe sẽ dùng của OpenAI

Store: MongoDB atlas

Generate: LLM