# Mô hình ngôn ngữ lớn (LLM)



LLM (Large Language Model – Mô hình ngôn ngữ lớn) là mô hình AI được huấn luyện trên khối dữ liệu văn bản khổng lồ để hiểu, tạo ra và dự đoán văn bản giống con người.

Cùng học nào!

## 1. LLM là gì?

LLM là một loại foundation model, dùng deep learning (chủ yếu là kiến trúc Transformer) để xử lý và tạo ra ngôn ngữ tự nhiên.

## 2. Khả năng chính

- ★ Hiểu ngữ cảnh
- ★ Tạo ra văn bản
- ★ Trả lời câu hỏi
- ★ Tóm tắt nội dung
- ★ Dịch ngôn ngữ
- ★ Viết code
- ★ Suy luận & giải quyết vấn đề
- ★ Và nhiều hơn nữa!

## 3. LLM hoạt động thế nào?

- Nhận văn bản đầu vào (prompt)
- ↓
- Chuyển văn bản thành token (số)
- ↓
- Xử lý token qua nhiều lớp Transformer
- ↓
- Dự đoán token tiếp theo
- ↓
- Lặp lại đến khi tạo xong phản hồi

Sinh văn bản tự hồi quy (auto-regressive) – mỗi lần một token

LLM không chỉ đoán từ – chúng hiểu quy luật trong ngữ cảnh.

Hãy hình dung LLM như một công cụ gợi ý tự động siêu thông minh – nó hiểu cả câu, chứ không chỉ từ cuối cùng.

## 4. Một số LLM tiêu biểu

- GPT series (OpenAI)
- Llama series (Meta)
- Gemini (Google)
- Claude (Anthropic)
- Mistral (Mistral AI)
- Và nhiều mô hình khác...

## 5. Thành phần cốt lõi của LLM

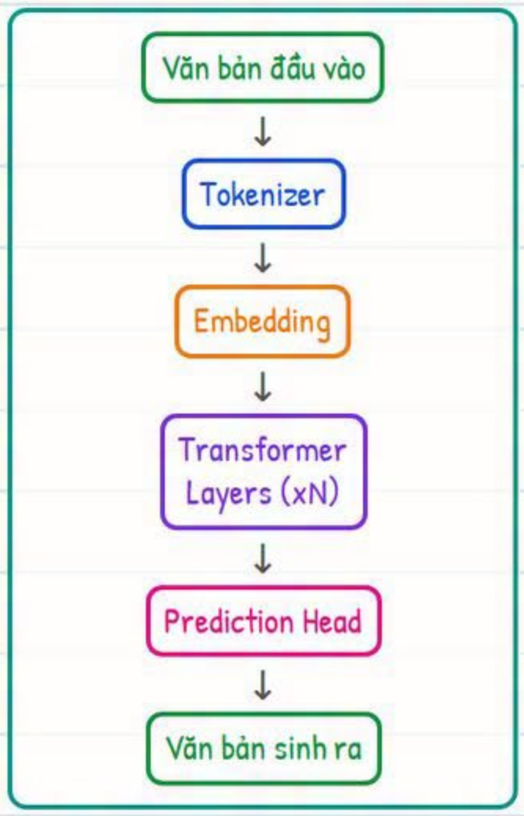
→ Tokenizer
→ Tách văn bản thành token

→ Embedding
→ Chuyển token thành vector số

→ Transformer Layers
→ Hiểu quan hệ trong văn bản

→ Prediction Head
→ Dự đoán xác suất token kế tiếp

→ Output Generation
→ Sinh văn bản cuối, từng token một




## 6. Ví von đời thực

Hãy tưởng tượng bạn đã đọc hàng triệu cuốn sách. Khi ai đó cho bạn một câu mở đầu, bạn dự đoán câu tiếp theo hợp lý nhất → đó chính là điều một LLM đang làm!

LLM học từ dữ liệu, không phải từ quy tắc.
Nhiều dữ liệu → hiểu tốt hơn.
Hiểu tốt hơn → trả lời tốt hơn.

# LLM hoạt động thế nào?

Ý tưởng lớn: Dự đoán từ tiếp theo!

2

Về bản chất, LLM là những bộ dự đoán token tiếp theo. Cho trước một đoạn văn bản, chúng dự đoán token có khả năng cao nhất đứng sau.

## 1. Mục tiêu huấn luyện

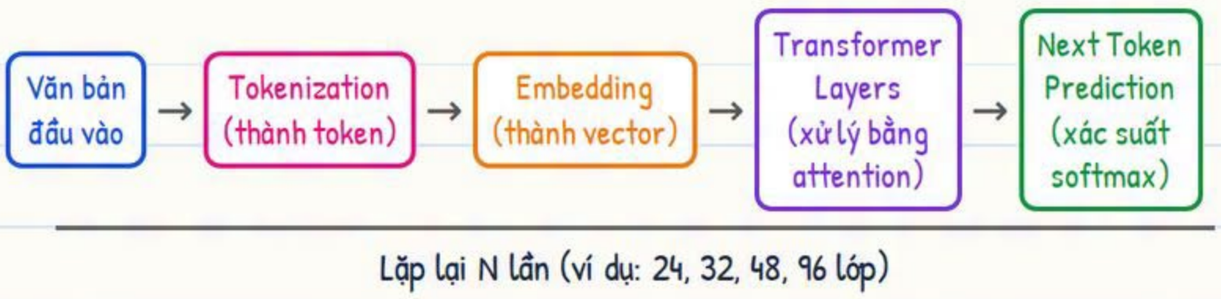
LLM được huấn luyện để tối thiểu hóa sai số dự đoán.

> Mục tiêu: Cho trước ngữ cảnh, dự đoán đúng token tiếp theo.

Chúng học từ hàng tỉ ví dụ và rút ra quy luật, mối quan hệ cùng tri thức về thế giới. ☆

## 2. Bộ khung Transformer

Hầu hết LLM dùng kiến trúc Transformer (được giới thiệu trong bài báo "Attention Is All You Need").

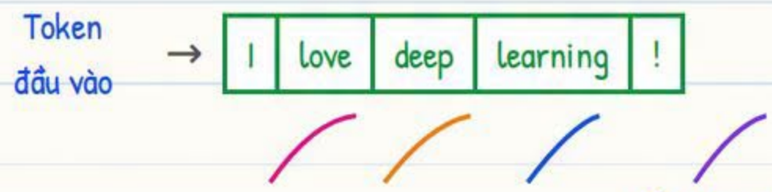



## 3. Luồng xử lý từng bước

1. Input Text → Bạn đưa vào một prompt.
2. Tokenization → Văn bản được tách thành token (từ / subword).
3. Embedding → Token được chuyển thành vector (dạng số).
4. Transformer Layers → Self-attention giúp các token "trò chuyện" với nhau và hiểu ngữ cảnh.
5. Prediction → Mô hình tính xác suất cho mọi token kế tiếp có thể.
6. Chọn token → Token khả năng cao nhất (hoặc được lấy mẫu) sẽ được thêm vào.
7. Lặp lại → Bước 3-6 lặp đến khi phản hồi hoàn tất.

## 4. Self-Attention (Điều kỳ diệu)

Self-attention cho phép mô hình tập trung vào những token quan trọng trong đầu vào khi đưa ra dự đoán.




> Nhờ đó mô hình nắm được các phụ thuộc xa và ngữ cảnh rộng.

## 5. Suy luận / Sinh văn bản

- Bắt đầu bằng một prompt.
- Dự đoán token tiếp theo.
- Nối token đó vào văn bản.
- Lặp đến khi gặp điều kiện dừng:

Stopping Tokens:

- ✓ <EOS>
- ✓ <|endoftext|>
- ✓ Xuống dòng / stop word tùy chỉnh

- ✓ Sinh ra token kết thúc
- ✓ Đạt độ dài tối đa
- ✓ Gặp stop word / quy tắc dừng

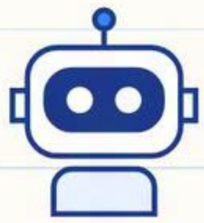
## 6. Temperature & Sampling

Điều khiển mức độ sáng tạo của đầu ra.

| Thiết lập | Hiệu quả | Trường hợp dùng |
|---|---|---|
| Temp thấp (0.1 - 0.3) | Đầu ra xác định, tập trung | Sự kiện, hỏi đáp, tóm tắt |
| Temp vừa (0.5 - 0.7) | Cân bằng sáng tạo và ổn định | Chatbot đa dụng |
| Temp cao (0.8 - 1.2) | Ngẫu nhiên và sáng tạo hơn | Truyện, thơ, brainstorming |

Tóm lại: LLM đọc, hiểu quy luật và dự đoán token tiếp theo lặp đi lặp lại, cho đến khi tạo ra một phản hồi có ý nghĩa và hữu ích!

# Kiến trúc Transformer



Attention làm nên sức mạnh!

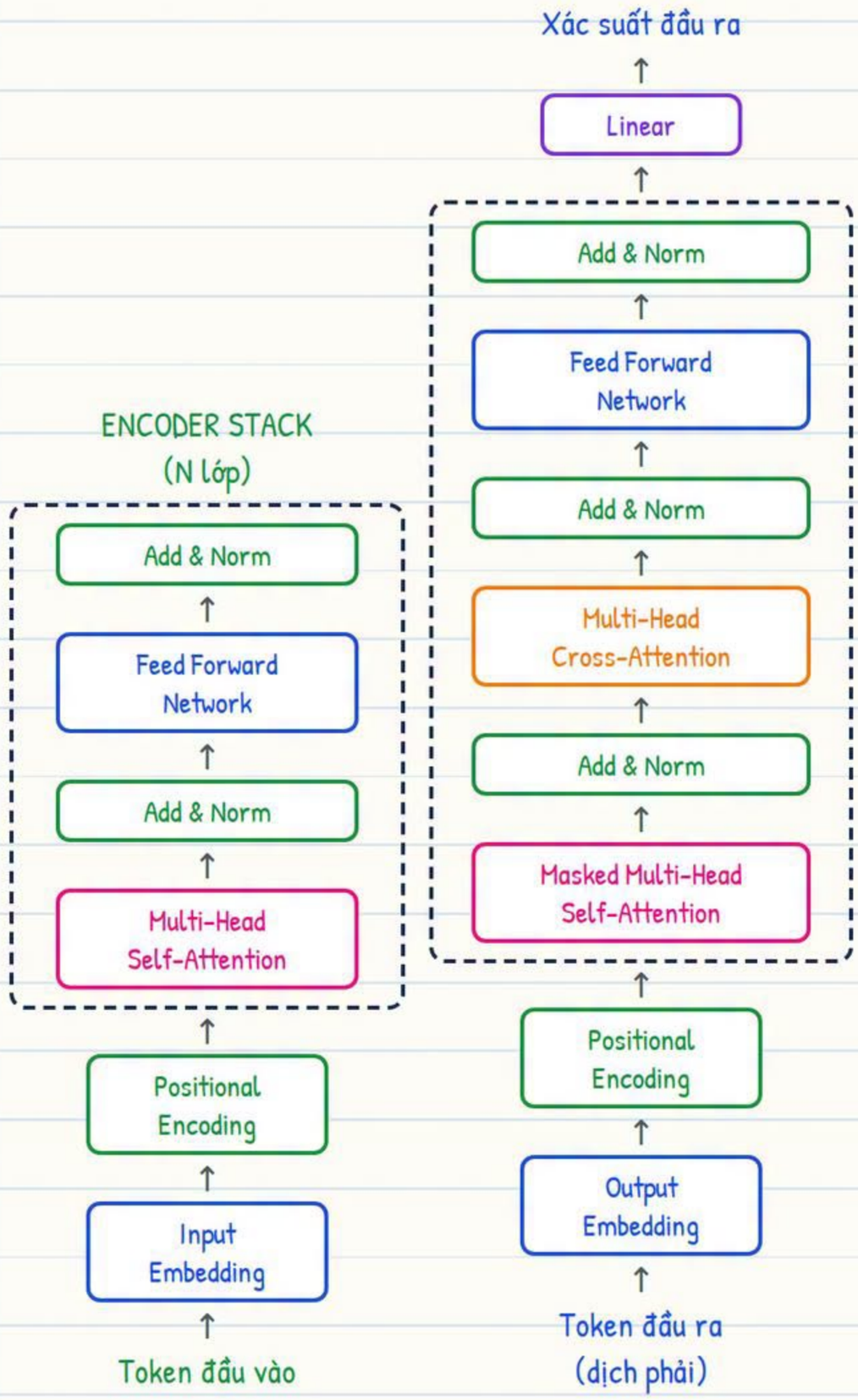


Kiến trúc Transformer, được giới thiệu trong bài báo "Attention Is All You Need" (2017), là bộ khung của hầu hết các LLM ngày nay.

## 1. Cấu trúc tổng thể

Transformer có hai phần chính:

- Encoder Stack
- Decoder Stack




N thường là 6, 12, 24 hoặc lớn hơn, tùy vào kích thước mô hình.

## 2. Các khối thành phần chính

Attention nhìn toàn bộ ngữ cảnh!

★ **Self-Attention**

Cho phép mọi token "chú ý" tới tất cả token khác trong chuỗi.

★ **Multi-Head Attention**

Chạy attention nhiều lần song song để nắm bắt các quan hệ khác nhau.

★ **Feed Forward Network (FFN)**

Mạng fully connected đơn giản, áp dụng riêng cho từng vị trí.

★ **Add & Norm**

Residual connection (Add) + Layer Normalization (Norm) giúp huấn luyện ổn định.

★ **Positional Encoding**

Thêm thông tin vị trí vào embedding, vì attention không phân biệt thứ tự.

Transformer loại bỏ RNN và chỉ dùng attention. Nhờ vậy việc huấn luyện diễn ra song song, nhanh hơn và xử lý phụ thuộc xa tốt hơn.

## 3. Attention (Trái tim)

Cho Query (Q), Key (K), Value (V):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- $QK^T$ cho ra điểm tương đồng.
- Chia cho $\sqrt{d_k}$ để ổn định.
- softmax cho ra trọng số.
- Nhân với V để có đầu ra.

Q = Tôi đang tìm gì?

K = Bạn có gì?

V = Đây là thông tin!

## 4. Vì sao Transformer hiệu quả đến vậy?

- ✓ Nắm bắt được phụ thuộc xa
- ✓ Song song hóa rất tốt
- ✓ Mở rộng tốt với dữ liệu & tham số khổng lồ
- ✓ Là nền tảng của các LLM mạnh mẽ

Nhiều dữ liệu + nhiều compute = LLM thông minh hơn

# Giải thích Self-Attention

Điều kỳ diệu đằng sau khả năng hiểu ngữ cảnh!

4

Self-Attention là đột phá then chốt giúp LLM hiểu ngữ cảnh bằng cách liên hệ mọi từ với mọi từ khác trong câu.

## 1. Trực giác

Khi đọc một câu, ta chú ý vào một số từ nhiều hơn những từ khác.

Ví dụ:

"Minh đã tới New York vì anh ấy rất thích pizza."

Khi dự đoán từ "pizza", mô hình nên chú ý nhiều hơn tới "thích", chứ không phải "New York".

## 2. Cách hoạt động (về mặt toán học)

Với một chuỗi token đầu vào, ta tạo ba vector cho mỗi token:

| Query (Q) | Key (K) | Value (V) |
|---|---|---|
| Tôi đang tìm gì? | Bạn đại diện cho điều gì? | Bạn có thông tin gì? |

Attention được tính như sau:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

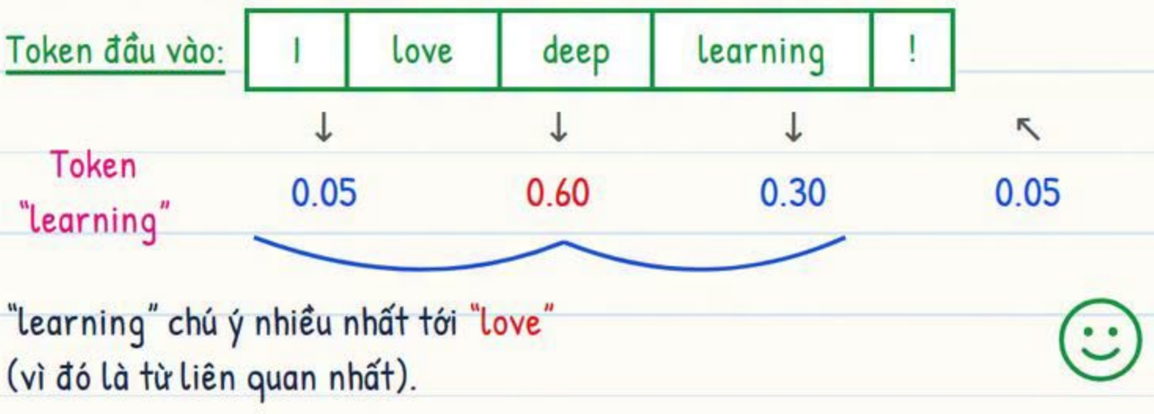
So khớp Q với tất cả K, rồi dùng điểm số để trộn các V

## 3. Từng bước một

1. Mỗi token được chuyển thành Q, K, V.
2. Tính điểm tương đồng: $Q \times K^T$
3. Chia tỉ lệ $1/\sqrt{d_k}$ để ổn định.
4. Dùng softmax để có trọng số attention.
5. Nhân trọng số với V để ra đầu ra.
6. Thực hiện song song cho mọi token.

Mỗi token tự quyết định sẽ chú ý bao nhiêu tới từng token khác!

## 4. Ví dụ nhỏ



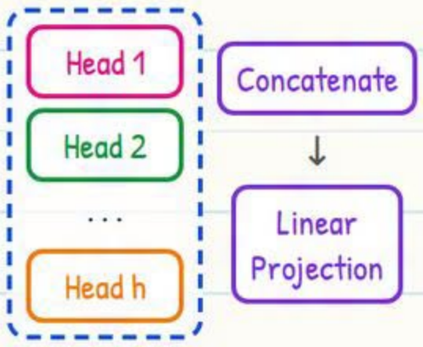

"learning" chú ý nhiều nhất tới "love" (vì đó là từ liên quan nhất).

## 5. Multi-Head Attention

Thay vì một attention, Transformer chạy h head song song, mỗi head học một loại quan hệ khác nhau.

Lợi ích:

- ★ Nắm bắt nhiều loại quan hệ
- ★ Biểu đạt phong phú hơn
- ★ Cải thiện hiệu năng mô hình




## 6. Positional Encoding

Vì Transformer không có tính hồi quy, ta thêm thông tin vị trí vào embedding của token.

Giúp mô hình biết được thứ tự của các từ.

Hai phương pháp phổ biến:

- Sinusoidal Positional Encoding (bài báo gốc)
- Learned Positional Embeddings

★ Không có thông tin vị trí, "chó cắn người" và "người cắn chó" sẽ trông y hệt nhau!

## 7. Điểm chính cần nhớ

- ✓ Self-Attention cho mô hình nhìn toàn bộ ngữ cảnh cùng lúc.
- ✓ Nó tính xem mỗi từ nên tập trung bao nhiêu vào từng từ khác.
- ✓ Song song hóa hoàn toàn → huấn luyện nhanh hơn.
- ✓ Là trái tim của mọi LLM hiện đại.

Self-Attention = Hiểu ngữ cảnh ở quy mô lớn!

## Ví von đời thực

Hãy tưởng tượng bạn đang ở trong một căn phòng ồn ào. Bạn nghe thấy tất cả mọi người, nhưng tập trung nhiều hơn vào người đang nói chuyện với bạn. Đó chính là self-attention!

# Huấn luyện LLM

Nhiều dữ liệu
+ nhiều compute
= LLM thông minh hơn! ♥



LLM được huấn luyện trên lượng dữ liệu văn bản khổng lồ (hàng tỉ từ) để có thể dự đoán token tiếp theo trong bất kỳ chuỗi nào.

## 1. Mục tiêu huấn luyện

Mục tiêu: Tối thiểu hóa khác biệt giữa token được dự đoán và token thực tế.

Nói đơn giản: Dạy mô hình đoán từ tiếp theo chính xác nhất có thể. ☺

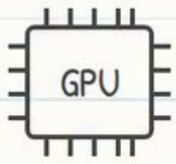
## 2. Hàm mất mát (Loss)

- LLM dùng Cross-Entropy Loss.
- Nó đo phân phối xác suất dự đoán cách phân phối thực tế bao xa.
- Loss càng thấp → dự đoán càng tốt.
- Công thức (cho một token):

$$Loss = -\log\left(P_{model}(correct\_token)\right)$$

## 3. Tối ưu hóa

- Optimizer thường dùng: AdamW (phổ biến nhất)
- Nó cập nhật trọng số để giảm loss.
- Cần rất nhiều compute (GPU / TPU).
- Việc huấn luyện có thể kéo dài hàng tuần hoặc hàng tháng!




## 4. Dữ liệu để huấn luyện

Chất lượng + đa dạng của dữ liệu rất quan trọng! ♥

Nguồn dữ liệu:

- Sách, bài viết, website
- Wikipedia
- Kho mã nguồn (ví dụ: GitHub)
- Diễn đàn, blog, tin tức, v.v.

Các bước tiền xử lý dữ liệu:

- ✓ Loại bỏ dữ liệu trùng lặp
- ✓ Làm sạch văn bản (bỏ nhiễu, ký tự đặc biệt...)
- ✓ Tokenize văn bản
- ✓ Tạo các chuỗi huấn luyện

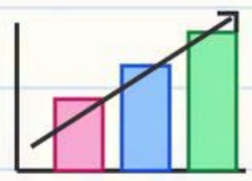
## 5. Quy trình huấn luyện (tổng quan)

Văn bản đầu vào → Tokenize → Mô hình (Transformer) → Dự đoán token kế → Tính loss → Cập nhật trọng số

Lặp lại hàng triệu / hàng tỉ lần

## 6. Quy mô rất quan trọng

★ Nhiều dữ liệu hơn → tổng quát hóa tốt hơn
★ Mô hình lớn hơn → học được nhiều quy luật hơn
★ Nhiều compute hơn → huấn luyện sâu & lâu hơn



Mở rộng quy mô = Hiệu năng tăng!

## 7. Ví von thực tế

Hãy tưởng tượng một học sinh đọc hàng triệu cuốn sách. Dần dần, cậu ấy bắt đầu dự đoán rất chính xác điều gì sẽ xuất hiện tiếp theo trong bất kỳ câu nào. LLM học đúng như vậy! ♥

## 8. Điểm chính cần nhớ

- ✓ LLM học bằng cách dự đoán token tiếp theo.
- ✓ Huấn luyện cần dữ liệu + compute khổng lồ.
- ✓ Cross-Entropy Loss dẫn đường cho việc học.
- ✓ Quy mô (dữ liệu, mô hình, compute) tạo nên trí tuệ.

Huấn luyện mới chỉ là khởi đầu. Sau đó, ta còn fine-tune, align và làm cho LLM thật sự hữu ích & an toàn khi dùng trong thực tế! ♥

# Chi tiết về Tokenization

Mô hình hiểu số, chứ không hiểu văn bản thô!



Tokenization là quá trình chuyển văn bản thô thành các token mà mô hình có thể hiểu được.

## 1. Token là gì?

Một token có thể là:

- Một từ
- Một phần của từ
- Một dấu câu
- Hoặc thậm chí một ký tự đơn lẻ

Các mô hình khác nhau dùng phương pháp tokenization khác nhau.

## 2. Ví dụ

Văn bản: "I love deep learning!"

Kết quả tokenization có thể là:

| I | love | deep | learn | ing | ! |
|---|---|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| 40 | 2293 | 2401 | 4087 | 62 | 0 |

(Đây là các token ID)

## 3. Subword Tokenization

Hầu hết LLM hiện đại (như GPT, Llama) dùng Subword Tokenization (ví dụ BPE hoặc WordPiece). Cách này tách những từ hiếm thành các mảnh nhỏ hơn.

Ví dụ:

Văn bản: "unbelievable"

Có thể được tách thành:

| un | bel | iev | able |
|---|---|---|---|
| 345 | 789 | 1234 | 567 |

Giúp mô hình xử lý được từ hiếm hoặc từ mới!

## 4. Vì sao dùng Subword Tokenization?

- ★ Xử lý được mọi từ (kể cả từ chưa từng gặp)
- ★ Giảm kích thước bộ từ vựng
- ★ Hiệu quả hơn
- ★ Phù hợp hơn với nhiều ngôn ngữ

Có thể bạn chưa biết:
ChatGPT (GPT-4) dùng một tokenizer tên là tiktoken (dựa trên BPE).

## 5. Luồng xử lý Tokenization

Văn bản thô (chuỗi ký tự) → Tiền xử lý (làm sạch) → Tokenizer (tách thành token) → Token ID (dạng số) → Đầu vào mô hình (cho embedding)

Ví dụ: "I love AI." → [I] [love] [AI] [.] → [40, 2293, 527, 13] → Embedding Vectors (đầu vào cho mô hình)

## 6. Token đặc biệt

LLM dùng một số token đặc biệt:

| Token | Công dụng |
|---|---|
| <BOS> | Bắt đầu chuỗi |
| <EOS> | Kết thúc chuỗi |
| <PAD> | Đệm (cho bằng độ dài) |
| <UNK> | Token không xác định |

Những token này giúp mô hình hiểu cấu trúc!

## 7. Tầm quan trọng

- ✓ Tokenization quyết định cách mô hình "nhìn" văn bản.
- ✓ Tokenization tốt = hiểu tốt hơn
- ✓ Ảnh hưởng tới tốc độ, chi phí và độ chính xác
- ✓ Token nhỏ hơn → chuỗi dài hơn nhưng chi tiết hơn
- ✓ Token lớn hơn → chuỗi ngắn hơn nhưng ít chi tiết

Tóm lại: Tokenization biến văn bản lộn xộn thành một danh sách số gọn gàng, để LLM có thể hiểu, xử lý và dự đoán!

# Embedding trong LLM

Những từ có nghĩa gần nhau sẽ có vector nằm gần nhau!



Embedding chuyển token (từ / subword) thành vector số. Những vector này nắm bắt ý nghĩa của từ, để mô hình có thể làm toán trên chúng!

## 1. Embedding là gì?

- Là biểu diễn vector dày đặc của một token.
- Ánh xạ token rời rạc → vector liên tục.
- Số chiều thường là 128, 256, 512, 768, 1024, 1536, 4096, ...
- Được học trong quá trình huấn luyện.

Ví dụ:

| Token | Embedding (vector 5 chiều) |
|---|---|
| king | [ 0.21, -0.13, 0.45, 0.02, -0.31] |
| queen | [ 0.22, -0.11, 0.46, 0.01, -0.32] |
| man | [-0.31, 0.27, -0.14, 0.33, 0.19] |
| woman | [-0.29, 0.25, -0.16, 0.34, 0.18] |

Từ giống nhau có vector giống nhau!

## 2. Embedding được học thế nào?

Trong quá trình huấn luyện, mô hình học embedding sao cho các token xuất hiện trong ngữ cảnh tương tự sẽ có vector tương tự.

Mục tiêu:

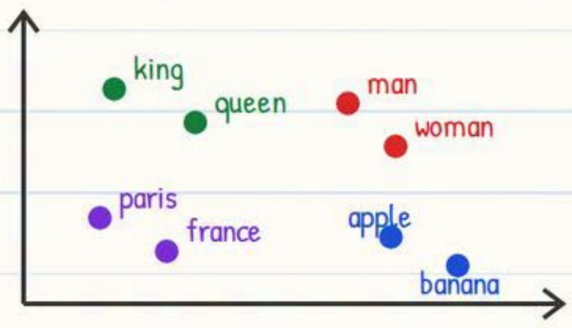
Từ dùng trong tình huống giống nhau → vector giống nhau
Từ dùng trong tình huống khác nhau → vector khác nhau

Ví von:

Giống như tấm bản đồ đặt các thành phố trong không gian 2D sao cho thành phố gần nhau thì nằm gần nhau, embedding biểu diễn từ trong không gian nhiều chiều.

## 3. Không gian embedding

Ví dụ không gian nhiều chiều (vẽ 2D cho đơn giản):




Những từ gần nghĩa nằm gần nhau trong không gian!

## 4. Số chiều rất quan trọng

- ★ Số chiều thấp → ít biểu đạt, dễ mất nghĩa
- ★ Số chiều cao → biểu đạt tốt hơn, nắm nhiều sắc thái hơn
- ★ Kích thước phổ biến: 128, 256, 384, 512, 768, 1024, 1536, 4096
- ★ Mô hình càng lớn thường dùng số chiều càng cao.

Embedding là nền tảng cho mọi khả năng hiểu của LLM.
Embedding tốt hơn → ngữ cảnh tốt hơn → dự đoán tốt hơn!

## 5. Embedding được dùng thế nào


Mô hình làm việc trên vector, không phải văn bản thô!

## 6. Ví dụ minh họa

Văn bản: "I love deep learning"
Tokens: [ I, love, deep, learning ]
Token IDs: [ 40, 2293, 2401, 4087 ]

Embeddings:
[ 0.12, -0.07, ..., 0.33]
[ 0.21, 0.11, ..., -0.11]
[-0.05, 0.32, ..., 0.10]
[ 0.08, -0.21, ..., 0.27]

Mỗi hàng là vector của một token!

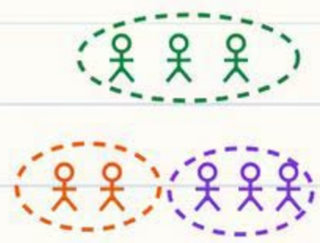
## 7. Điểm chính cần nhớ

- ✓ Embedding biến từ thành số.
- ✓ Chúng nắm bắt ý nghĩa và các mối quan hệ.
- ✓ Từ gần nghĩa nằm gần nhau trong không gian vector.
- ✓ Chúng được học trong quá trình huấn luyện.
- ✓ Embedding là bước đầu tiên để hiểu văn bản!

## 8. Ví von thực tế

Hãy nghĩ về embedding như các nhóm bạn trong trường học.

- Bạn thân (nghĩa giống nhau) ngồi cạnh nhau.
- Các nhóm khác nhau thì ngồi cách xa.
- Không gian embedding cho thấy ai giống ai!



Embedding chính là ngôn ngữ của ý nghĩa.
Embedding càng tốt, LLM càng thông minh!

# Cách LLM sinh văn bản

LLM không "suy nghĩ" như con người. Chúng dự đoán token kế tiếp có khả năng cao nhất! ♥



LLM sinh văn bản mỗi lần một token, dựa trên xác suất.

## 1. Quy trình sinh văn bản

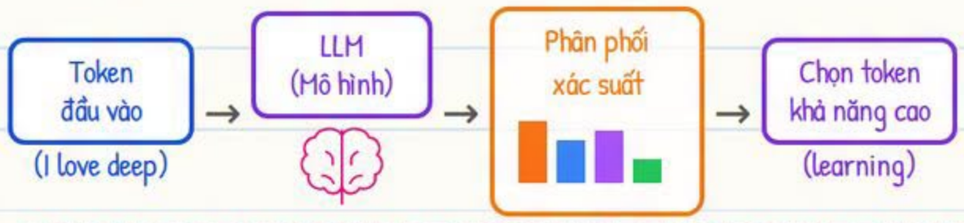
1. Bạn đưa vào một prompt.
2. Mô hình chuyển nó thành token.
3. Mô hình dự đoán token tiếp theo.
4. Token đó được thêm vào chuỗi.
5. Bước 3-4 lặp đến khi gặp điều kiện dừng.

Điều kiện dừng:

- Sinh ra token <EOS>
- Đạt độ dài tối đa
- Người dùng dừng việc sinh

## 2. Dự đoán token kế tiếp

Cho một chuỗi token, mô hình xuất ra phân phối xác suất trên toàn bộ bộ từ vựng cho token tiếp theo.




Công thức (khái niệm):

$$P(\text{token kế tiếp} \mid \text{các token trước đó})$$

Mô hình dùng toàn bộ token trước đó làm ngữ cảnh.

## 3. Chiến lược decoding

Các chiến lược khác nhau quyết định cách chọn token tiếp theo.

| Chiến lược | Cách hoạt động | Trường hợp dùng |
|---|---|---|
| Greedy Search | Chọn token có xác suất cao nhất | Nhanh, xác định |
| Top-k Sampling | Chọn ngẫu nhiên trong k token đầu | Đa dạng hơn |
| Top-p (Nucleus) Sampling | Chọn từ tập nhỏ nhất có tổng xác suất ≥ p | Đa dạng cân bằng |
| Temperature | Điều khiển độ ngẫu nhiên (cao = ngẫu nhiên hơn) | Kiểm soát sáng tạo |

★ Không có chiến lược nào tốt nhất cho mọi trường hợp! ♥

## 4. Ví dụ từng bước

Prompt: "I love deep"

| Bước | Chuỗi hiện tại | Token kế (kèm xác suất) | Token chọn |
|---|---|---|---|
| 1 | I love deep | learning (0.42), thinking (0.22), work (0.10), blue (0.05) ... | learning |
| 2 | I love deep learning | everyday (0.35), a (0.25), so (0.10), it (0.07) ... | everyday |
| 3 | I love deep learning everyday | . (0.60), ! (0.15), and (0.08), because (0.04) ... | . |

Kết quả cuối: "I love deep learning everyday."

## 5. Điểm quan trọng

- ★ LLM không lưu tri thức như một cơ sở dữ liệu. Chúng sinh nội dung dựa trên quy luật đã học.
- ★ Chúng có thể sai một cách rất tự tin (hallucination).
- ★ Chất lượng prompt + ngữ cảnh ảnh hưởng lớn tới đầu ra.
- ★ Nhiều ngữ cảnh hơn → đầu ra tốt hơn (trong giới hạn).
- ★ Chiến lược decoding đổi cân bằng giữa sáng tạo và chính xác.

Rác vào = Rác ra!

## 6. Yếu tố ảnh hưởng tới việc sinh

- ✓ Kích thước mô hình & dữ liệu huấn luyện
- ✓ Chất lượng prompt / ngữ cảnh
- ✓ Chiến lược & tham số decoding (top-k, top-p, temperature)
- ✓ Số token tối đa được phép
- ✓ System instruction / vai trò (ví dụ: "Bạn là một trợ lý hữu ích")

Thay đổi nhỏ có thể dẫn tới đầu ra rất khác! ♥

## 7. Điểm chính cần nhớ

- ✓ LLM sinh văn bản bằng cách dự đoán token tiếp theo.
- ✓ Mọi thứ đều là token (từ, phần của từ, ký hiệu).
- ✓ Xác suất chi phối mọi quyết định.
- ✓ Prompt tốt hơn + chiến lược tốt hơn = kết quả tốt hơn.
- ✓ Hiểu quy trình → dùng LLM hiệu quả hơn!

## 8. Ví von đời thực

Hãy nghĩ về LLM như bàn phím gợi ý nhưng mạnh hơn rất nhiều. Bạn gõ "I love" → bàn phím gợi ý "you", "pizza", "coding"... Bạn chọn một → nó lại gợi ý từ tiếp theo. LLM cũng làm vậy, chỉ khác là ở quy mô khổng lồ!


LLM là những bộ dự đoán quy luật rất mạnh mẽ.
Hiểu rõ quy trình, dùng công cụ khôn ngoan và tạo ra những điều tuyệt vời!

# Kiến trúc của LLM

Transformer là động cơ đằng sau hầu hết các LLM mạnh mẽ! ♥



Hầu hết LLM hiện đại đều dựa trên kiến trúc Transformer. Cùng phân tích các khối thành phần chính nhé!

## 1. Kiến trúc tổng thể

LLM đi theo cấu trúc Encoder-Decoder (Decoder-only).

Token đầu vào → Token Embedding → Transformer Blocks (x N) → LM Head + Softmax → Xác suất token kế tiếp

★ Decoder-only nghĩa là mô hình chỉ dùng phần decoder của transformer, xếp chồng N lần.

## 2. Token Embedding

Chuyển mỗi token thành một vector dày đặc (biểu diễn được học).

| Token | I | love | deep | learning |
|---|---|---|---|---|
| Embedding (Vector) | [0.12, -0.21, 0.35, ...] | [0.35, 0.18, -0.11, ...] | [0.07, -0.33, -0.11, ...] | [0.42, 0.11, -0.31, ...] |

+ Positional Encoding được thêm vào để mô hình biết thứ tự của các token.

Embedding nắm bắt ý nghĩa và quan hệ giữa các token. ♥

## 3. Transformer Block (Decoder Block)

Mỗi block được lặp lại N lần và có hai phần chính:

1. Masked Multi-Head Self-Attention
2. Feed Forward Neural Network (FFN)

Masked Multi-Head Self-Attention
↓
Add & Norm
↓
Feed Forward Network (FFN)
↓
Add & Norm

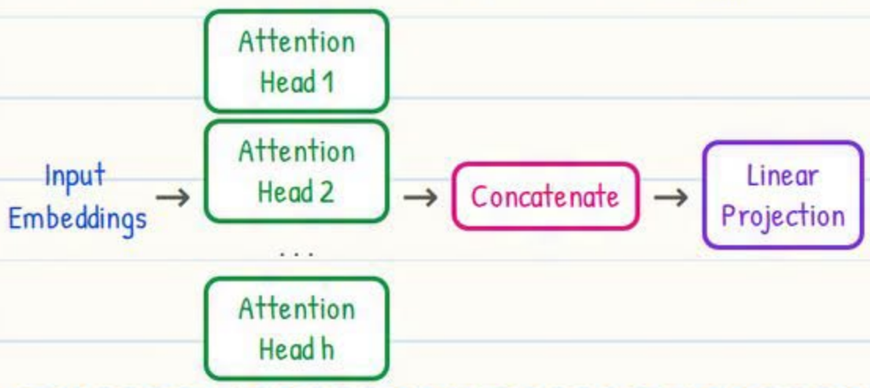
Ý chính:

- Masked attention đảm bảo mô hình không thể "nhìn thấy" token tương lai.
- Add & Norm = Residual connection + Layer Normalization (giúp huấn luyện ổn định).

♥

## 4. Masked Multi-Head Self-Attention

- Cho phép mỗi token chú ý tới tất cả token trước đó.
- "Masked" nghĩa là các token tương lai bị che đi.
- Multi-head = mô hình nhìn thông tin từ nhiều không gian biểu diễn khác nhau.




Vì sao cần nhiều head? Các head khác nhau tập trung vào những mối quan hệ khác nhau. ★

## 5. Feed Forward Network (FFN)

Áp dụng độc lập cho từng vị trí. Thường là MLP 2 lớp kèm hàm kích hoạt (ví dụ GELU).

Linear ($d_{model} \rightarrow d_{ff}$) → GELU (Activation) → Linear ($d_{ff} \rightarrow d_{model}$)

Thêm tính phi tuyến và tăng năng lực của mô hình. ☺

## 6. LM Head + Softmax

- Trạng thái ẩn cuối cùng từ block cuối được đưa vào một lớp linear (LM Head).
- Sau đó Softmax chuyển logits thành xác suất trên toàn bộ bộ từ vựng.


Token có xác suất cao nhất chính là token kế tiếp được dự đoán. ★

## 7. Siêu tham số quan trọng

- ★ $d_{model}$ : Kích thước ẩn (ví dụ 768, 1024, 4096)
- ★ N : Số transformer block (ví dụ 24, 32, 80)
- ★ h : Số attention head (ví dụ 12, 16, 32)
- ★ $d_{ff}$ : Kích thước ẩn của feed-forward (ví dụ 4x $d_{model}$)
- ★ Vocab Size : Số token duy nhất (ví dụ 32K, 50K, 100K)
- ★ Max Context : Độ dài chuỗi tối đa (ví dụ 2K, 4K, 8K)

Nhiều tham số hơn → mạnh hơn, nhưng cần nhiều dữ liệu & compute hơn! ♥

## 8. Ghép tất cả lại

Cho một prompt, LLM dự đoán token lần lượt từng cái một.

Prompt (Văn bản) → Mô hình (Transformer) → Dự đoán token kế → Nối token & lặp lại

Quá trình tự hồi quy này tiếp tục đến khi gặp token dừng hoặc đạt độ dài tối đa. ☺

Kiến trúc Transformer đã thay đổi NLP mãi mãi.
LLM mở rộng ý tưởng này lên hàng tỉ tham số và dữ liệu khổng lồ!

# Huấn luyện, Suy luận & Ứng dụng

LLM chỉ được huấn luyện một lần, nhưng được dùng hàng tỉ lần! ♥

10

Sau kiến trúc là phần kỳ diệu – huấn luyện mô hình, dùng nó để đưa ra dự đoán (inference), và áp dụng vào đời sống thực tế!

## 1. Pre-Training (Nền tảng)

LLM trước tiên được pre-train trên khối dữ liệu văn bản khổng lồ (web, sách, bài viết, code, v.v.).

### Điều gì diễn ra?


- ✓ Mô hình học quy luật, ngữ pháp, kiến thức, khả năng suy luận.
- ✓ Không có nhiệm vụ cụ thể – chỉ dự đoán token tiếp theo.
- ✓ Được huấn luyện trên hàng tỉ token.

★ Mục tiêu: Làm cho mô hình thông minh một cách tổng quát.

## 2. Fine-Tuning / Instruction Tuning

Mô hình đã pre-train được fine-tune trên dữ liệu instruction – response chất lượng cao để trở nên hữu ích và an toàn.

Ví dụ dữ liệu:

| Instruction | Response |
|---|---|
| Quang hợp là gì? | Quang hợp là quá trình cây xanh... |
| ... | ... |

Biến một mô hình thông minh thành trợ lý hữu ích! ♥

Lợi ích:

- ★ Tuân theo chỉ dẫn
- ★ Câu trả lời tốt hơn
- ★ Đầu ra an toàn hơn
- ★ Phù hợp với con người hơn

## 3. Học tăng cường từ phản hồi của con người (RLHF)

Phản hồi của con người giúp mô hình đưa ra câu trả lời tốt hơn và an toàn hơn.

Đầu ra mô hình → Phản hồi con người → Reward Model → Reinforcement Learning → Mô hình tốt hơn

RLHF giúp:

- Giảm đầu ra có hại / thiên lệch
- Tăng tính hữu ích
- Phù hợp với giá trị của con người

Tóm lại: Con người dẫn dắt mô hình trở nên tốt hơn từng bước. ☺

## 4. Inference (Đưa ra dự đoán)

Inference là khi mô hình đã huấn luyện được dùng để sinh phản hồi cho prompt mới.

Lúc này mô hình không học – nó chỉ dự đoán! ♥

User Prompt → LLM (Mô hình) → Phản hồi sinh ra

Điểm chính:

- Không cập nhật trọng số trong lúc inference.
- Nhanh và dùng được theo thời gian thực.
- Chất lượng phụ thuộc vào huấn luyện + prompt.

## 5. Ứng dụng trong thực tế

LLM có thể dùng theo vô số cách. Dưới đây là một vài ứng dụng phổ biến:

| Tạo nội dung | Chatbot / Trợ lý | Giáo dục | Code & phát triển | Kinh doanh | Sáng tạo & giải trí |
|---|---|---|---|---|---|
| • Bài blog | • Chăm sóc khách hàng | • Giải thích khái niệm | • Sinh code | • Tóm tắt | • Thơ / truyện |
| • Truyện | • Trợ lý cá nhân | • Tóm tắt chủ đề | • Hỗ trợ debug | • Phân tích dữ liệu | • Lời bài hát |
| • Email | • Bot hỏi đáp FAQ | • Hỗ trợ học tập | • Giải thích code | • Viết báo cáo | • NPC trong game |
| • Nội dung mạng xã hội | • Trợ lý ảo | • Học ngoại ngữ | • Viết tài liệu | • Tạo ý tưởng | • Brainstorming |

★ Từ học sinh đến doanh nghiệp – ai cũng có thể hưởng lợi từ LLM! ♥

## 6. Thách thức & hạn chế

- ✗ Có thể tạo ra thông tin sai hoặc "ảo giác".
- ✗ Đầu ra đôi khi thiên lệch hoặc thiếu công bằng.
- ✗ Cần compute & dữ liệu khổng lồ để huấn luyện.
- ✗ Vấn đề riêng tư, bảo mật & nguy cơ lạm dụng.
- ✗ Không thay thế được trí tuệ con người!

Ghi nhớ: LLM là công cụ mạnh mẽ, không phải nhà tiên tri hoàn hảo. ☺

## 7. Tổng kết

- ✓ Pre-train → Học từ dữ liệu khổng lồ
- ✓ Fine-tune → Học cách trở nên hữu ích
- ✓ RLHF → Con người giúp nó tốt hơn
- ✓ Inference → Mô hình dự đoán, không học
- ✓ Ứng dụng → Vô vàn khả năng!

LLM chính là tương lai của AI.
Hiểu chúng hôm nay, kiến tạo tương lai ngày mai! ♥