



# NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN

## 504045

# HIỂU, FINE-TUNE & ĐÁNH GIÁ MÔ HÌNH TRANSFORMER TRÊN DỮ LIỆU TIẾNG VIỆT

Dự án Cuối kỳ

Giảng viên hướng dẫn:  
**PGS.TS Lê Anh Cường**

Thành viên thực hiện:  
**Đoàn Thống Lĩnh**  
**Nguyễn Cao Kỳ**  
**Nguyễn Hà My**

# CẤU TRÚC BÁO CÁO

## I - Tổng quan

## II - Mô hình Transformer

- Encoder-only
- Encoder-Decoder
- Decoder-only

## III - Mô tả dữ liệu

## IV - Phương pháp & chi tiết huấn luyện

## V - Đánh giá kết quả

## I - TỔNG QUAN

- Tổng quan dự án Generative QA tiếng Việt
- Áp dụng Transformer:
  - Encoder-only,
  - Decoder-only,
  - Encoder-Decoder
- Xây dựng dữ liệu QA từ Wikipedia + LLM

## Giới thiệu & Mục tiêu dự án

- Tập trung vào bài toán Generative QA tiếng Việt
- Sử dụng 3 kiến trúc Transformer tiêu biểu
- Sinh dữ liệu QA bằng mô hình Gemini từ Wikipedia
- Fine-tune mô hình & đánh giá hiệu năng

## Giới thiệu & Mục tiêu dự án

- Generative QA → mô hình tự sinh câu trả lời hoàn chỉnh
  - *Nguồn dữ liệu QA tiếng Việt còn hạn chế*
  - Dự án tạo dữ liệu + fine-tune 3 loại Transformer
- 
- Nghiên cứu sâu kiến trúc Transformer
  - Xây dựng bộ dữ liệu QA tiếng Việt
  - Fine-tune các mô hình trên cùng dataset
  - Đánh giá & so sánh hiệu năng

## Bài toán lựa chọn

- Bài toán Generative Question Answering
- Hệ thống sinh câu trả lời dựa vào context
- Ứng dụng: Chatbot, giáo dục, tìm kiếm thông minh

Lý do lựa chọn:

- Áp dụng được cho 3 kiến trúc Transformer
- QA là tác vụ trung tâm của NLP
- Nhiều ứng dụng thực tiễn & tiềm năng

## QA tổng quát đa lĩnh vực (open-domain QA)

Open-domain Question Answering (ODQA) là dạng bài toán mà người dùng có thể đặt bất kỳ câu hỏi nào, thuộc bất kỳ lĩnh vực nào (khoa học, lịch sử, đời sống, IT, văn học, y tế, pháp luật,...).

Mô hình phải:

- tự suy luận,
- tự tìm thông tin liên quan,
- tự tổng hợp và sinh câu trả lời.

Môi trường open-domain thường mô phỏng giống trợ lý ảo / chatbot tri thức.

## QA tổng quát đa lĩnh vực (open-domain QA)

Open-domain Question Answering (ODQA) là dạng bài toán mà người dùng có thể đặt bất kỳ câu hỏi nào, thuộc bất kỳ lĩnh vực nào (khoa học, lịch sử, đời sống, IT, văn học, y tế, pháp luật,...).

Mô hình phải:

- tự suy luận,
- tự tìm thông tin liên quan,
- tự tổng hợp và sinh câu trả lời.

Môi trường open-domain thường mô phỏng giống trợ lý ảo / chatbot tri thức.

Đặc điểm	Mô tả
<b>Không giới hạn chủ đề</b>	Người dùng có thể hỏi bất kỳ lĩnh vực nào
<b>Nguồn kiến thức mở</b>	Từ Internet, Wikipedia, hoặc kiến thức nội bộ của mô hình
<b>Cần khả năng suy luận và tổng hợp</b>	Không chỉ trích xuất từ câu cụ thể
<b>Phụ thuộc vào LLM</b>	Vì LLM chưa kiến thức rộng trong tham số mô hình
<b>Sinh câu trả lời dạng tự nhiên</b>	Không chỉ trả lời ngắn, mà có thể giải thích, phân tích

## Dữ liệu sử dụng

- Nguồn: Wikipedia tiếng Việt
- Tạo dữ liệu tổng hợp bằng mô hình Gemini
- Tham khảo: [UIT-ViQuAD \(2020\)](#)

### Quy trình tạo dữ liệu:

- Crawl văn bản từ Wikipedia
- Chia dữ liệu thành nhiều lô (~200 context/lô)
- Sinh QA bằng LLM
- Tổng hợp thành dataset hoàn chỉnh

## Cấu trúc dữ liệu

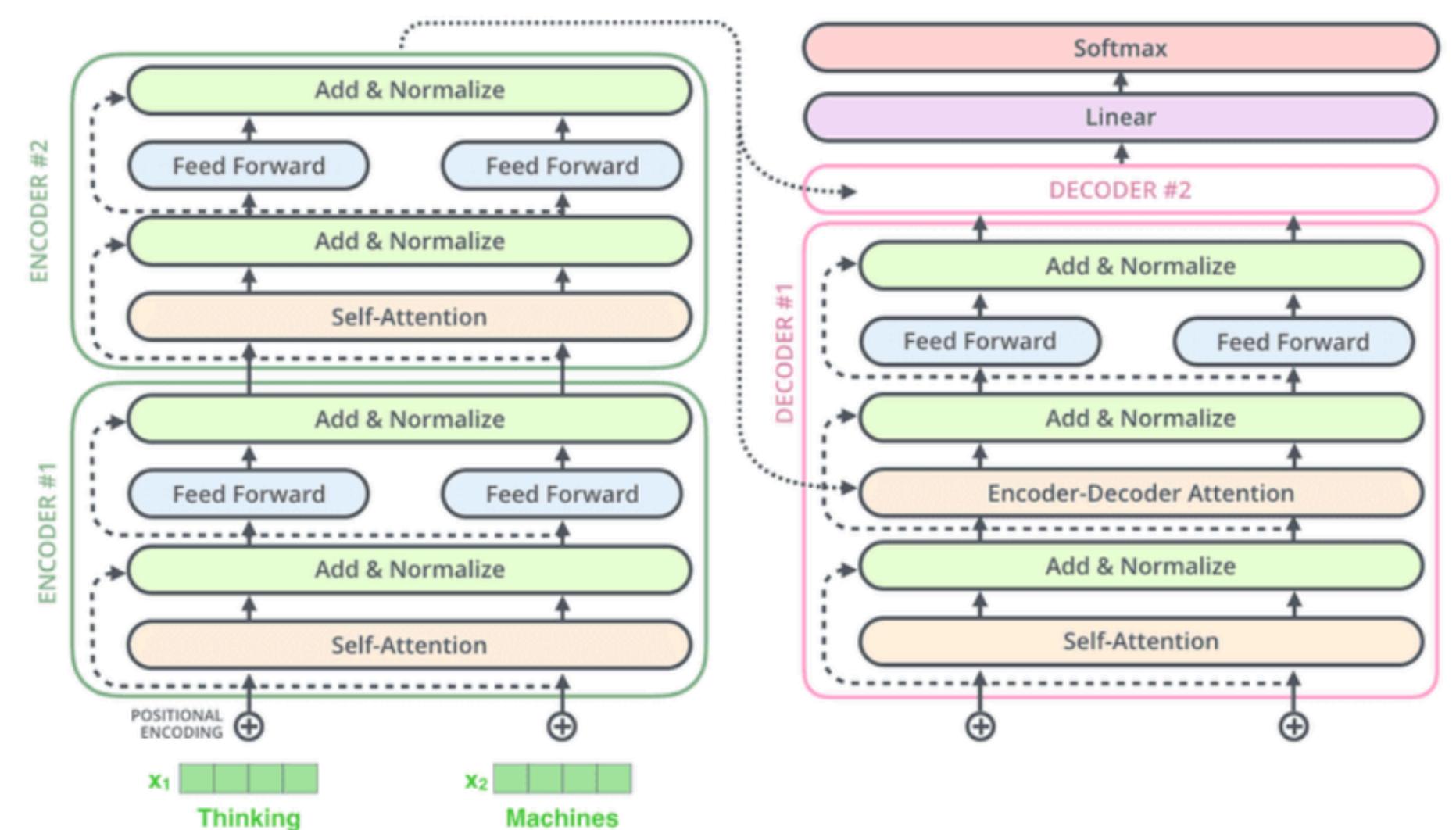
- context – question – answer
- Ví dụ: “Ai là bạn cùng lớp của Yuu?” → Koizumi

### Mục tiêu đầu ra:

- Bộ dữ liệu QA tiếng Việt chất lượng cao
- 3 mô hình Transformer đã fine-tune
- Đánh giá bằng metric & LLM-based evaluation
- So sánh ưu/nhược điểm từng kiến trúc

## II – MÔ HÌNH TRANSFORMER

- Transformer: kiến trúc Encoder–Decoder dựa hoàn toàn vào Attention
- Loại bỏ RNN, thay bằng *Self-Attention* song song
- Thêm Positional Encoding để giữ trật tự từ
- Cốt lõi: Self-Attention cho phép học phụ thuộc xa

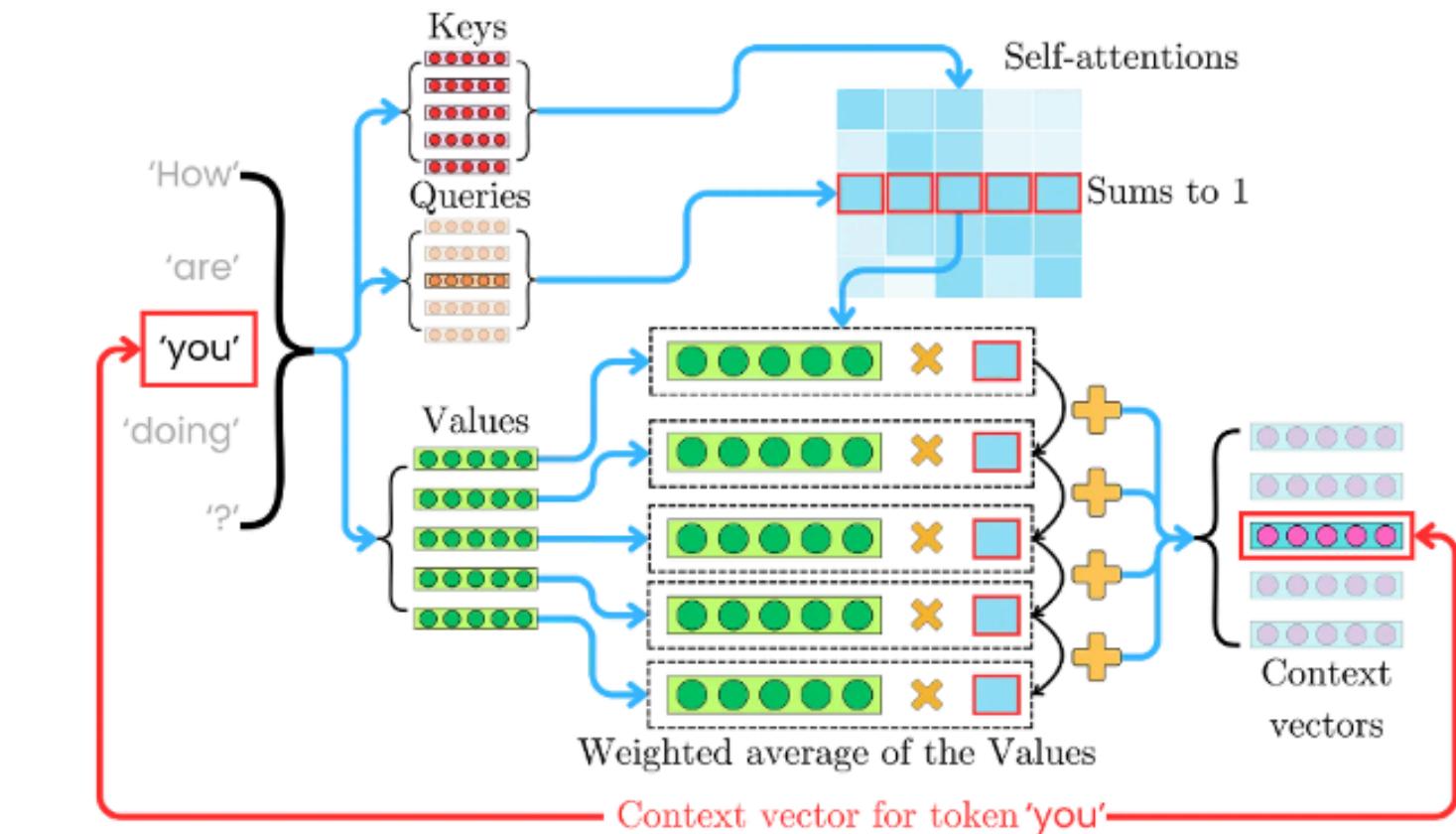
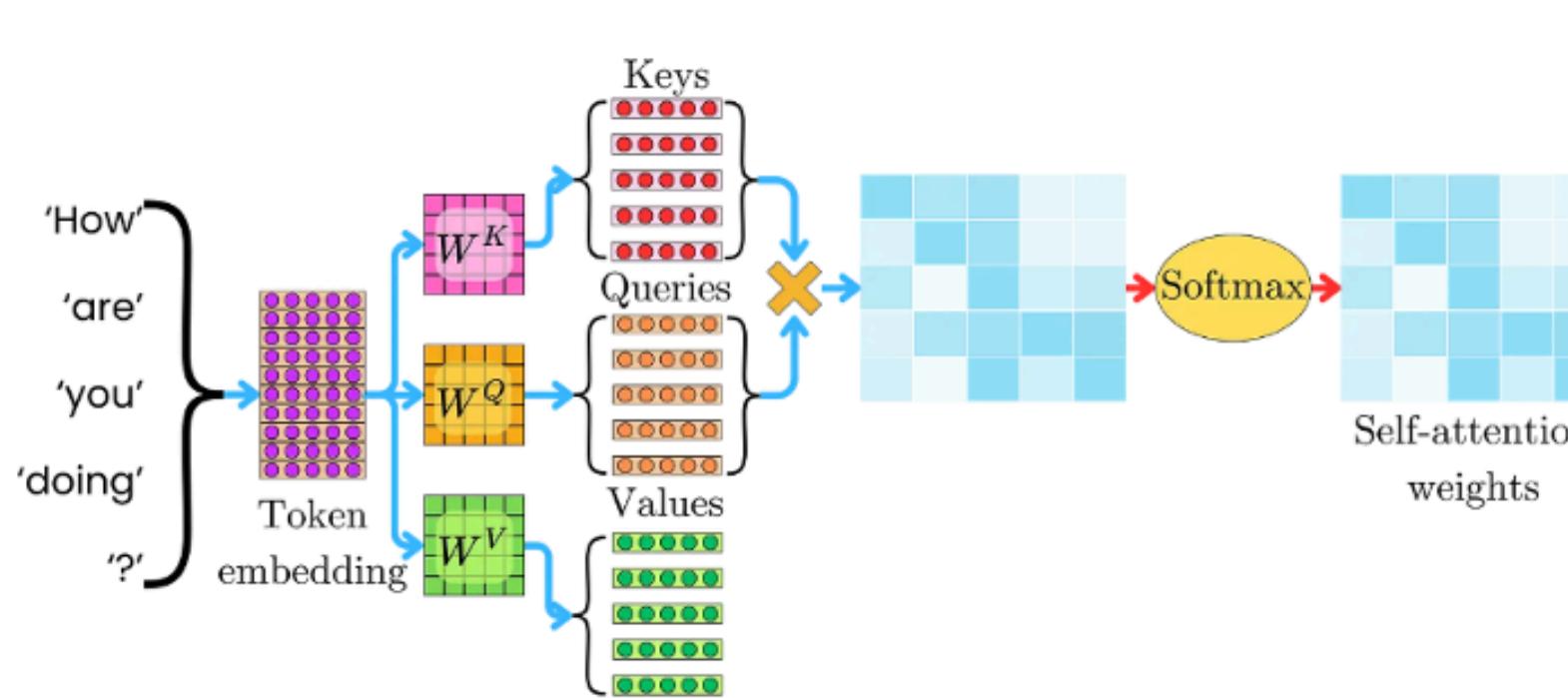


## Self-Attention – Công thức

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

- $QK^T$ : điểm chú ý
- Softmax: chuẩn hóa trọng số
- Nhân với  $V$ : tổng hợp thông tin
- Mô hình hóa phụ thuộc dài hạn hiệu quả

# Self-Attention – Công thức



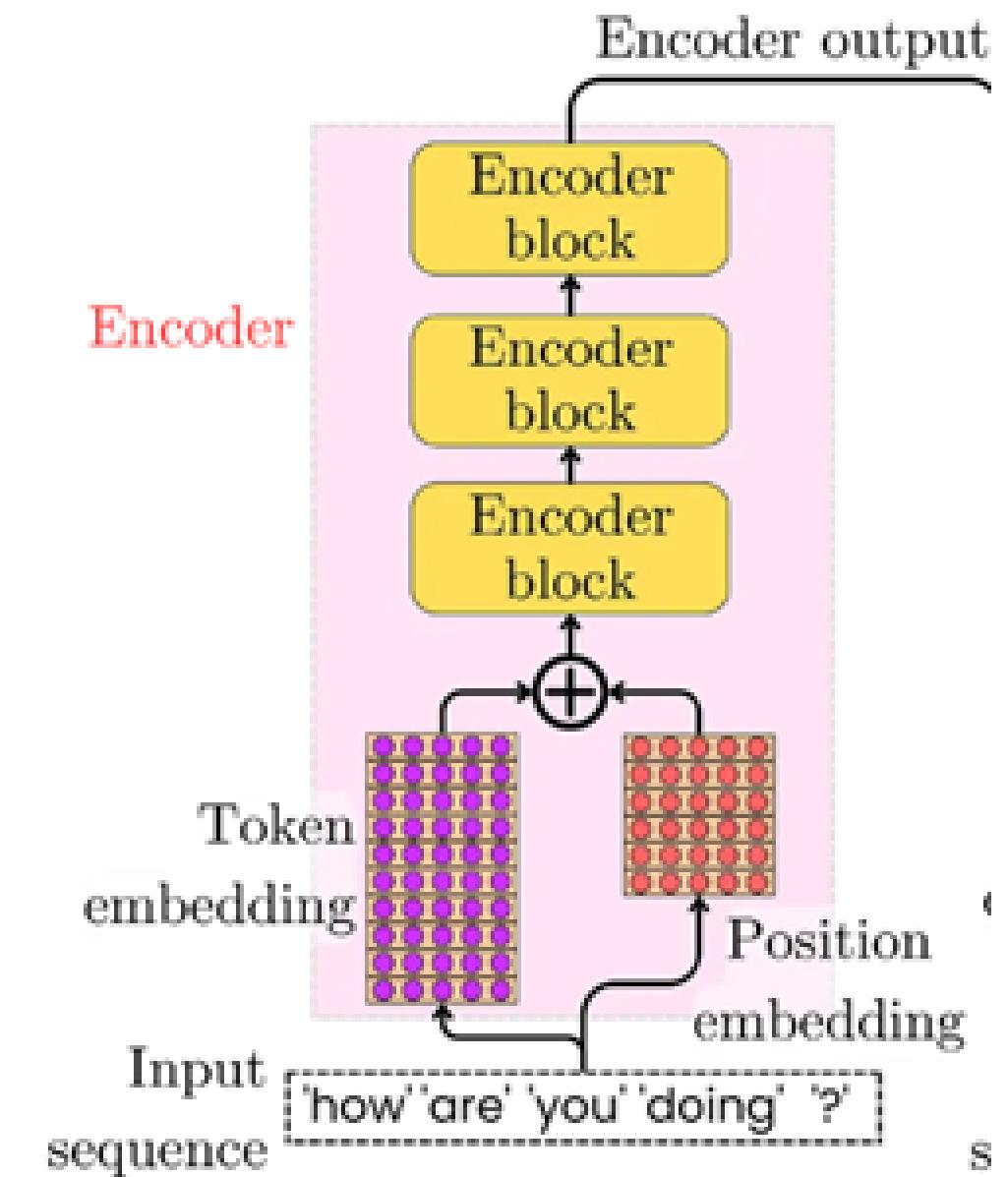
- Mỗi token → Ánh xạ sang Query, Key, Value
- Điểm chú ý =  $\text{Query}_i \cdot \text{Key}_j$
- Softmax → trọng số chú ý
- Tổng hợp Value theo trọng số để tạo vector ngữ cảnh
- Đầu ra token i phụ thuộc vào toàn bộ token khác

## Multi-Head Self-Attention

- Nhiều đầu attention chạy song song
- Mỗi đầu học một loại quan hệ ngữ nghĩa khác nhau
- Ghép lại để tạo biểu diễn giàu ngữ cảnh
- Ví dụ từ 'bank': hiểu đúng nghĩa dựa vào các từ liên quan như deposits/safe hoặc river

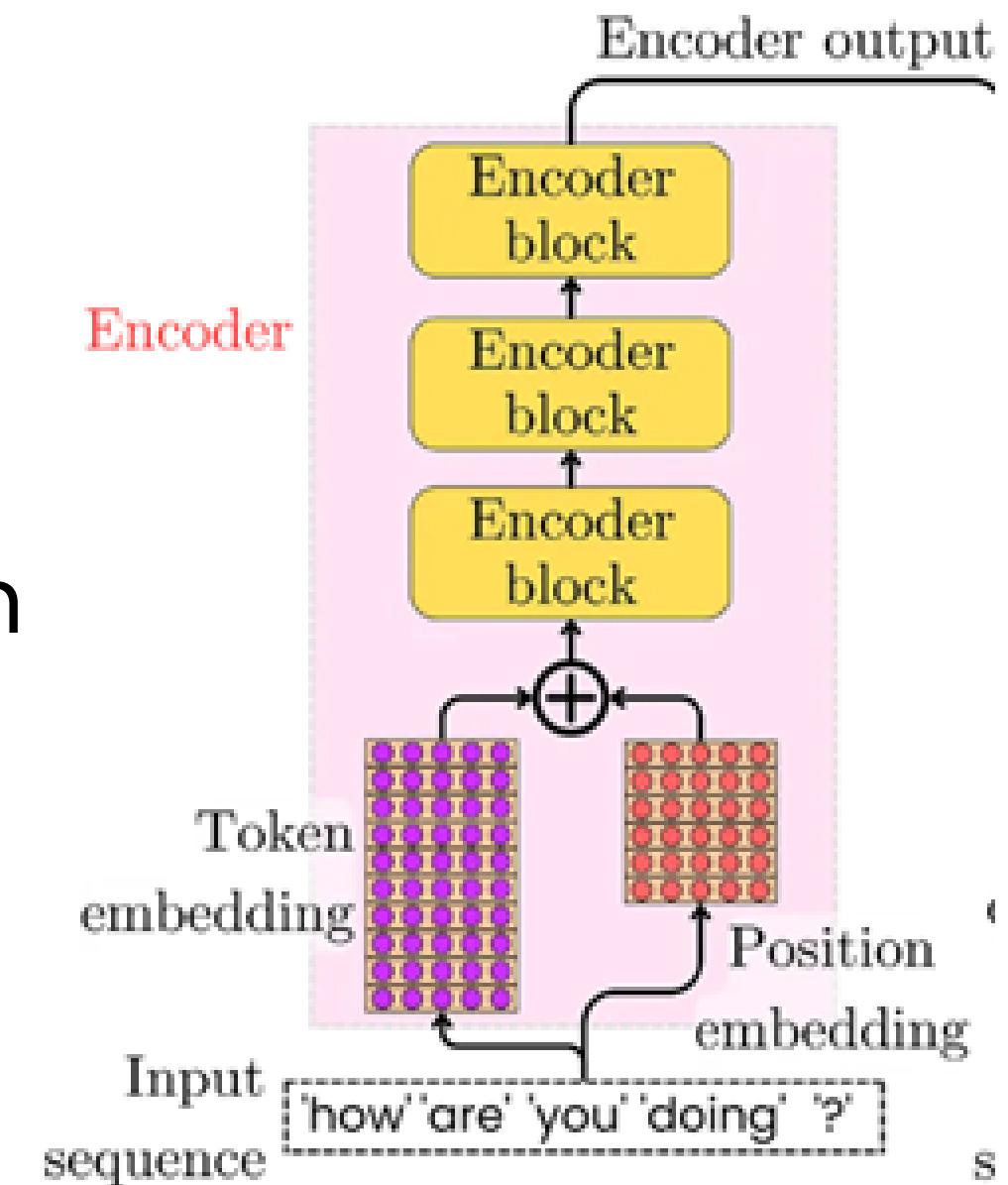
## Encoder-only

- Kiến trúc Encoder-Only sử dụng chỉ phần Encoder của Transformer.
- Encoder-only là mô hình không sinh văn bản mà tập trung hoàn toàn vào việc hiểu câu, phân tích ngữ nghĩa hai chiều.
- Được xây dựng từ nhiều tầng encoder giống nhau ghép lại.
- Khi câu được đưa vào mô hình, nó sẽ đi qua lớp embedding, được thêm thông tin vị trí, sau đó đi qua hàng loạt các tầng encoder.
- Mỗi tầng có nhiệm vụ trích xuất ngữ nghĩa ngày càng sâu hơn.



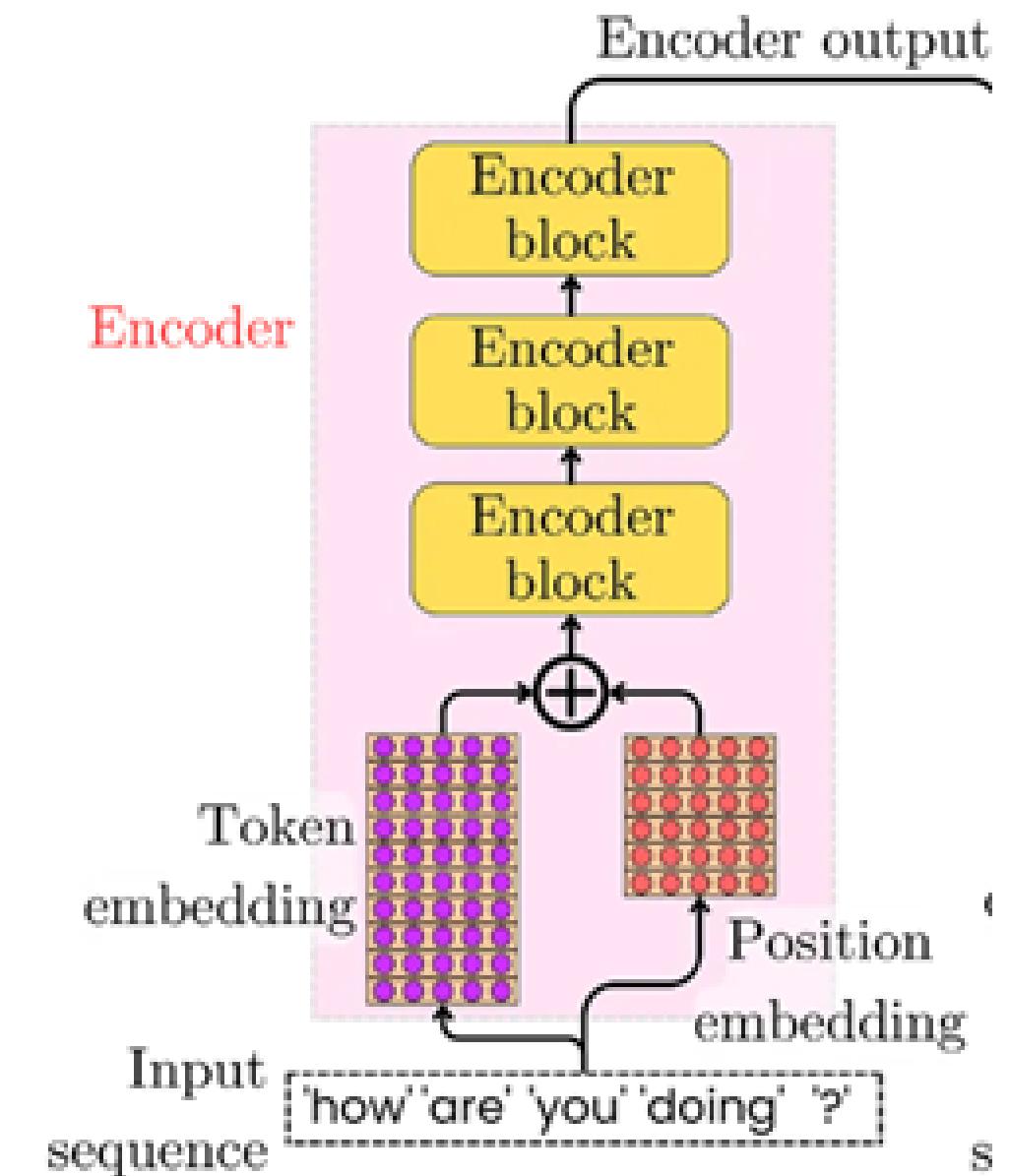
## Encoder-only

- Mỗi tầng encoder có hai khối chính.
- Khối thứ nhất là Multi-Head Self-Attention, giúp mô hình xem xét mối quan hệ giữa các token. Một đặc trưng quan trọng là dùng self-attention hai chiều, nghĩa là một token có thể nhìn thấy toàn bộ các token khác trong câu, không bị giới hạn bởi hướng trái sang phải như decoder.
- Khối thứ hai là Feed-Forward Network, một mạng nơ-ron đơn giản hai lớp để biến đổi đặc trưng. Giữa các khối đều có residual connection và layer normalization nhằm ổn định gradient, giúp mô hình học tốt hơn và cho phép chồng nhiều tầng lên nhau.



## Encoder-only

- Self-attention là cơ chế giúp mô hình tính xem một token cần tập trung vào những token nào khác trong câu.
- Mỗi token sẽ được ánh xạ thành Query, Key và Value. Từ đó mô hình tính mức độ liên quan giữa các token bằng hàm softmax.
- Vì là encoder, cơ chế này không dùng mask, nên mỗi token được phép nhìn toàn bộ câu => giúp mô hình hiểu ngữ cảnh hai chiều và nắm được đầy đủ tầm nghĩa của câu.



## Encoder-only với bài toán QA

### Tại sao áp dụng model Encoder-only cho bài toán QA ?

- Với cơ chế self-attention hai chiều, encoder-only có khả năng nắm bắt mối quan hệ giữa câu hỏi và đoạn văn trong toàn bộ ngữ cảnh.
- Điều này giúp mô hình xác định chính xác phần thông tin liên quan trong đoạn văn, từ đó trích xuất câu trả lời.
- Các mô hình encoder-only như BERT đã từng đạt kết quả SOTA trong các bộ dữ liệu QA dạng trích xuất như SQuAD => Điều đó chứng minh kiến trúc này rất mạnh trong nhiệm vụ đọc hiểu và tìm câu trả lời trực tiếp từ đoạn văn.

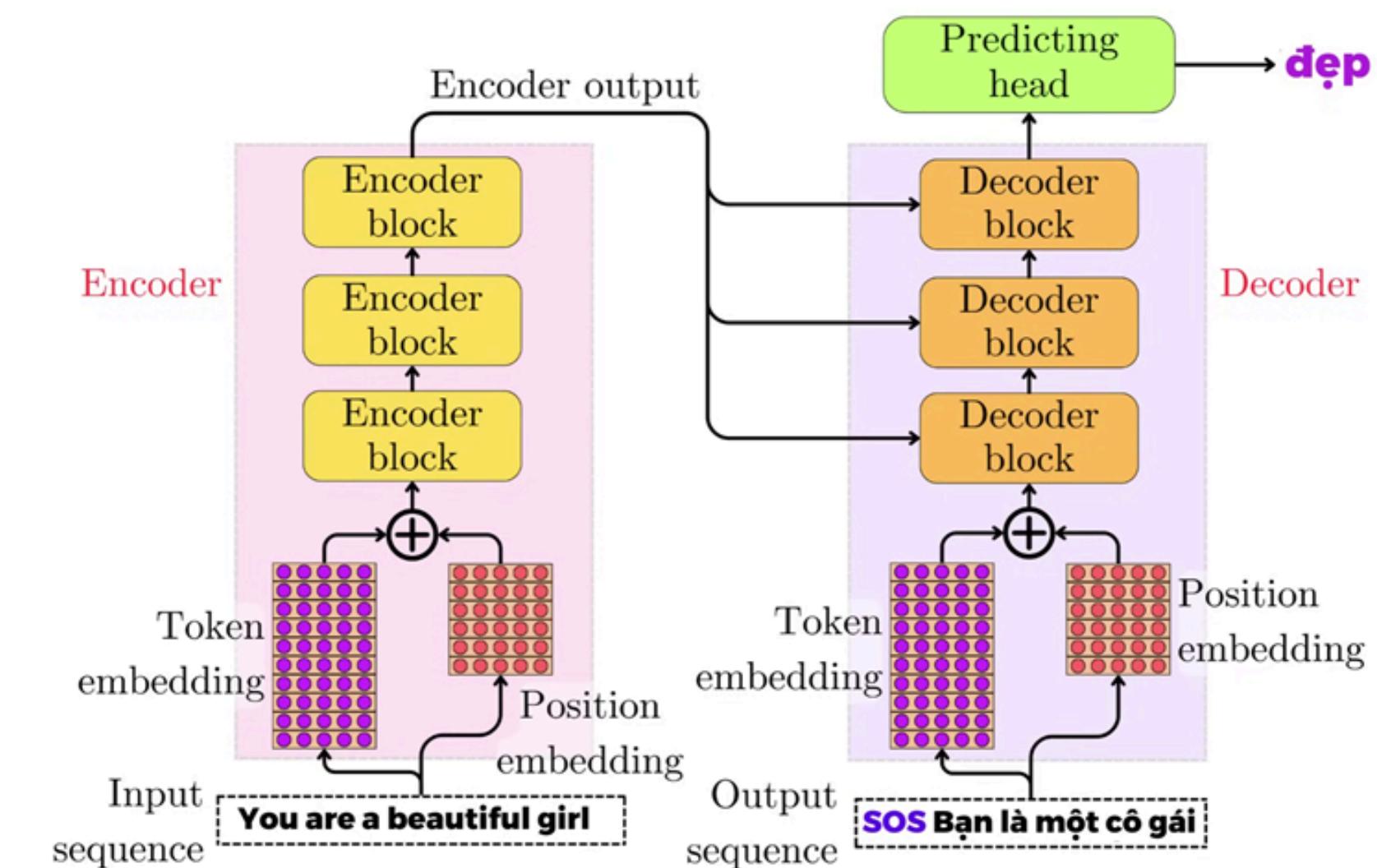
## Encoder-only với bài toán QA

### Encoder-only làm gì trong bài toán QA ?

- Trong bài toán QA dạng trích xuất, encoder-only nhận đầu vào là câu hỏi và đoạn văn ghép lại, sau đó biến chúng thành các vector ngữ nghĩa chứa đầy đủ thông tin ngữ cảnh.
- Qua nhiều tầng encoder, mô hình tạo ra các vector ngữ nghĩa sâu cho từng token.
- Từ các vector này, mô hình dự đoán hai thứ: vị trí bắt đầu và vị trí kết thúc của câu trả lời trong đoạn văn.

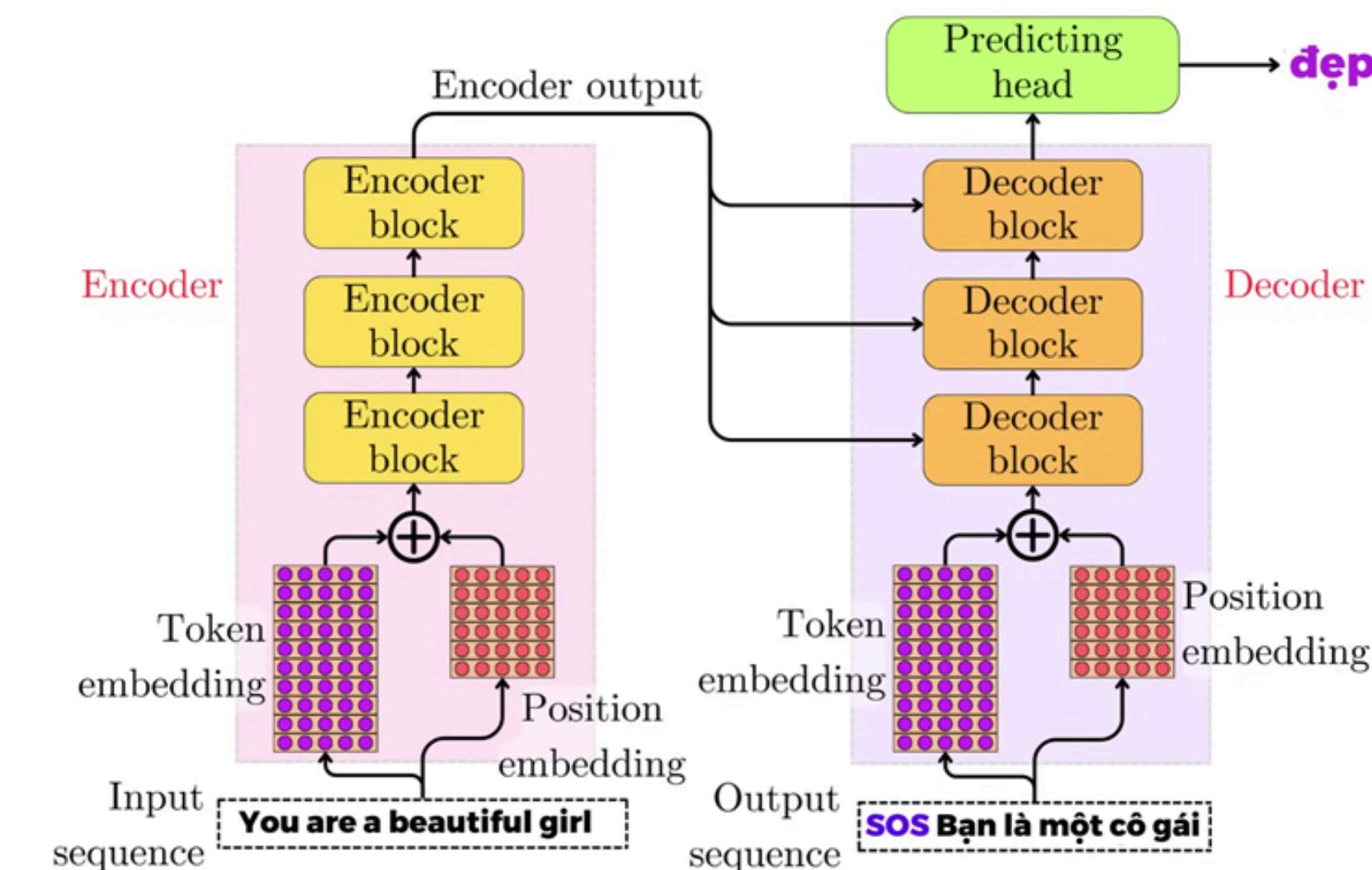
## II – MÔ HÌNH TRANSFORMER

- Kiến trúc encoder-decoder ra đời từ Seq2Seq và phát triển mạnh mẽ với Transformer. Đầu vào sẽ đi qua bộ mã hóa để chuyển thành biểu diễn ngữ cảnh, sau đó bộ giải mã sinh ra chuỗi đầu ra dựa trên biểu diễn đó.
- Mô hình encoder-decoder vừa hiểu được trọn vẹn ngữ nghĩa đầu vào, vừa có khả năng sinh ra đầu ra mới.
- Mô hình này có thể được huấn luyện theo nhiều cách, phổ biến là denoising.
- Mô hình sẽ học cả việc hiểu ngữ cảnh đầy đủ lẫn sinh đoạn đã mất.
- Ứng dụng: dịch máy, hỏi đáp tạo sinh, sinh mô tả từ dữ liệu,...



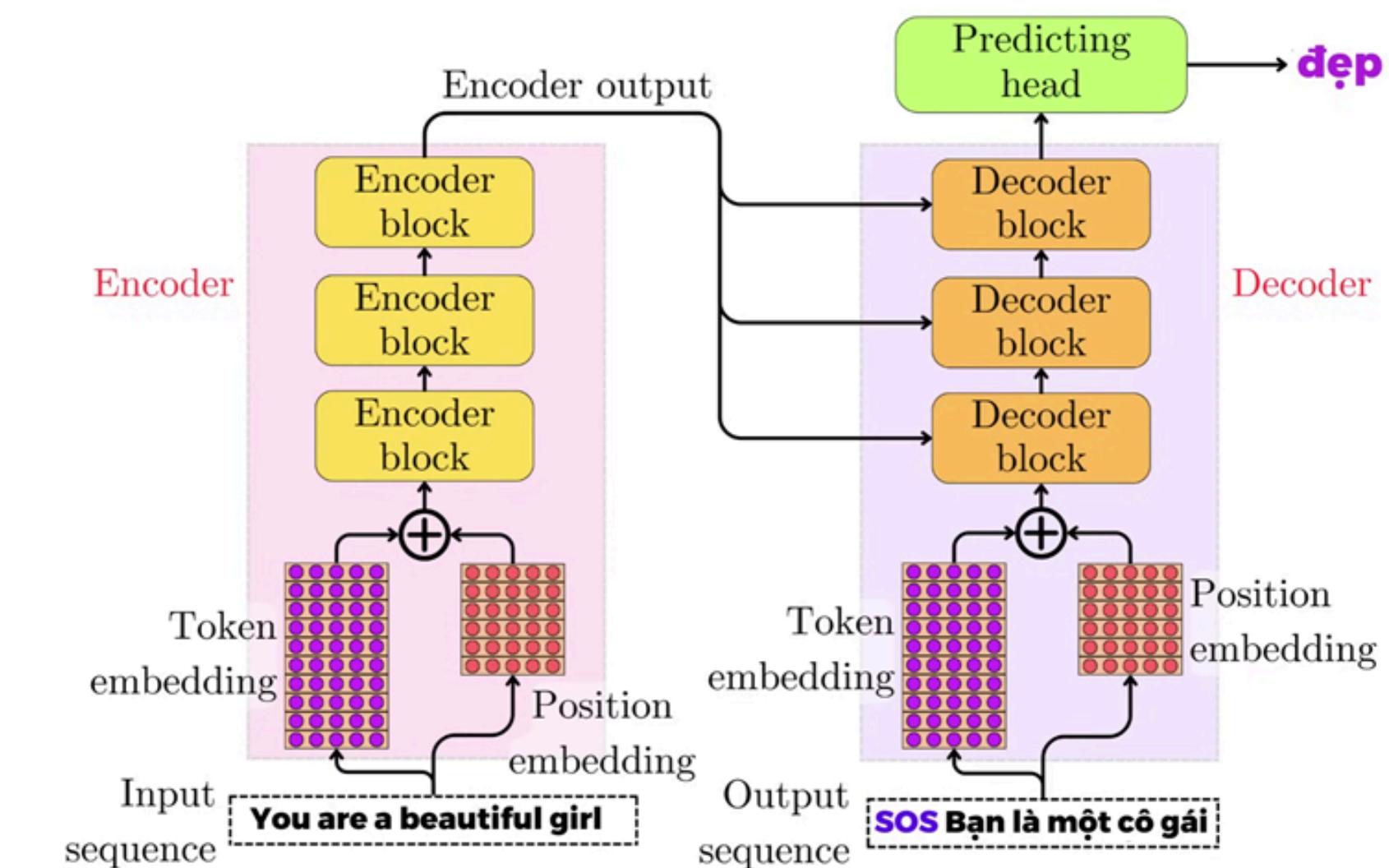
## II – MÔ HÌNH TRANSFORMER

- Input sequence: You are a beautiful girl. Đầu tiên câu tiếng Anh được chia thành token. Ví dụ: [You] [are] [a] [beautiful] [girl].
- Token Embedding + Position Embedding: Token embedding: biến từng token thành vector. Position embedding: thêm vị trí của token trong câu để Transformer “biết thứ tự”. Hai embedding được cộng lại → tạo ra biểu diễn đầu vào cho Encoder.
- Đi qua các Encoder Blocks: Mỗi block gồm: Self-attention giúp mỗi từ nhìn thấy toàn bộ câu để hiểu ngữ nghĩa tổng thể và Feed-forward network tinh chỉnh biểu diễn vector sau attention.
- Encoder output: Sau khi qua 3 block, ta thu được một loạt vector biểu diễn toàn bộ câu. Encoder output này sẽ được gửi sang Decoder.



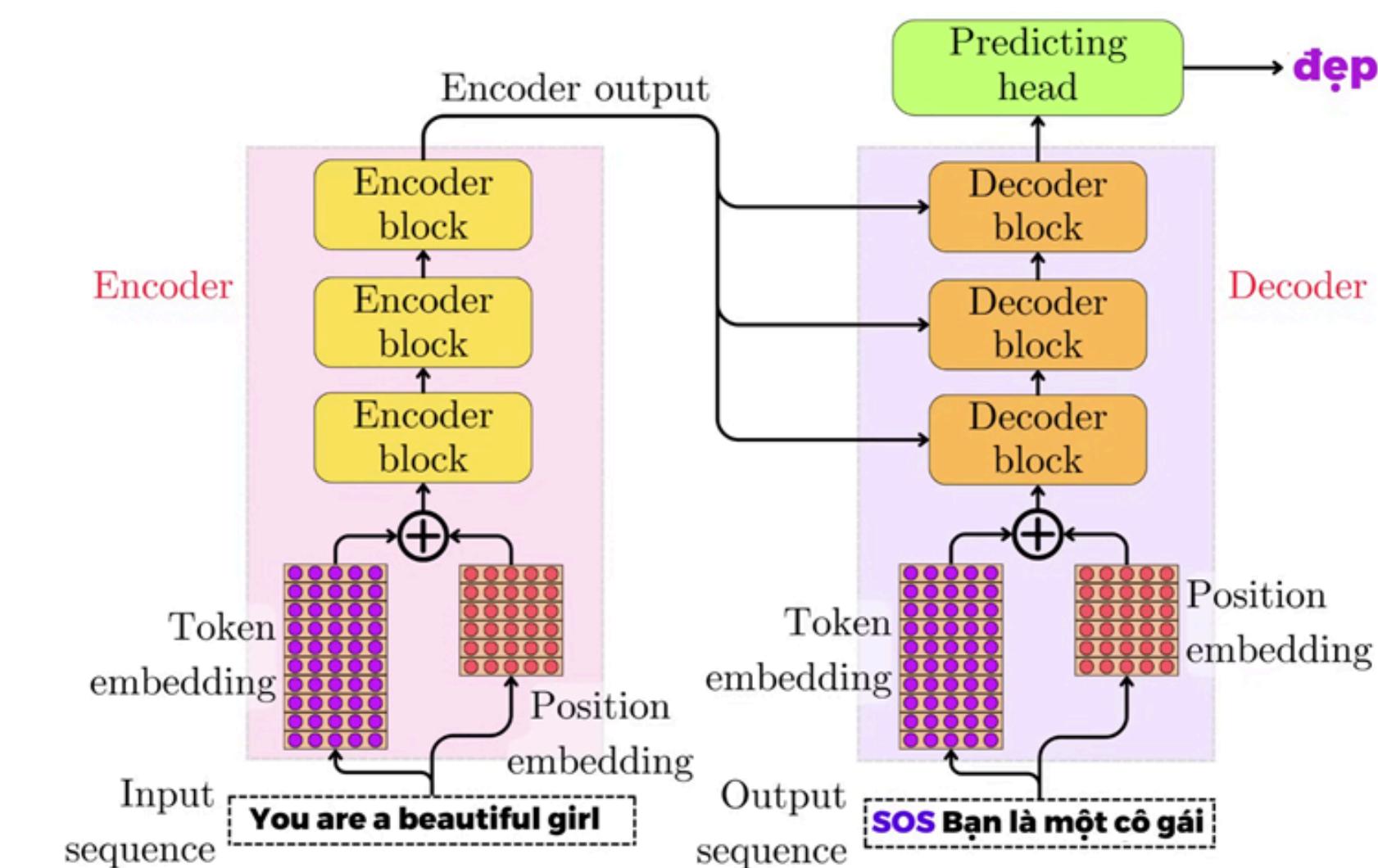
## II – MÔ HÌNH TRANSFORMER

- Đến luồng decoder: Decoder không sinh toàn bộ câu một lần mà nó sinh từng token một, theo chiều trái → phải. chuỗi đầu ra hiện tại là: SOS Bạn là một cô gái. (“SOS” = Start Of Sentence → token bắt đầu chuỗi).
- Token Embedding + Position Embedding: Chuỗi đã sinh được (“SOS Bạn là một cô gái”) được: Tokenize → embedding, cộng thêm position embedding tạo ra chuỗi vector đưa vào Decoder.
- Đi vào Decoder Block: Mỗi Decoder block cũng có 3 phần gồm
  - a. Masked Self-Attention: Các token chỉ được nhìn các token phía trước, không được nhìn token tương lai. Ví dụ: khi đang dự đoán token kế tiếp, token “gái” chỉ nhìn: SOS → Bạn → là → một → cô.
  - b. Cross-Attention với Encoder Output: Decoder lấy biểu diễn hiện tại của output và “nhìn sang” biểu diễn từ Encoder để biết phần nào của câu tiếng Anh liên quan. Ví dụ khi chuẩn bị sinh token “đẹp”, attention sẽ tập trung mạnh vào token “beautiful” của encoder.
  - c. Feed-Forward Layer: Tăng độ phi tuyến và tinh chỉnh vector.



## II – MÔ HÌNH TRANSFORMER

- Predicting Head → Sinh token kế tiếp: Kết quả cuối từ decoder block 3 đi vào Linear layer và Softmax. Softmax sẽ gán xác suất cho tất cả token trong vocab. Token có xác suất cao nhất → được sinh ra. Trong hình, token được sinh là “đẹp”



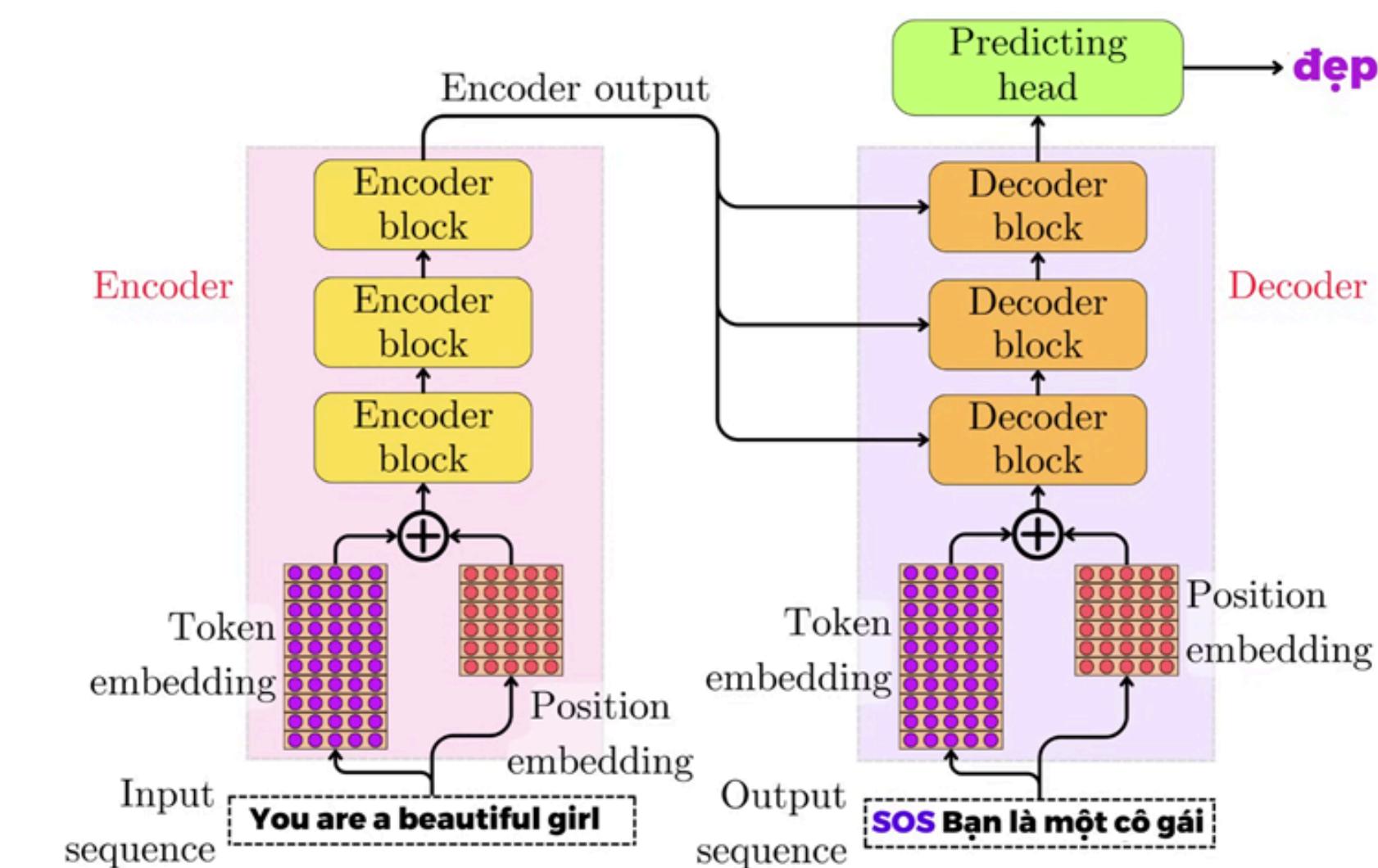
## II – MÔ HÌNH TRANSFORMER

### Tại sao áp dụng model Encoder-only cho bài toán QA ?

- Vì QA là bài toán “biến đổi chuỗi → chuỗi” (sequence-to-sequence)
- Vì QA yêu cầu mô hình “hiểu sâu” câu hỏi trước khi sinh câu trả lời
- Vì Decoder có khả năng SINH NGÔN NGỮ tự nhiên (Generative)
- Vì nhiều dạng QA yêu cầu PHẢI sinh câu trả lời mới, không chỉ chọn

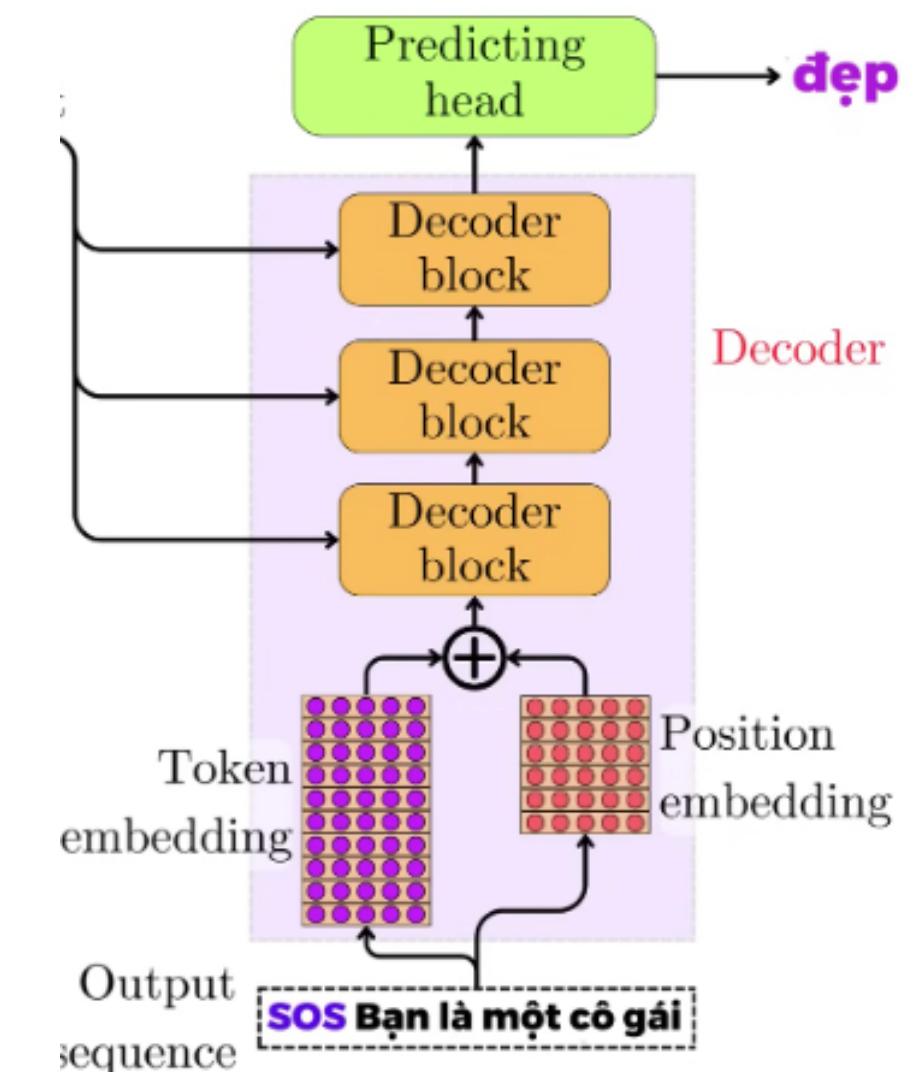
## II – MÔ HÌNH TRANSFORMER

- Predicting Head → Sinh token kế tiếp: Kết quả cuối từ decoder block 3 đi vào Linear layer và Softmax. Softmax sẽ gán xác suất cho tất cả token trong vocab. Token có xác suất cao nhất → được sinh ra. Trong hình, token được sinh là “đẹp”



## II – DECODER ONLY

- Kiến trúc decoder-only sử dụng chỉ phần Decoder của Transformer, gồm nhiều lớp chồng nhau chứa:
  - Masked Multi-Head Self-Attention → dùng causal mask để mỗi token chỉ nhìn được token trước nó
  - Feed-Forward Network
  - Residual connection + LayerNorm
- Không có encoder riêng, không có attention hai chiều.
- Toàn bộ mô hình hoạt động theo tự hồi quy (auto-regressive).



## Cơ chế attention một chiều (uni-directional)

- Token t chỉ có thể nhìn: [token 1, token 2, ..., token t-1]
- Không được nhìn token tương lai.
- Điều này quan trọng vì:
  - Mô hình học đúng bản chất sinh văn bản từ trái sang phải.
  - Mô hình không thể đọc toàn bộ câu trong đầu, phải dự đoán từng bước dựa trên quá khứ.

Huấn luyện bằng Causal Language Modeling (CLM)

- Dự đoán token tiếp theo dựa trên chuỗi token trước đó.
- $P(\text{token}_t | \text{token} < t)$

Ví dụ chuỗi:

"Transformer rất mạnh trong"

Mô hình phải dự đoán token tiếp theo là:

"NLP" hoặc "xử lý ngôn ngữ", tùy context học được.

## Cơ chế attention một chiều (uni-directional)

Tập huấn luyện CLM thường cực lớn (internet-scale), giúp mô hình học:

- kiến thức ngôn ngữ
- kiến thức thế giới (world knowledge)
- phong cách viết
- mối quan hệ giữa câu hỏi – câu trả lời (nếu được fine-tune thêm)

Decoder-only trong bài toán Question Answering (QA)

- Input của mô hình
  - Khác với kiến trúc encoder-decoder, QA với decoder-only cần ghép câu hỏi và (nếu có) đoạn context thành một chuỗi duy nhất.
  - Cấu trúc điển hình:
    - <|system|> Trả lời dựa trên đoạn văn sau.
    - <|context|> ...
    - <|question|> Câu hỏi: ...
    - <|answer|>
  - Mô hình không được quyền đọc toàn câu trả lời trước khi sinh, mà phải sinh từng token tiếp theo dựa vào chuỗi trước đó.

## Cách mô hình hiểu câu hỏi

Mô hình không có encoder, nên:

- Nó hiểu câu hỏi dựa vào kiến trúc self-attention bên trong chuỗi input.
- Tất cả thông tin đều nằm trong bên trái token đang dự đoán.
- Kiến thức Q&A được học từ:
  - pretraining (mô hình đã từng đọc dạng hỏi đáp)
  - instruction tuning
  - supervision fine-tuning (SFT)
  - RLHF hoặc DPO (nếu có)
- Điểm quan trọng:

Self-attention giúp câu trả lời có thể “tham chiếu” lại câu hỏi và đoạn context đã đọc trước đó.

# Quá trình trả lời QA

- Nhập chuỗi: context + question
  - Masked self-attention dùng toàn bộ quá khứ để tính trọng số
  - Mô hình sinh token đầu tiên của câu trả lời
  - Ghép token đó vào chuỗi
  - Lặp lại bước 2→4 cho đến khi gặp token kết thúc hoặc vượt giới hạn độ dài
- QA được xem như dạng mở rộng của Next Token Prediction.

## Ưu điểm của Decoder-only cho QA

Ưu điểm	Giải thích
<b>Sinh câu trả lời tự nhiên, linh hoạt</b>	Vì bản chất sinh văn bản
<b>Không cần kiến trúc encoder riêng</b>	Đơn giản hơn encoder-decoder
<b>Học được kiến thức QA từ pretraining</b>	Nhờ học trên dữ liệu lớn
<b>Hiệu quả cao khi kết hợp hướng dẫn (instruction tuning)</b>	Giúp mô hình trở thành chatbot tốt

## Nhược điểm

Nhược	Giải thích
<b>Không có attention hai chiều</b>	→ không phân tích toàn câu cùng lúc, dễ hiểu sai nếu context dài
<b>Phụ thuộc mạnh vào prompt</b>	Cần thiết kế prompt chuẩn
<b>Không tối ưu cho nhiệm vụ đọc hiểu chính xác (extractive QA)</b>	Vì không có encoder để mã hóa ngữ cảnh hai chiều
<b>Dễ “bịa” thông tin (hallucination)</b>	Vì bản chất là mô hình sinh

## Vai trò của Qwen trong giải bài toán QA

Qwen Series = Kiến trúc Decoder-only

- Ví dụ: Qwen 0.6B, 1.8B, 7B, 14B...

Hỗ trợ nhiều ngôn ngữ, trong đó có tiếng Việt

→ phù hợp cho QA tiếng Việt

→ có tokenizer tối ưu cho đa ngôn ngữ

Được huấn luyện trên dữ liệu khổng lồ, bao gồm:

- hội thoại
- QA mở
- mã nguồn
- kiến thức thế giới

→ giúp Qwen rất mạnh trong generative QA.

Instruction Tuning + RLHF

Qwen-AI đã tinh chỉnh:

- theo hướng dẫn (instruction tuning)
- theo phản hồi con người (RLHF)

→ giúp mô hình “hiểu câu hỏi” tốt hơn so với base model.

### III – MÔ TẢ DỮ LIỆU

- Crawl ~200 đoạn văn từ Wikipedia tiếng Việt thuộc nhiều chủ đề (lịch sử, địa lý, khoa học, giải trí...).
- Mỗi bài viết được chia nhỏ theo paragraph để tạo các đoạn ngữ cảnh ngắn, rõ ràng và độc lập nội dung.
- Sử dụng mô hình Gemini để sinh ra các cặp câu hỏi – câu trả lời tương ứng. Cụ thể, với mỗi đoạn văn, gửi prompt yêu cầu Gemini tạo ra 3 cặp hỏi-đáp độc lập, đảm bảo câu hỏi được trả lời dựa hoàn toàn vào thông tin trong đoạn văn.

.....

Bạn là một hệ thống tạo cặp Câu hỏi/câu trả lời (QA) chuyên nghiệp. Sử dụng **chỉ** nội dung từ đoạn văn được cung cấp dưới đây để tạo ra 3 cặp QA độc lập.

**YÊU CẦU ĐẦU VÀO:**

- CHỦ ĐỀ: {topic}
- ĐOẠN VĂN: {context\_text}

**QUY TẮC BẮT BUỘC:**

1. **Số lượng:** Phải tạo ra **ĐÚNG 3** cặp QA.
2. **Định dạng:** Chỉ trả về một mảng JSON (Array of Objects), không kèm bất kỳ lời dẫn hay giải thích nào khác.
3. **Cấu trúc QA:** Mỗi đối tượng JSON phải có **ĐÚNG 3** khóa: "question", "answer".
4. **Câu hỏi:** Phải rõ ràng, tự nhiên, và KHÔNG được tham chiếu đến từ ngữ như "trong đoạn văn", "theo văn bản trên", v.v.
5. **Câu trả lời:** Phải chính xác, ngắn gọn, và **chỉ** dựa trên thông tin có trong phần ĐOẠN VĂN.

**ĐỊNH DẠNG ĐẦU RA JSON TUYỆT ĐỐI:**

[

### III – MÔ TẢ DỮ LIỆU

- Mỗi mẫu dữ liệu có 3 trường:  
context – đoạn văn từ Wikipedia  
question – câu hỏi được sinh  
answer – câu trả lời trích trực tiếp từ đoạn văn

```
{  
    "topic": "Manaurie",  
    "context": "Manaurie là một xã của Pháp, nằm ở tỉnh Dordogne trong vùng Aquitaine của Pháp. Xã này có diện tích 9,97 km2, dân số năm 2006 là 158 người. Xã nằm ở khu vực có độ cao trung bình 78 m trên mực nước biển."  
}
```

```
{  
    "context": "Manaurie là một xã của Pháp, nằm ở tỉnh Dordogne trong vùng Aquitaine của Pháp. Xã này có diện tích 9,97 km2, dân số năm 2006 là 158 người. Xã nằm ở khu vực có độ cao trung bình 78 m trên mực nước biển.",  
    "question": "Manaurie là gì?",  
    "answer": "Manaurie là một xã của Pháp.",  
,  
{  
    "context": "Manaurie là một xã của Pháp, nằm ở tỉnh Dordogne trong vùng Aquitaine của Pháp. Xã này có diện tích 9,97 km2, dân số năm 2006 là 158 người. Xã nằm ở khu vực có độ cao trung bình 78 m trên mực nước biển.",  
    "question": "Manaurie nằm ở đâu?",  
    "answer": "Manaurie nằm ở tỉnh Dordogne trong vùng Aquitaine của Pháp.",  
,  
{  
    "context": "Manaurie là một xã của Pháp, nằm ở tỉnh Dordogne trong vùng Aquitaine của Pháp. Xã này có diện tích 9,97 km2, dân số năm 2006 là 158 người. Xã nằm ở khu vực có độ cao trung bình 78 m trên mực nước biển.",  
    "question": "Dân số của Manaurie vào năm 2006 là bao nhiêu?",  
    "answer": "Dân số của Manaurie vào năm 2006 là 158 người."  
}
```

### III – MÔ TẢ DỮ LIỆU

#### Tiền xử lý dữ liệu

- Làm sạch & chuẩn hóa văn bản
- Loại bỏ ký tự thừa, khoảng trắng không cần thiết, xuống dòng dư.
- Giữ nguyên dấu tiếng Việt và định dạng chữ.
- Xóa token trống hoặc quá ngắn.
- Chuẩn hóa khoảng trắng trong context, question, answer.
- Loại bỏ ký tự markdown/HTML rác.
- Chuyển đổi sang HuggingFace Dataset
- Từ list các dict JSON → Dataset.from\_list().
- Dễ dàng chia train/val/test và gán nhãn.
- Chuẩn bị cho tokenization theo từng mô hình.

### III – MÔ TẢ DỮ LIỆU

#### Tokenization cho từng kiến trúc mô hình

- PhoBERT sử dụng SentencePiece tokenizer với văn bản đã được tách từ.
- Để tokenization chính xác, chuỗi đầu vào được ghép theo cấu trúc “question + context”, sau đó toàn bộ được đưa qua tokenizer để tạo input\_ids và attention\_mask.
- Vì PhoBERT phục vụ bài toán trích xuất câu trả lời, nhóm xác định vị trí bắt đầu và kết thúc của câu trả lời trực tiếp trong đoạn văn gốc, rồi ánh xạ các vị trí ký tự đó sang vị trí token trong chuỗi đã token hóa.
- Đồng thời, mọi token đặc biệt và token của câu hỏi đều được tính offset để đảm bảo vị trí span không bị lệch. Độ dài chuỗi được cố định ở 256 token bằng cơ chế padding và truncation.
- Tokenizer PhoBERT trả về bốn trường chính cho mỗi mẫu: input\_ids, attention\_mask, start\_positions, end\_positions.

### III – MÔ TẢ DỮ LIỆU

#### **Tokenization cho từng kiến trúc mô hình**

- ViT5 sử dụng SentencePiece tokenizer theo định dạng text-to-text, nghĩa là đầu vào và đầu ra đều là chuỗi văn bản. Nhóm xây dựng đầu vào dạng prompt rõ ràng: “question: ... context: ...”.
- Sau đó tokenizer T5 sẽ mã hóa thành input\_ids và attention\_mask, với giới hạn độ dài 256 token. Phần đáp án được tokenizer mã hóa riêng để tạo ra labels với giới hạn 128 token.
- Theo chuẩn T5 của HuggingFace, các token <pad> trong labels được thay bằng -100, giúp mô hình bỏ qua loss tại các vị trí đệm. Tokenization được áp dụng tự động cho toàn bộ dataset bằng phương thức map(), tạo ra các tensor phù hợp cho huấn luyện mô hình encoder-decoder.

### III – MÔ TẢ DỮ LIỆU

#### Tokenization cho từng kiến trúc mô hình

- Qwen hoạt động theo cơ chế causal language modeling, nên tokenizer sẽ mã hóa toàn bộ prompt + answer trong một chuỗi duy nhất. Nhóm xây dựng prompt theo kiểu instruction-tuning:
  - #### Ngữ cảnh: {context}
  - #### Câu hỏi: {question}
  - #### Trả lời: {answer}
- Tokenizer Qwen (tương thích đa ngôn ngữ, gồm tiếng Việt) mã hóa toàn bộ chuỗi này thành input\_ids. Để mô hình học sinh câu trả lời, phần token thuộc prompt được gán nhãn -100, còn phần token thuộc answer giữ nguyên để tính loss. Chuỗi token được giới hạn ở 600 token để đảm bảo vừa đủ cho ngữ cảnh + câu hỏi + đáp án. Bộ xử lý batch sử dụng DataCollator của HuggingFace để thực hiện padding động.
- Kết quả tokenization gồm input\_ids và labels đã được mask phù hợp cho huấn luyện mô hình decoder-only.