

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CẦN THƠ
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



TRẦN QUỐC KHANG
MÃ SỐ SINH VIÊN: B1913236

ỨNG DỤNG HỌC SÂU XÂY DỰNG CHATBOT HỖ
TRỢ QUẢNG BÁ CÁC MÓN ĂN ĐẶC SẢN CỦA
ĐỒNG BẰNG SÔNG CỬU LONG

LUẬN VĂN TỐT NGHIỆP
NGÀNH: KHOA HỌC MÁY TÍNH
MÃ SỐ: 748 0101

GIẢNG VIÊN HƯỚNG DẪN
PGS. TS. GVCC. PHẠM NGUYỄN KHANG

NĂM 2023

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CẦN THƠ
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

TRẦN QUỐC KHANG
MÃ SỐ SINH VIÊN: B1913236

ỨNG DỤNG HỌC SÂU XÂY DỰNG CHATBOT HỖ
TRỢ QUẢNG BÁ CÁC MÓN ĂN ĐẶC SẢN CỦA
ĐỒNG BẰNG SÔNG CỬU LONG

USING DEEP LEARNING TO BUILD A CHATBOT
SUPPORTING THE PROMOTION OF SPECIALTY DISHES IN
MEKONG DELTA

LUẬN VĂN TỐT NGHIỆP
NGÀNH: KHOA HỌC MÁY TÍNH
MÃ SỐ: 748 0101

GIẢNG VIÊN HƯỚNG DẪN
PGS. TS. GVCC. PHẠM NGUYÊN KHANG

NĂM 2023

LỜI CẢM ƠN

Tôi xin gửi lời cảm ơn đến PGS. TS. GVCC. Phạm Nguyên Khang, người đã tận tình giúp đỡ tôi trong quá trình học tập, nghiên cứu và hoàn thành luận văn này.

Tôi xin bày tỏ lòng biết ơn đến các Thầy Cô trong Khoa Khoa học máy tính, các Thầy Cô giảng dạy tại Trường Công nghệ thông tin và Truyền thông, Trường Đại học Cần Thơ đã giúp đỡ tôi trong suốt quá trình học tập và nghiên cứu.

Sau cùng tôi xin gửi lời cảm ơn gia đình và bạn bè đã luôn ủng hộ tôi, động viên cũng như là giúp đỡ và hỗ trợ tôi trong suốt thời gian qua.

Trong quá trình nghiên cứu và thực hiện đề tài luận văn sẽ không tránh khỏi nhiều sai sót và hạn chế, kính mong nhận được sự chỉ dẫn và đóng góp của quý Thầy Cô để bài luận văn của tôi được hoàn thiện hơn.

Tôi xin chân thành cảm ơn!

Cần Thơ, ngày 12 tháng 5 năm 2023

Tác giả



Trần Quốc Khang

TÓM TẮT

Dựa vào hình ảnh các món ăn ở khu vực đồng bằng sông Cửu Long, cùng các câu hỏi xoay quanh món ăn: Món ăn này tên là gì? Nổi tiếng ở đâu? Nguyên liệu chính bao gồm những gì? Cách làm như thế nào? Từ đó, xây dựng ứng dụng chatbot hỗ trợ quảng bá các món ăn đặc sản của đồng bằng sông Cửu Long.

Đề tài trình bày cách tiếp cận các mô hình Transformer-based (như ViT [1] cho dữ liệu hình ảnh, BERT [2]/PhoBERT [3] với dữ liệu văn bản, hoặc ViLT [4] xử lý đồng thời dữ liệu hình ảnh và văn bản), để huấn luyện mô hình Visual Question Answering – VQA cho tác vụ phân lớp. Sau đó, sử dụng mô hình VQA như Visual Encoder kết hợp với Decoder là GPT-2 [5] xây dựng một mô hình Visual Encoder – Decoder cho tác vụ sinh câu trả lời.

Tập dữ liệu thực nghiệm là một phần của tập dữ liệu 30VNFood [6] và VinaFood21 [7] với 7694 hình ảnh các món ăn ở đồng bằng sông Cửu Long.

Kết quả đánh giá các mô hình VQA dựa trên chỉ số Accuracy đạt được kết quả khá cao. Giá trị Accuracy Mô hình 1: ViT [1] và BERT [2] trên tiếng Anh là 94%, ViT [1] và PhoBERT [3] trên tiếng Việt xấp xỉ 95%, với Mô hình 2: ViLT [4] đạt hơn 92% trên tiếng Anh.

Với Mô hình 3, mô hình sinh câu trả lời sử dụng ViLT [4] kết hợp GPT-2 [5], chỉ tiếng Anh, đạt kết quả lần lượt là 49.92, 39.26, 47.53 tương ứng với các chỉ số ROUGE-1, ROUGE-2, ROUGE-L trong phương pháp đánh giá ROUGE [8].

Cuối cùng, áp dụng các mô hình đã được huấn luyện vào xây dựng chatbot.

ABSTRACT

Based on the images of dishes in the Mekong Delta along with questions about the dishes such as: What is the name of this dish? Where is it famous? What are the main ingredients? How is it made? An application chatbot will be built to promote the specialty dishes of the Mekong Delta.

This thesis outlines a method for training a Visual Question Answering (VQA) model for classification tasks using Transformer-based models, such as ViT [1] for image data, BERT [2]/PhoBERT [3] for text data, or ViLT [4] for simultaneous processing of image and text data. After that, a Visual Encoder-Decoder model for the task of generating sentences will be built using the VQA model as a Visual Encoder and a GPT-2 [5] as a Decoder.

The experimental dataset, which includes 7,694 photos of dishes from the Mekong Delta, is a subset of the datasets 30VNFood [6] and VinaFood21 [7].

The accuracy metric was used to evaluate the VQA models, and the results were relatively good. For Model 1: ViT [1] and BERT [2], the accuracy scores for English and Vietnamese are 94% and 95%, respectively, while the accuracy score for Model 2: ViLT [4] is over 92% on English-only.

According to the ROUGE [8] evaluation method, Model 3's answer sentence generation model, on English only, which used ViLT [4] along with GPT-2 [5], yielded results of 49.92, 39.26, and 47.53 for the ROUGE-1, ROUGE-2, and ROUGE-L, respectively.

Finally, the trained models were applied to build a chatbot.

LỜI CAM ĐOAN

Tôi xin cam đoan Luận văn tốt nghiệp “Ứng dụng học sâu xây dựng chatbot hỗ trợ quảng bá các món ăn đặc sản của đồng bằng sông Cửu Long” là công trình nghiên cứu của riêng tôi. Ngoài các trích dẫn, tài liệu tham khảo đã được ghi nguồn đầy đủ. Các số liệu, kết quả nêu trong luận văn là trung thực và chưa từng được công bố trong các công trình khác. Nếu không đúng như đã nêu trên, tôi xin hoàn toàn chịu trách nhiệm về lỗi của mình.

Cần Thơ, ngày 12 tháng 5 năm 2023

Người cam đoan



Trần Quốc Khang

MỤC LỤC

LỜI CẢM ƠN -----	3
TÓM TẮT -----	4
ABSTRACT -----	5
LỜI CAM ĐOAN -----	6
MỤC LỤC -----	7
DANH SÁCH BẢNG-----	10
DANH SÁCH HÌNH-----	11
DANH MỤC TỪ VIẾT TẮT -----	14
A. PHẦN MỞ ĐẦU -----	15
1. Lý do chọn đề tài -----	15
2. Mục tiêu nghiên cứu-----	16
3. Các nghiên cứu có liên quan-----	17
4. Đối tượng và phạm vi nghiên cứu-----	17
5. Phương pháp nghiên cứu -----	18
6. Cấu trúc luận văn -----	18
B. PHẦN NỘI DUNG -----	19
Chương 1. Cơ sở lý thuyết của đề tài -----	19
1. Trả lời câu hỏi từ hình ảnh (Visual Question Answering – VQA) -----	19
2. Nhúng từ (Word Embedding) -----	19
2.1. One-Hot Encoding -----	20
2.2. Word2Vec [12]-----	21
Huấn luyện mô hình Word2Vec [12] -----	22
2.3. Subword Embedding-----	24
2.3.1. Byte Pair Encoding – BPE [13]-----	24
Thuật toán huấn luyện (training) -----	24
2.3.2. WordPiece [15] [16]-----	25
Thuật toán huấn luyện (training) -----	25
3. Mô hình Seq2Seq [17] và cơ chế Attention (Attention Mechanism) -----	26
3.1. Sơ lược về mô hình Seq2Seq [17] trong bài toán dịch-----	26
3.2. Attention Mechanism-----	28
3.3. Áp dụng cơ chế Attention vào mô hình Seq2Seq [17] -----	30
4. Mô hình Transformer [9]-----	32
4.1. Đầu vào (Input) và đầu ra (Output) của Transformer [9] -----	35
4.2. Embedding và Position Encoding-----	36
4.2.1. Embedding -----	36
4.2.2. Positional Encoding-----	37
4.3. Chi tiết các thành phần có trong Transformer [9] -----	41
4.3.1. Multi-Head Attention-----	41

4.3.2. Scaled Dot-Product Attention-----	42
4.3.3. Position-wise Feed-Forward -----	43
4.3.4. Add & Norm -----	44
4.3.5. Linear và Softmax-----	45
4.4. Encoder và Decoder trong Transformer -----	45
5. Mô hình BERT [2] -----	46
Ví dụ: Sử dụng BERT để phân loại văn bản -----	47
5.1. Kiến trúc tổng quát của mô hình BERT [2]-----	47
5.2. Biểu diễn đầu vào và đầu ra của BERT [2] -----	49
Đầu vào (Input)-----	49
Đầu ra (Output)-----	50
5.3. Masked Language Model – MLM -----	51
5.4. Next Sequence Prediction – NSP hay Two-sequence tasks -----	52
5.5. Một số mô hình cụ thể-----	53
5.6. Trích xuất đặc trưng sử dụng BERT [2]-----	53
6. Mô hình ViT [1] -----	55
6.1. Kiến trúc tổng quan của ViT [1] -----	55
Kiến trúc của mô hình Vision Transformer (ViT) [1] -----	56
6.2. Tạo các Patch từ hình ảnh đầu vào -----	57
6.3. Làm phẳng các Patch -----	58
6.4. Encoder trong Vision Transformer [1] -----	59
Layer Normalization (LayerNorm - LN) -----	60
Multi-Head Attention (MSA)-----	61
Multi-Layer Perceptron (MLP)-----	62
Multi-Layer Perceptron (MLP) Head-----	63
7. Mô hình GPT-2 [5]-----	64
7.1. Từ Transformer đến GPT-2 -----	64
7.2. Kiến trúc tổng quan của GPT-2 [5] -----	67
7.3. Biểu diễn đầu vào của GPT-2 [5]-----	67
7.4. Biểu diễn đầu ra của GPT-2 [5]-----	69
8. Mô hình ViLT [4]-----	70
9. Từ BERT [2] đến PhoBERT [3] -----	71
10. Phương pháp sinh từ: Greedy Search hay Beam Search?-----	72
10.1. Greedy Search-----	72
10.2. Beam Search-----	72
11. Phương pháp đánh giá mô hình-----	73
11.1. Đánh giá mô hình phân lớp với Accuracy -----	73
11.2. Đánh giá mô hình sinh tự động câu trả lời với ROUGE [8]-----	73
Chương 2. Phương pháp thực hiện-----	75
1. Thu thập dữ liệu-----	75
2. Tiền xử lý dữ liệu -----	76
3. Huấn luyện mô hình 1: VQA sử dụng ViT [1] và BERT [2] / PhoBERT [3] -----	77
4. Huấn luyện mô hình 2: VQA sử dụng ViLT [4] -----	78
5. Huấn luyện mô hình 3: Sinh câu trả lời với ViLT [4] và GPT-2 [5]-----	79
6. Đánh giá mô hình -----	80
7. Xây dựng ứng dụng chatbot đơn giản-----	80

Chương 3. Thực nghiệm -----	81
1. Xây dựng tập dữ liệu-----	81
2. Huấn luyện mô hình-----	83
2.1. Mô hình 1: VQA sử dụng ViT [1] và BERT [2] / PhoBERT [3] -----	85
2.2. Mô hình 2: VQA sử dụng ViLT [4] -----	86
2.3. Mô hình 3: Sinh câu trả lời với ViLT [4] và GPT-2 [5]-----	86
3. Kết quả đánh giá sau khi chạy mô hình-----	86
3.1. Mô hình 1: VQA sử dụng ViT [1] và BERT [2] / PhoBERT [3] -----	86
3.2. Mô hình 2: VQA sử dụng ViLT [4] -----	87
3.3. Mô hình 3: Sinh câu trả lời với ViLT [4] và GPT-2 [5]-----	87
4. Xây dựng ứng dụng chatbot đơn giản-----	88
Phía server (Server-side) -----	89
Phía client (Client-side) -----	90
C. PHẦN KẾT LUẬN-----	92
1. Kết quả đạt được-----	92
2. Thảo luận và hướng phát triển -----	92
PHỤ LỤC -----	95

DANH SÁCH BẢNG

Bảng 1. Kết quả huấn luyện của Mô hình 1: VQA sử dụng ViT [1] và BERT [2] / PhoBERT [3]	-----	86
Bảng 2. Kết quả huấn luyện của Mô hình 2: VQA sử dụng ViLT [4]	-----	87
Bảng 3. Kết quả huấn luyện của Mô hình 2: Sinh câu trả lời với ViLT [4] và GPT-2 [5]	-----	87

DANH SÁCH HÌNH

Hình 1. Biểu diễn ma trận mã hoá one-hot -----	20
Hình 2. Một từ đôi khi chỉ có nghĩa khi đứng cạnh các từ khác (ví dụ: sinh) -----	21
Hình 3. Word2Vec [12] tạo ra các word embedding bằng cách sử dụng mối quan hệ của từ với các từ xung quanh để mô tả từ trong không gian đa chiều. -----	21
Hình 4. Biểu diễn word embedding trên không gian đa chiều-----	21
Hình 5. Kiến trúc cơ bản mô hình Word2Vec [12] -----	22
Hình 6. Huấn luyện Word2Vec [12]: Skip-gram -----	22
Hình 7. Huấn luyện Word2Vec [12]: CBOW -----	23
Hình 8. Biểu diễn Word2Vec [12] trên không gian đa chiều từ Tensorflow Projector -----	23
Hình 9. Mô hình Seq2Seq [17] cho bài toán dịch -----	26
Hình 10. Vấn đề của context vector trong mô hình Seq2Seq [17] -----	27
Hình 11. Sử dụng các trạng thái ẩn của Encoder trong Decoder-----	27
Hình 12. Cơ chế Attention trong mô hình Seq2Seq [17] -----	28
Hình 13. Ví dụ về Từ điển (Dictionary) -----	29
Hình 14. Mô tả quá trình truy vấn với Query là 2018 -----	29
Hình 15. Mô tả quá trình tính toán giá trị Attention -----	29
Hình 16. Mô tả chi tiết quá trình tính toán Attention-----	30
Hình 17. Key và Value được sử dụng tính toán giá trị Attention trong mô hình Seq2Seq [17] là các trạng thái ẩn của Encoder -----	30
Hình 18. Query được sử dụng để tính toán giá trị Attention trong mô hình Seq2Seq [17] là các trạng thái ẩn của Decoder -----	31
Hình 19. Mô tả quá trình tính toán giá trị Attention từ Q (trạng thái ẩn của Decoder), K và V (trạng thái ẩn của Encoder)-----	31
Hình 20. Mô tả quá trình tính toán giá trị Attention trong mô hình Seq2Seq [17]-----	32
Hình 21. Kiến trúc mô hình Transformer [9] -----	33
Hình 22. Hình mô tả quá trình lan truyền ngược trong mô hình mạng RNN-----	34
Hình 23. Mô hình Transformer [9] (trái) và mô hình Seq2Seq [17] (phải) -----	34
Hình 24. Mô tả đầu vào (Input) và đầu ra (Output) của mô hình Transformer [9] -----	35
Hình 25. Tầng Input Embedding và Output Embedding trong mô hình Transformer [9] -----	36
Hình 26. Positional Encoding trong mô hình Transformer [9]-----	37
Hình 27. Quá trình thêm thông tin thứ tự vào dữ liệu đầu vào-----	38
Hình 28. Thể hiện vị trí của từ trong câu-----	39
Hình 29. Công thức tính $PE(pos, 2i)$ và $PE(pos, 2i + 1)$ -----	39
Hình 30. Ma trận mã hoá vị trí của câu “I am a student”-----	39
Hình 31. Giá trị trong ma trận mã hoá vị trí của câu “I am a student”-----	40
Hình 32. Multi-Head Attention trong Transformer -----	41
Hình 33. Multi-Head Attention trong Transformer [9] -----	41
Hình 34. Scaled Dot-Product Attention trong Transformer [9] -----	42
Hình 35. Minh họa quá trình tính toán giá trị Attention z từ bộ ba Q, K và V. -----	43
Hình 36. Feed Forward trong Transformer [9]-----	43
Hình 37. Add & Norm trong Transformer [9] -----	44

Hình 38. Linear và Softmax trong Transformer [9]-----	45
Hình 39. Mô tả cấu trúc Encoder - Decoder trong Transformer-----	45
Hình 40. (1) Pre-training tasks và (2) Sentence classification-----	46
Hình 41. Mô hình BERT cho bài toán phân loại văn bản-----	47
Hình 42. Các phiên bản của BERT [2]-----	47
Hình 43. Số lượng Encoder L trong mô hình BERT [2]-----	48
Hình 44. Mô hình tổng quát của BERT -----	48
Hình 45. Mô phỏng đầu vào của BERT [2] -----	49
Hình 46. Mô phỏng đầu vào của BERT [2] với các khối Encoder-----	49
Hình 47. Đầu ra của mô hình BERT [2] -----	50
Hình 48. Sử dụng SLP để phân lớp từ đầu ra của BERT [2]-----	50
Hình 49. MLM che giấu 15% số từ trong đầu vào và yêu cầu mô hình dự đoán từ còn thiếu. ---	51
Hình 50. Mô tả tác vụ NSP hay Two-sequence tasks trong quá trình tiền huấn luyện BERT [2]	52
Hình 51. Các mô hình cụ thể cho các tác vụ khác nhau của mô hình BERT [2]-----	53
Hình 52. Trích xuất đặc trưng sử dụng BERT [2]-----	54
Hình 53. Lựa chọn contextualized embedding từ các đầu ra Encoder-----	54
Hình 54. Mô hình tổng quát của ViT [1] -----	56
Hình 55. Chia hình ảnh đầu vào thành các Patch(es)-----	57
Hình 56. Làm phẳng Patch(es)-----	58
Hình 57. Quá trình tạo ra đầu vào hoàn chỉnh cho mô hình ViT-----	58
Hình 58. Mô tả các thành phần của Encoder trong ViT -----	59
Hình 59. Quá trình tính toán z_L từ z_0 -----	60
Hình 60. Layer Norm trong Encoder của Vision Transformer-----	60
Hình 61. Công thức tính LayerNorm -----	61
Hình 62. Minh họa quá trình tính toán trong MSA -----	61
Hình 63. Hàm kích hoạt GELU trong MLP -----	62
Hình 64. Phân lớp với MLP Head-----	63
Hình 65. BERT [2] được xây dựng dựa trên Transformer's Encoder [9], trong khi GPT-2 [5] là Transformer's Decoder [9]-----	64
Hình 66. Encoder block trong Transformer gốc-----	64
Hình 67. Decoder block trong Transformer gốc-----	65
Hình 68. Masked Self-Attention trong GPT-2 [5] -----	65
Hình 69. Minh họa Self-Attention và Masked Self-Attention-----	66
Hình 70. Mô hình Transformer-Decoder -----	66
Hình 71. Mô hình GPT-2 [5]-----	67
Hình 72. Biểu diễn đầu vào của mô hình GPT-2 [5]-----	67
Hình 73. Ma trận wte với kích thước từ điển vocab_size = 50.257 -----	68
Hình 74. Ma trận wpe, với seq_length = 1024 (context_size), hidden_state tương ứng với phiên bản sử dụng -----	68
Hình 75. Biểu diễn đầu ra của các khối Decoder-----	69
Hình 76. Biểu diễn đầu ra của mô hình GPT-2 [5] -----	69
Hình 77. Biểu diễn kết quả đầu ra của mô hình GPT-2 [5]-----	70
Hình 78. Tổng quan kiến trúc mô hình ViLT và các tác vụ chính [4]-----	70

Hình 79. Mô tả quá trình sử dụng Greedy Search để sinh từ -----	72
Hình 80. Mô tả quá trình sử dụng Beam Search để sinh từ -----	72
Hình 81. Ví dụ về ROUGE-1 -----	74
Hình 82. Mô hình các bước thực hiện -----	75
Hình 83. Biểu đồ thống kê số lượng hình của từng lớp trong tập dữ liệu. -----	76
Hình 84. Quá trình tiền xử lý dữ liệu để tạo tập tin *.csv-----	76
Hình 85. Kiến trúc mô hình sử dụng BERT và ViT để phân lớp -----	77
Hình 86. pooler_output (khung màu hồng) chính là vector mang đặc trưng của toàn bộ dữ liệu đầu vào được dùng cho tác vụ phân lớp ở đầu ra của mô hình BERT [2]-----	77
Hình 87. Tương tự, ViT [1] cũng có pooler_output (hình tròn màu đỏ), được làm đầu vào cho MLP Head cho tác vụ phân lớp. -----	78
Hình 88. Mô tả cơ bản về kiến trúc của mô hình ViLT [4] -----	79
Hình 89. Mô tả cơ bản về kiến trúc của Mô hình 3-----	79
Hình 90. Một phần nội dung của tập tin <i>data.json</i> -----	81
Hình 91. Một phần các mẫu trong tập dữ liệu train của tập dữ liệu sinh tự động câu trả lời---	82
Hình 92. Thống kê số lượng câu hỏi theo lớp trên tập dữ liệu train-----	82
Hình 93. Thống kê số lượng câu hỏi theo lớp trên tập dữ liệu test -----	83
Hình 94. Sử dụng Tokenizer để chuyển đổi câu thành vector số-----	84
Hình 95. Sử dụng Image-Processor để chuẩn hoá ma trận số từ hình ảnh-----	84
Hình 96. Mô tả cách hoạt động của Data Collator -----	85
Hình 97. Nhắc lại kiến trúc của Mô hình 1 -----	85
Hình 98. Sử dụng Gradio tạo giao diện đơn giản tương tác với mô hình-----	88
Hình 99. Quy trình tiếp nhận và xử lý dữ liệu trong server -----	89
Hình 100. Giao diện chính của ứng dụng chatbot (Ảnh từ iPhone 14 Pro Max ảo)-----	90
Hình 101. Chuyển đổi ngôn ngữ sang tiếng Việt (Ảnh từ iPhone 14 Pro Max ảo) -----	91
Hình 102. Ứng dụng chatbot hỗ trợ tiếng Việt (Ảnh từ iPhone 12 thật) -----	91

DANH MỤC TỪ VIẾT TẮT

STT	Từ viết tắt	Nghĩa của từ
1	VQA	Visual Question Answering
2	NLP	Natural Language Processing
3	ViT	Vision Transformer
4	BERT	Bidirectional Encoder Representations from Transformers
5	ViLT	Vision-and-Language Transformer
6	GPT-2	Generative Pre-trained Transformer 2
7	CBOW	Countinous Bag of Words
8	BPE	Byte-Pair Encoding
9	Seq2Seq	Sequence to Sequence
10	RNN	Recurrent Neural Network
11	LSTM	Long Short-Term Memory
12	BPTT	Back Propagation Through Time
13	MLM	Masked Language Model
14	NSP	Next Sequence Prediction
15	SLP	Single-Layer Perceptron
16	MLP	Multi-Layer Perceptron
17	LN	Layer Normalization
18	MSA	Multi-Head (Self)-Attention
19	ROUGE	Recall-Oriented Understudy for Gisting Evaluation
20	GPU	Graphics Processing Unit
21	LCS	Longest Common Subsequence
22	API	Application Programming Interface

A. PHẦN MỞ ĐẦU

Phần mở đầu sẽ trình bày sự cần thiết của đề tài cũng như là giới thiệu các công trình nghiên cứu có liên quan, đối tượng và phạm vi nghiên cứu đề tài, cuối cùng là cấu trúc luận văn.

1. Lý do chọn đề tài

Thị giác máy tính (Computer Vision) là một lĩnh vực của khoa học máy tính giúp cho các máy tính có khả năng nhận diện, phân tích và xử lý các hình ảnh và video một cách tự động. Thị giác máy tính có thể được áp dụng trong nhiều lĩnh vực khác nhau, chẳng hạn như trong y tế, an ninh, công nghiệp và thương mại điện tử. Với sự phát triển của thị giác máy tính, các máy tính ngày nay có thể nhận diện khuôn mặt, đếm số lượng đối tượng trong một bức ảnh, phát hiện và phân loại các đối tượng trong video và rất nhiều ứng dụng khác. Thị giác máy tính giúp cho các máy tính có khả năng nhận biết thế giới xung quanh chúng và xử lý thông tin một cách nhanh chóng và chính xác, mở ra nhiều tiềm năng ứng dụng hứa hẹn cho tương lai.

Ngoài việc nhận diện, phân tích và xử lý hình ảnh, thị giác máy tính còn có khả năng giải quyết các vấn đề trả lời câu hỏi từ hình ảnh (Visual Question Answering – VQA). Với VQA, các máy tính có thể trả lời các câu hỏi được đặt ra từ một bức ảnh hoặc một đoạn video. Ví dụ, nếu có một bức ảnh về một con mèo đang nằm trên một chiếc ghế, VQA sẽ có thể trả lời các câu hỏi như “Con vật trên bức ảnh là gì?”, “Nó đang nằm ở đâu?”, hoặc “Con vật đó có màu gì?”.

VQA là một lĩnh vực mới nổi trong thị giác máy tính, tuy nhiên nó đã thu hút sự quan tâm của nhiều nhà nghiên cứu và các công ty công nghệ hàng đầu thế giới. Nếu được phát triển thành công, VQA có thể được ứng dụng rộng rãi trong các lĩnh vực như du lịch, thương mại điện tử, y tế và an ninh, mang lại nhiều tiện ích cho cuộc sống và công việc của chúng ta.

Xử lý ngôn ngữ tự nhiên (Natural Language Processing – NLP) cũng là một lĩnh vực của khoa học máy tính giúp cho các máy tính có khả năng hiểu và thực hiện các nhiệm vụ liên quan đến ngôn ngữ của con người một cách hiệu quả, chẳng hạn như tương tác giữa người và máy, cải thiện hiệu quả giao tiếp giữa con người với con người, hoặc đơn giản là nâng cao hiệu quả xử lý văn bản và lời nói. NLP đã được áp dụng rộng rãi trong nhiều lĩnh vực của cuộc sống, chẳng hạn như nhận dạng giọng nói, dịch máy, rút trích thông tin và trả lời câu hỏi, và còn nhiều ứng dụng khác. Với NLP, các máy tính hiện nay có thể nhận diện các mẫu ngôn ngữ nói, dịch ngôn ngữ, trích xuất thông tin quan trọng và cung cấp các câu trả lời thông minh cho các câu hỏi.

Kết hợp giữa NLP và VQA có thể mang lại nhiều lợi ích và tiện ích cho việc xử lý thông tin từ hình ảnh và video. Khi kết hợp NLP vào VQA, các máy tính sẽ có khả năng hiểu và trả lời các câu hỏi phức tạp hơn từ hình ảnh và video. NLP sẽ giúp cho máy tính có thể phân tích và hiểu ý nghĩa của các từ, cụm từ và ngữ cảnh được sử dụng trong câu hỏi. Điều này sẽ giúp cho máy tính có khả năng trả lời các câu hỏi có tính chất phức tạp hơn, như các câu hỏi về ngữ nghĩa, tương phản hoặc suy luận. Ví dụ, nếu có một hình ảnh về một chiếc ô tô đang chạy trên đường, NLP có thể giúp cho máy tính hiểu được nghĩa của câu hỏi “Chiếc ô tô đó đi đến đâu?” và có thể trả lời “Nó đang đi trên đường”.

Kết hợp giữa NLP và VQA cũng có thể giúp cho máy tính có khả năng phát triển các mô hình ngôn ngữ tự nhiên để hiểu và tương tác với con người một cách tự nhiên hơn. Ví dụ, nếu có một bức ảnh về một chiếc xe bus, một câu hỏi như “Có thể tôi đến trường bằng xe bus không?” sẽ được máy tính hiểu và trả lời một cách tự nhiên hơn.

Đồng bằng sông Cửu Long đã và đang là một trong những điểm đến du lịch nổi tiếng và hấp dẫn ở Việt Nam. Với sự đa dạng về cảnh quan thiên nhiên, văn hóa, lịch sử và ẩm thực, du khách đến với vùng đất này sẽ có những trải nghiệm tuyệt vời. Trong đó, ẩm thực đặc sản là một yếu tố không thể bỏ qua.

Đồng bằng sông Cửu Long là vùng đất đa dạng về nguồn lương thực và thực phẩm, với các loại đặc sản nổi tiếng như bánh xèo, bánh tét, cơm tấm... Chatbot hỗ trợ quảng bá các món ăn đặc sản ở đồng bằng sông Cửu Long – được phát triển nhằm giới thiệu những món ăn đặc sản đến với du khách từ khắp nơi trên thế giới. Điều này giúp khách du lịch có cơ hội tìm hiểu và trải nghiệm ẩm thực đặc trưng của vùng đất này.

Bên cạnh đó, việc xây dựng **Chatbot hỗ trợ quảng bá các món ăn đặc sản ở đồng bằng sông Cửu Long** không chỉ góp phần phát triển kinh tế du lịch của vùng đất này mà còn góp phần giữ gìn và phát huy những giá trị văn hóa, lịch sử và ẩm thực đặc trưng của vùng đất đồng bằng sông Cửu Long.

2. Mục tiêu nghiên cứu

Đề tài nghiên cứu phương pháp xây dựng chatbot hỗ trợ quảng bá các món ăn đặc sản ở đồng bằng sông Cửu Long – dựa trên Transformer’s Encoder [9] (mô hình BERT [2] cho văn bản kết hợp mô hình ViT [1] cho hình ảnh và mô hình ViLT cho cả văn bản và hình ảnh) cùng với Transformer’s Decoder [9] (mô hình GPT-2 [5]) bằng cách trích xuất đặc trưng từ hình ảnh món ăn và câu hỏi liên quan đến món ăn, từ đó phân lớp hoặc sinh ra câu trả lời phù hợp.

3. Các nghiên cứu có liên quan

Trả lời câu hỏi từ hình ảnh (VQA) đã và đang rất được quan tâm.

Wonjae Kim, Bokyung Son và Ildoo Kim đã đề xuất mô hình ViLT [4] đạt 71.26% trên tập dữ liệu kiểm tra của VQAv2¹.

Junnan Li, Dongxu Li, Caiming Xiong và Steven Hoi đề xuất mô hình BLIP [10] đạt 77.54% trên dập dữ liệu kiểm tra của VQA.

Mô hình GIT [11] được Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu và Lijuan Wang đề xuất năm 2022 đạt 78.81% trên tập dữ liệu VQAv2.

Nhìn chung, cả ba mô hình trên, ViLT, BLIP và GIT đều là các Multimodal có khả năng xử lý được cả dữ liệu hình ảnh và văn bản một cách đồng thời.

Ngoài ra, các mô hình này đều sử dụng kiến trúc của mô hình Transformer [9], sử dụng cơ chế Attention để kết hợp thông tin từ nhiều nguồn dữ liệu khác nhau (hình ảnh và văn bản).

Bên cạnh đó, ViLT, BLIP và GIT đều được huấn luyện trên một tập dữ liệu lớn do đó cả ba đều có khả năng Transfer Learning tốt.

4. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu của đề tài bao gồm:

- Tham khảo các nghiên cứu liên quan đến đề tài.
- Mô hình VQA với đầu vào bao gồm hình ảnh món ăn và câu hỏi liên quan đến món ăn, sử dụng BERT [2] và ViT [1] đối với tiếng Anh, PhoBERT [3] và ViT [1] dành cho tiếng Việt.
- Mô hình VQA dựa trên kiến trúc của ViLT [4] (tiếng Anh).
- Mô hình sinh câu trả lời từ hình ảnh món ăn và câu hỏi liên quan đến món ăn, sử dụng ViLT [4] và GPT-2 [5] dành cho tiếng Anh.
- Phương pháp đánh giá mô hình.

Phạm vi nghiên cứu của luận văn này là các hình ảnh về món ăn đặc sản của đồng bằng sông Cửu Long, kèm theo những câu hỏi và câu trả lời liên quan đến món ăn bằng tiếng Anh và tiếng Việt.

¹ <https://visualqa.org>

5. Phương pháp nghiên cứu

Phương pháp nghiên cứu của đề tài chủ yếu dựa trên các nguồn tài liệu khác nhau được công bố trên internet. Tìm hiểu về học sâu, các mô hình trích xuất đặc trưng văn bản – đặc trưng ảnh, các phương pháp biểu diễn văn bản, các phương pháp đánh giá mô hình. Từ đó, kết hợp các phương pháp, kỹ thuật, công cụ khác nhau nhằm giải quyết vấn đề mà đề tài đề ra.

6. Cấu trúc luận văn

Nội dung bao gồm 03 phần chính:

- Phần mở đầu: giới thiệu tổng quan của đề tài, trình bày sơ lược về thị giác máy tính, trả lời câu hỏi từ hình ảnh, xử lý ngôn ngữ tự nhiên cũng như sự cần thiết của đề tài. Các nghiên cứu có liên quan đã được thực nghiệm; đối tượng và phạm vi nghiên cứu của đề tài, sau cùng là phương pháp nghiên cứu để thực hiện đề tài.
- Phần nội dung: gồm 03 chương:
 - Chương 1 – Cơ sở lý thuyết: sẽ giới thiệu tổng quan về bài toán trả lời câu hỏi từ hình ảnh, các khái niệm liên quan của các phương pháp thực hiện đề tài, như: Khái niệm về nhúng từ (Word Embedding), thuật toán Greedy Search – Beam Search, mô hình Transformer [9], mô hình BERT [2], mô hình ViT [1], mô hình GPT-2 [5], mô hình ViLT [4], các phương pháp đánh giá mô hình.
 - Chương 2: mô tả phương pháp thực hiện đề tài bao gồm các bước: thu thập dữ liệu, tiền xử lý dữ liệu, huấn luyện mô hình cũng như quá trình đánh giá để tinh chỉnh các thông số của mô hình. Sau cùng, phát triển một ứng dụng chatbot đơn giản trên điện thoại di động.
 - Chương 3: tiến hành thực nghiệm trên mô hình, kết quả đạt được sau khi đánh giá mô hình.
- Phần kết luận và đề xuất: tổng kết kết quả đạt được của nghiên cứu và đề xuất hướng phát triển cho đề tài.

B. PHẦN NỘI DUNG

Chương 1. Cơ sở lý thuyết của đề tài

Chương 1 trình bày khái quát về các hướng tiếp cận mục tiêu, trình bày các khái niệm liên quan, cũng như là các phương pháp thực hiện và đánh giá mô hình.

1. Trả lời câu hỏi từ hình ảnh (Visual Question Answering – VQA)

Visual Question Answering (VQA) là một bài toán trong lĩnh vực học sâu (Deep Learning) kết hợp giữa xử lý hình ảnh và xử lý ngôn ngữ tự nhiên.

Mô hình VQA nhận đầu là một hình ảnh và một câu hỏi, sau đó mô hình sẽ trả lời câu hỏi bằng một câu trả lời đúng cho câu hỏi đó. Bài toán này đòi hỏi khả năng hiểu ngữ nghĩa của câu hỏi và hình ảnh, khả năng phân tích và trích xuất đặc trưng từ hình ảnh và khả năng ánh xạ từ hình ảnh và câu hỏi sang một câu trả lời đúng.

VQA được ứng dụng trong nhiều lĩnh vực, từ robot hỗ trợ cho người khuyết tật đến hệ thống hỗ trợ khách hàng tự động.

2. Nhúng từ (Word Embedding)

Nhúng từ (Word Embedding) thực hiện ánh xạ các từ hoặc cụm từ sang dạng vector số (thường là số thực). Các vector từ được biểu diễn theo phương pháp nhúng từ thể hiện được ngữ nghĩa của các từ, từ đó nhận ra được mối quan hệ giữa các từ với nhau. Nhúng từ được chia thành 02 loại:

- Frequency-based embedding: dựa vào tần suất xuất hiện của các từ để tạo ra các vector số đại diện cho từ. Tiêu biểu là TF-IDF-Vectorizer², CountVectorizer³.
- Prediction-based embedding: xây dựng các vector số đại diện cho từ dựa trên mô hình dự đoán. Word2Vec [12] là đại diện tiêu biểu nhất, Word2Vec được xây dựng trên mô màng neuron gồm 3 tầng: Tầng Input, tầng ẩn và tầng Output (Hình 5). Mục đích chính của mạng neuron này học các trọng số để biểu diễn vector số đặc trưng cho từ.

Trong việc xử lý đầu vào ở dạng văn bản, do máy tính chỉ có thể hiểu được các đầu vào dạng số nên chúng ta cần phải chuyển các từ về dạng số để máy tính có thể hiểu được. Dưới đây là một số phương pháp thông dụng.

² https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

³ https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

2.1. One-Hot Encoding

Trong xử lý ngôn ngữ tự nhiên (NLP), One-Hot Encoding được sử dụng để biểu diễn các từ trong văn bản. Kỹ thuật này chuyển đổi một từ thành một vector đặc trưng có độ dài bằng với số lượng từ trong từ điển và chỉ có một giá trị bằng 1 tại vị trí tương ứng với từ đó, còn lại là giá trị 0.

Việc biểu diễn các từ bằng One-Hot Encoding giúp cho việc xử lý ngôn ngữ dễ dàng hơn, vì ta có thể sử dụng các phép toán đơn giản như phép nhân và cộng với nhau để tính toán độ tương đồng giữa các từ.

Tuy nhiên, One-Hot Encoding cũng có một số hạn chế, như kích thước của vector đặc trưng sẽ rất lớn nếu số lượng từ trong từ điển quá lớn, và việc tính toán độ tương đồng giữa các từ không phản ánh được sự liên hệ ngữ nghĩa giữa chúng.

Giả sử ta có một tập từ vựng gồm 5 từ: “apple”, “banana”, “orange”, “grape”, và “watermelon”. Để mã hóa các từ này thành các vector one-hot, chúng ta tạo ra một ma trận với số hàng và số cột bằng với số lượng từ vựng. Trong trường hợp này, chúng ta sẽ tạo ra một ma trận 5×5 , với mỗi hàng đại diện cho một từ vựng, và mỗi cột đại diện cho một phần tử trong vector one-hot.

	apple	banana	orange	grape	waterm -elon
apple	1	0	0	0	0
banana	0	1	0	0	0
orange	0	0	1	0	0
grape	0	0	0	1	0
waterm -elon	0	0	0	0	1

Danh sách từ
trong từ điển

Ma trận mã hoá one-hot tương ứng

Hình 1. Biểu diễn ma trận mã hoá one-hot

Ví dụ, để mã hóa từ “apple” thành vector one-hot, ta sẽ đặt giá trị 1 ở vị trí đầu tiên của vector, vị trí đại diện cho từ “apple” và giá trị 0 cho các vị trí còn lại. Tương tự, để

mã hóa từ “banana”, ta sẽ đặt giá trị 1 ở vị trí thứ hai của vector, vị trí đại diện cho từ “banana” và giá trị 0 cho các vị trí còn lại. Quá trình này sẽ được lặp lại cho tất cả các từ trong từ vựng, và ta sẽ thu được ma trận các vector one-hot đại diện cho toàn bộ từ vựng. (Hình 1)

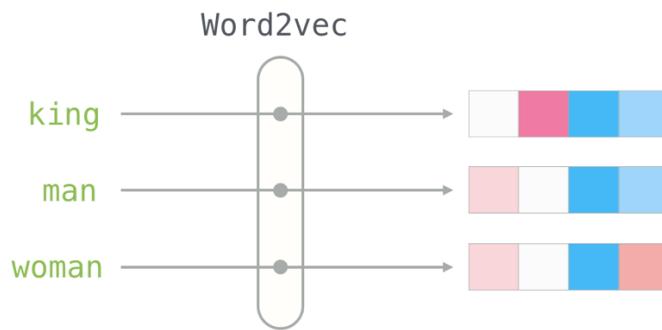
2.2. Word2Vec [12]

Word2Vec [12] là một trong những kỹ thuật tạo ra word embedding được giới thiệu vào 2013.

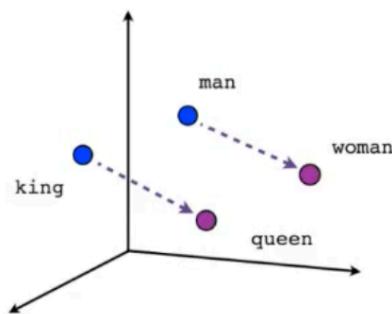


Hình 2. Một từ đôi khi chỉ có nghĩa khi đứng cạnh các từ khác (ví dụ: sinh)

Xét các câu “học sinh đến trường sớm”, “môn Sinh học là một môn khó”. Dễ dàng nhận thấy, nếu từ “sinh” chỉ đúng riêng lẻ thì đây là từ không có nghĩa (Hình 2). Do đó, Word2Vec [12] sẽ tạo ra các word embedding bằng cách dựa trên mối quan hệ giữa từ với các từ xung quanh.



Hình 3. Word2Vec [12] tạo ra các word embedding bằng cách sử dụng mối quan hệ của từ với các từ xung quanh để mô tả từ trong không gian đa chiều.

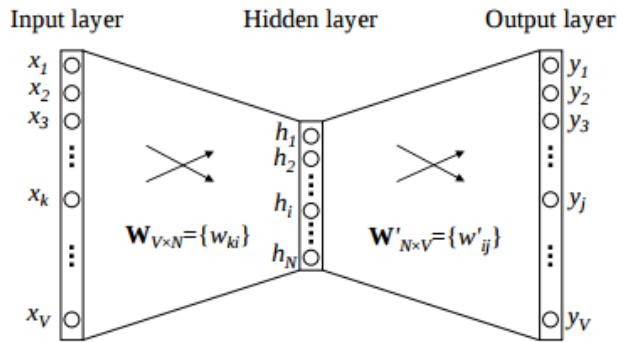


Hình 4. Biểu diễn word embedding trên không gian đa chiều

Nhận thấy, word embedding của một từ là một vector có kích thước là $d \rightarrow$ Chúng ta có thể biểu diễn vector này trên một không gian d -chiều (Hình 4).

Tiếp tục, xét các câu “người đàn ông (man) này thật mạnh mẽ!”, “người phu nữ (woman) này thật mạnh mẽ!”. Theo đó, cụm từ “đàn ông” có thể thay thế cho cụm từ “phu nữ” hoặc ngược lại, mà vẫn tạo thành một câu có nghĩa, hay cụm từ “đàn ông” và “phu nữ” có khả năng được sử dụng trong cùng một ngữ cảnh.

Ngoài việc, sử dụng mối liên hệ giữ từ với các từ xung quanh, Word2Vec [12] có khả năng biểu diễn các từ trong ngữ cảnh giống nhau thành các vector số thực có độ tương đồng cao được tính theo công thức Cosine-similarity⁴. Nói cách khác, khi xét hai word embedding đại diện cho hai từ, nếu giá trị của công thức Cosine-similarity càng gần 1, thì ngữ cảnh của chúng càng giống nhau.

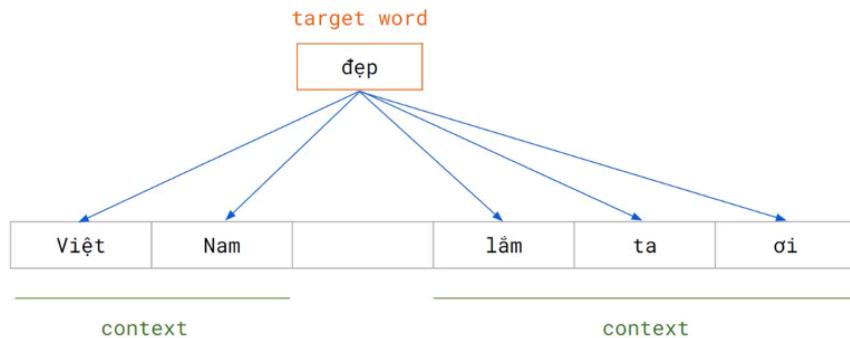


Hình 5. Kiến trúc cơ bản mô hình Word2Vec [12]

Huấn luyện mô hình Word2Vec [12]

Cách 1. Skip-gram:

- Đầu vào (Input) sẽ là một từ.
- Đầu ra (Output) là các từ xung quanh của từ đầu vào. (Hình 6)

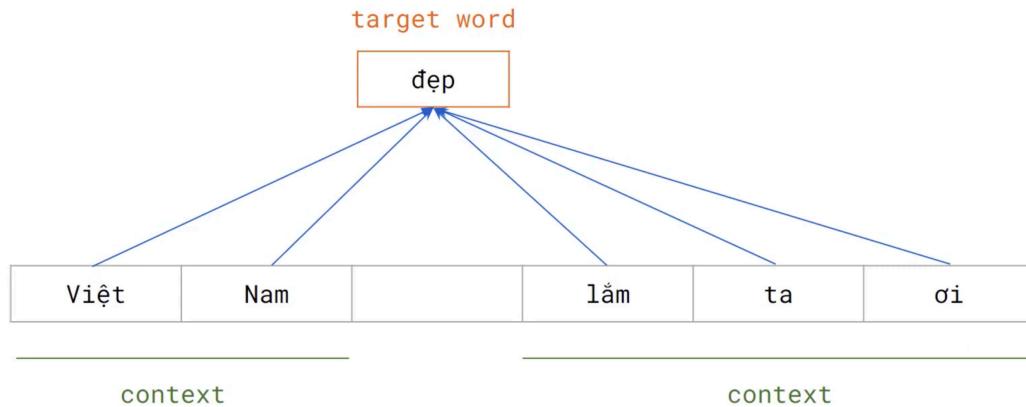


Hình 6. Huấn luyện Word2Vec [12]: Skip-gram

⁴ https://en.wikipedia.org/wiki/Cosine_similarity

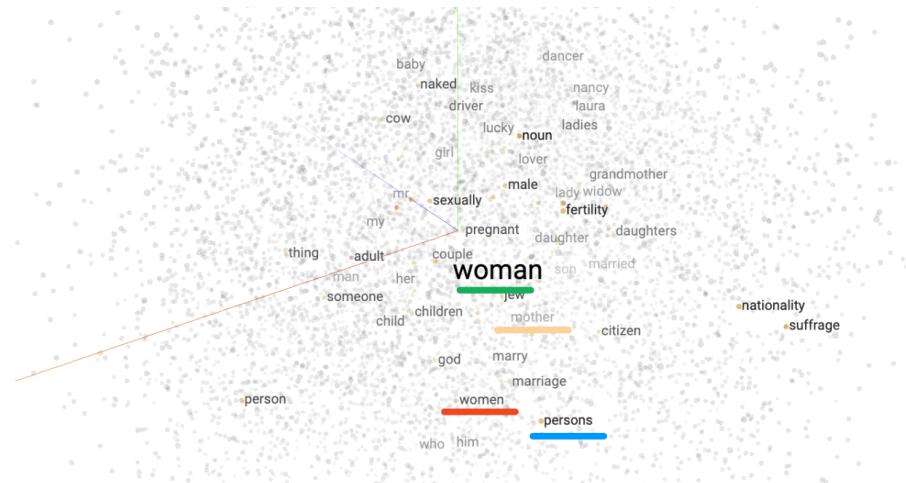
Cách 2. Countinuous Bag of Words – CBOW:

- Đầu vào (Input): một câu đã bị lược bỏ đi một từ.
 - Đầu ra (Output): từ còn thiếu phù hợp với câu ở đầu vào. (Hình 7)



Hình 7. Huấn luyện Word2Vec [12]: CBOW

Đầu vào của cả hai cách huấn luyện (1) và (2) đều sử dụng One-Hot Encoding (mục 2.1) để biểu diễn các từ trong câu.



Hình 8. Biểu diễn Word2Vec [12] trên không gian đa chiều từ Tensorflow Projector⁵

Hình 8, các word embedding được tạo ra bởi Word2Vec được biểu diễn trên không gian đa chiều. Nếu chúng ta xét từ “woman”, dễ dàng nhận thấy các từ như “mother”, “women”, “persons”, ... sẽ có độ tương đồng cao với từ “woman”.

⁵ <https://projector.tensorflow.org>

2.3. Subword Embedding

Để hạn chế các từ nằm ngoài từ điển cũng như là các từ có mối quan hệ gần nhau được thể hiện bằng các vector khác nhau như “cat” với “cats”. Khái niệm subword embedding ra đời để khắc phục hạn chế trên.

2.3.1. Byte Pair Encoding – BPE [13]

Byte-Pair Encoding (BPE) ban đầu được phát triển như một thuật toán để nén văn bản, sau đó được OpenAI sử dụng để tách từ khi tiền huấn luyện mô hình GPT.

BPE được sử dụng bởi rất nhiều mô hình dựa trên Transformer, bao gồm GPT, GPT-2 [5], RoBERTa [14], BART, DeBERTa.

Thuật toán huấn luyện (training)

Quá trình huấn luyện BPE bắt đầu bằng việc tính toán tập hợp duy nhất (unique set) các từ được sử dụng trong tập dữ liệu (văn bản) (sau khi hoàn thành các bước chuẩn hóa và tiền xử lý), sau đó xây dựng từ điển với các từ vựng (vocabulary) là tất cả các ký tự trong tập hợp duy nhất (unique set) đã được tính toán trước đó. Ví dụ đơn giản, giả sử tập văn bản của chúng ta sử dụng năm từ: “hug”, “pug”, “pun”, “bun”, “hugs” \Rightarrow Từ điển sẽ bao gồm các từ vựng cơ bản (basic vocabulary) [“b”, “g”, “h”, “n”, “p”, “s”, “u”].

Tuy nhiên, trong thực tế, từ điển sẽ phải bao gồm thêm toàn bộ tất cả các ký tự ASCII, và có thể thêm một vài ký tự Unicode.

Lý do là khi mã hoá một ký tự không có trong tập dữ liệu huấn luyện, thì ký tự đó sẽ được chuyển đổi thành unknown token (hay thường thấy với ký hiệu <OOV> - Out of Vocabulary). Đó là một trong những nguyên nhân tại sao nhiều mô hình xử lý ngôn ngữ tự nhiên (NLP) rất kém khi phân tích nội dung có chứa biểu tượng cảm xúc, ví dụ như emoji.

Sau khi có được bộ từ vựng cơ bản (base vocabulary), chúng ta sẽ thêm các token mới cho đến khi đạt được kích thước từ vựng mong muốn bằng phép gộp (merges) – là các quy tắc để gộp hai phần tử của bộ từ vựng hiện có thành một phần tử mới. Do đó, ban đầu các phép gộp (merges) này sẽ tạo ra các token với hai ký tự, và dần dần theo qua trình huấn luyện các subword dài hơn sẽ được tạo ra.

Trong quá trình huấn luyện, thuật toán BPE sẽ tìm kiếm cặp token liên tiếp phổ biến nhất trong các token đã có và cặp token phổ biến nhất đó sẽ được gộp (merges) với nhau, và quá trình này sẽ tiếp tục lặp lại.

Giả sử, chúng ta có tần suất xuất hiện của các từ như sau: từ “hug” xuất hiện 10 lần trong tập văn bản, “pug” 5 lần, “pun” 12 lần, “bun” 4 lần và “hugs” 5 lần, tương ứng với ký hiệu dưới dạng từ điển: (“hug”, 10); (“pug”, 5); (“pun”, 12); (“bun”, 4); (“hugs”, 5).

Chúng ta bắt đầu quá trình huấn luyện bằng cách tách mỗi từ thành các ký tự: (“h” “u” “g”, 10), (“p” “u” “g”, 5), (“p” “u” “n”, 12), (“b” “u” “n”, 4), (“h” “u” “g” “s”, 5).

Tiếp theo chúng ta xem xét các cặp ký tự. Cặp (“h”, “u”) xuất hiện trong các từ “hug” và “hugs”, nên tổng cộng xuất hiện 15 lần trong tập văn bản. Tuy nhiên, nó không phải là cặp xuất hiện nhiều nhất, cặp xuất hiện nhiều lần nhất là cặp (“u”, “g”), xuất hiện trong “hug”, “pug” và “hugs”, với tổng cộng 20 lần trong tập từ vựng.

Như vậy, quy tắc ghép cặp đầu tiên mà tokenizer học được là (“u”, “g”) → “ug”, có nghĩa là “ug” sẽ được thêm vào từ điển và các cặp này sẽ được ghép lại trong tất cả các từ của bộ văn bản. Khi kết thúc giai đoạn này, bộ từ vựng và bộ văn bản trông như sau:

Vocabulary	[“b”, “g”, “h”, “n”, “p”, “s”, “u”, “ug”]
Corpus	(“h” “ug”, 10), (“p” “ug”, 5), (“p” “u” “n”, 12), (“b” “u” “n”, 4), (“h” “ug” “s”, 5)

Chúng ta tiếp tục thực hiện quá trình trên cho đến khi đạt được kích thước từ vựng (vocab_size) mà chúng ta mong muốn.

2.3.2. WordPiece [15] [16]

WordPiece là tokenization algorithm được Google phát triển để tiền huấn luyện (pre-training) cho BERT. Sau đó, nó đã được sử dụng lại trong khá nhiều mô hình Transformer dựa trên BERT, chẳng hạn như DistilBERT, MobileBERT.

Thuật toán huấn luyện (training)

Giống như BPE, WordPiece bắt đầu từ một từ vựng nhỏ (small vocabulary) bao gồm các token đặc biệt được sử dụng bởi mô hình (như <CLS>, <SEP>, <PAD> trong BERT) và bảng chữ cái ban đầu.

Do WordPiece xác định các subword bằng cách thêm một tiền tố (như ## trong BERT), mỗi từ ban đầu sẽ được phân tách bằng cách thêm tiền tố đó vào tất cả các ký tự bên trong từ. Vì vậy, ví dụ như “word” sẽ bị phân tách như sau: w, ##o, ##r, ##d.

Do đó, từ điển ban đầu chứa tất cả các ký tự xuất hiện ở đầu một từ và các ký tự xuất hiện trong một từ đi kèm với tiền tố được quy định bởi WordPiece.

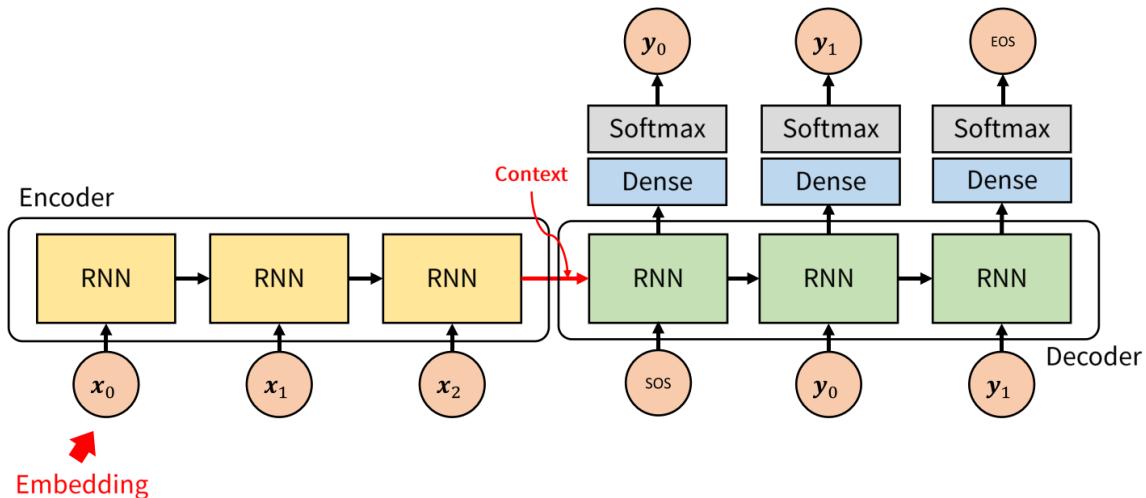
Sau đó, giống như BPE, nhưng WordPiece thay đổi quy tắc để chọn cặp ghép nối (merges). Thay vì chọn cặp phổ biến nhất, WordPiece tính điểm cho mỗi cặp bằng công thức sau:

$$score = \frac{\text{freq_of_pair}}{\text{freq_of_first_element} \times \text{freq_of_second_element}}$$

3. Mô hình Seq2Seq [17] và cơ chế Attention (Attention Mechanism)

3.1. Sơ lược về mô hình Seq2Seq [17] trong bài toán dịch

Trong phần này, chúng ta sẽ cùng tìm hiểu về cách mà mô hình Seq2Seq [17] đã sử dụng cơ chế Attention.



Hình 9. Mô hình Seq2Seq [17] cho bài toán dịch

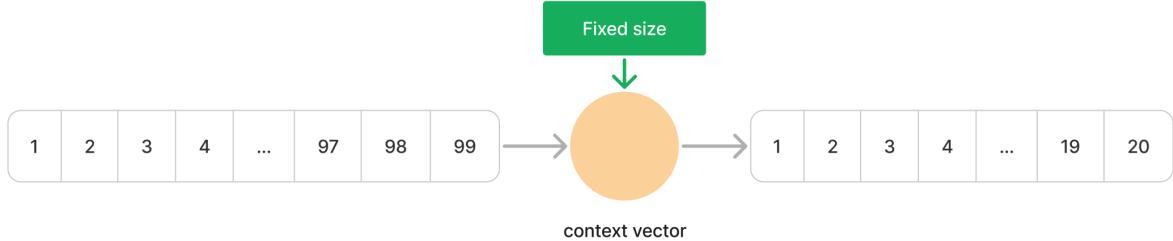
Hình 9 mô tả một mô hình Seq2Seq [17] cho bài toán dịch. Đầu tiên chúng ta sử dụng một phương pháp Word Embedding bất kỳ, chẳng hạn như Word2Vec (mục 2.2) để chuyển đổi các từ thành vector số thực gọi là Embedding. Sau đó, các Embedding tương ứng với mỗi từ sẽ được truyền vào Encoder dưới dạng đầu vào.

Trong Encoder, mỗi Embedding của từ và các trạng thái ẩn (hidden state) của RNN sẽ được kết hợp để nén thông tin và đầu ra cuối cùng của Encoder sẽ trở thành context vector. Nói cách khác, mục đích của Encoder chính là tạo ra context vector. Context vector là một vector đơn giản bao gồm các số thực. Kích thước của context vector có thể được thiết lập ban đầu khi khởi tạo mô hình, thông thường có thể là 256, 512, 1024 và nhiều giá trị khác.

Phần đầu tiên trong Decoder sẽ nhận đầu vào là context vector cùng với $\langle \text{SOS} \rangle$ (Start Of Sequence – bắt đầu chuỗi) đại diện cho phần bắt đầu của câu. Sau khi đi qua RNN, các trạng thái ẩn (hidden state) sẽ được tiếp tục kết hợp với step tiếp theo, sau đó sẽ đi qua tầng Dense và Softmax để sinh ra y_0 . Sau đó y_0 được sử dụng như là đầu vào của RNN một lần nữa. Nói cách khác, RNN trong Decoder nhận vào y_i kèm các trạng thái ẩn và tạo ra y_{i+1} .

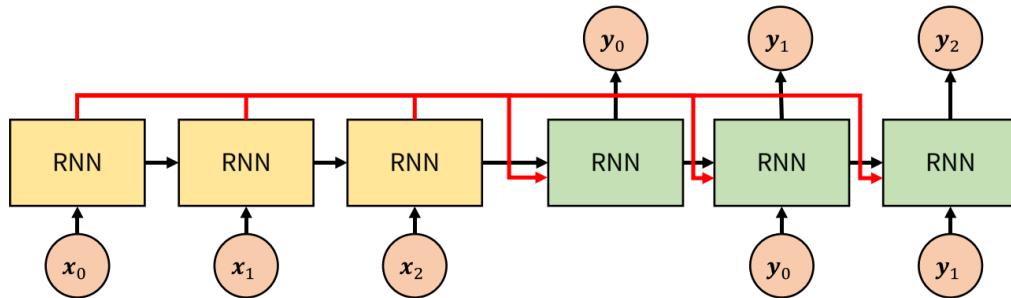
Có thể thấy trong suốt quá trình này, việc Decoder chỉ nhận duy nhất một vector là context vector từ kết quả của Encoder làm đầu vào có thể gây ra một số vấn đề.

Gradient Vanishing là một vấn đề có thể xảy ra khi sử dụng chỉ context vector làm đầu vào cho Decoder. Lý do là do Encoder và Decoder được hoàn toàn tách rời với nhau dựa trên context vector, vì vậy mối quan hệ giữa đầu vào và đầu ra rất yếu, dẫn đến vấn đề Gradient Vanishing khi huấn luyện với back-propagation.



Hình 10. Vấn đề của context vector trong mô hình Seq2Seq [17]

Hơn nữa, cách tiếp cận này có thể dẫn đến mất mát thông tin do việc mã hóa ý nghĩa của mỗi từ thông qua RNN và hợp nhất chúng vào trong một vector context duy nhất. Như Hình 10, việc ánh xạ thông tin của 99 vector nhúng từ thành một vector context duy nhất có thể là không thực tế.



Hình 11. Sử dụng các trạng thái ẩn của Encoder trong Decoder

Để hạn chế việc mất thông tin, chúng ta có thể dùng trạng thái ẩn (hidden state) của Encoder trong Decoder như Hình 11. Khi sửa đổi mô hình theo hướng này chúng ta có thể đạt được 4 lợi ích như sau:

- Không cần “ép buộc” các Embedding của nhiều từ thành một context vector duy nhất.
- Decoder có thể sử dụng các trạng thái ẩn của Encoder một cách linh hoạt hơn. Chẳng hạn, chúng ta có thể thiết kế cơ chế (mechanism) để tập trung (chú ý) hơn vào các trạng thái (state) mà Decoder muốn tập trung (chú ý) hơn.
- Trong RNN, xảy ra vấn đề về việc quên đi các dữ liệu đã nhập vào trước đó. Mặc dù các mô hình như LSTM đã cải thiện vấn đề này, tuy nhiên, đối với dữ liệu đã nhập vào trước đó, ảnh hưởng đến đầu ra vẫn thấp hơn. Do đó, nếu ta sử dụng tất

cả các trạng thái ẩn của Encoder như Hình 11, vẫn đề về việc quên đi các dữ liệu trước có sẽ được cải thiện.

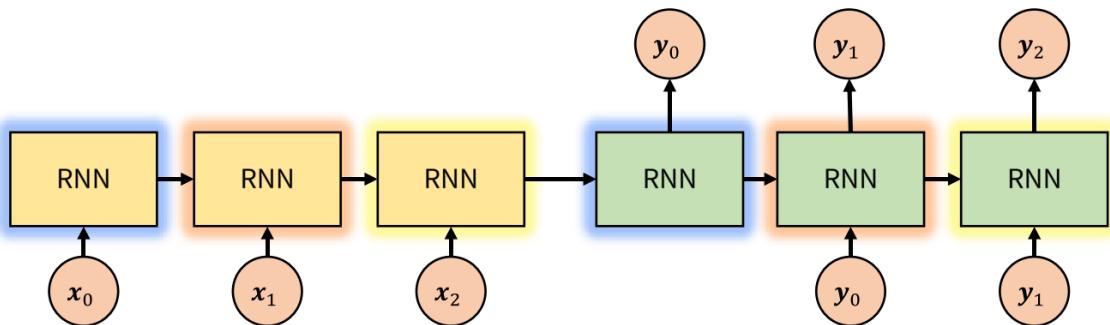
- Do sự xuất hiện của Gradient Highway⁶, nên mô hình sẽ chống lại vấn đề Gradient Vanishing một cách tốt hơn.

Tuy nhiên, trong trường hợp này vẫn có một số vấn đề phát sinh. Trước tiên, số lượng đầu vào được chuyển đến Decoder phụ thuộc vào đầu vào của Encoder. Nghĩa là không thể cố định mô hình cho các câu có độ dài khác nhau. Kích thước tương ứng với vector màu đỏ (Hình 11) sẽ thay đổi theo độ dài của câu.

Vấn đề khác là nếu chỉ đơn giản chuyển tất cả các trạng thái ẩn (hidden state) cho Decoder thì kích thước đầu vào sẽ trở nên rất lớn có thể ảnh hưởng xấu đến hiệu suất của mô hình.

Để giải quyết các vấn đề này, mô hình dựa trên cơ chế Attention đã được phát minh.

3.2. Attention Mechanism



Hình 12. Cơ chế Attention trong mô hình Seq2Seq [17]

Attention có nghĩa theo từ điển là “chú ý” hoặc “tập trung”. Ý nghĩa này đề cập đến việc giúp cho Decoder có thể tập trung vào thông tin nào của Encoder để đưa ra dự đoán đầu ra. Đây chính là ý tưởng cơ bản của cơ chế Attention.

Để hiểu hơn về cơ chế Attention, chúng ta tìm hiểu về các thuật ngữ sau:

- Query (Q): Thông tin truy vấn – Chúng ta đang cần tìm kiếm cái gì?
- Key (K): Khoá – Giá trị tham chiếu được sử dụng khi tìm kiếm dữ liệu được lưu trữ.
- Value (V): Giá trị – Dữ liệu được lưu trữ.

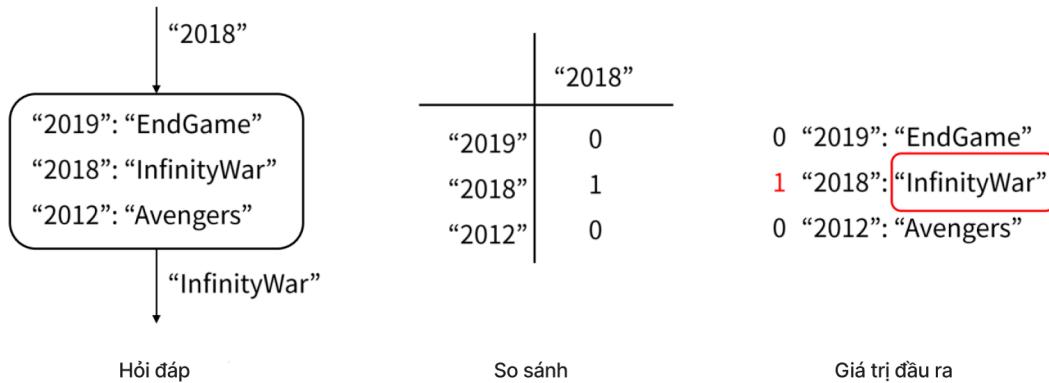
⁶ Gradient Highway là một kỹ thuật trong Deep Learning, được sử dụng để giải quyết vấn đề của Gradient Vanishing trong quá trình lan truyền ngược gradient. Kỹ thuật này cho phép gradient truyền qua nhiều lớp liên tiếp mà không bị giảm mất dần như thường thấy trong mạng RNN, giúp cải thiện khả năng học của mô hình. Trong mô hình Seq2Seq [17], Gradient Highway được tạo ra nhờ việc kết hợp trạng thái ẩn (hidden state) của Encoder và Decoder, giúp thông tin có thể truyền qua các lớp mạng một cách hiệu quả hơn, từ đó cải thiện khả năng học của mô hình.

- Dictionary: Từ điển – Tập hợp gồm các cặp Key: Value.

```
{
    "Nation": "Vietnam",
    "City": "Cantho",
}
```

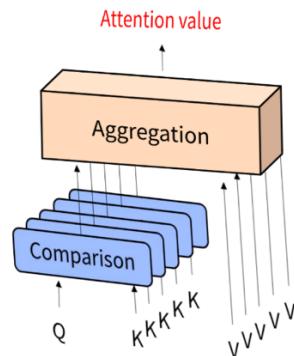
Hình 13. Ví dụ về Từ điển (Dictionary)

Trong ví dụ như Hình 13, “Nation” và “City” là các Key (K), “Vietnam” và “Cantho” là các Value (V) tương ứng với Key (K).



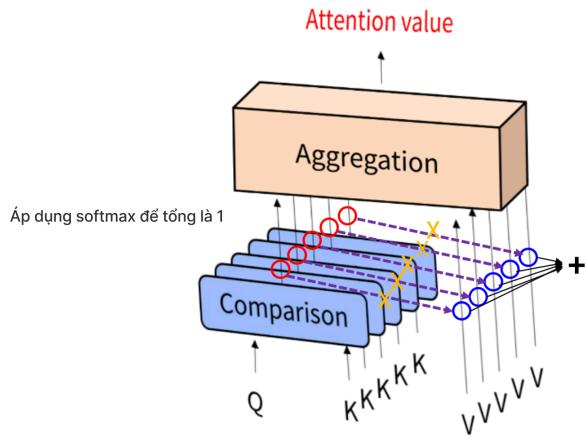
Hình 14. Mô tả quá trình truy vấn với Query là 2018

Thuật ngữ truy vấn – Query: như được mô tả trên Hình 14, khi đầu vào của truy vấn (Query) là “2018”, chúng thực hiện việc truy xuất bằng cách tìm kiếm giá trị khoá (Key) tương ứng với Query và sau đó xuất ra giá trị (Value) tương ứng với Key đó. Việc lựa chọn Key tương ứng với Query, có thể là chọn Key giống như Query, hay chọn Key sao cho tương tự nhất tuỳ vào bài toán cụ thể.



Hình 15. Mô tả quá trình tính toán giá trị Attention

Trong Attention, chúng ta sẽ so sánh Query với các Key rồi dựa trên đó tổng hợp (Aggregation) các Value tương ứng với Key theo Attention Score để tạo ra Attention Value. (Hình 15)



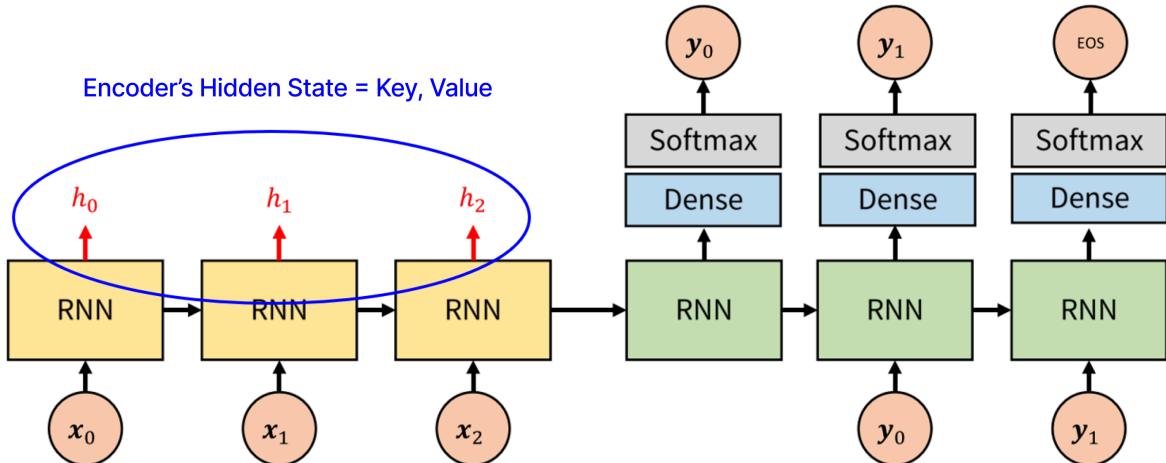
Hình 16. Mô tả chi tiết quá trình tính toán Attention

Lưu ý rằng, ở đây chỉ có một Query, và nó sẽ được tính toán độ tương đồng với các Key phù hợp với Query (Hình 16).

Giá trị Attention (Attention Value) được tính bằng *tổng* của *tích* giữa giá trị hình tròn màu đỏ tương ứng với độ tương đồng và giá trị hình tròn màu xanh dương tương ứng với giá trị. Độ tương đồng hoạt động như một trọng số được nhân vào. Phép tính được thực hiện bằng cách lấy tích vô hướng và tổng của các vector.

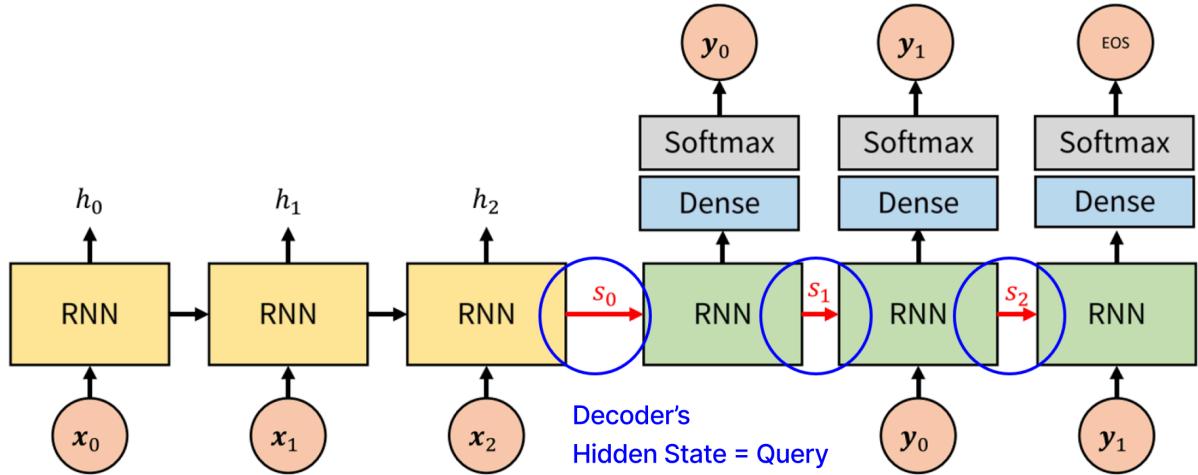
Trong Attention, *Query* là trạng thái ẩn của *Decoder*. Nó là trạng thái ẩn của timestep trước đó được đưa vào RNN của Decoder. *Key* và *Value* là trạng thái ẩn của *Encoder*, có nghĩa là Key và Value giống nhau và có số lượng Key bằng số lượng từ. Lưu ý rằng *hầu hết* các mạng Attention đều sử dụng cùng giá trị cho Key và Value.

3.3. Áp dụng cơ chế Attention vào mô hình Seq2Seq [17]



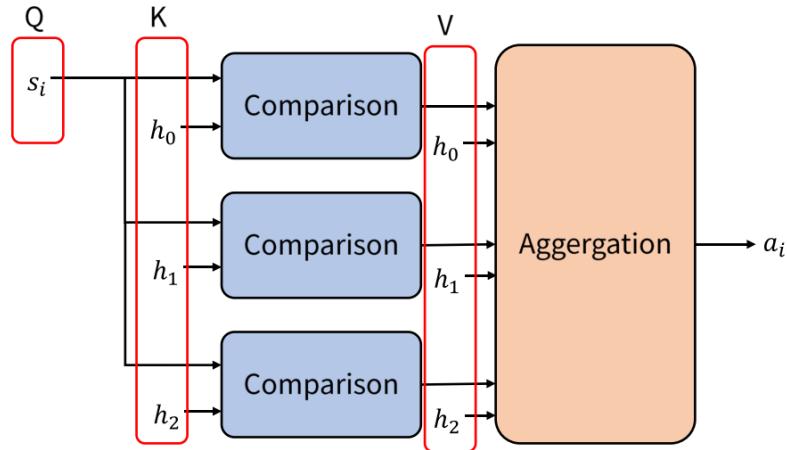
Hình 17. Key và Value được sử dụng tính toán giá trị Attention trong mô hình Seq2Seq [17] là các trạng thái ẩn của Encoder

Như đã đề cập trước đó, Key và Value trong Attention sẽ là trạng thái ẩn của Encoder. Do đó, trong Hình 17, Key = Value = h_i .



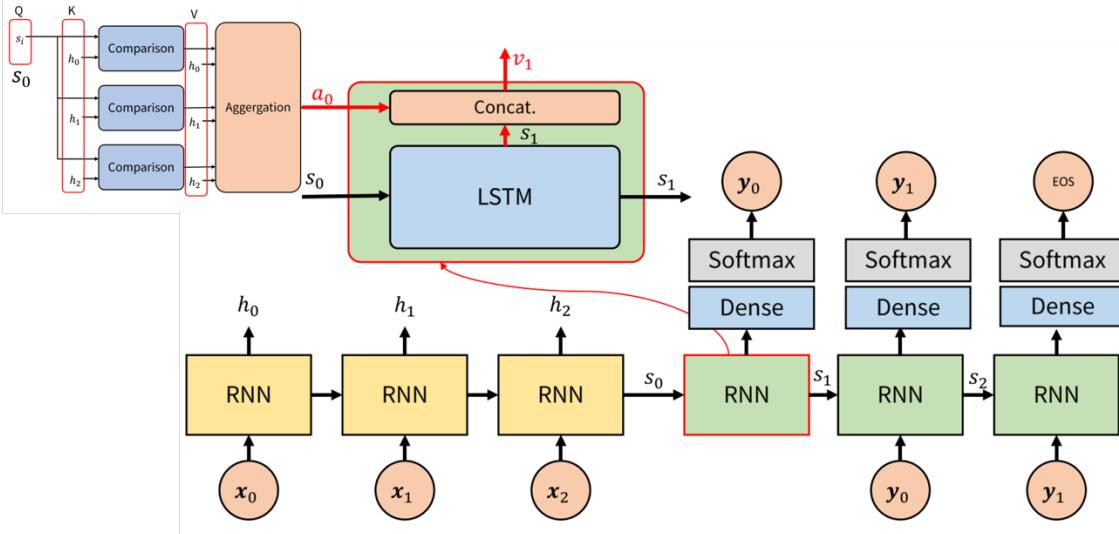
Hình 18. Query được sử dụng để tính toán giá trị Attention trong mô hình Seq2Seq [17] là các trạng thái ẩn của Decoder

Tương tự, Query trong Attention là trạng thái ẩn của Decoder hay $Query = s_i$. (Hình 18)



Hình 19. Mô tả quá trình tính toán giá trị Attention từ Q (trạng thái ẩn của Decoder), K và V (trạng thái ẩn của Encoder)

Các phép tính liên quan đến Attention tuân theo cơ chế như đã đề cập ở trên được mô tả như Hình 19. Tại Decoder, Query s_i được nhập vào và thông qua phép tính tính toán độ tương đồng giữa Query này với tất cả các giá trị Key h_i (trong Hình 19 ví dụ là h_0, h_1, h_2) để tạo ra Giá trị Attention. Giá trị tương đồng này được trải qua phép tính softmax để trở thành giá trị xác suất và tổng các giá trị này sẽ là 1.



Hình 20. Mô tả quá trình tính toán giá trị Attention trong mô hình Seq2Seq [17]

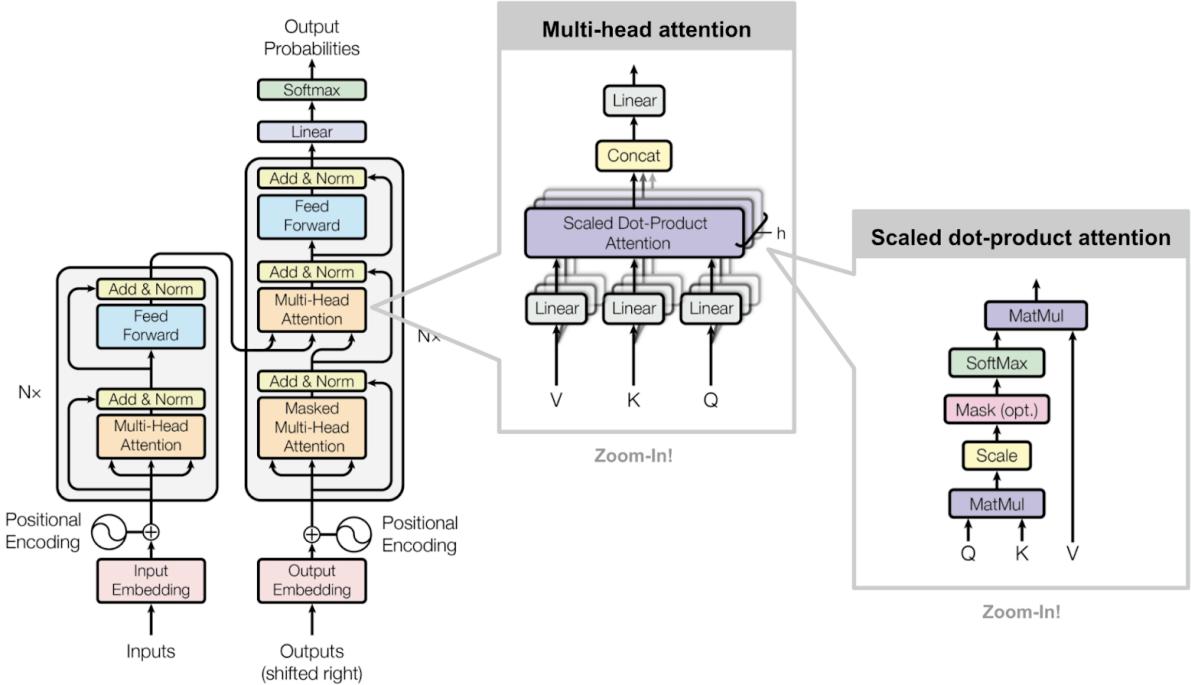
Trước tiên, trạng thái ẩn của Decoder được lấy từ RNN (LSTM) và tính toán để tạo ra $s_i \rightarrow s_{i+1}$. Sau đó, ta ghép Attention Value a_i và s_{i+1} bằng cách concat ($v_i = [s_i; a_{i-1}]$) để tạo ra v_{i+1} . Giá trị này được đưa qua một tầng kết nối đầy đủ (Fully Connected Layer) và softmax để tính toán và đưa ra kết quả cuối cùng là y_i . (Hình 20)

4. Mô hình Transformer [9]

Transformer là một mô hình học sâu được giới thiệu bởi Vaswani và các cộng sự [9] vào năm 2017, đây là mô hình đã đạt được kết quả nổi bật trong nhiều nhiệm vụ xử lý ngôn ngữ tự nhiên (NLP). Transformer được xây dựng dựa trên cơ chế Attention (Attention mechanism, tạm dịch: cơ chế tập trung), cho phép mô hình có khả năng “tập trung” vào các phần khác nhau của đầu vào (Input) trong quá trình học.

Mô hình Transformer [9] sử dụng kiến thức *đa đầu vào* (Multi-Head Input) và *đa đầu ra* (Multi-Head Output) để xử lý đầu vào và đầu ra theo cách đồng thời, tức là không cần xử lý tuần tự từng phần như các mô hình trước đây. Điều này giúp giải quyết vấn đề độ dài phụ thuộc trong ngôn ngữ (Long-range Dependency) và giúp mô hình có khả năng hiểu ngữ cảnh (Contextual Understanding) của dữ liệu đầu vào (Input).

Mô hình Transformer [9] đã được ứng dụng rộng rãi trong nhiều tác vụ xử lý ngôn ngữ tự nhiên, bao gồm dịch máy, phân loại văn bản, dự đoán từ, tóm tắt văn bản, hỏi đáp, và nhiều tác vụ ngôn ngữ tự nhiên khác. BERT (Bidirectional Encoder Representations from Transformers) [2], GPT (Generative Pre-trained Transformer), và T5 (Text-to-Text Transfer Transformer) là những mô hình NLP nổi tiếng được xây dựng trên kiến thức của mô hình Transformer.

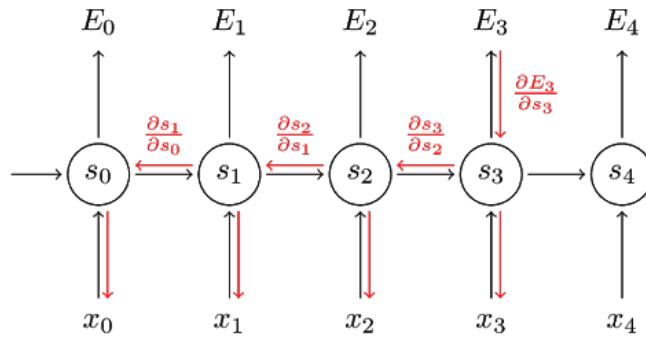


Hình 21. Kiến trúc mô hình Transformer [9]

Mô hình Transformer [9] có cấu trúc như trong Hình 21. Mô hình trên là một nghiên cứu nổi tiếng vì đạt được hiệu suất cao nhất trong bài toán dịch máy mà không sử dụng RNN hay CNN, chỉ sử dụng các phép tính cơ bản như tầng kết nối đầy đủ (Fully Connected Layer – FC Layer) và các khối Attention, và được biết đến là một nghiên cứu tiên phong trong lĩnh vực này.

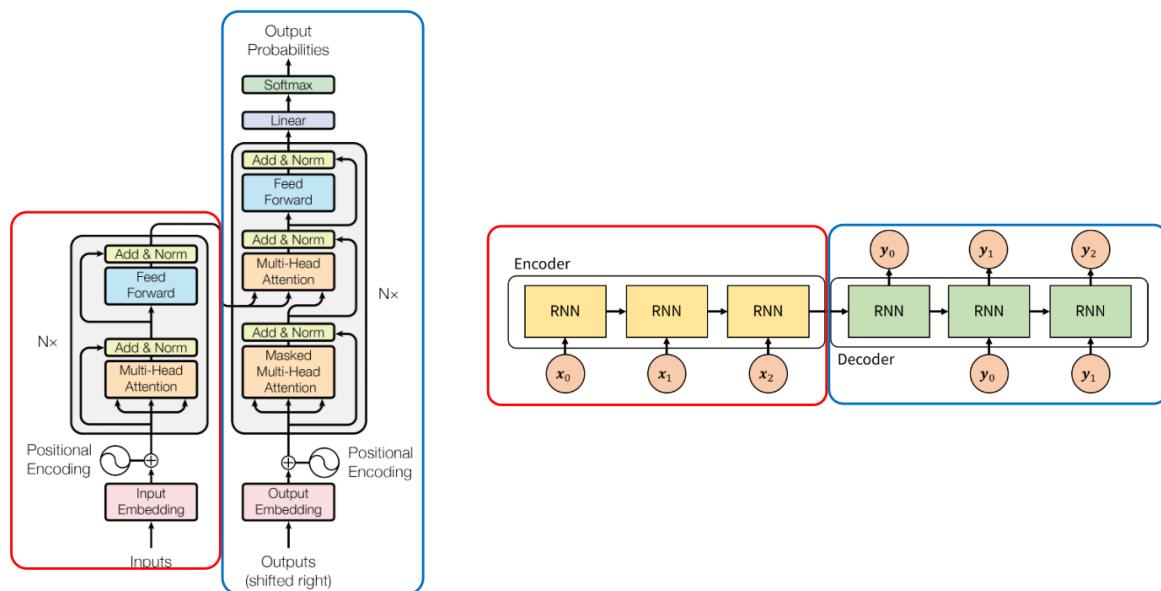
Trước tiên, chúng ta sẽ tìm hiểu tổng quan về kiến trúc của mô hình:

- Mô hình Transformer [9] sử dụng định dạng mã hóa - giải mã (Encoder - Decoder) tương tự như Seq2Seq [17].
- Những khối được đề xuất trong Transformer [9], bao gồm Scaled Dot-Product Attention và Multi-Head Attention, là nhân tố cốt lõi của thuật toán này, và chúng được xếp hàng loạt và thực hiện song song (parallel) trong mô hình.
- Trong kiến trúc của RNN, do có cấu trúc Back Propagation Through Time – BPTT, mô hình phải được triển khai dọc theo thời gian và yêu cầu các phép tính tuần tự. Do đó, vấn đề hiệu quả tính toán kém có thể xảy ra (Hình 22). Trong khi đó, trong kiến trúc của Transformer, không có cấu trúc BPTT tương tự và tính toán có thể được thực hiện song song, do đó mô hình có khả năng hoạt động hiệu quả hơn so với kiến trúc RNN.



Hình 22. Hình mô tả quá trình lan truyền ngược trong mô hình mạng RNN

- Trong mô hình Transformer [9], khác với RNN, tính toán có thể được thực hiện song song, do đó cần có cơ chế biểu diễn vị trí của từ đang được tính toán trong câu. Đó là lý do tại sao Encoder và Decoder trong Transformer [9] sử dụng mã hóa vị trí (Positional Encoding) để đảm bảo các thông tin về vị trí của các từ trong câu được đưa vào mô hình.



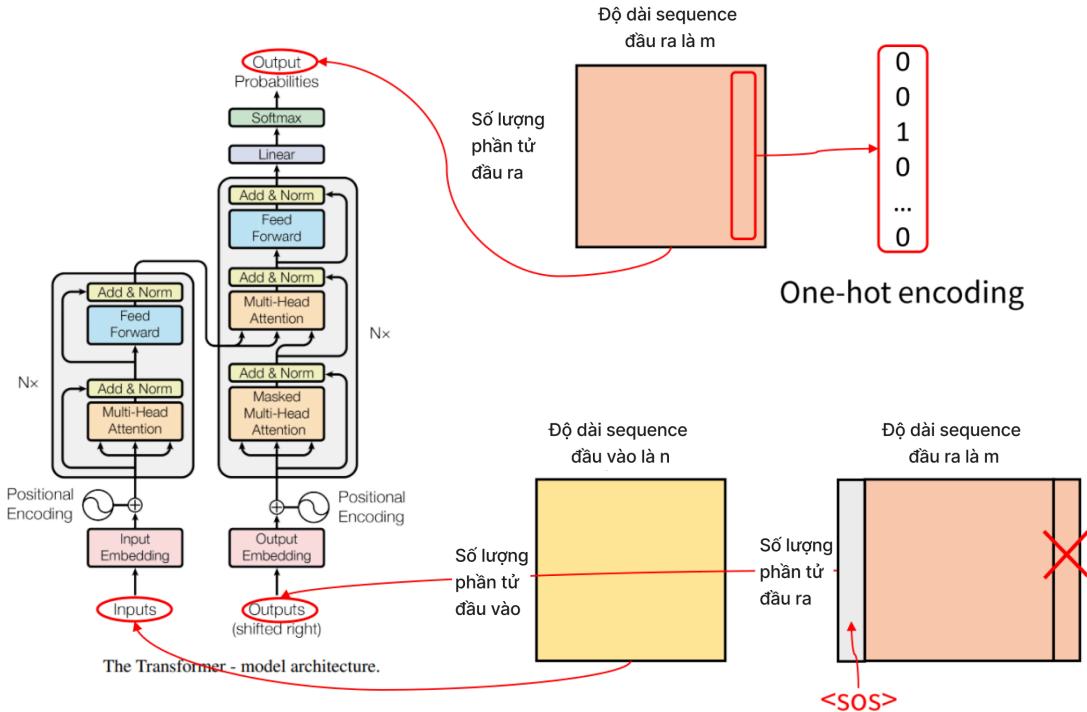
Hình 23. Mô hình Transformer [9] (trái) và mô hình Seq2Seq [17] (phải)

Nhìn chung điểm khác biệt giữa mô hình Transformer [9] và Seq2Seq [17] có thể tóm tắt như sau:

- Seq2Seq [17] có cấu trúc gồm Encoder (màu đỏ) và Decoder (màu xanh), trong đó mỗi Encoder và Decoder sử dụng RNN (Hình 23). Khi truyền thông tin từ Encoder sang Decoder, thông tin được lưu trữ trong context vector và được gửi sang Decoder.
- Đối với Transformer [9] mặc dù cũng có cấu trúc gồm Encoder (màu đỏ) và Decoder (màu xanh) và với mũi tên truyền từ cuối cùng của Encoder sang Decoder, có cấu trúc tương tự như Seq2Seq [17]. Điểm khác biệt lớn nhất là trong Seq2Seq [17],

toàn bộ đầu vào phải được xử lý bởi Encoder trước khi bắt đầu quá trình tính toán của Decoder. Trong khi đó, trong mô hình Transformer [9], tính toán của Encoder và Decoder xảy ra đồng thời.

4.1. Đầu vào (Input) và đầu ra (Output) của Transformer [9]



Hình 24. Mô tả đầu vào (Input) và đầu ra (Output) của mô hình Transformer [9]

Tiếp theo, chúng ta sẽ tìm hiểu về dạng dữ liệu đầu vào và đầu ra của mô hình Transformer [9].

Vì mô hình Transformer [9] được phát triển cho dịch máy tự nhiên ngôn ngữ, vì vậy chúng ta sẽ giải thích dựa trên vector từ đầu vào và đầu ra.

Nếu nhìn tổng thể hình thức đầu vào và đầu ra, nó trông giống như một ma trận. Vậy cột và hàng trong ma trận đó đại diện cho cái gì? Chúng ta sẽ đi tìm hiểu.

Đầu tiên, chúng ta sẽ xem xét về đầu vào (Input) của mô hình Transformer [9]:

- Mỗi từ được biểu diễn dưới dạng One-Hot Encoding (mục 2.1). Vì vậy, mỗi cột vector tương ứng với một từ và chiều dài của vector, tức là số hàng trong ma trận, phụ thuộc vào số lượng từ khác nhau trong đầu vào.
- Trong khi đó, số lượng cột tương ứng với độ dài của chuỗi từ (Sequence) được sử dụng. Nếu câu có 10 từ, thì kích thước của cột trong ma trận sẽ là 10.

Tiếp theo, chúng ta sẽ xem xét đầu ra (Output):

- Vấn đề đầu tiên được xử lý trong Transformer [9] là bài toán dịch thuật. Do đó, dạng đầu ra cũng là các từ, tuy nhiên kích thước của ma trận đầu vào và đầu ra có thể khác nhau.
- Ví dụ, nếu đầu vào là tiếng Anh và đầu ra là tiếng Hàn, số lượng từ sử dụng cũng sẽ khác nhau, và thậm chí những câu có ý nghĩa tương đương cũng có thể sử dụng số lượng từ khác nhau.
- Vì vậy, kích thước của đầu vào và đầu ra có thể khác nhau, nhưng thành phần cấu thành lại giống nhau. Kích thước của mỗi cột vector trong đầu ra, tức là số hàng trong ma trận, phụ thuộc vào số lượng từ khác nhau trong đầu ra và chiều dài của cột trong ma trận đại diện cho độ dài chuỗi từ được sử dụng trong đầu ra.

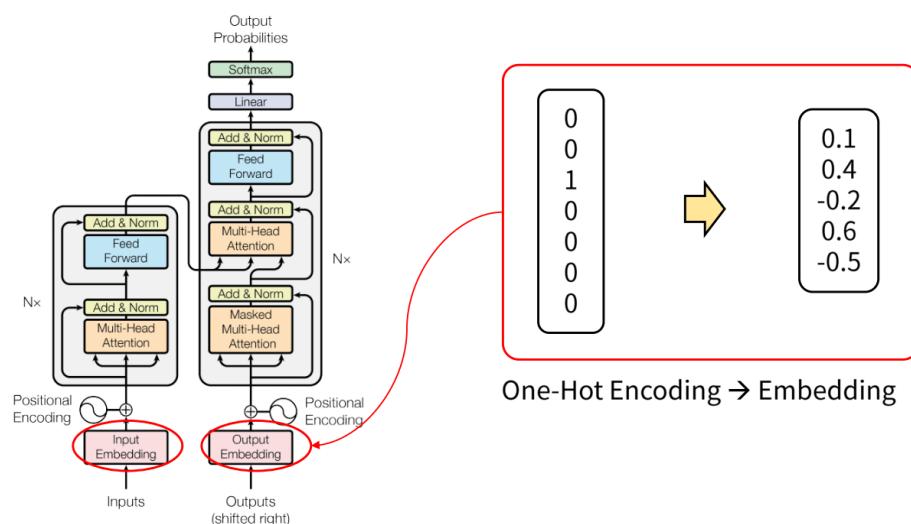
Trong khi đối với đầu vào (Input) thì chỉ có một loại, nhưng khi xem xét đầu ra (Output) thì có 2 loại đầu ra được hiển thị. Đầu ra ở phía trên trong Hình 24 là đầu ra thực tế được tạo ra thông qua mô hình Transformer [9], trong khi đầu ra ở phía dưới đại diện cho đầu ra được tạo ra bởi mô hình Transformer [9] và được sử dụng lại như đầu vào tiếp theo.

Vector cột đầu tiên của đầu ra được sử dụng lại là $\langle \text{SOS} \rangle$ (Start of Sequence – bắt đầu chuỗi) và vector cột cuối cùng được đánh dấu X là $\langle \text{EOS} \rangle$ (End of Sequence – kết thúc chuỗi).

Vì vậy, phần được ghi chú *shifted right* có thể được hiểu là một cấu trúc trong đó mỗi cột của đầu ra được *dịch sang phải* một cột và được sử dụng lại như đầu vào tiếp theo.

4.2. Embedding và Position Encoding

4.2.1. Embedding



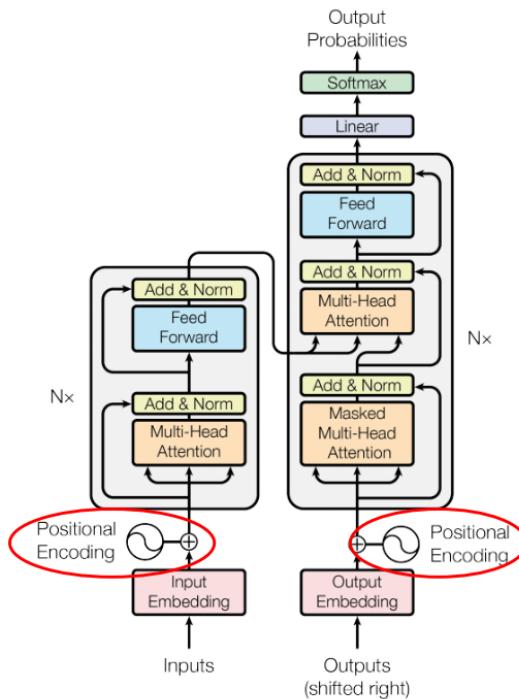
Hình 25. Tầng Input Embedding và Output Embedding trong mô hình Transformer [9]

Đầu tiên, chúng ta có thể giảm số chiều bằng cách chuyển đổi vector mã hóa one-hot được sử dụng cho đầu vào sang dạng số thực. Giống như Hình 25, chúng ta có thể giảm số chiều bằng cách chuyển đổi vector mã hóa one-hot sang dạng số thực.

Trong Transformer [9], Input Embedding và Output Embedding đều là tầng Embedding để chuyển đổi đầu vào thành vector với số thực có chiều là d .

4.2.2. Positional Encoding

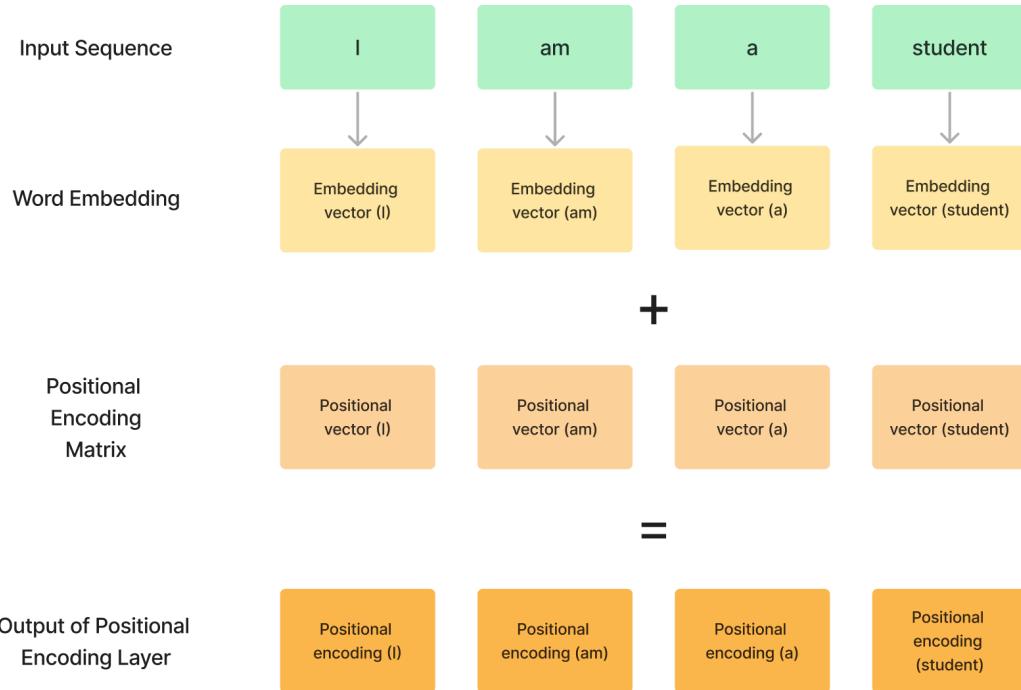
Trước Transformer [9], các mô hình như RNN hay LSTM đã được sử dụng để xử lý dữ liệu đầu vào dạng chuỗi (ví dụ: câu), nhưng do trong các mô hình này dữ liệu được đưa vào một cách tuần tự (có thứ tự) nên dẫn đến tốc độ tính toán chậm.



Hình 26. Positional Encoding trong mô hình Transformer [9]

Đối với Transformer [9], một trong những ưu điểm của mô hình này là có khả năng xử lý đầu vào một cách song song, tức không phải xử lý dữ liệu một cách tuần tự như RNN hay LSTM. Tuy nhiên, việc xử lý dữ liệu một cách song song lại dẫn đến thông tin về vị trí bị mất đi \Rightarrow Do đó, để giải quyết việc bị mất thông tin về vị trí, Transformer [9] đã sử dụng mã hoá vị trí (Positional Encoding).

Vị trí của Positional Encoding trong mô hình Transformer [9] được mô tả như trong Hình 26.



Hình 27. Quá trình thêm thông tin thứ tự vào dữ liệu đầu vào

Dữ liệu đầu vào là câu (Input Sequence) khi được đưa vào Transformer [9], các từ trong câu sẽ được chuyển đổi thành vector số thực thông qua tầng Embedding. Trong quá trình chuyển đổi, đầu vào là các vector one-hot và được biến đổi thành vector số thực có số chiều thấp hơn.

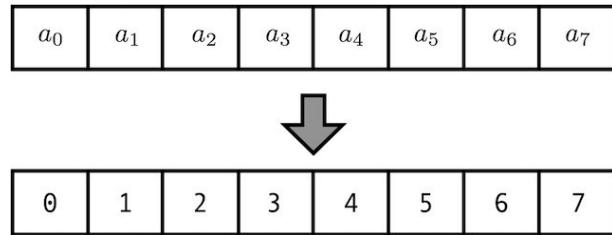
Tuy nhiên, một số thông tin về vị trí của từ trong câu sẽ bị mất đi trong quá trình chuyển đổi. Để giải quyết vấn đề này, Transformer [9] sử dụng mã hoá vị trí (Positional Encoding) để mã hóa thông tin về vị trí của từ trong câu.

Cụ thể, Positional Encoding tạo ra một vector số thực có kích thước bằng với vector số thực ứng với từ đã được chuyển đổi bởi tầng Embedding và giá trị của vector này thể hiện vị trí của từ trong câu, vector thể hiện vị trí sẽ được cộng vào vector số thực của từ để tạo ra một vector mới mang cả thông tin về từ và vị trí của từ.

Quá trình trên được minh họa trên Hình 27.

Tiếp theo, chúng ta sẽ tìm hiểu quá trình mà Positional Encoding tạo ra các vector số thực tương ứng với vị trí của từ trong câu.

Trong Transformer [9], vị trí của từ trong câu được gán tăng dần tương ứng, chẳng hạn, đầu vào là câu gồm 8 từ $a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$, thì vị trí của từ tương ứng sẽ là 0, 1, 2, 3, 4, 5, 6, 7. (Hình 28)



Hình 28. Thể hiện vị trí của từ trong câu

Với mô hình Transformer [9], các vector số thực mang thông tin về vị trí được tạo ra dựa trên công thức được trình bày như Hình 29.

Theo đó, công thức sẽ được chia thành hai trường hợp theo chỉ số của vector mang thông tin về vị trí: chỉ số chẵn ($2i$) và chỉ số lẻ ($2i+1$).

Chỉ số chẵn tương ứng với hàm **sin**, ngược lại chỉ số lẻ ứng với hàm **cos**. Số 10000 là số được đề xuất bởi Transformer [9].

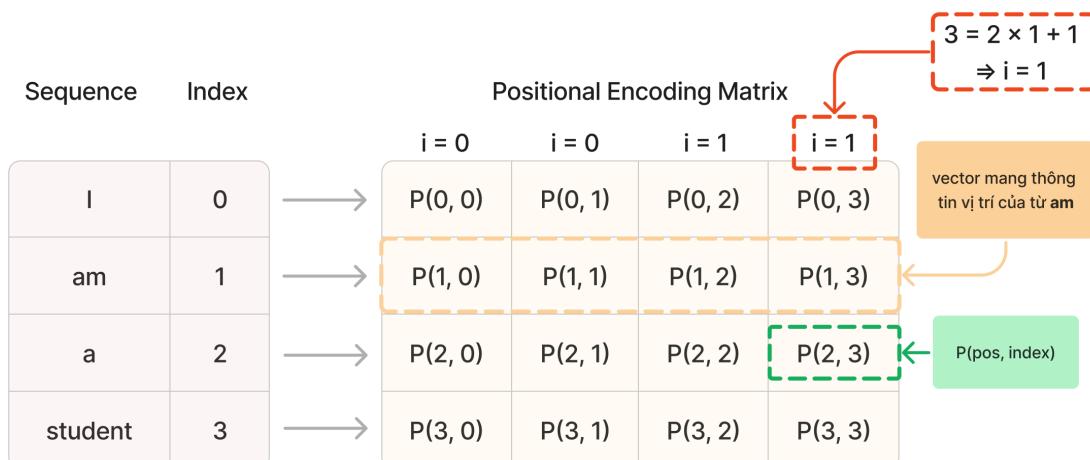
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$0 \leq i \leq d_{model}/2$$

Hình 29. Công thức tính $PE(pos, 2i)$ và $PE(pos, 2i + 1)$

Để rõ hơn, chúng ta sẽ xét Positional Encoding Matrix với $d_{model} = 3$ của câu “I am a student” (Hình 30).



Hình 30. Ma trận mã hoá vị trí của câu “I am a student”

Với mỗi phần từ trong ma trận mã hoá vị trí (Positional Encoding Matrix) ta sẽ dựa theo công thức ở Hình 29 để tính toán dựa theo cặp (pos, index) của từng ô trong ma trận.

Trong đó, pos là vị trí của từ trong câu (Hình 28), index là chỉ số của vector mang thông tin vị trí.

Chẳng hạn, xét $P(2, 3)$, ta có pos = 2 và index = 3 do index = 3 là số lẻ ($3 = 2 \times 1 + 1$) nên $P(2, 3) = PE(2, 1) = \cos(2/10)$ với $i = 1$.

Sau khi tính toán toàn bộ các ô trong ma trận mã hoá vị trí, ta sẽ được các vector số thực tương ứng với vị trí của các từ trong câu.

Kết quả ví dụ sau khi tính toán các ô trong ma trận mã hoá vị trí ở Hình 30, được thể hiện như sau (Hình 31):

Positional Encoding Matrix

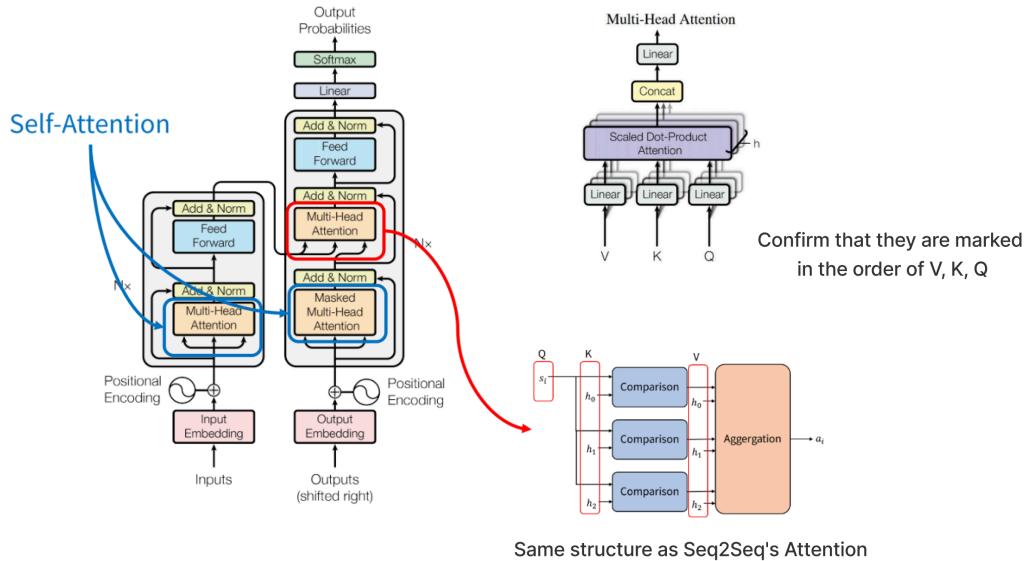
	$i = 0$	$i = 0$	$i = 1$	$i = 1$
I	$P(0, 0) = \sin(0) = 0$	$P(0, 1) = \cos(0) = 1$	$P(0, 2) = \sin(0) = 0$	$P(0, 3) = \cos(0) = 1$
am	$P(1, 0) = \sin(1/1) = 0.84$	$P(1, 1) = \cos(1/1) = 0.54$	$P(1, 2) = \sin(1/10) = 0.10$	$P(1, 3) = \cos(1/10) = 0.99$
a	$P(2, 0) = \sin(2/1) = 0.91$	$P(2, 1) = \cos(2/1) = -0.42$	$P(2, 2) = \sin(2/10) = 0.2$	$P(2, 3) = \cos(2/10) = 0.98$
student	$P(3, 0) = \sin(3/1) = 0.14$	$P(3, 1) = \cos(3/1) = -0.99$	$P(3, 2) = \sin(3/10) = 0.3$	$P(3, 3) = \cos(3/10) = 0.96$

Hình 31. Giá trị trong ma trận mã hoá vị trí của câu “I am a student”

Cuối cùng, cộng vector số thực thể hiện vị trí của từ tương ứng với vector số thực của từ được tính toán bởi tầng Embedding, ta được kết quả đầu ra khi đi qua Positional Encoding.

4.3. Chi tiết các thành phần có trong Transformer [9]

4.3.1. Multi-Head Attention



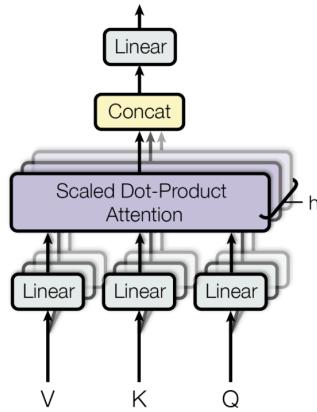
Hình 32. Multi-Head Attention trong Transformer

Hình 32, mô tả các Multi-Head Attention trong kiến trúc mô hình Transformer [9].

Các phần được đánh dấu bởi màu xanh dương là Self-Attention – đồng nghĩa với cả ba vector Value (V), Key (K), và Query (Q) là như nhau (bằng nhau). Multi-Head Attention trong Encoder sẽ không bao gồm Mask, còn Multi-Head Attention trong Decoder lại sử dụng Mask → Masked Multi-Head Attention, lý do việc Decoder sử dụng Mask là bởi K và V không thể xuất hiện trước Q.

Multi-Head Attention được đánh dấu bởi màu đỏ trong Decoder sẽ sử dụng các hidden state từ Encoder làm giá trị cho vector K và V, còn lại giá trị của vector Q được nhận từ Decoder. Có thể thấy, cấu trúc ở phần này tương tự với Seq2Seq's Attention.

Tiếp theo, chúng ta sẽ đi sâu vào tìm hiểu bên trong Multi-Head Attention.



Hình 33. Multi-Head Attention trong Transformer [9]

Hình 33 mô tả cấu trúc của Multi-Head Attention. Đầu vào của Multi-Head Attention bao gồm ba vector V, K và Q tương tự như các đầu vào được sử dụng trong các Attention truyền thống.

Tuy nhiên, trong quá trình tính toán, có sự khác biệt giữa Multi-Head Attention với các Attention khác, thay vì tính toán duy nhất mô lần trên các vector V, K, Q có kích thước d_{model} , Multi-Head Attention chia nhỏ đầu vào là các vector V, K, Q thành h vector có kích thước nhỏ hơn, lần lượt là d_v, d_k, d_k để làm đầu vào cho h -Scaled Dot-Product Attention (mục 4.3.2).

Trong đó,

(1) h : số lượng tầng Attention song song (còn được gọi là heads) là 8;

$$(2) d_k = d_v = \frac{d_{model}}{h} = 64.$$

Với $d_{model} = 512$. (kích thước của hidden_state)

Các thông số ở (1) và (2) là số liệu của tác giả trong bài báo gốc [9].

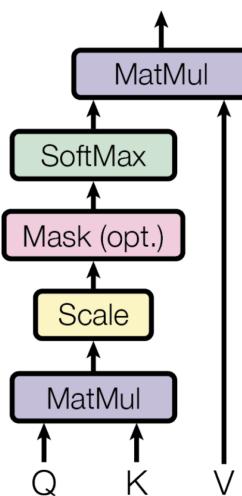
Quá trình tính toán h -Scaled Dot-Product Attention sẽ được diễn ra đồng thời (song song). Sau đó, kết quả của h -Scaled Dot-Product Attention sẽ được Concat lại trước khi đi qua tầng Linear để có được kết quả cuối cùng là đầu ra của tầng Multi-Head Attention.

4.3.2. Scaled Dot-Product Attention

Yếu tố cốt lõi của Transformer [9] chính là Scaled Dot-Product Attention.

Các đầu vào được sử dụng trong Attention (mục 3.2) là Query (Q), Key (K), và Value (V). Do đó, cấu trúc của Scaled Dot-Product Attention (Hình 34) cũng sử dụng cùng cấu trúc này: Q (Query), K (Key), V (Value).

Lưu ý, như đã trình bày ở mục 4.3.1, thì các vector Q, K, V ở đầu vào của Scaled Dot-Product Attention có kích thước là lượt là d_k, d_k, d_v , không phải d_{model} .



Hình 34. Scaled Dot-Product Attention trong Transformer [9]

Hàm so sánh giữa Q và K được thực hiện bằng phép nhân chấm (Dot-Product) và được điều chỉnh (Scale) bằng cách chia cho $\sqrt{d_k}$.

Phép nhân chấm (Dot-Product) tương đương với tích vô hướng (Inner-product) và được ký hiệu là nhận ma trận (MatMul).

Sau đó, Mask (nếu có) được sử dụng để ngăn chặn Attention đến các kết nối không hợp lệ (illegal connection). Các kết nối không hợp lệ liên quan đến khái niệm Self-Attentionⁱ.

$$\text{softmax} \left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}$$

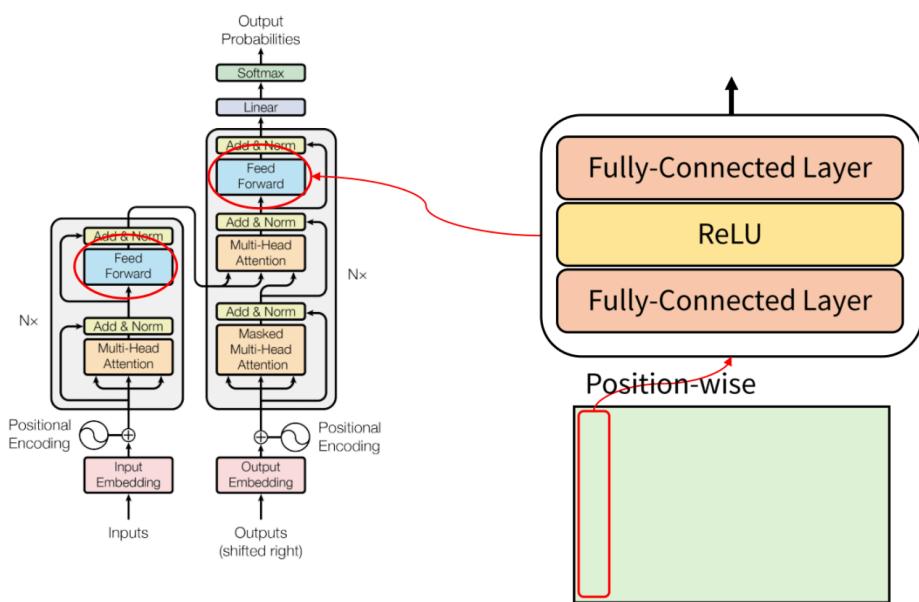
$$= \mathbf{z}$$

Hình 35. Minh họa quá trình tính toán giá trị Attention z từ bộ ba Q , K và V .

Cuối cùng, chúng ta kết hợp đầu ra của Softmax, tức là giá trị độ tương đồng, và V để tính toán giá trị Attention như Hình 35.

Đầu ra của Scaled Dot-Product Attention là một vector có kích thước là d_v .

4.3.3. Position-wise Feed-Forward

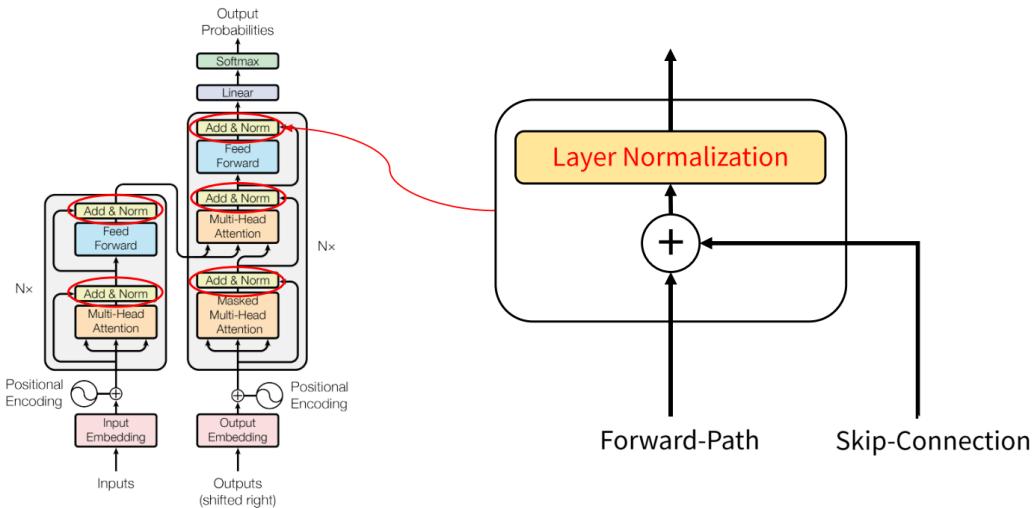


Hình 36. Feed Forward trong Transformer [9]

“Position-wise Feed-Forward” có nghĩa là Feed-Forward dựa trên vị trí của từng từ. Do đó, Position-wise Feed-Forward sẽ thực hiện trên các cột vector đại diện cho từng từ đầu vào. (Hình 36)

Cụ thể, các vector đầu vào (là các vector đại diện cho từ) sẽ được truyền qua tầng FC – Layer (Fully Connected Layer) với hàm kích hoạt ReLU, và cuối cùng đi qua một FC – Layer nữa. Đầu ra của tầng FC – Layer thứ hai cũng chính ra đầu ra của Position-wise Feed-Forward.

4.3.4. Add & Norm



Hình 37. Add & Norm trong Transformer [9]

Trong Transformer [9], Add & Norm (Hình 37) được sử dụng để kết hợp thông tin từ các tầng khác nhau trong mạng. Trong quá trình này, đầu ra của một tầng sẽ được cộng với đầu vào ban đầu của tầng đó (skip-connection), sau đó chuẩn hóa lại với Layer Normalization [18] để tạo ra đầu ra cuối cùng.

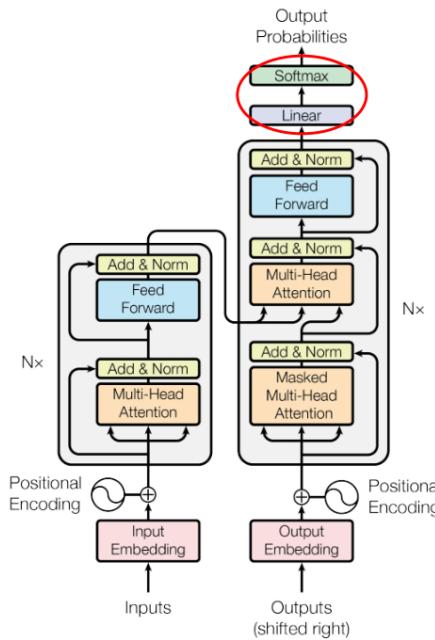
Cụ thể,

(1) **Bước cộng (Add):** Trong bước này, đầu ra của lớp Attention được cộng với vector đầu vào ban đầu. Việc cộng này giúp cập nhật thông tin từ các phần khác nhau của kiến trúc và giúp tránh hiện tượng mất thông tin (Vanishing Gradient) trong quá trình huấn luyện.

(2) **Bước chuẩn hóa (Norm):** Sau khi thực hiện bước cộng, giá trị đầu ra được chuẩn hóa bằng cách tính toán giá trị trung bình và độ lệch chuẩn của các giá trị đầu ra. Quá trình chuẩn hóa giúp cải thiện tính ổn định của mô hình và giảm hiện tượng biến dạng (covariate shift⁷) trong quá trình huấn luyện.

⁷ Covariate shift xảy ra khi phân phối các biến (distribution of variables) trong dữ liệu huấn luyện khác với dữ liệu thực tế hoặc dữ liệu kiểm tra.

4.3.5. Linear và Softmax

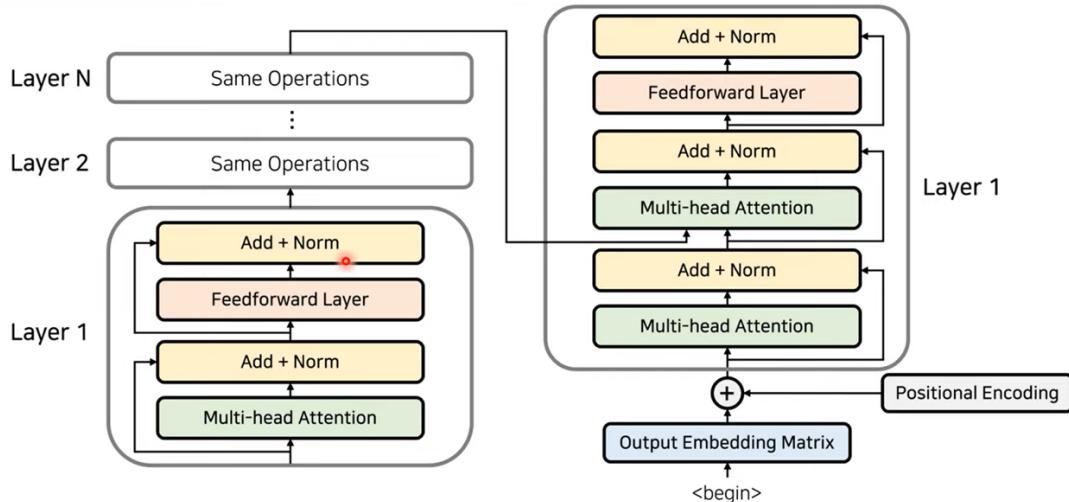


Hình 38. Linear và Softmax trong Transformer [9]

Dữ liệu đưa qua tầng tuyến tính (Linear) cuối cùng nhằm điều chỉnh kích thước đầu ra phù hợp với số lượng từ có trong từ điển (vocab_size).

Sau khi qua tầng Linear, đầu ra được đưa vào Softmax để tính toán xác suất của các lớp đầu ra (các từ trong từ điển). Softmax đưa ra các xác suất đối với mỗi lớp đầu ra và chọn ra lớp có xác suất cao nhất làm đầu ra của mô hình. (Hình 38)

4.4. Encoder và Decoder trong Transformer



Hình 39. Mô tả cấu trúc Encoder - Decoder trong Transformer

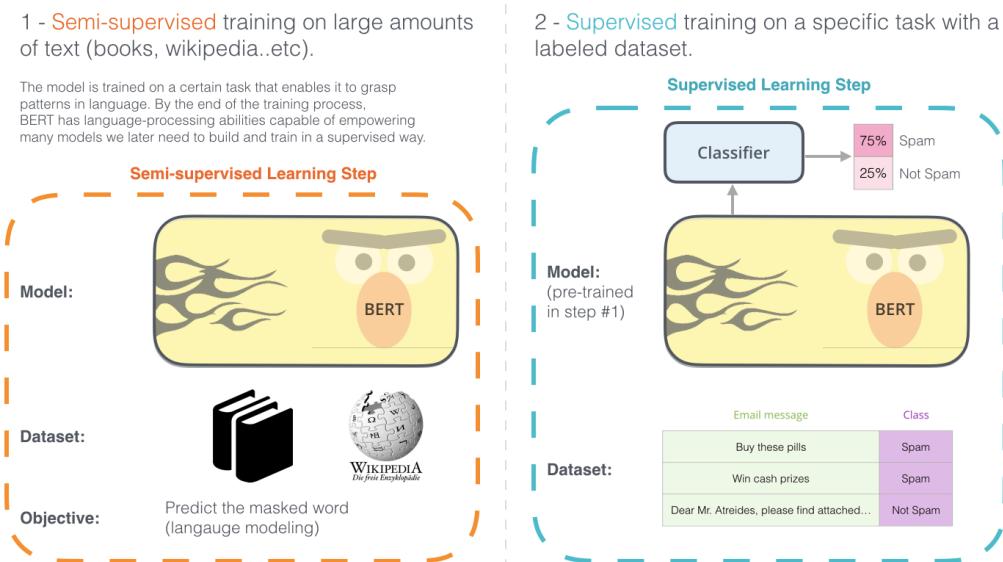
Cấu trúc Encoder – Decoder được mô tả như Hình 39.

Đầu vào và đầu ra của Encoder có cùng kích thước. Do đó, cấu trúc Encoder có thể được lặp lại nhiều lần để sử dụng một cách dễ dàng.

Như Hình 39 ở trên, Encoder có thể được lặp lại nhiều lần như một khối để xử lý đầu vào và đưa ra đầu ra có cùng kích thước. Tương tự, Decoder cũng có thể được lặp lại nhiều lần dưới dạng khối để giải mã đầu ra.

5. Mô hình BERT [2]

BERT (Bidirectional Encoder Representations from Transformers) [2] là một mô hình ngôn ngữ tiên tiến trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP) được phát triển bởi Google vào năm 2018. BERT [2] được xây dựng trên kiến trúc Transformer [9], một mô hình học sâu dựa trên cơ chế Attention (mục 3.2).



Hình 40. (1) Pre-training tasks và (2) Sentence classification

BERT [2] được huấn luyện trên một lượng lớn dữ liệu văn bản từ các nguồn khác nhau, bao gồm các tài liệu trên web, các trang báo, tạp chí, sách và các bài báo khoa học. Quá trình huấn luyện của BERT [2] được thực hiện trên một tập hợp các nhiệm vụ phụ (pre-training tasks) trước khi được fine-tuning trên các tác vụ cụ thể như phân loại văn bản, trích xuất thông tin, hoặc dịch máy (Hình 40). Với việc sử dụng các pre-training tasks, BERT [2] có khả năng hiểu được ngôn ngữ tự nhiên một cách toàn diện hơn so với các mô hình truyền thống.

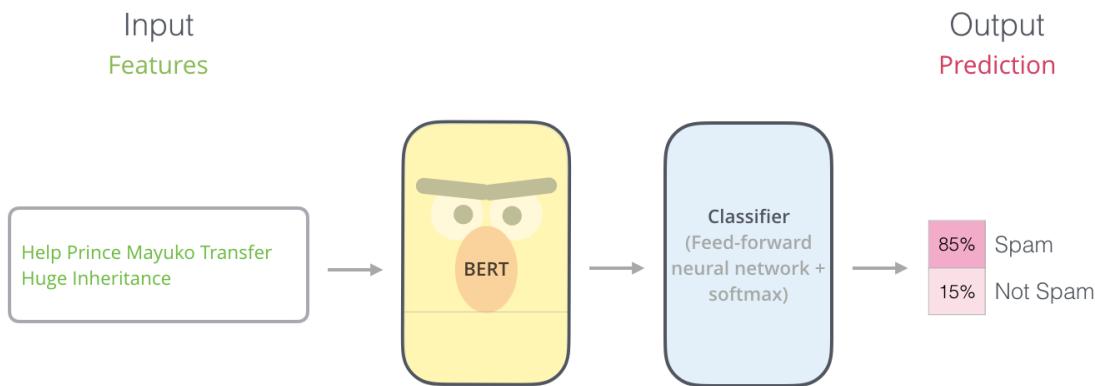
Một trong những điểm nổi bật của BERT [2] là khả năng xử lý ngôn ngữ tự nhiên hai chiều (bidirectional), có nghĩa là nó có thể hiểu được ngữ cảnh của từ cần xử lý không chỉ dựa trên các từ trước đó mà còn dựa trên các từ sau đó. Với tính hai chiều này, BERT [2] được huấn luyện hai tác vụ NLP khác nhau nhưng có liên quan mật thiết với nhau, đó là Masked Language Model (MLM) và Next Sequence Prediction (NSP). Mục đích của

huấn luyện MLM là ẩn đi một từ trong câu và để thuật toán dự đoán xem từ nào đã được ẩn dựa trên ngữ cảnh. Còn mục đích của NSP nhằm dự đoán hai câu đã cho có kết nối logic với nhau không, tuân tự không hay chỉ là ngẫu nhiên. Điều này giúp BERT [2] đạt được kết quả rất tốt trên nhiều tác vụ trong NLP.

Theo đó, BERT [2] được coi là một trong những mô hình NLP tiên tiến nhất hiện nay và đã đạt được nhiều thành công trong việc giải quyết các vấn đề liên quan đến xử lý ngôn ngữ tự nhiên như phân loại văn bản, trả lời câu hỏi và dịch máy.

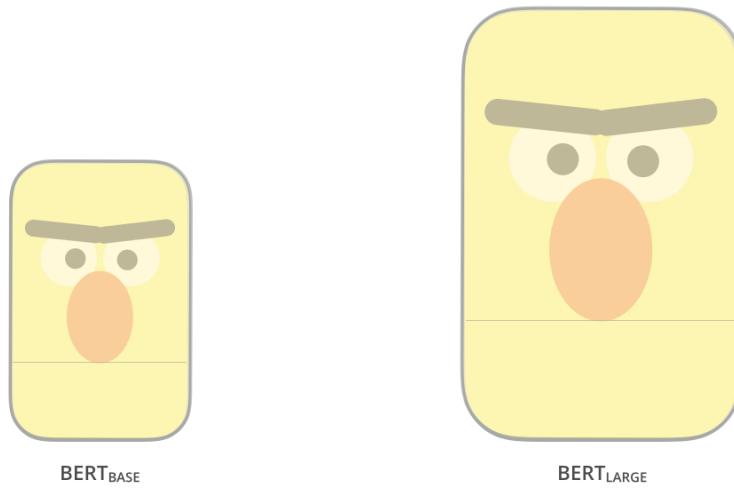
Ví dụ: Sử dụng BERT để phân loại văn bản

Cách đơn giản nhất để sử dụng BERT [2] là sử dụng để phân loại một đoạn văn bản duy nhất. Mô hình sẽ có dạng như Hình 41.



Hình 41. Mô hình BERT cho bài toán phân loại văn bản

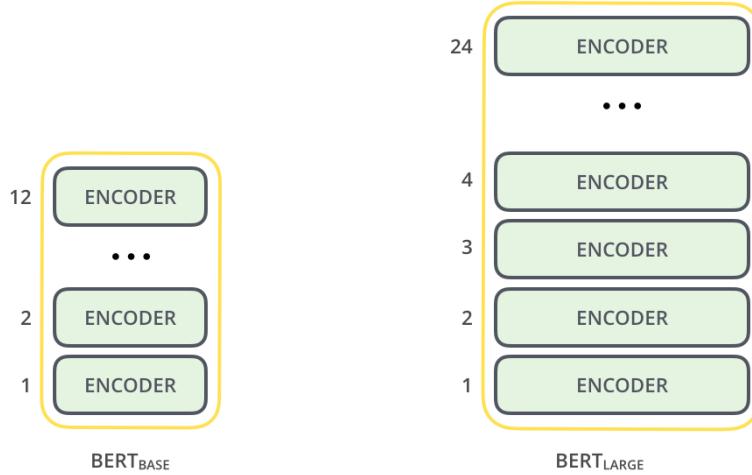
5.1. Kiến trúc tổng quát của mô hình BERT [2]



Hình 42. Các phiên bản của BERT [2]

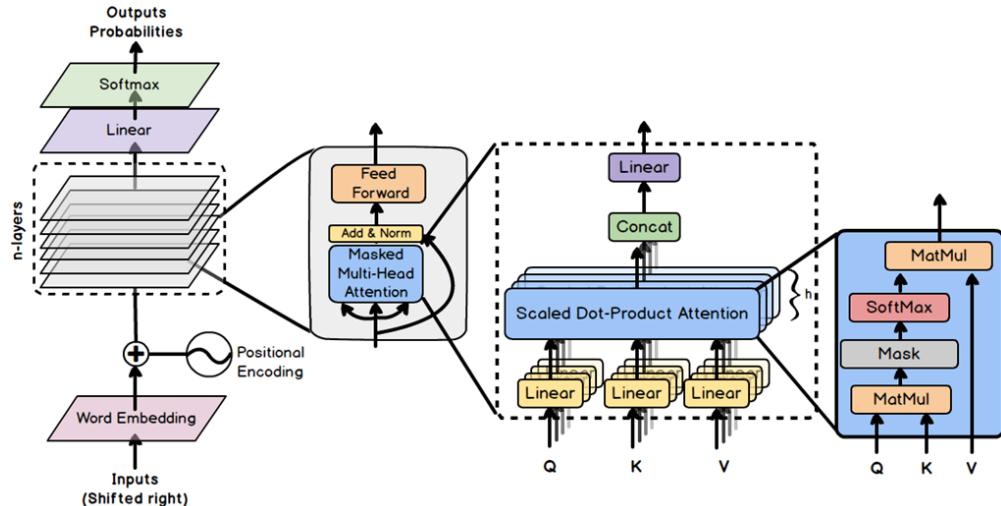
Trong bài báo gốc của BERT [2], tác giả đã giới thiệu hai phiên bản, gồm: BERT_{BASE}, BERT_{LARGE}.

Về cơ bản, BERT [2] là một mô hình xử lý ngôn ngữ tự nhiên được huấn luyện dựa trên Encoder của Transformer [9].



Hình 43. Số lượng Encoder L trong mô hình BERT [2]

Cả hai loại của mô hình BERT [2] đều có một số lượng lớn các Encoder (tác giả gọi là các khối Transformer) – $L = 12$ cho phiên bản $BERT_{BASE}$ và $L = 24$ cho phiên bản $BERT_{LARGE}$. (Hình 43)

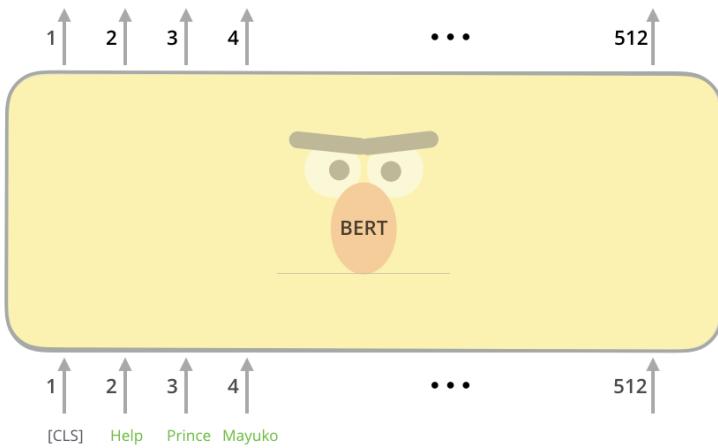


Hình 44. Mô hình tổng quát của BERT

Các thông số chính trong một mô hình BERT [2]: L là số tầng Transformers (blocks) được sử dụng với kích thước của các tầng ẩn là H và số lượng Scaled Dot-Product Attention ở tầng Multi-Head Attention là A .

- $BERT_{BASE}$: $L = 12$, $H = 768$, $A = 12$. Tổng tham số: 110M.
- $BERT_{LARGE}$: $L = 24$, $H = 1024$, $A = 16$. Tổng tham số: 340M.

5.2. Biểu diễn đầu vào và đầu ra của BERT [2]

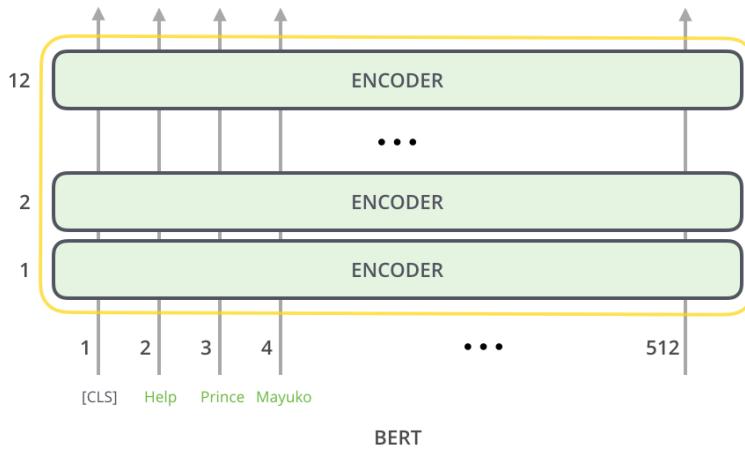


Hình 45. Mô phỏng đầu vào của BERT [2]

Đầu vào (Input)

Token đầu tiên ở đầu vào luôn là [CLS] token. CLS là viết tắt của Classification. Tác dụng của [CLS] token sẽ được trình bày ở phần Đầu ra (Output).

Nhìn chung, cũng giống như Encoder trong mô hình Transformer gốc [9], BERT [2] nhận đầu vào là một chuỗi các từ, có thể bao gồm một hoặc nhiều câu. Các câu sẽ được phân biệt bởi [SEP] token, và sử dụng [PAD] token (nếu cần thiết) để đưa toàn bộ các chuỗi về cùng một kích thước được cấu hình trước.



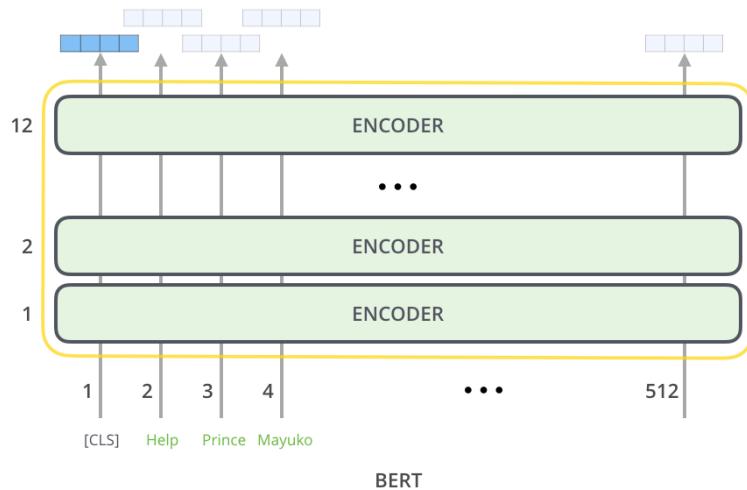
Hình 46. Mô phỏng đầu vào của BERT [2] với các khối Encoder

Đầu vào sẽ đi qua từng Encoder một, tức đầu ra của Encoder thứ l sẽ là đầu vào cho Encoder thứ $l + 1$. (với $l = 1..L - 1$).

Đầu ra của Encoder thứ L cũng chính ra đầu ra (Output) của mô hình BERT [2].

Đầu ra (Output)

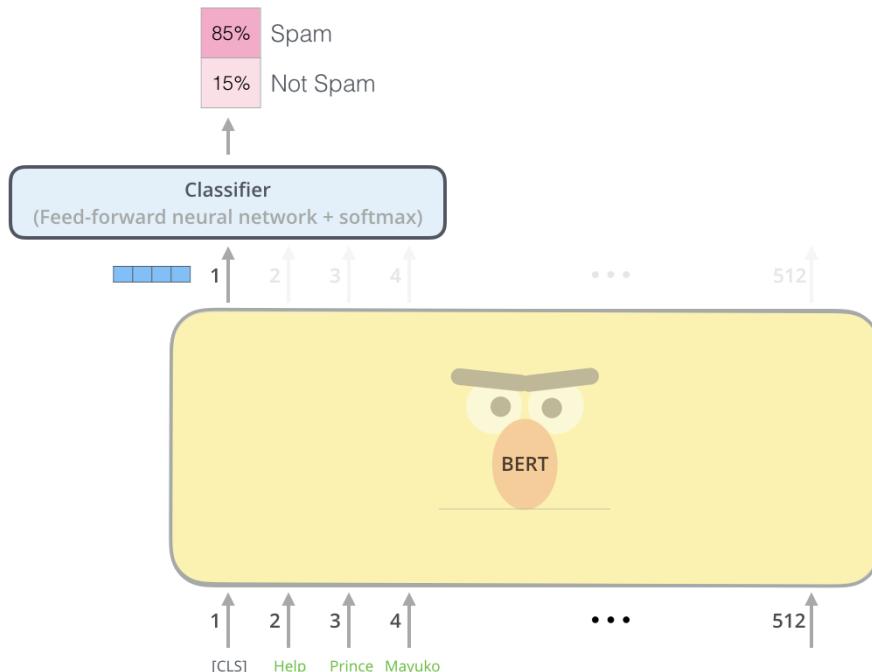
Với mỗi vị trí tương ứng với token ở đầu vào, thì ở đầu ra sẽ cho ra một vector có kích thước `hidden_size H`, với mô hình BERT_{BASE} $H = 768$.



Hình 47. Đầu ra của mô hình BERT [2]

Đối với tác vụ phân lớp văn bản, chúng ta sẽ chỉ tập trung vào vector ở vị trí đầu tiên của đầu ra, tương ứng với vị trí ở đầu vào là [CLS] token (vector màu xanh đậm ở Hình 47), do vector này đại diện cho đặc trưng của dữ liệu đầu vào.

Do đó, để phân lớp văn bản, chúng ta sẽ xây dựng một bộ Classifier mà theo bài báo gốc, bộ Classifier này chỉ là Single-Layer Perceptron (SLP), đã đạt được kết quả cao.



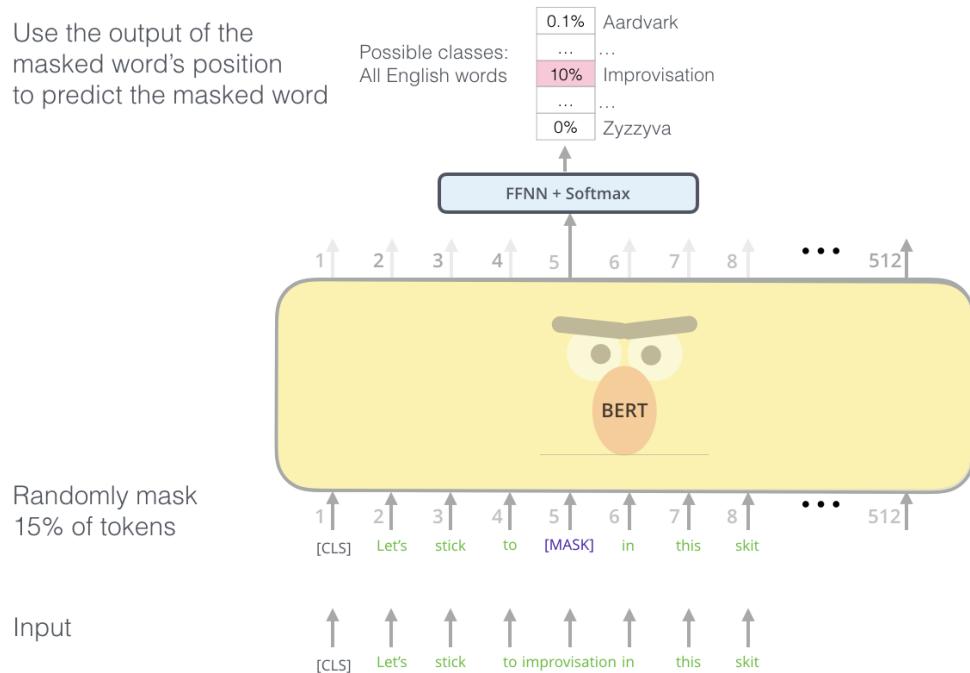
Hình 48. Sử dụng SLP để phân lớp từ đầu ra của BERT [2]

Hình 48 trình bày việc sử dụng vector đại diện cho [CLS] token ở đầu ra để phân lớp một bài toán có hai nhãn “Spam”, “Not Spam” sử dụng SLP với hàm kích hoạt là Softmax. Tương tự, nếu chúng ta có nhiều hơn hai nhãn, ví dụ, đối với bài toán phân loại email có thể có các nhãn “Spam”, “Not Spam”, “Social” và “Promotion”, thì chúng ta chỉ cần tùy chỉnh số lượng neuron ở đầu ra của SLP.

5.3. Masked Language Model – MLM

Masked Language Model (MLM) là một tác vụ (task) được sử dụng trong quá trình tiền huấn luyện (pre-training) của BERT [2]. Với MLM, một phần tử ngẫu nhiên trong câu được chọn và sau đó được che đi. Mô hình sẽ nhận diện từ xung quanh từ bị che đi và cố gắng dự đoán từ bị che đó.

Điều đặc biệt về cách sử dụng MLM trong BERT [2] là tất cả các từ trong câu đều có thể được che. Trong khoảng 15% trường hợp, từ được chọn để che sẽ được thay thế bằng [MASK] để đảm bảo rằng mô hình sẽ không chỉ nhớ từ đó, mà phải học cách dự đoán từ đó bằng cách sử dụng các từ khác trong câu.



Hình 49. MLM che giấu 15% số từ trong đầu vào và yêu cầu mô hình dự đoán từ còn thiếu.

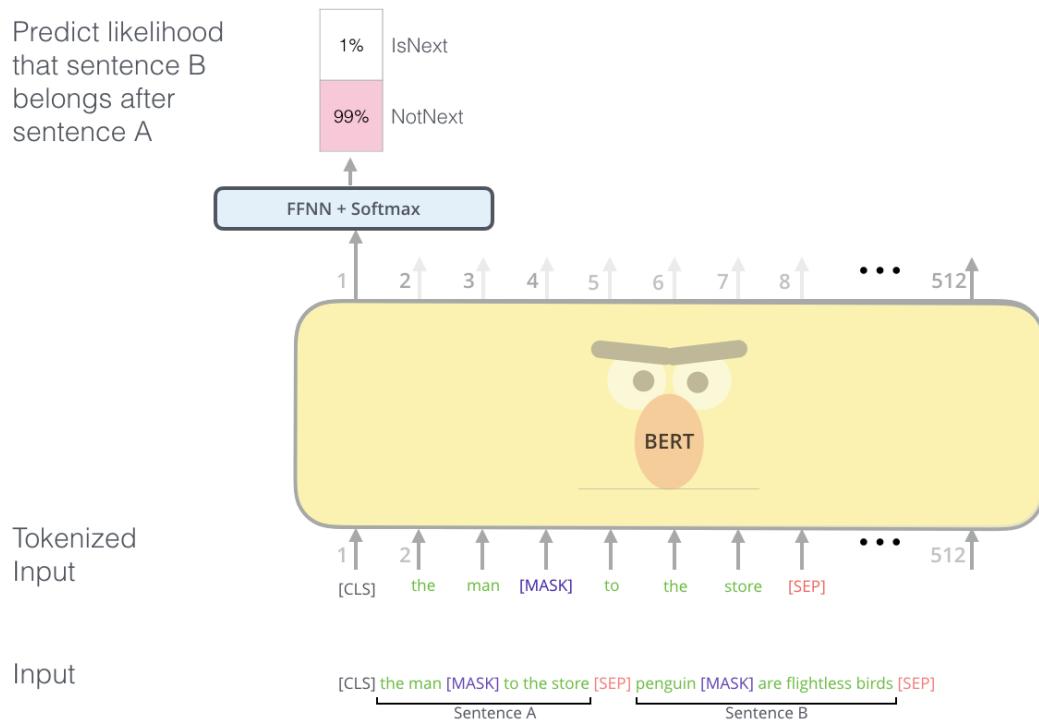
Về mặt kỹ thuật, việc dự đoán các từ bị che ở đầu ra yêu cầu thêm các tầng để phân lớp (classifier) nhận đầu vào là đầu ra của BERT [2]. Như Hình 49, chúng ta sẽ thêm một tầng kết nối đầy đủ (Fully Connected Layer) ngay sau BERT [2], tiếp theo đó sẽ sử dụng softmax để tính xác suất nhằm biết được từ bị che là từ gì. Số lượng units của tầng kết nối đầy đủ phải đúng bằng số lượng kích thước của từ điển (vocab_size).

Hàm mất mát (loss function) của BERT [2] sẽ bỏ qua mất mát từ những từ không bị che mà chỉ đưa vào mất mát của những từ bị che.

5.4. Next Sequence Prediction – NSP hay Two-sequence tasks

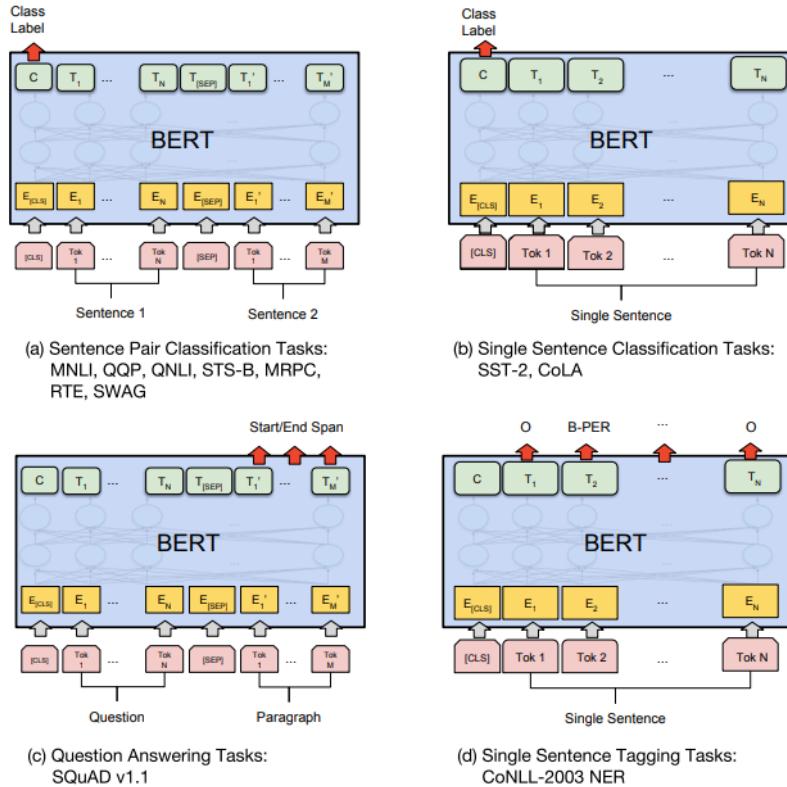
Two-sentence tasks là một phần quan trọng trong quá trình tiền huấn luyện (pre-training) của BERT [2] nhằm giúp mô hình hiểu được mối quan hệ giữa các câu. Nói cách khác, BERT [2] không chỉ quan tâm đến từng câu một mà còn nhìn vào mối quan hệ giữa các câu trong văn bản.

Trong quá trình tiền huấn luyện (pre-training), BERT [2] sẽ đưa cho mô hình một cặp câu (A, B) và yêu cầu mô hình dự đoán xem câu B có phải là câu tiếp theo của câu A hay không (Hình 50). Bằng cách này, mô hình sẽ học được cách đặt câu hỏi dựa trên nội dung của câu trước đó, đồng thời làm quen với các mối quan hệ giữa các câu.



Hình 50. Mô tả tác vụ NSP hay Two-sequence tasks trong quá trình tiền huấn luyện BERT [2]

5.5. Một số mô hình cụ thể



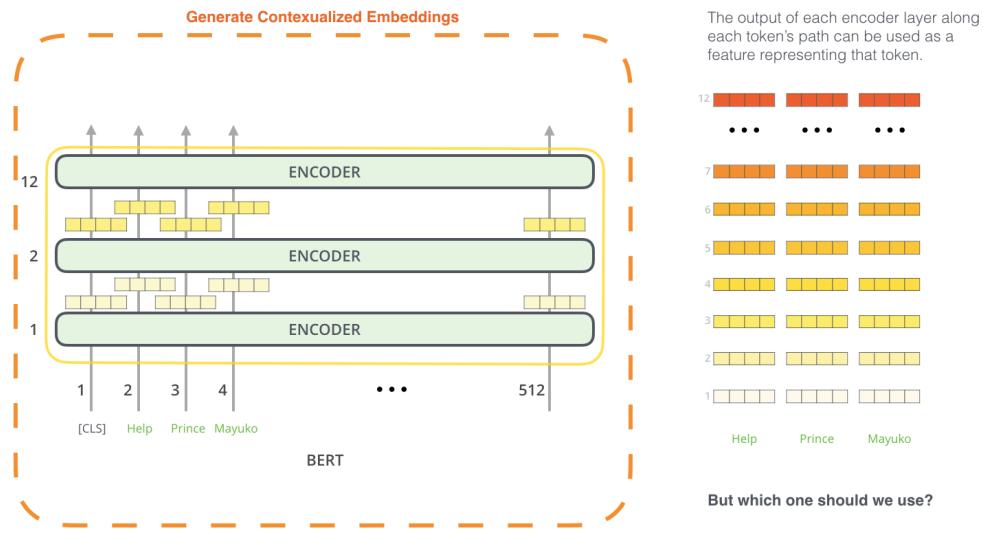
Hình 51. Các mô hình cụ thể cho các tác vụ khác nhau của mô hình BERT [2]

Các mô hình cụ thể cho các tác vụ khác nhau được thể hiện trong bài báo của BERT [2] được thể hiện trên Hình 51. Trong đó, các tác giả của bài báo đã chỉ ra nhiều cách để sử dụng BERT [2] cho các tác vụ khác nhau. Ví dụ, để phân loại văn bản, BERT [2] có thể được sử dụng như một bộ mã hóa (Encoder), trong đó lớp cuối cùng được thay đổi để phù hợp với mục đích phân lớp.

5.6. Trích xuất đặc trưng sử dụng BERT [2]

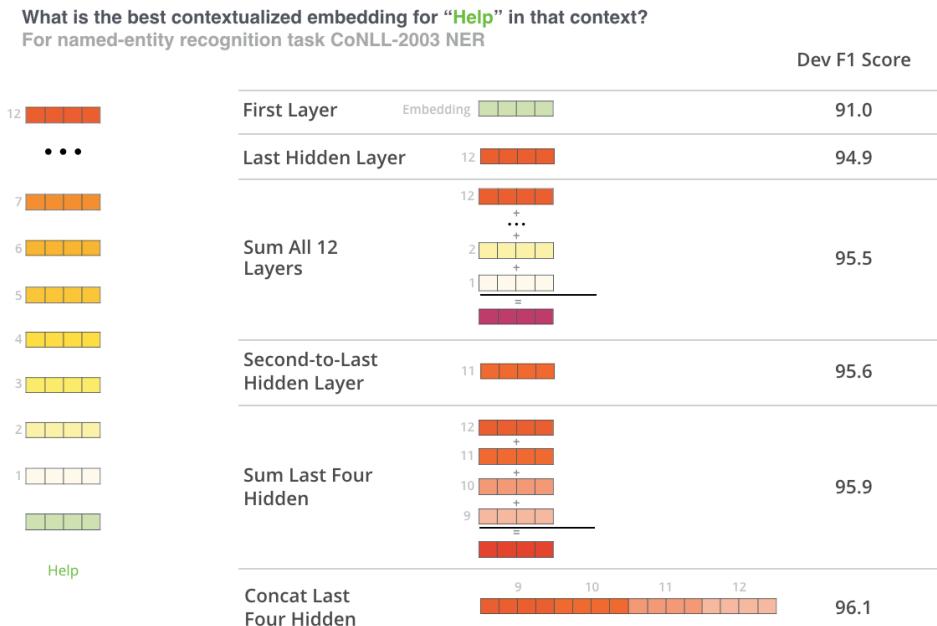
Phương pháp fine-tuning không phải là cách duy nhất để sử dụng BERT [2], chúng ta có thể sử dụng BERT [2] đã được pre-trained để tạo ra các contextualized word embeddingⁱⁱ.

Trong quá trình huấn luyện BERT [2], mỗi từ trong câu đầu vào được mã hóa dưới dạng một vector đặc trưng bằng cách sử dụng đầu ra của các tầng Encoder. Điều này cho phép mỗi từ được biểu diễn như là một vector đặc trưng phụ thuộc vào ngữ cảnh của từ đó trong câu. Các vector đặc trưng này được gọi là các contextualized word embeddings, và chúng có thể được sử dụng để biểu diễn các từ trong các tác vụ xử lý ngôn ngữ tự nhiên khác nhau.



Hình 52. Trích xuất đặc trưng sử dụng BERT [2]

Đầu ra của mỗi tầng Encoder trên đường đi của mỗi từ có thể được sử dụng như một đặc trưng đại diện cho từ đó (Hình 52). Vậy đâu là sự lựa chọn tốt nhất để dùng làm contextualized embedding. Trong bài báo [2], tác giả xem xét sáu lựa chọn được mô tả như Hình 53 bên dưới cùng với Dev F1-score⁸.



Hình 53. Lựa chọn contextualized embedding từ các đầu ra Encoder

⁸ Dev F1-score là một chỉ số đánh giá hiệu suất của mô hình trong quá trình đánh giá (validation) trên tập dữ liệu kiểm tra. Nó được tính bằng cách sử dụng công thức F1-score trên tập dữ liệu kiểm tra và được sử dụng để so sánh và đánh giá hiệu suất của các mô hình khác nhau trong cùng một tác vụ. Chỉ số này càng cao thì mô hình càng tốt.

6. Mô hình ViT [1]

Vision Transformer (ViT) [1] là một kiến trúc mạng neuron sử dụng kiến trúc Transformer [9] trong xử lý ảnh.

ViT [1] được thiết kế để giải quyết các vấn đề trong việc sử dụng mạng neuron truyền thống cho các tác vụ xử lý ảnh, như kích thước đầu vào cố định và sự phụ thuộc vào kiến trúc của mạng. ViT [1] sử dụng một phương pháp gọi là Patch Embedding, trong đó mỗi vùng của ảnh được chuyển đổi thành một vector số, sau đó truyền vào kiến trúc Transformer để tính toán. Với cách tiếp cận này, ViT [1] cho phép một mô hình có thể học cách thực hiện các tác vụ liên quan đến ảnh trên một tập dữ liệu lớn mà không cần biết trước kích thước ảnh đầu vào hoặc kiến trúc mạng cụ thể.

Do đó, ViT [1] đã đạt được những kết quả ấn tượng trên các tác vụ xử lý ảnh như phân loại ảnh, phát hiện vật thể, nhận dạng khuôn mặt và nhận dạng hành động trong video.

6.1. Kiến trúc tổng quan của ViT [1]

Để hiểu rõ hơn về cấu trúc của mô hình, trước hết chúng ta sẽ tìm hiểu về ưu điểm và nhược điểm của Vision Transformer [1].

Vision Transformer [1] có những ưu điểm đáng chú ý sau:

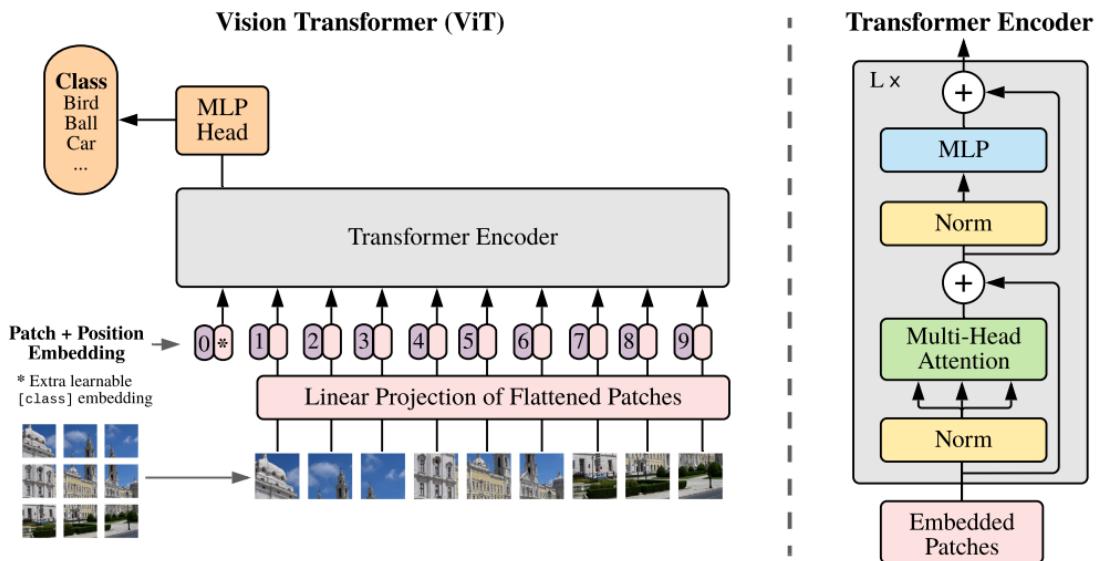
- Vision Transformer được xây dựng dựa trên kiến trúc Transformer [9], giúp cho việc mở rộng mô hình trở nên dễ dàng hơn nhờ sử dụng các thành phần Attention đã được tối ưu hóa. Trong khi đó, các mô hình attention khác với kiến trúc đặc thù cho một nhiệm vụ nhất định thường rất khó để mở rộng sang các nhiệm vụ khác.
- Transformers được chứng minh là có hiệu suất tốt trong việc học large-scale model⁹, do đó Vision Transformer [1] cũng có thể tận dụng được hiệu quả này.
- Trong quá trình transfer learning, Vision Transformer [1] sử dụng ít tài nguyên tính toán hơn so với CNN, giúp tiết kiệm thời gian và chi phí.

Bên cạnh đó, nhược điểm của Vision Transformer [1] là Yêu cầu nhiều dữ liệu hơn so với CNN do thiếu Inductive biasⁱⁱⁱ. Điều này là do Vision Transformer [1] có giới hạn trong việc sử dụng thông tin không gian của ảnh, dẫn đến có thể giảm hiệu suất trên một số tập dữ liệu cụ thể.

⁹ Large-scale model là mô hình mạng neuron có số lượng tham số lớn hơn so với các mô hình thông thường, thường được huấn luyện trên tập dữ liệu lớn. Mô hình lớn hơn cho phép mô hình học được những đặc trưng phức tạp và có khả năng dự đoán tốt hơn trên tập dữ liệu mới. Tuy nhiên, huấn luyện các mô hình lớn hơn yêu cầu nhiều tài nguyên tính toán và bộ nhớ, do đó cần sử dụng các công nghệ tiên tiến để xử lý và giải quyết vấn đề này.

Kiến trúc của mô hình Vision Transformer (ViT) [1]

Vision Transformer [1] là một mô hình áp dụng Encoder của Transformer [9] đã được sử dụng trong xử lý ngôn ngữ tự nhiên vào xử lý ảnh. Do đó, giống như mô hình Transformer [9], nó cũng nhận đầu vào là một chuỗi các token chứ không phải một vector. Để làm điều này, hình ảnh được chia thành các khối (Patch) có kích thước cố định và được chuyển đổi thành vector, tạo thành một chuỗi token để được sử dụng làm đầu vào cho mô hình Transformer [9]. Qua đó, mô hình có thể áp dụng Transformer [9] vào xử lý ảnh trong khi vẫn giữ được thông tin không gian của hình ảnh.



Hình 54. Mô hình tổng quát của ViT [1]

Trong Transformer [9], dữ liệu chuỗi được nhúng (Embedding) và thêm mã hóa vị trí (Positional Encoding). Vision Transformer [1] cũng tuân theo cùng quy trình. Quá trình được thực hiện theo thứ tự như sau:

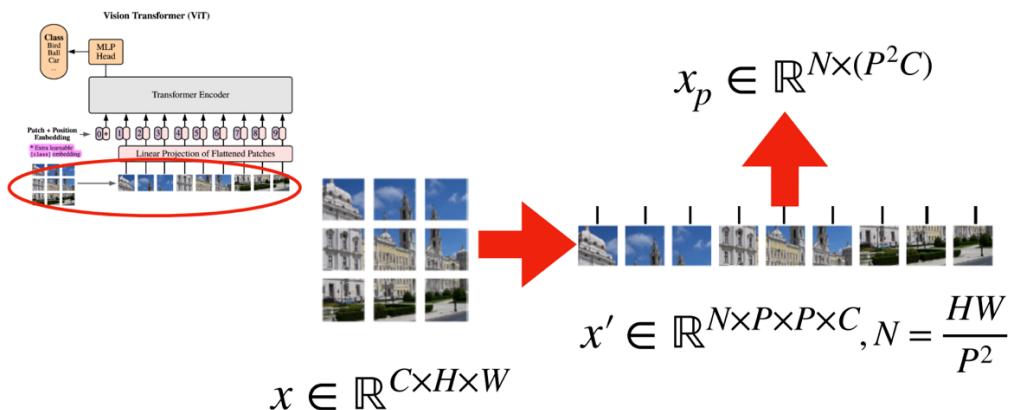
- ① Để tạo ra dữ liệu đầu vào có cấu trúc tương tự Transformer [9], hình ảnh được chia thành các Patch và sắp xếp chúng dưới dạng chuỗi, từ trái qua phải và từ trên xuống dưới.
- ② Tiếp theo, các Patch được làm phẳng để chuyển đổi chúng thành các vector, tương tự như quá trình Embedding trong Transformer [9].
- ③ Sau đó, các vector được truyền qua một phép toán tuyến tính để thực hiện quá trình nhúng (Embedding).
- ④ Một [class] token (đại diện cho lớp) được thêm vào kết quả của quá trình Embedding.

⑤ Cuối cùng, kết hợp thêm mã hóa vị trí (Positional Encoding) vào đầu vào đã có thêm [class] token, đầu vào của Vision Transformer [1] được hoàn thành. Vì vị trí của mỗi Patch trong hình ảnh là rất quan trọng, nên mã hóa vị trí cũng được áp dụng vào.

Sau khi hoàn tất các bước ① đến ⑤ để tạo ra đầu vào, Transformer's Encoder [9] được lặp lại L lần. Cuối cùng, ở đầu ra, là một chuỗi các vector, mỗi vector đại diện cho trạng thái ẩn (hidden state) của từ tương ứng ở đầu vào. Ngoài chuỗi các vector, đầu ra còn bao gồm một token đặc biệt gọi là [class], được sử dụng để đại diện cho toàn bộ chuỗi và được thêm vào đầu chuỗi. Chúng ta sẽ sử dụng đầu ra này để làm đầu vào của MLP Head và sử dụng MLP Head này để phân loại các lớp.

Quá trình trên được mô tả như Hình 54 và được trình bày chi tiết ở các mục tiếp theo.

6.2. Tạo các Patch từ hình ảnh đầu vào



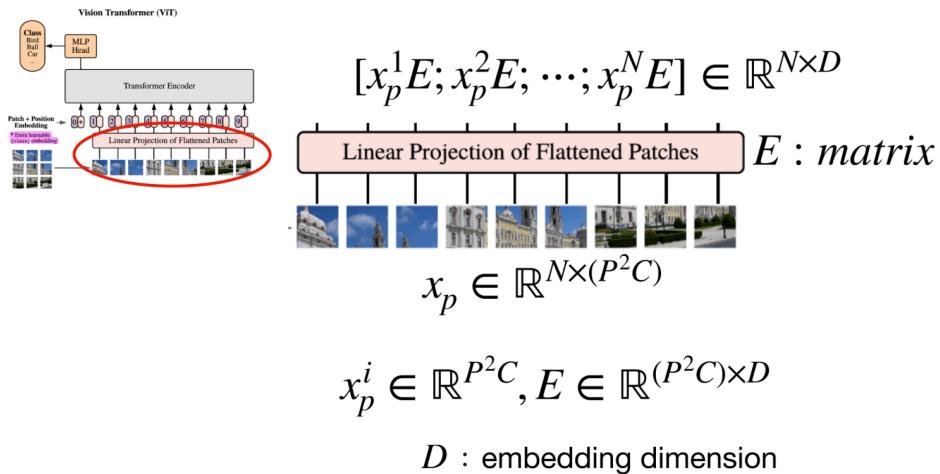
Hình 55. Chia hình ảnh đầu vào thành các Patch(es)

Trong Hình 55, (C , H , W) tương ứng với Channel (kênh), Height (chiều cao), Width (chiều rộng) và P đại diện cho kích thước của mỗi Patch, do đó mỗi Patch sẽ có kích thước là (C , P , P) và được tạo thành bằng cách chia ảnh đầu vào thành các Patch không trùng lặp. Tổng số Patch(es) là N .

Mỗi Patch sẽ được làm phẳng (Flatten) ở bước tiếp theo, và sẽ trở thành một vector có kích thước $P^2 \times C$, tương tự cũng sẽ có N vector tương ứng với N Patch(es).

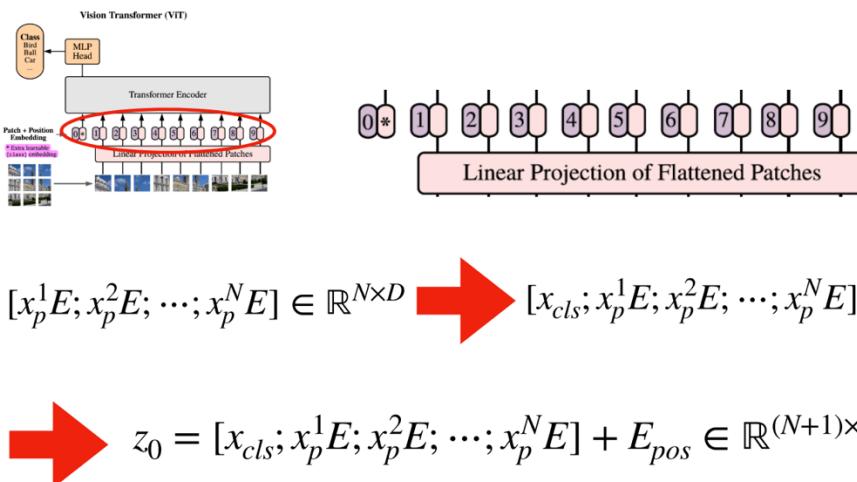
Ví dụ, nếu ảnh đầu vào có kích thước (3, 256, 256) với $P = 16$, kích thước của mỗi Patch sẽ là (3, 16, 16) và tổng số Patch $N = 16 \times 16 = 256$. Khi làm phẳng mỗi Patch sẽ tạo thành vector có kích thước là $3 \times 16 \times 16 = 768$ và sẽ có $N = 256$ vector ứng với $N = 256$ Patch(es). Khi biểu diễn dưới dạng chuỗi dữ liệu (sequence data) sẽ có kích thước là (256, 768).

6.3. Làm phẳng các Patch



Hình 56. Làm phẳng Patch(es)

Các x_p đã được tạo ra từ bước trước đó, ở bước này, các x_p sẽ được nhân với ma trận E . Kích thước của ma trận E là $(P^2 \times C, D)$ với D là số chiều của Embedding, giúp biến đổi các vector có kích thước $P^2 \times C$ thành vector có kích thước D . Do đó, kích thước ban đầu của ma trận tập hợp các x_p là $(N, P^2 \times C)$ và sau khi nhân với ma trận E sẽ trở thành (N, D) . Do đó, nếu tính theo batch, mỗi batch sẽ có kích thước là $(\text{batch_size}, N, D)$. Quá trình này được mô tả như Hình 56.



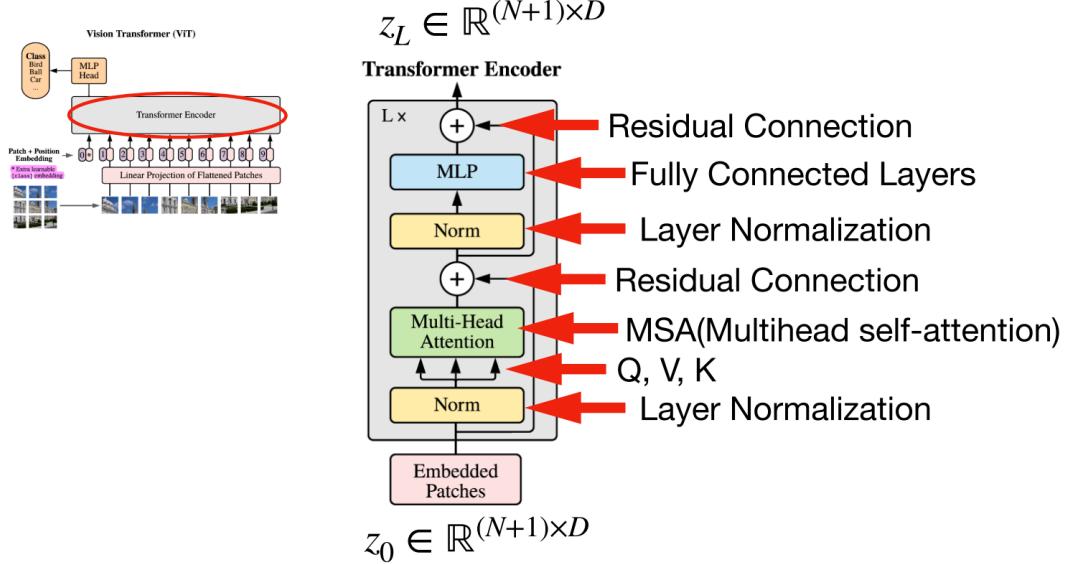
Hình 57. Quá trình tạo ra đầu vào hoàn chỉnh cho mô hình ViT

Sau quá trình trên (Hình 56), chúng ta sẽ thêm [class] token vào mỗi phía trước vector như Hình 57 để tạo ra một ma trận có kích thước $(N + 1, D)$ thay vì (N, D) .

Cuối cùng, chúng ta cộng thêm ma trận $(N+1, D)$ này với ma trận Positional Encoding để hoàn thành quá trình chuẩn bị dữ liệu đầu vào. (Hình 57)

Ví dụ, xét tập dữ liệu CIFAR-10 với mỗi ảnh có kích thước là (3, 32, 32). Với $P = 4$, ta có $N = 64$. Khi đó ma trận tập hợp các x_p sẽ có kích thước là $(64, 4 \times 4 \times 3)$, hay $(64, 48)$. Với $D = 128$, kết quả sau khi áp dụng Embedding sẽ có kích thước là $(64, 128)$. Sau khi thêm [class] token sẽ thành $(65, 128)$. Cuối cùng cộng thêm Positional Encoding để tạo ra z_0 có kích thước $(65, 128)$.

6.4. Encoder trong Vision Transformer [1]



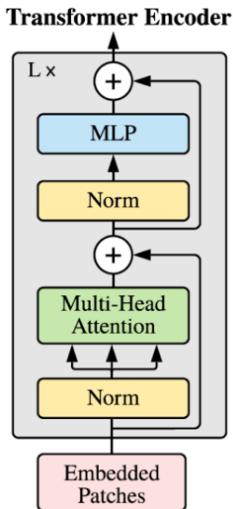
Hình 58. Mô tả các thành phần của Encoder trong ViT

Cũng giống như trong Transformer [9], bộ mã hóa (Encoder) của Vision Transformer [1] giữ cho đầu vào và đầu ra có cùng kích thước để lặp lại L lần.

Kiến trúc Vision Transformer [1] được sử dụng có một số khác biệt so với Encoder gốc của Transformer [9]. Trong Encoder gốc của Transformer [9], Multi-Head Attention được thực hiện trước Layer Normalization, nhưng ta có thể thấy thứ tự ngược lại được áp dụng trong kiến trúc của Vision Transformer [1].

Bắt đầu từ giá trị đầu vào z_0 , kết quả đầu ra của Encoder sẽ là z_L sau khi lặp lại L lần. Quá trình tính toán được mô tả như Hình 58.

Multi-Head Attention được sử dụng trong kiến trúc này là Self-Attention, do đó ta có thể gọi nó là Multi-Head Self-Attention (MSA).



$$z'_l = MSA(LN(z_{l-1})) + z_{l-1}$$

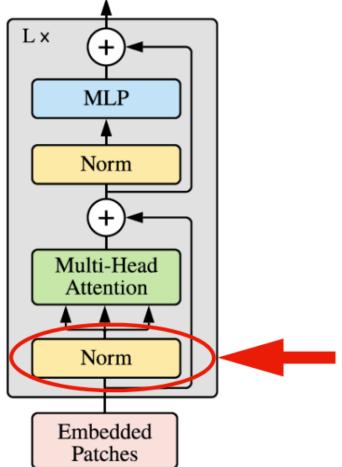
$$z_l = MLP(LN(z'_l)) + z'_l \quad l = 1, 2, \dots, L$$

Hình 59. Quá trình tính toán z_L từ z_0

Trong Hình 59, ngoài MSA, ta có LN là Layer Normalization, MLP là Multi-Layer Perceptron.

Layer Normalization (LayerNorm - LN)

Transformer Encoder



Layer Normalization: Thực hiện chuẩn hóa cho mỗi đặc trưng
(tức là chuẩn hóa theo chiều D)

Hình 60. Layer Norm trong Encoder của Vision Transformer

Layer Normalization (Hình 60): thực hiện chuẩn hóa cho mỗi đặc trưng.

Do Encoder lặp lại L lần, chúng ta sẽ gọi đầu vào là z_i ứng với lần lặp thứ i .

$$z_i = [z_i^1, z_i^2, z_i^3, \dots, z_i^N, z_i^{N+1}] \quad (1)$$

Layer Norm tuân theo công thức sau (Hình 61):

$$\text{LN}(z_i^j) = \gamma \frac{z_i^j - \mu_i}{\sigma_i} + \beta \quad (2)$$

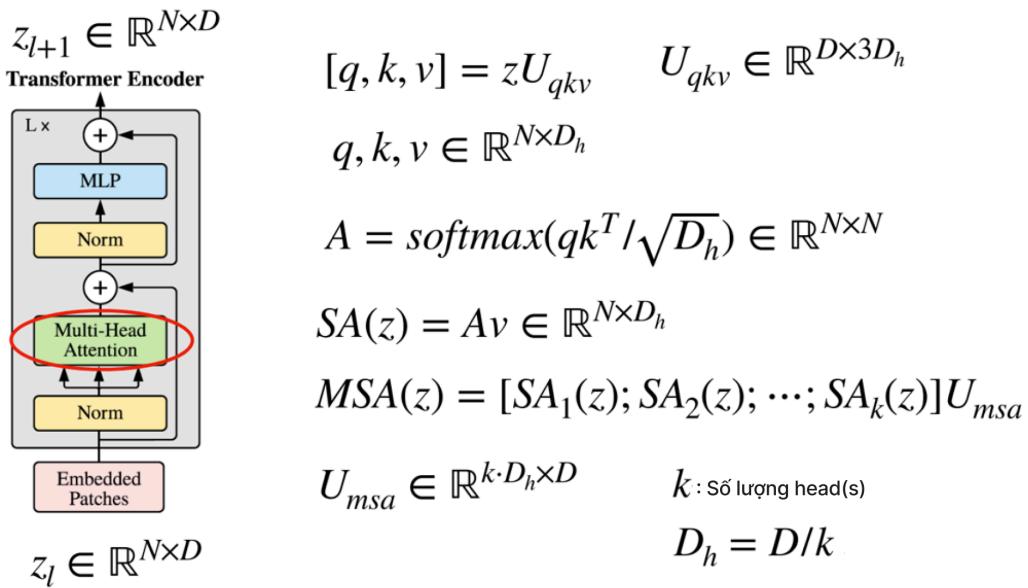
$$= \gamma \frac{z_i^j - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta \quad (3)$$

Hình 61. Công thức tính LayerNorm

Trong đó:

- μ_i là giá trị trung bình của feature thứ i trong vector đầu vào z_i .
- σ_i^2 là phương sai của feature thứ i trong vector đầu vào z_i .
- ϵ là một hằng số nhỏ được cộng vào tránh trường hợp phương sai bằng 0.
- γ và β là các hệ số học được, giúp cho mô hình có thể học được những giá trị tốt hơn để chuẩn hóa.

Multi-Head Attention (MSA)



Hình 62. Minh họa quá trình tính toán trong MSA

Đầu vào và đầu ra của Multi-Head Attention (Hình 62) có thể được biểu diễn:

$$z_l \in \mathbb{R}^{N+1 \times D} \rightarrow z_l \in \mathbb{R}^{N \times D}$$

Theo cấu trúc Attention, q , k và v tạo ra từ đầu vào z bằng phép nhân ma trận với các ma trận w_q , w_k , và w_v tương ứng.

$$q = z \cdot w_q \quad (w_q \in \mathbb{R}^{D \times D_h}) \quad (4)$$

$$k = z \cdot w_k \quad (w_k \in \mathbb{R}^{D \times D_h}) \quad (5)$$

$$v = z \cdot w_v \quad (w_v \in \mathbb{R}^{D \times D_h}) \quad (6)$$

$$[q, k, v] = z \cdot U_{qkv} \quad (U_{qkv} \in \mathbb{R}^{D \times 3D_h}) \quad (7)$$

Thay vì tính toán q , k và v riêng lẻ, chúng ta có thể sử dụng công thức (7) để tính toán q , k và v một cách đồng thời.

$$A = \text{softmax}\left(\frac{q \cdot k^T}{\sqrt{D_h}}\right) \in R^{N \times N} \quad (8)$$

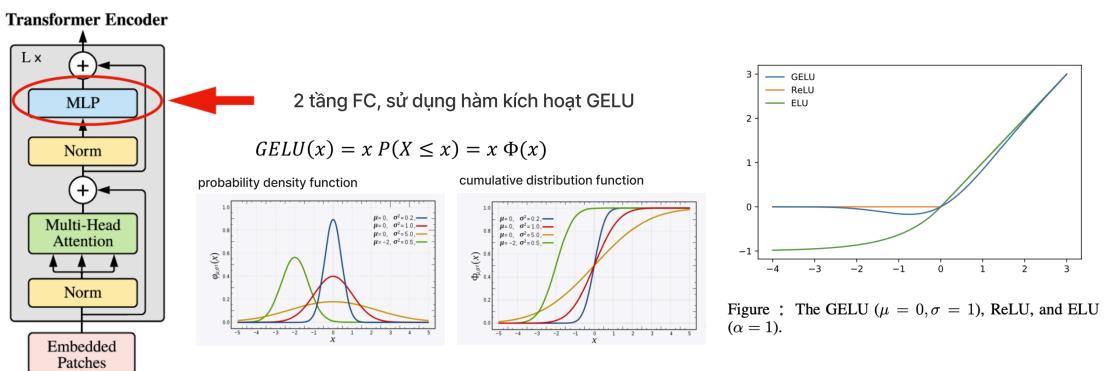
$$\text{SA}(z) = A \cdot v \in R^{N \times D_h} \quad (9)$$

$$\text{MSA}(z) = [\text{SA}_1(z); \text{SA}_2(z); \dots; \text{SA}_k(z)]U_{msa} \quad (10)$$

Chúng ta có thể sử dụng các công thức (8) và (9) để lấy kết quả Self-Attention của mỗi head, sau đó sử dụng công thức (10) để kết hợp kết quả Self-Attention của mỗi head và thực hiện phép toán tuyến tính (Linear) để thu được kết quả Multi-Head Attention cuối cùng.

Các kết quả Self-Attention được kết hợp với nhau bằng cách sử dụng phép toán tuyến tính (Linear) trong phương trình (10) sẽ cho ra kết quả có kích thước là (N, D_h, k) . Kích thước của U_{msa} là (k, D_h, D) , do đó kết quả của phép nhân sẽ có kích thước (N, D_h, D) . Qua đó, chúng ta có thể điều chỉnh kích thước của kết quả để có cùng kích thước với đầu vào của Transformer Encoder, tức là (N, D) .

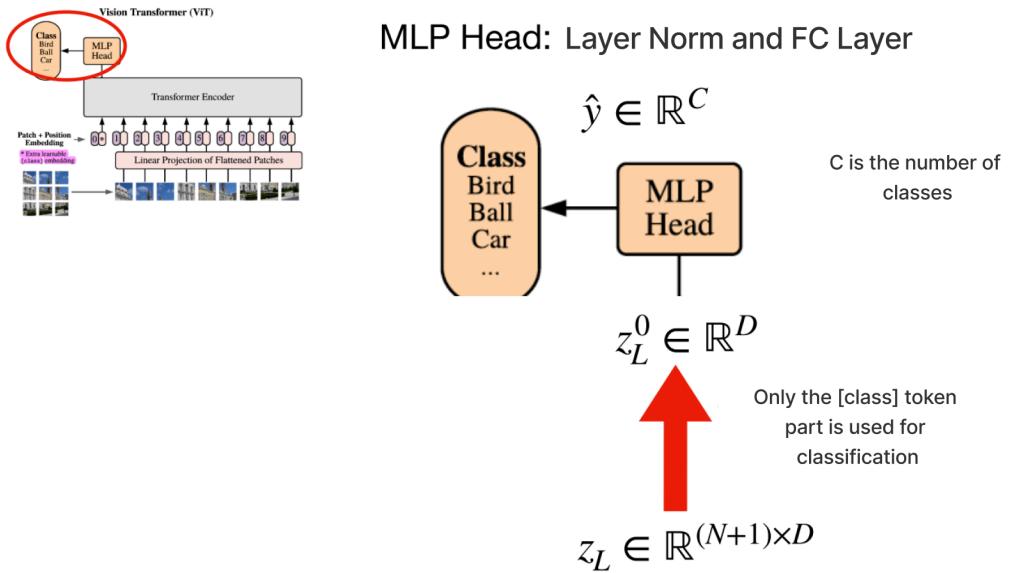
Multi-Layer Perceptron (MLP)



Hình 63. Hàm kích hoạt GELU trong MLP

Chúng ta sẽ đi qua quá trình cuối cùng của Transformer's Encoder [9], trong đó bao gồm hai tầng kết nối đầy đủ (FC Layer) kết hợp với GELU Activation [19]. GELU [19] sử dụng tích của giá trị đầu vào và phân phối chuẩn tích lũy (cumulative distribution function) của giá trị đầu vào. Hàm này cũng khả vi (tức có đạo hàm) tại mọi điểm và có thể sử dụng như một hàm kích hoạt, vì nó không phải là hàm tăng đơn điệu, và do đó có thể điều chỉnh giá trị của đầu vào theo tỉ lệ so với các giá trị đầu vào khác, điều này cho phép phân tích xác suất. (Hình 63)

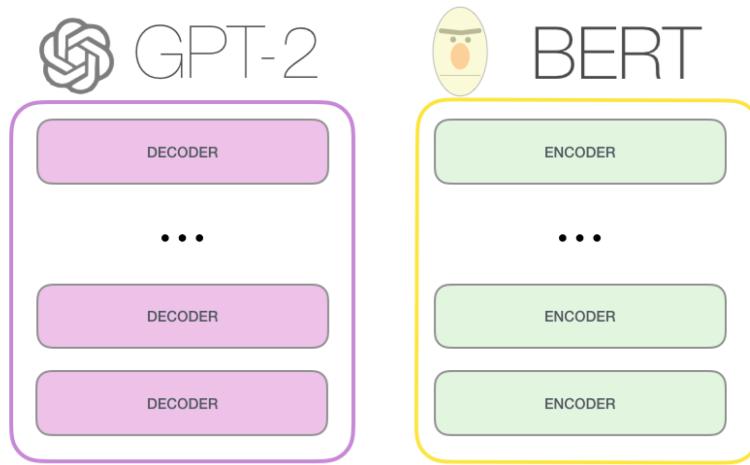
Multi-Layer Perceptron (MLP) Head



Hình 64. Phân lớp với MLP Head

Đầu ra cuối cùng của L lần lặp được sử dụng để giải quyết bài toán phân loại. Một MLP được sử dụng để phân loại lớp bằng cách sử dụng [class] token. (Hình 64)

7. Mô hình GPT-2 [5]



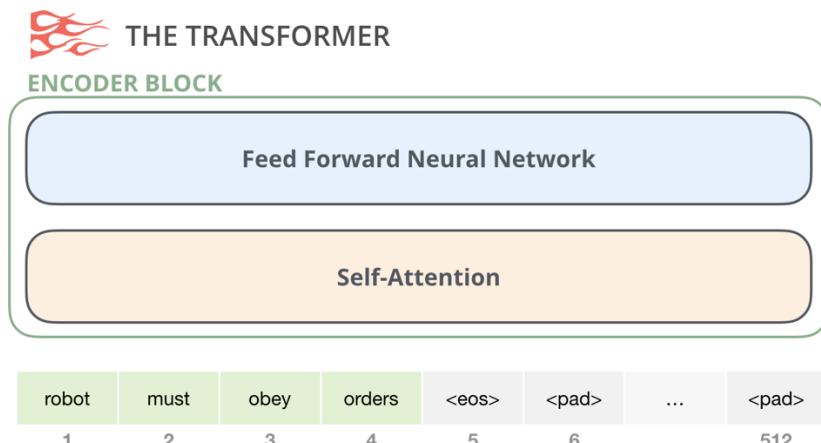
Hình 65. BERT [2] được xây dựng dựa trên Transformer's Encoder [9], trong khi GPT-2 [5] là Transformer's Decoder [9]

GPT-2 [5] được xây dựng bằng các khối giải mã (Decoder) trong Transformer [9]. Trong khi đó, BERT [2] sử dụng các khối mã hóa (Encoder) trong Transformer [9]. Chúng ta sẽ xem xét sự khác biệt này trong phần tiếp theo. Nhưng một sự khác biệt chính giữa hai mô hình này là GPT-2 [5], giống như các mô hình ngôn ngữ truyền thống khác, đầu ra của GPT-2 [5] chỉ là một token tại một thời điểm.

7.1. Từ Transformer đến GPT-2

Trong bài báo gốc của Transformer [9], tác giả đã giới thiệu hai loại Transformer blocks.

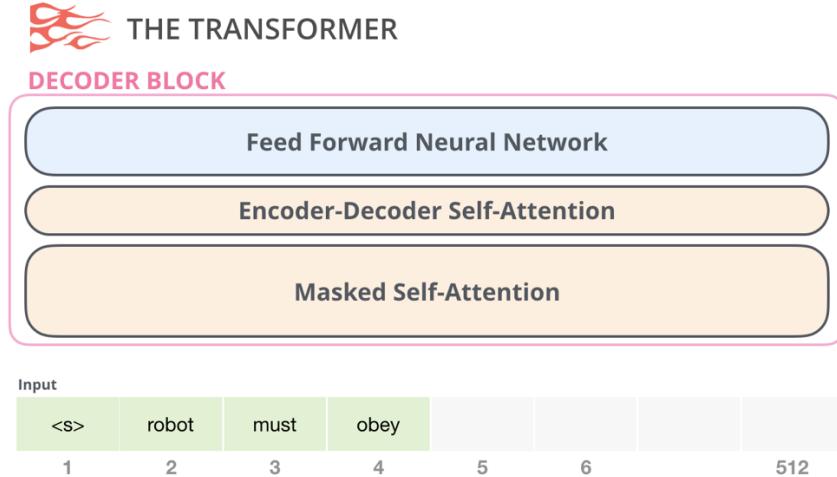
(1) Encoder block



Hình 66. Encoder block trong Transformer gốc

Khối Encoder trong Transformer gốc [9] có khả năng xử lý đầu vào có độ dài câu tối đa 512. Nếu câu có độ ngắn hơn 512, chúng ta chỉ cần thêm vào các `<pad>` token. (Hình 66)

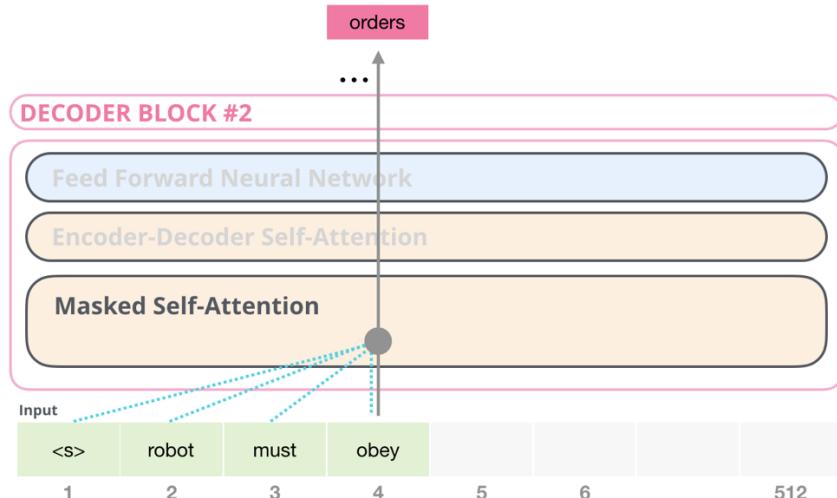
(2) Decoder block



Hình 67. Decoder block trong Transformer gốc

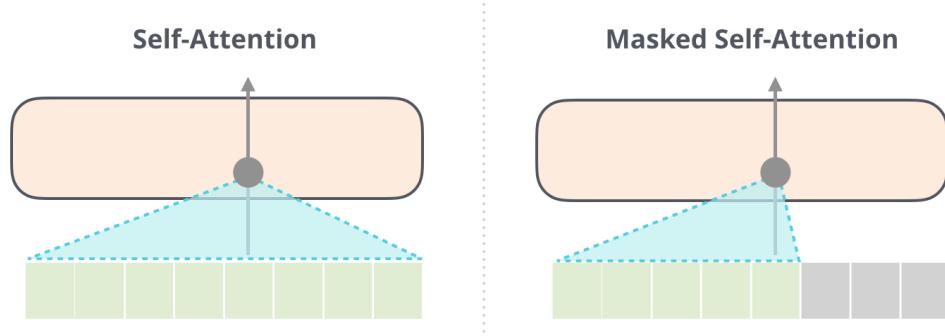
Khối Decoder trong Transformer gốc [9] là một biến thể “nhỏ” về mặt kiến trúc của khối Encoder: được thêm vào một tầng cho phép Decoder tập trung đến các đoạn cụ thể từ Encoder. (Hình 67)

Một điểm khác biệt quan trọng trong Self-Attention, là nó che giấu các thông tin từ tương lai – không phải bằng cách thay thế bằng [mask] như BERT [2], mà bằng cách can thiệp vào quá trình tính toán Self-Attention để ngăn các thông tin từ các từ bên phải của vị trí đang được tính toán.



Hình 68. Masked Self-Attention trong GPT-2 [5]

Ví dụ như Hình 68, hãy xem xét đường đi của vị trí #4, chúng ta có thể thấy rằng nó chỉ được phép tập trung đến các từ hiện tại và trước đó.

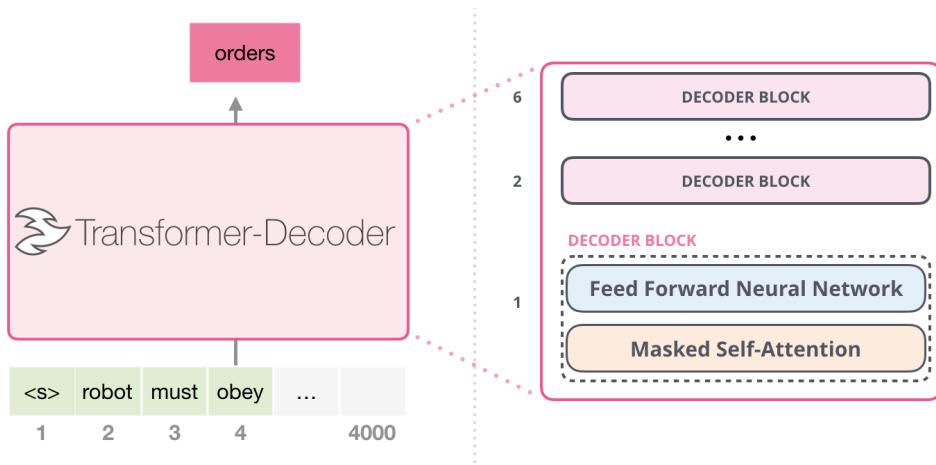


Hình 69. Minh họa Self-Attention và Masked Self-Attention

Việc phân biệt giữa Self-Attention (được sử dụng trong BERT [2]) và Masked Self-Attention (được sử dụng trong GPT-2 [5]) là rất quan trọng (Hình 69). Một khối Self-Attention thông thường cho phép một vị trí nhìn vào các từ ở phía bên phải của nó. Còn Masked Self-Attention thì ngăn không cho điều đó xảy ra.

Sau bài báo gốc về Transformer [9], một bài báo khác [20] đề xuất một cấu trúc khác của khối Transformer có khả năng làm việc trong tác vụ mô hình ngôn ngữ. Mô hình này loại bỏ khối mã hóa (Encoder) của Transformer [9]. Vì vậy, mô hình này được gọi là “Transformer – Decoder”. Mô hình mô hình ngôn ngữ dựa trên Transformer [9] này bao gồm một ngăn xếp gồm 6 khối Transformer Decoder [9]. (Hình 70)

(3) Decoder-only block



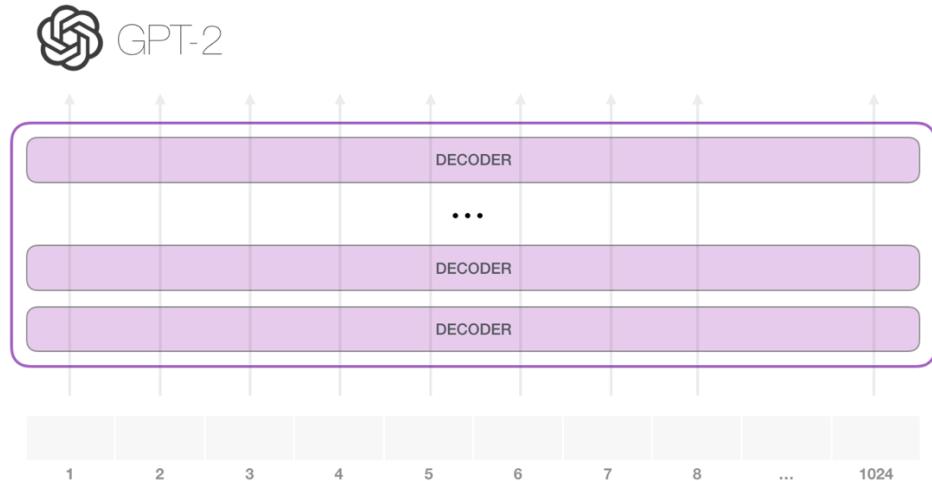
Hình 70. Mô hình Transformer-Decoder

Các khối Decoder (Decoder block) là hoàn toàn giống nhau và khá giống với khối Decoder trong mô hình Transformer gốc [9]. Tuy nhiên trong mô hình này tác giả đã loại bỏ đi tầng Self-Attention thứ hai.

Lưu ý rằng trong mô hình Transformer – Decoder [20] có thể nhận đầu vào với độ dài câu lên đến 4000 token thay vì 512 token như mô hình Transformer gốc [9].

Mô hình GPT-2 [5] sử dụng kiến trúc như mô hình Transformer – Decoder [20] này.

7.2. Kiến trúc tổng quan của GPT-2 [5]



Hình 71. Mô hình GPT-2 [5]

Như đã trình bày ở mục 7.1, mô hình GPT-2 [5] sử dụng kiến trúc Transformer – Decoder [20], bao gồm nhiều khối Decoder được xếp chồng lên nhau như một ngăn xếp.

GPT-2 [5] có thể xử lý đầu vào với độ dài tối đa 1024 token (ứng với tối đa 1024 từ).

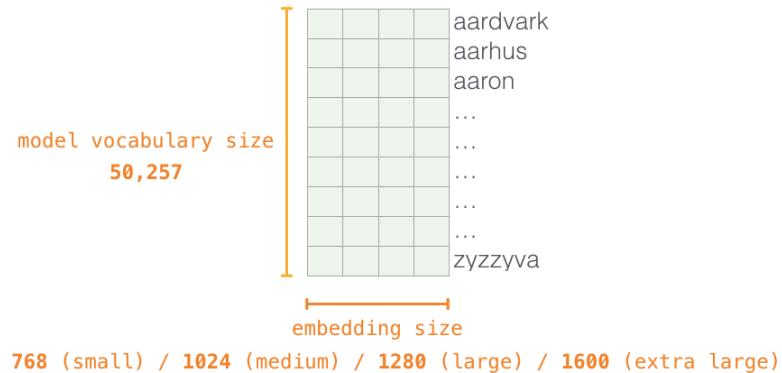
7.3. Biểu diễn đầu vào của GPT-2 [5]



Hình 72. Biểu diễn đầu vào của mô hình GPT-2 [5]

Đầu vào của GPT-2 [5] bao gồm Token Embedding (wte) và Positional Encoding (wpe). Cũng giống như như BERT [2], GPT-2 [5] cũng bắt đầu với token mang ý nghĩa bắt đầu, cụ thể ở đây là <s> (Hình 72), nhưng trong thực tế giá trị của token này lại là <endoftext> hay <eos> - End Of Sequence.

Token Embeddings (wte)

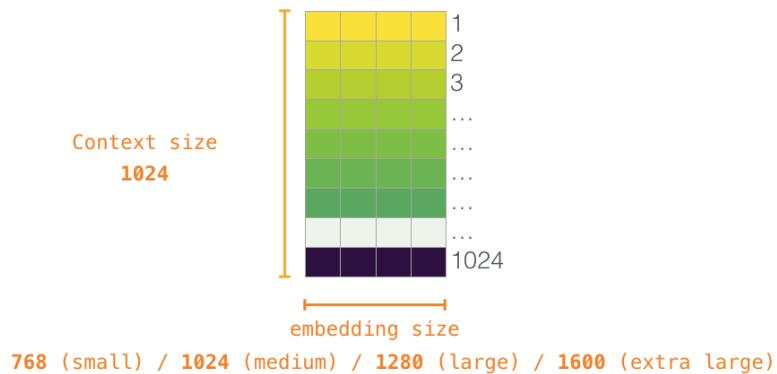


Hình 73. Ma trận wte với kích thước từ điển vocab_size = 50.257

Token Embedding (wte) là một ma trận có kích thước (vocab_size, hidden_size), trong đó: mỗi hàng tương ứng với một từ trong từ điển và mỗi cột tương ứng với một vector có kích thước hidden_state. (Hình 73)

Đầu vào được chia thành các token, mỗi token sẽ được biểu diễn dưới dạng một one-hot vector và được nhân với ma trận wte để tạo ra một vector có kích thước hidden_state biểu diễn cho từ đó.

Positional Encodings (wpe)

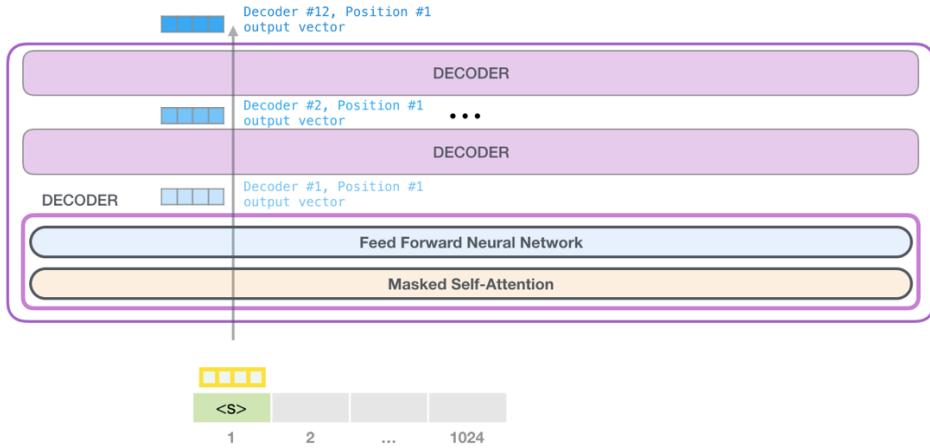


Hình 74. Ma trận wpe, với seq_length = 1024 (context_size), hidden_state tương ứng với phiên bản sử dụng

Positional Encoding (wpe) cũng là một ma trận có kích thước (seq_length, hidden_size), trong đó: mỗi hàng tương ứng với một vị trí của token trong câu và mỗi cột tương ứng với vector có kích thước hidden_state. (Hình 74)

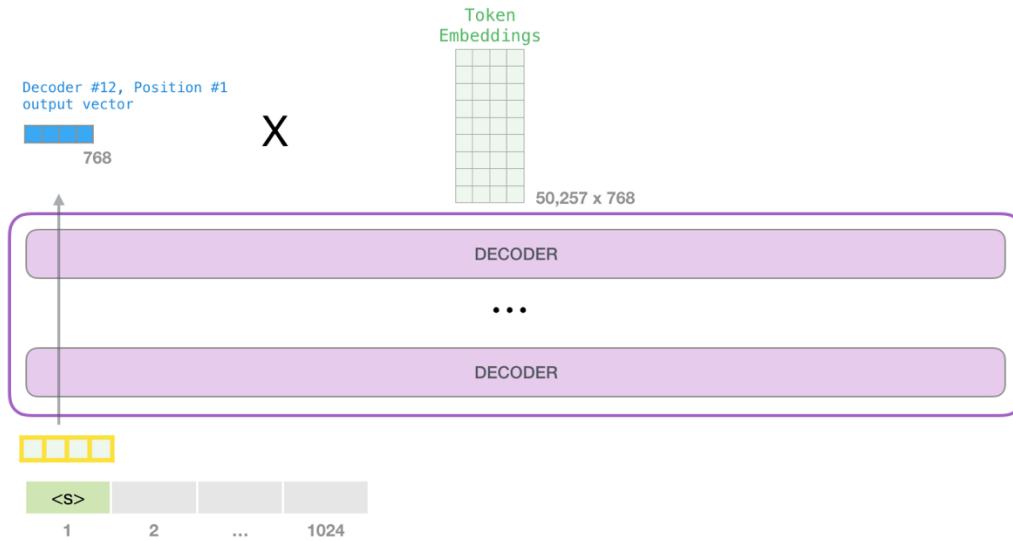
Mỗi phần tử của ma trận wpe được tính bằng cách sử dụng hàm số đã được định nghĩa (hàm sin, cos – được trình bày ở mục 4.2.2) để biểu diễn vị trí của token đó trong câu. Ma trận wpe được cộng vào ma trận wte trước khi được đưa vào mô hình.

7.4. Biểu diễn đầu ra của GPT-2 [5]



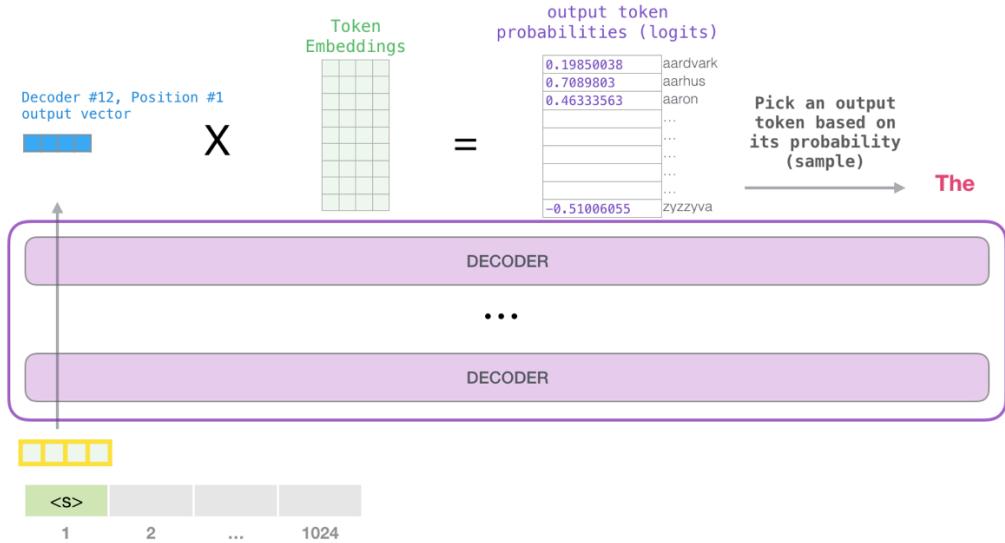
Hình 75. Biểu diễn đầu ra của các khối Decoder

Tương tự như BERT [2], với mỗi tầng Decoder đều sẽ có các vector là đầu ra tương ứng với vị trí của các token. Đầu ra của Decoder thứ l sẽ là đầu vào cho Decoder thứ $l + 1$. (với $l = 1..L - 1$, L là số lượng tầng Decoder). (Hình 75)



Hình 76. Biểu diễn đầu ra của mô hình GPT-2 [5]

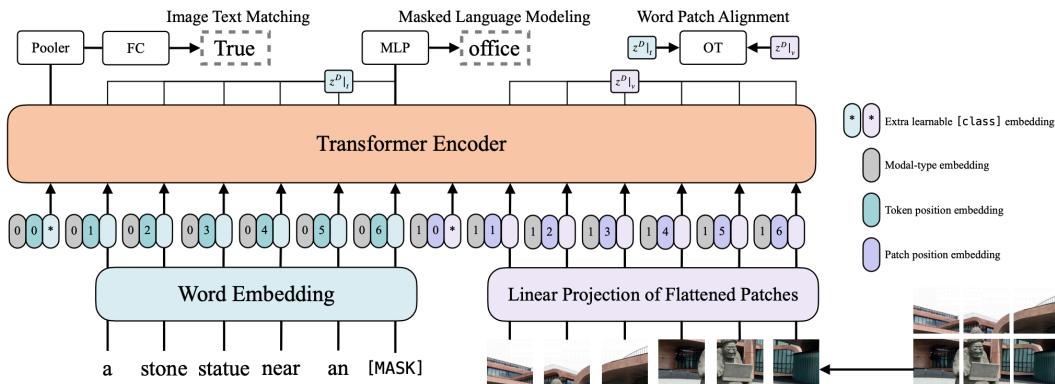
Đầu ra tại tầng Decoder thứ L cũng chính là đầu ra của GPT-2 [5]. Tuy nhiên, để được đầu ra cuối cùng, ta phải nhân từng vector ở đầu ra của tầng Decoder thứ L với ma trận wte ban đầu. (Hình 76)



Hình 77. Biểu diễn kết quả đầu ra của mô hình GPT-2 [5]

Kết quả của phép nhân sẽ tạo thành một vector cột có kích thước từ điển (`vocab_size`). Sau đó tuỳ vào tác vụ cụ thể mà tiến hành các bước tiếp theo, ví dụ, sử dụng GPT-2 [5] để sinh câu trả lời, thì tại mỗi thời điểm t chúng ta sẽ chọn ra từ có xác suất cao nhất trong vector cột có kích thước từ điển ứng với vị trí của `<s>` token để làm đầu ra cho mô hình. (Hình 77)

8. Mô hình ViLT [4]



Hình 78. Tổng quan kiến trúc mô hình ViLT và các tác vụ chính [4]

Về cơ bản, ViLT [4] có kiến trúc mô hình (Hình 78) tương tự như việc kết hợp ViT [1] với BERT [2] để xử lý đồng thời dữ liệu hình ảnh và văn bản trong một mô hình duy nhất.

Tuy nhiên cũng có một số tuỳ chỉnh khác so với kiến trúc gốc của mô hình ViT [1] hay BERT [2].

Cũng như các mô hình Transformer-based khác, ViLT [4] sẽ phải cộng thêm Positional Embedding vào Input Embedding để có được thông tin về vị trí. Do ViLT [4] xử lý đồng thời dữ liệu hình ảnh và văn bản, vì vậy, đối với ViLT [4], ngoài việc cộng thêm Positional Embedding thì tác giả cộng thêm Model-type Embedding để phân biệt được đầu vào là token (từ) hay patch (một phần của ảnh).

Trong quá trình khởi tạo mô hình, tác giả đã sử dụng các trọng số (weights) của mô hình pre-trained ViT [1], thay vì BERT [2] để tiếp tục huấn luyện. Các thông số quan trọng trong mô hình ViLT [4]:

(1) Kích thước `hidden_size` $H = 768$.

(2) Số tầng Transformer's Encoder $D = 12$ (ký hiệu là D thay vì L như trong mô hình ViT/BERT). Số heads $A = 12$.

(3) Kích thước `patch_size` $P = 32$ thay vì 16 như ViT.

(4) Kích thước đầu ra của tầng MLP cuối cùng trong Encoder là 3072.

Ngoài ra, tùy vào tác vụ cụ thể mà tác giả có thể chỉnh sửa các thông số trên cho phù hợp.

Quá trình pre-training (tiền huấn luyện) của mô hình ViLT [4], tác giả thực hiện trên hai tác vụ chính là Image-Text-Matching (ITM) và Masked Language Modeling (MLM) (Hình 78). Tương tự BERT [2], ViLT [4] cũng sử dụng WordPiece (mục 2.3.2) cho việc phân tách từ trong câu.

9. Từ BERT [2] đến PhoBERT [3]

PhoBERT [3] là một pre-trained model được huấn luyện trên tập dữ liệu lớn bao gồm các văn bản tiếng Việt. Tương tự BERT [2], PhoBERT [3] bao gồm hai phiên bản PhoBERT_{BASE} và PhoBERT_{LARGE} và có kiến trúc tương tự BERT_{BASE} và BERT_{LARGE}.

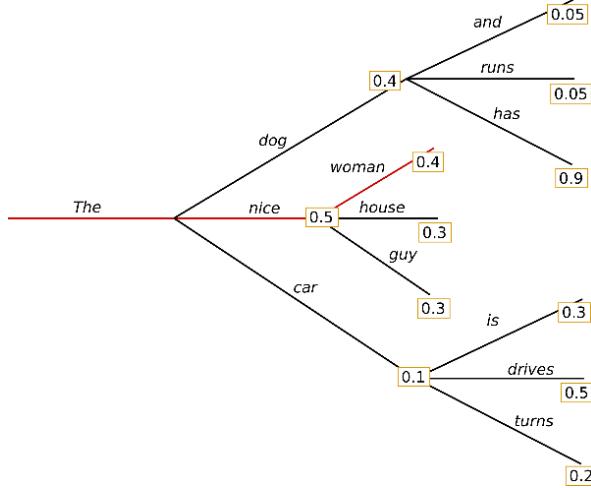
BERT [2] và PhoBERT [3] đều là các mô hình học sâu dựa trên kiến trúc Transformer [9] để thực hiện tác vụ xử lý ngôn ngữ tự nhiên. Tuy nhiên, có một số khác biệt cơ bản giữa chúng:

- Số lượng từ điển: BERT [2] sử dụng bộ từ điển có kích thước 30.000 từ, trong khi PhoBERT [3] sử dụng bộ từ điển có kích thước lớn hơn là 64.000 từ, được xây dựng trên cơ sở của ngôn ngữ tiếng Việt.
- Tiền xử lý dữ liệu: BERT [2] được huấn luyện trên dữ liệu tiếng Anh, trong khi PhoBERT [3] được huấn luyện trên dữ liệu tiếng Việt. Điều này có nghĩa là tiền xử lý dữ liệu của BERT [2] và PhoBERT [3] khác nhau. Ví dụ, BERT [2] sử dụng word tokenization cho tiếng Anh, trong khi PhoBERT [3] sử dụng word tokenization cho tiếng Việt. Do đó, PhoBERT [3] đạt được hiệu suất cao trên các bộ dữ liệu tiếng Việt, và được coi là một trong những mô hình tiên tiến nhất cho xử lý ngôn ngữ tự nhiên trong tiếng Việt.

10. Phương pháp sinh từ: Greedy Search hay Beam Search?

10.1. Greedy Search

Greedy Search là phương pháp tìm kiếm đơn giản, chỉ đơn giản là chọn từ có xác suất cao nhất là từ tiếp theo: $w_t = \text{argmax}_w P(w|w_{1:t-1})$ tại mỗi timestep t .

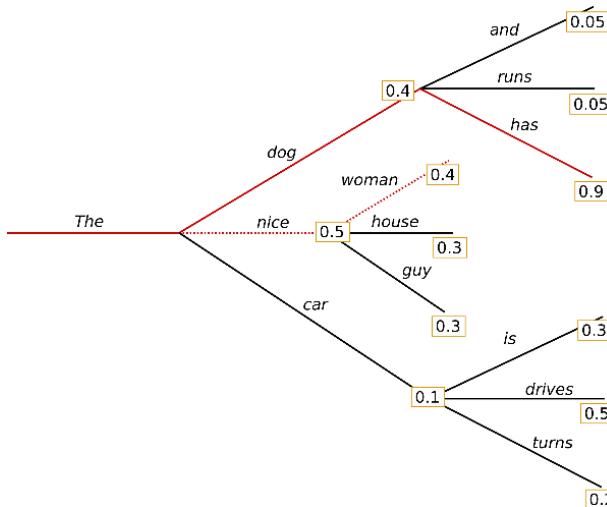


Hình 79. Mô tả quá trình sử dụng Greedy Search để sinh từ

Bắt đầu từ từ khóa "The", thuật toán chọn từ tiếp theo có xác suất cao nhất là "nice" và tiếp tục chọn những chữ tiếp theo có xác suất cao nhất cho đến khi đủ số từ. Do đó, như Hình 79, kết quả đầu ra là chuỗi từ "The", "nice", "woman" với xác suất tổng thể là $0.5 \times 0.4 = 0.2$.

10.2. Beam Search

Beam Search là phương pháp giảm thiểu nguy cơ bỏ lỡ các chuỗi từ có xác suất cao bằng cách duy trì `num_beams` ứng viên có xác suất cao nhất tại mỗi bước và chọn ứng viên cuối cùng có xác suất cao nhất. Hình 80 minh họa với `num_beams` = 2:



Hình 80. Mô tả quá trình sử dụng Beam Search để sinh từ

Tại timestep 1, ngoài giả thuyết có khả năng (“The”, “nice”), Beam Search cũng theo dõi giả thuyết có khả năng thứ hai (“The”, “dog”). Tại timestep 2, Beam Search tìm thấy chuỗi từ (“The”, “dog”, “has”) có xác suất 0.36 cao hơn so với (“The”, “nice”, “woman”), có xác suất 0.20.

Beam Search sẽ tìm kiếm đầu ra có xác suất cao hơn so với Greedy Search, nhưng không đảm bảo tìm thấy đầu ra có xác suất cao nhất. Beam Search sẽ giữ lại một số lượng giới hạn (`num_beams`) các đường đi ở mỗi bước để tìm kiếm, do đó có thể bỏ qua những đường đi không được lưu giữ trong số lượng này dù chúng có xác suất cao hơn. Tuy nhiên, số lượng đường đi giữ lại càng lớn thì Beam Search càng có khả năng tìm thấy đường đi tốt nhất. Nhưng, việc tăng `num_beams` cũng tăng đáng kể thời gian tính toán và bộ nhớ yêu cầu cho quá trình dự đoán. Do đó, việc lựa chọn `num_beams` cần cân nhắc giữa độ chính xác và hiệu suất tính toán.

11. Phương pháp đánh giá mô hình

11.1. Đánh giá mô hình phân lớp với Accuracy

Accuracy là một chỉ số đánh giá mức độ chính xác của mô hình phân lớp. Nó được tính bằng tỉ lệ phân loại đúng trên tổng số lượng mẫu trong tập dữ liệu.

$$\text{Accuracy} = \frac{\text{Số lượng dự đoán chính xác trên tất cả các lớp}}{\text{Tổng số lượng mẫu}}$$

Ví dụ, nếu mô hình phân lớp được huấn luyện trên tập dữ liệu gồm 1000 mẫu và dự đoán đúng 900 mẫu, thì giá trị accuracy của mô hình sẽ là 0.9 hoặc 90%.

11.2. Đánh giá mô hình sinh tự động câu trả lời với ROUGE [8]

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [8] là một tập các metric được sử dụng để đánh giá chất lượng của các bài tóm tắt hoặc các câu trả lời được sinh ra bởi các mô hình tự động như mô hình sinh câu trả lời tự động, các mô hình tóm tắt văn bản tự động... ROUGE đo lường độ giống nhau giữa một văn bản thực tế (reference) và một văn bản được sinh ra tự động (generated/hypothesis) bằng cách tính toán các chỉ số dựa trên các N-gram của các từ trong văn bản. ROUGE [8] được sử dụng rộng rãi trong nghiên cứu về xử lý ngôn ngữ tự nhiên, đặc biệt là trong lĩnh vực tóm tắt văn bản và trả lời câu hỏi dựa trên nội dung hình ảnh (VQA).

ROUGE-N: là một hệ thống đánh giá độ đo đánh giá độ tương đồng giữa hai đoạn văn bằng cách tính toán số lần xuất hiện của các cụm từ chính xác có độ dài N trong cả hai đoạn văn. Giá trị của ROUGE nằm trong khoảng từ 0 đến 1, và giá trị càng gần 1 thì câu trả lời tự động càng giống với câu trả lời tham chiếu được tạo ra bởi con người.

$$\text{ROUGE}_N = \frac{\sum_{\text{gram}_N \in \text{reference}} \text{Count}_{\text{match}}(\text{gram}_N)}{\sum_{\text{gram}_N \in \text{reference}} \text{Count}(\text{gram}_N)}$$

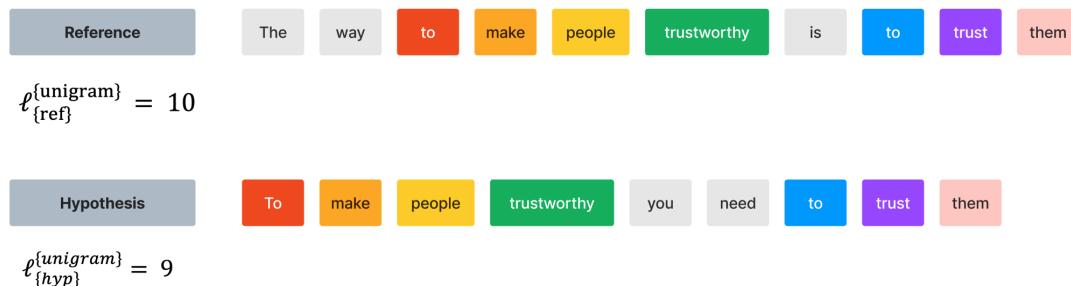
Một thẻ hiện của ROUGE-N là ROUGE-1 – tính độ chính xác của từ đơn hay unigram (1-gram) trong câu trả lời. Công thức tính:

$$\text{ROUGE}_1 = \frac{\sum_{\text{unigram} \in \text{reference}} \text{Count}_{\text{match}}(\text{unigram})}{\sum_{\text{unigram} \in \text{reference}} \text{Count}(\text{unigram})}$$

hay,

$$\text{ROUGE}_1 = \frac{\sum_{\text{unigram} \in \text{reference}} \text{Count}_{\text{match}}(\text{unigram})}{l_{\{\text{ref}\}}^{\{\text{unigram}\}}}$$

Ví dụ tính ROUGE-1:



Hình 81. Ví dụ về ROUGE-1

$$\text{Ta có, } \text{ROUGE}_1 = \frac{7}{10} = 0.7. \text{ (Hình 81)}$$

ROUGE-L hay ROUGE-LCS sử dụng chiều dài của dãy con chung dài nhất (Longest Common Subsequence – LCS) giữa hai đoạn văn bản để tính toán độ tương đồng. Với độ đo này, các từ có thể xuất hiện ở bất kỳ vị trí nào trong văn bản, không cần phải cùng liên tiếp nhau như trong ROUGE-N. ROUGE-LCS cho phép đánh giá độ chính xác của các từ trùng lặp và đánh giá các văn bản có cùng nội dung nhưng khác biệt về cú pháp.

Công thức:

$$\left\{ \begin{array}{l} R_{\text{LCS}} = \frac{\text{LCS}(\text{reference}, \text{hypothesis})}{l_{\{\text{ref}\}}^{\{\text{unigram}\}}} \\ P_{\text{LCS}} = \frac{\text{LCS}(\text{reference}, \text{hypothesis})}{l_{\{\text{hyp}\}}^{\{\text{unigram}\}}} \\ \text{ROUGE}_{\text{LCS}} = \frac{(1+\beta^2)R_{\text{LCS}}P_{\text{LCS}}}{R_{\text{LCS}}+\beta^2P_{\text{LCS}}} \end{array} \right.$$

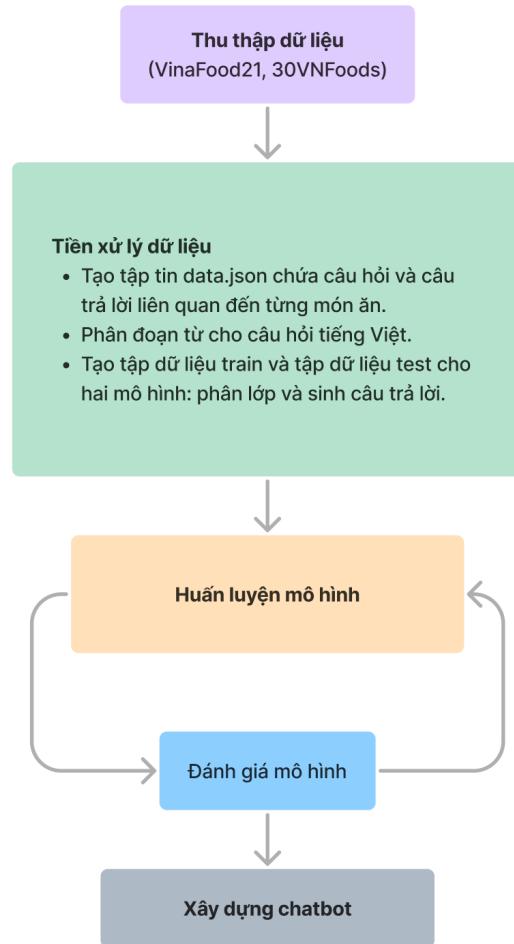
Ví dụ, theo Hình 81 ta có $\text{LCS}(\text{reference}, \text{hypothesis}) = 7$, do đó:

$$R_{\text{LCS}} = \frac{7}{10}, P_{\text{LCS}} = \frac{7}{9}, \text{ROUGE}_{\text{LCS}} = \frac{(1+\beta^2) \times 49}{70 + \beta^2 \times 63}$$

Trọng số $\beta = 1$, $\text{ROUGE}_{\text{LCS}} = \frac{98}{133} \approx 0.73684$.

Chương 2. Phương pháp thực hiện

Trong chương này, luận văn sẽ trình bày phương pháp thực hiện mô hình VQA và mô hình sinh câu trả lời với đầu vào là một hình ảnh và một câu hỏi, cũng như là các bước thực hiện đề tài. Quy trình thực hiện được trình bày như Hình 80.



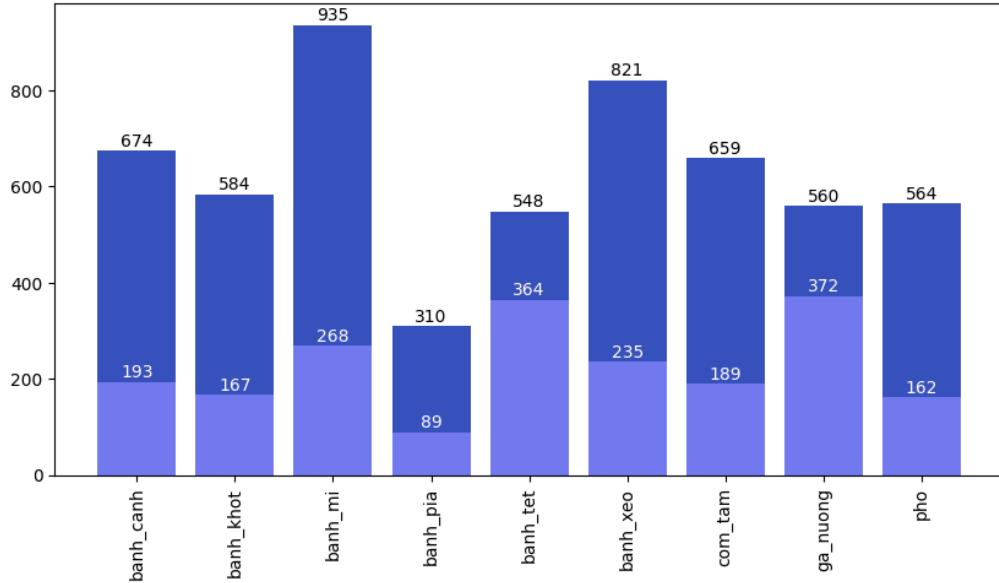
Các bước thực hiện ở Hình 82 sẽ được trình bày chi tiết ở các mục bên dưới.

1. Thu thập dữ liệu

Dữ liệu hình ảnh các món ăn là một phần của tập dữ liệu VinaFood21 [7] và 30VNFood [6] được các tác giả [7, 6] thu thập từ nhiều nguồn trên internet.

Dữ liệu thuộc 09 lớp: bánh canh, bánh khọt, bánh mì, bánh pía, bánh tét, bánh xèo, cơm tấm, gà nướng, phở. Tổng số hình ảnh trên tập dữ liệu train là 5655, tập dữ liệu test là 2039.

Số lượng hình ảnh trên từng lớp được thống kê như Hình 83.

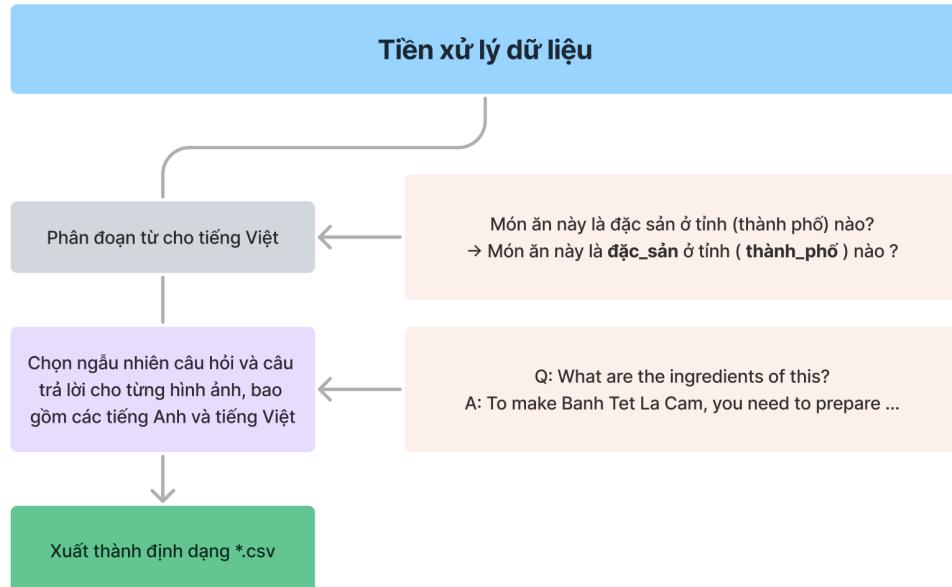


Hình 83. Biểu đồ thống kê số lượng hình của từng lớp trong tập dữ liệu.

2. Tiền xử lý dữ liệu

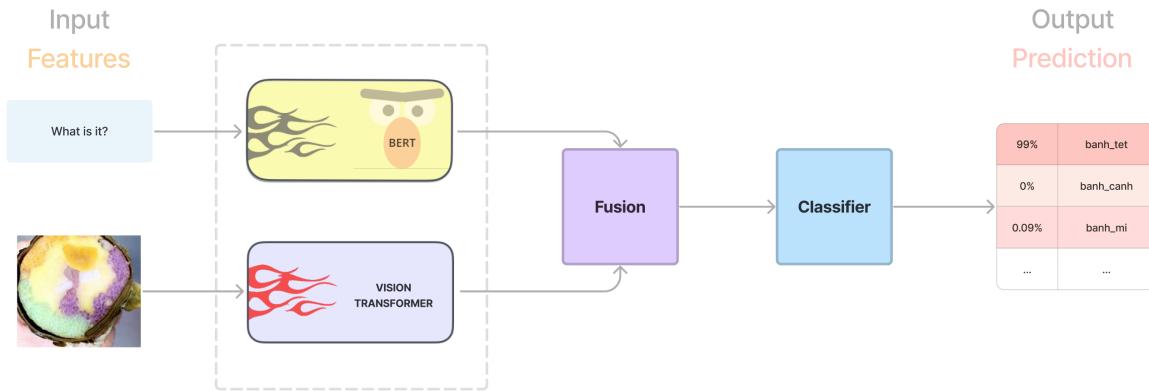
Tiền xử lý dữ liệu là bước quan trọng trong việc giải quyết bất kỳ bài toán nào trước khi đưa vào quá trình huấn luyện mô hình, nó ảnh hưởng trực tiếp đến kết quả huấn luyện mô hình.

Tiền xử lý dữ liệu gồm các bước được trình bày ở Hình 84: phân đoạn từ cho tiếng Việt sử dụng RDRSegmenter [21], chọn ngẫu nhiên câu hỏi và câu trả lời cho từng hình ảnh món ăn cho trong tập dữ liệu train và tập dữ liệu test. Cuối cùng xuất thành định dạng *.csv.



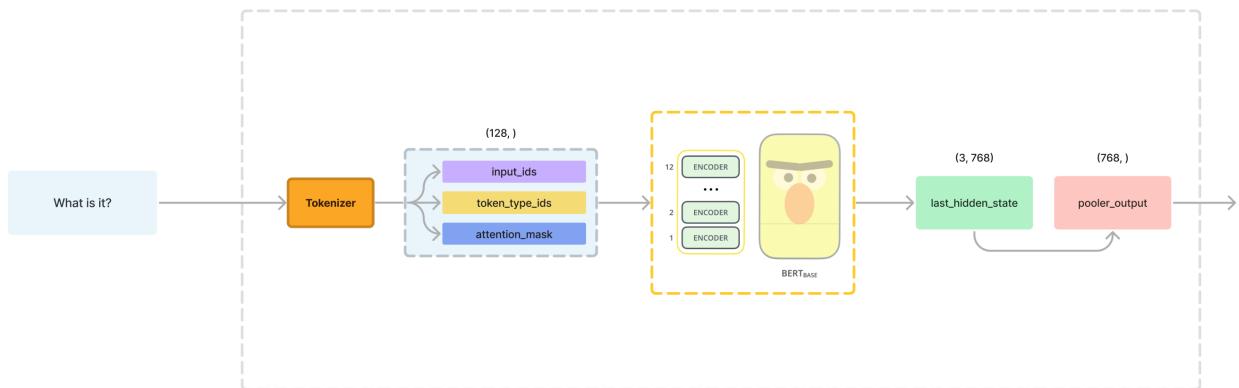
Hình 84. Quá trình tiền xử lý dữ liệu để tạo tập tin *.csv

3. Huấn luyện mô hình 1: VQA sử dụng ViT [1] và BERT [2] / PhoBERT [3]

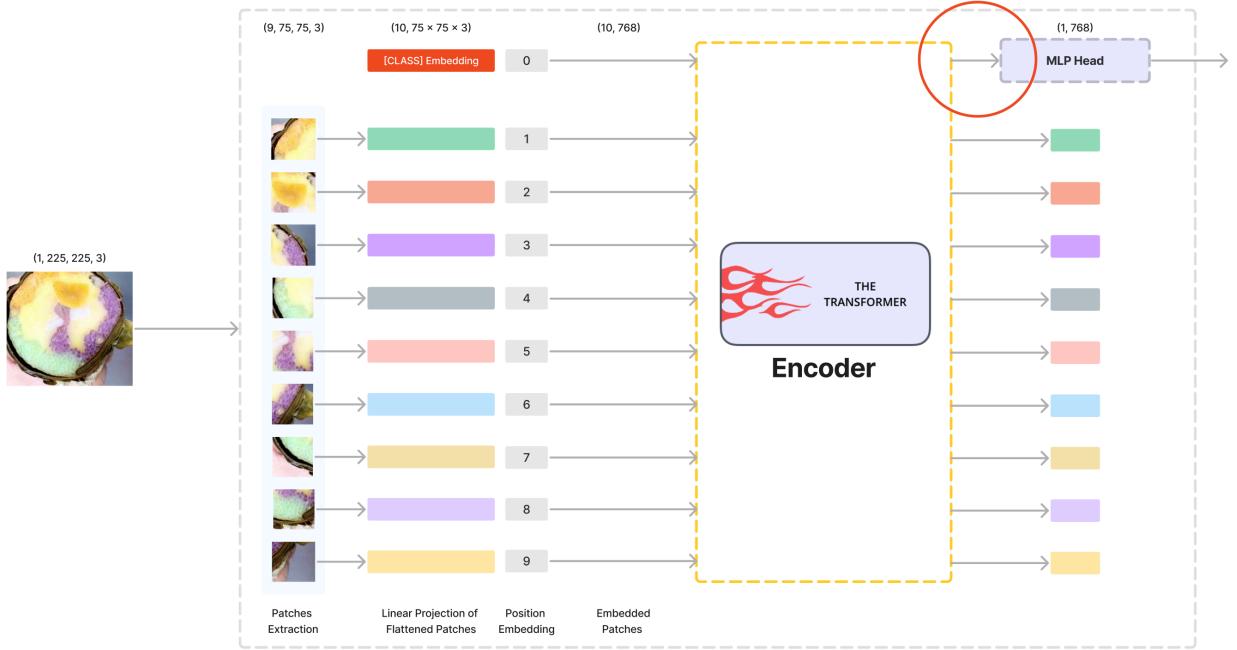


Hình 85. Kiến trúc mô hình sử dụng BERT và ViT để phân lớp

Ở Mô hình 1, được mô tả như Hình 85, tôi đã kết hợp ViT [1] và BERT [2] (hoặc PhoBERT [3] cho tiếng Việt) để tạo ra một mô hình VQA. Trong mô hình, tôi đã sử dụng BERT [2] (PhoBERT [3]) để trích xuất đặc trưng của câu hỏi và ViT [9] để trích xuất đặc trưng của hình ảnh món ăn.



Hình 86. pooler_output (khung màu hồng) chính là vector mang đặc trưng của toàn bộ dữ liệu đầu vào được dùng cho tác vụ phân lớp ở đầu ra của mô hình BERT [2]



Hình 87. Tương tự, ViT [1] cũng có pooler_output (hình tròn màu đỏ), được làm đầu vào cho MLP Head cho tác vụ phân lớp.

Như đã trình bày ở Chương 1, cả BERT [2] (hay PhoBERT [3]) và ViT [1] đều dựa trên kiến trúc của Transformer's Encoder [9], vì vậy, ở đầu ra của tầng Encoder thứ L đều sẽ bao gồm một vector mang đặc trưng đại diện cho toàn bộ dữ liệu đầu vào. Do đó, tôi đã tiến hành tạo ra một SLP (hàm kích hoạt ReLU) – gọi là Fusion để kết hợp hai vector đặc trưng này lại với nhau. Sau đó, đưa đầu ra của Fusion vào Classifier để tiến hành phân lớp.

Cụ thể,

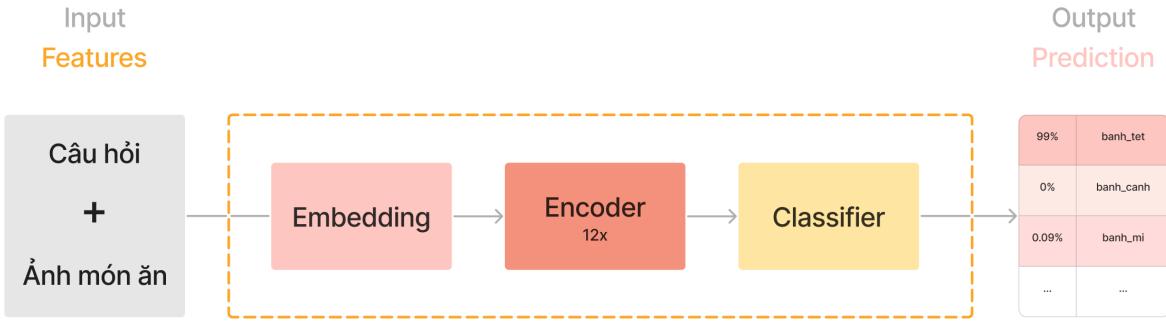
(1) Fusion là một SLP nhận vector đầu vào là một vector có kích thước `hidden_size` của BERT cộng với `hidden_size` của ViT (do tôi đã sử dụng phiên bản base cho cả hai mô hình, nên đầu vào sẽ có kích thước $768 + 768 = 1536$), và đầu ra là một vector có chiều `intermediate_dim = 512` (do tôi quy định).

(2) Classifier cũng là một SLP với kích thước vector đầu vào là `intermediate_dim = 512`, đầu ra là số lượng nhãn trong tập dữ liệu.

4. Huấn luyện mô hình 2: VQA sử dụng ViLT [4]

Với Mô hình 2, tôi đã sử dụng lại kiến trúc từ mô hình ViLT [4]. Sau đó, tinh chỉnh lại bộ Classifier (Hình 88, khung màu vàng) với số lượng lớp phù hợp.

Về cơ bản, mô hình ViLT [4] có kiến trúc như Hình 88.

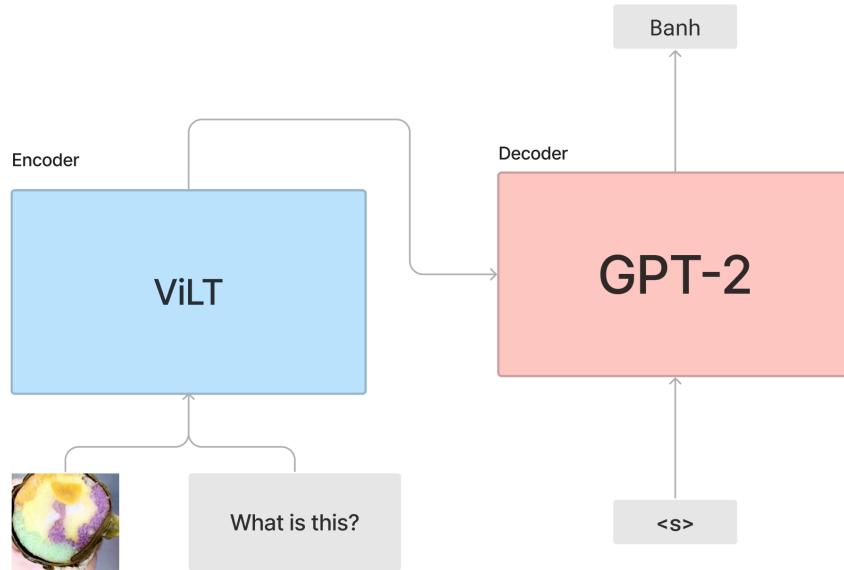


Hình 88. Mô tả cơ bản về kiến trúc của mô hình ViLT [4]

Cụ thể,

- (1) Embedding: bao gồm Text Embedding và Patch Embedding.
- (2) Encoder: là 12 khối Transformer's Encoder [9].
- (3) Classifier: MLP với hai tầng kết nối đầy đủ.

5. Huấn luyện mô hình 3: Sinh câu trả lời với ViLT [4] và GPT-2 [5]



Hình 89. Mô tả cơ bản về kiến trúc của Mô hình 3

Mô hình 3 được dựa trên kiến trúc mô hình Vision Encoder – Decoder được tích hợp trong thư viện Huggingface Transformers¹⁰, cho phép sử dụng một mô hình Transformer-based Vision bất kỳ làm Encoder. Tương tự, với Decoder sẽ là một mô hình ngôn ngữ (language model).

¹⁰ https://huggingface.co/docs/transformers/model_doc/vision-encoder-decoder

6. Đánh giá mô hình

Sử dụng chỉ số Accuracy để đánh giá việc phân lớp cho mô hình VQA và phương pháp đánh giá ROUGE [8] để đánh giá kết quả mô hình sinh câu trả lời.

Sau khi huấn luyện và điều chỉnh mô hình để đạt được mức độ ổn định. Tiến hành đánh giá kiểm tra mô hình thông qua các chỉ số Accuracy (với mô hình 1 và 2), ROUGE-1, ROUGE-2, ROUGE-L (với mô hình 3).

7. Xây dựng ứng dụng chatbot đơn giản

Để xây dựng một ứng dụng chatbot đơn giản, tôi đã sử dụng một số framework và thư viện như sau:

(1) Phía server (Server-side): Sử dụng Flask¹¹ để tạo ra API có đầu vào (Input) là một cặp (pair) dữ liệu: (hình ảnh món ăn, câu hỏi liên quan đến món ăn). Dữ liệu đầu ra (Output) là câu trả lời cho câu hỏi liên quan đến món ăn.

(2) Phía client (Client-side): Sử dụng React Native¹² kết hợp với thư viện Flyer Chat¹³ để tạo ra một ứng dụng trên điện thoại di động có giao diện chính là một khung chat, và có khả năng tương tác được với Server-side thông qua API được Server-side cung cấp.

¹¹ <https://flask.palletsprojects.com/en/2.3.x/quickstart>

¹² <https://reactnative.dev>

¹³ <https://docs.flyer.chat/react-native/chat-ui>

Chương 3. Thực nghiệm

Ở chương này, tôi tiến hành thực nghiệm lại các bước đã đề cập ở chương 2 và thảo luận kết quả đạt được trong quá trình thực nghiệm.

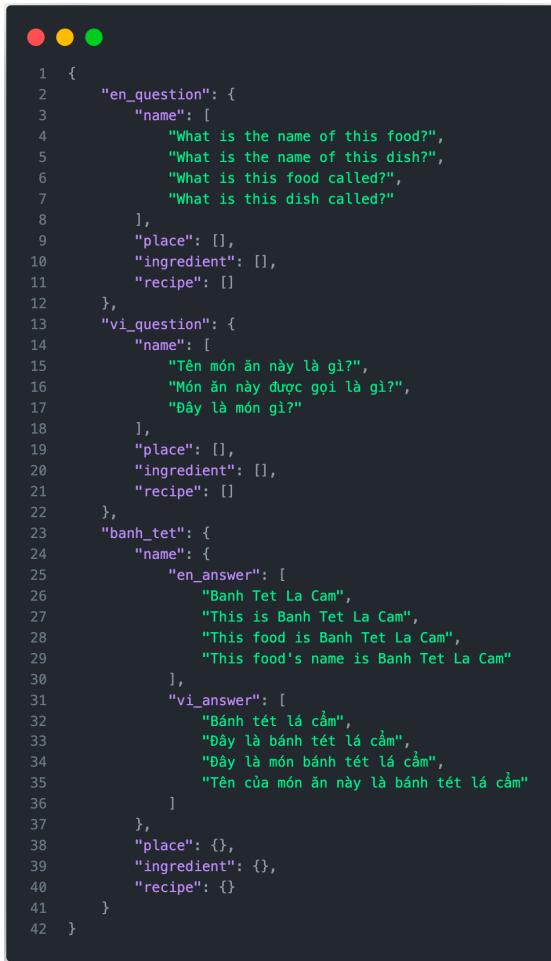
1. Xây dựng tập dữ liệu

Nhằm kiểm định độ chính xác của mô hình, ở đề tài này, tôi tiến hành thực nghiệm trên cả tiếng Anh và tiếng Việt – đối với mô hình phân lớp.

Tôi sử dụng một phần hình ảnh các món ăn của tập dữ liệu VinaFood21 [7] và 30VNFoods [6] để tiến hành xử lý dữ liệu.

Xử lý dữ liệu

Bước 1. Tạo tập tin *data.json* bao gồm các câu hỏi và câu trả lời liên quan đến từng món ăn.



```
1  {
2      "en_question": {
3          "name": [
4              "What is the name of this food?",
5              "What is the name of this dish?",
6              "What is this food called?",
7              "What is this dish called?"
8          ],
9          "place": [],
10         "ingredient": [],
11         "recipe": []
12     },
13    "vi_question": {
14        "name": [
15            "Tên món ăn này là gì?",
16            "Món ăn này được gọi là gì?",
17            "Đây là món gì?"
18        ],
19        "place": [],
20        "ingredient": [],
21        "recipe": []
22    },
23    "banh_tet": {
24        "name": [
25            "en_answer": [
26                "Banh Tet La Cam",
27                "This is Banh Tet La Cam",
28                "This food is Banh Tet La Cam",
29                "This food's name is Banh Tet La Cam"
30            ],
31            "vi_answer": [
32                "Bánh tết lá cẩm",
33                "Đây là bánh tết lá cẩm",
34                "Đây là món bánh tết lá cẩm",
35                "Tên của món ăn này là bánh tết lá cẩm"
36            ]
37        },
38        "place": {},
39        "ingredient": {},
40        "recipe": {}
41    }
42 }
```

Hình 90. Một phần nội dung của tập tin *data.json*

Bước 2. Với mỗi hình ảnh thuộc từng món ăn, lựa chọn ngẫu nhiên các câu hỏi và câu trả lời tương ứng để tạo thành một mẫu dữ liệu.

Sau quá trình xử lý dữ liệu, tổng số mẫu trong tập dữ liệu để phân lớp: 23013 mẫu (tập train là 16915 mẫu, tập test là 6098 mẫu); đối với tập dữ liệu để sinh tự động câu trả lời: 23064 mẫu (tập train là 16949 mẫu, tập test là 6115 mẫu). Hình 91 là một phần của các mẫu trong tập dữ liệu train của tập dữ liệu sinh tự động câu trả lời.

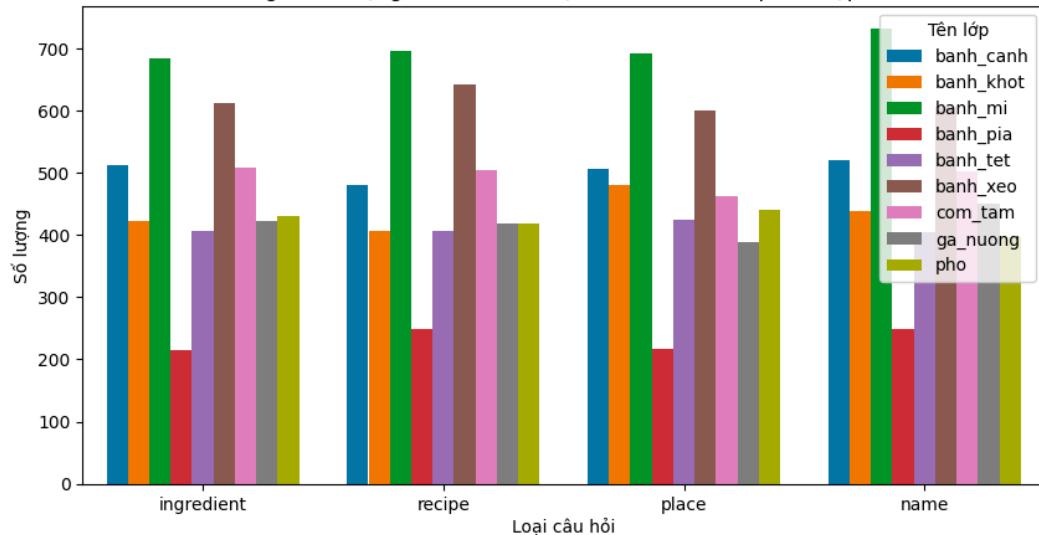
	image_id	question_vi	label_vi	question_en	label_en	class_name	question_type
0	train/banh_canh/823.jpg	Nguyên_lieu của món ăn này là gì ?	Nguyên_lieu_chính để làm bánh_canh gồm bột gạo...	What is this food made of?	The main ingredients to make Banh Canh include...	banh_canh	ingredient
1	train/banh_canh/189.jpg	Cách chế_biéin món ăn này ?	Để làm món bánh_canh , đầu_tiên người ta phải ...	How do you cook this?	To make Banh Canh, first, people have to prepa...	banh_canh	recipe
2	train/banh_canh/77.jpg	Món ăn này làm từ những gì ?	Nguyên_lieu_chính để làm bánh_canh gồm bột gạo...	What are the ingredients of this meal?	The main ingredients to make Banh Canh include...	banh_canh	ingredient
3	train/banh_canh/837.jpg	Món ăn này là đặc_sản ở đâu ?	Món ăn này nổi_tiếng ở miền Tây , đặc_biéit là ...	Where can I get specialty meal?	It is popular in the Mekong Delta, especially ...	banh_canh	place
4	train/banh_canh/638.jpg	Món ăn này nổi_tiếng ở đâu ?	Bánh_canh cũng là món ăn phô_biéin ở miền Tây ...	What is this food's origin?	This is popular in the Mekong Delta, especial...	banh_canh	place
5	train/banh_canh/162.jpg	Cách chế_biéin món ăn này ?	Bánh_canh là món ăn phô_biéin và rất được ua_ch...	How do you make this?	Banh Canh is a popular and very popular dish i...	banh_canh	recipe
6	train/banh_canh/176.jpg	Món ăn này là từ tỉnh (thành_phố) nào ?	Bánh_canh cũng là món ăn phô_biéin ở miền Tây ...	What province (city) is this food from?	This is popular in the Mekong Delta, especial...	banh_canh	place
7	train/banh_canh/88.jpg	Đây là món ăn gì ?	Món ăn này được gọi là bánh_canh	What do you call this dish?	Banh Canh	banh_canh	name
8	train/banh_canh/348.jpg	Đây là món ăn gì ?	Món ăn này được gọi là bánh_canh	What do you call this meal?	This food is called Banh Canh	banh_canh	name
9	train/banh_canh/360.jpg	Tên của đồ_ăn này là gì ?	Món ăn này được gọi là bánh_canh	What is this?	This food is called Banh Canh	banh_canh	name

Hình 91. Một phần các mẫu trong tập dữ liệu train của tập dữ liệu sinh tự động câu trả lời

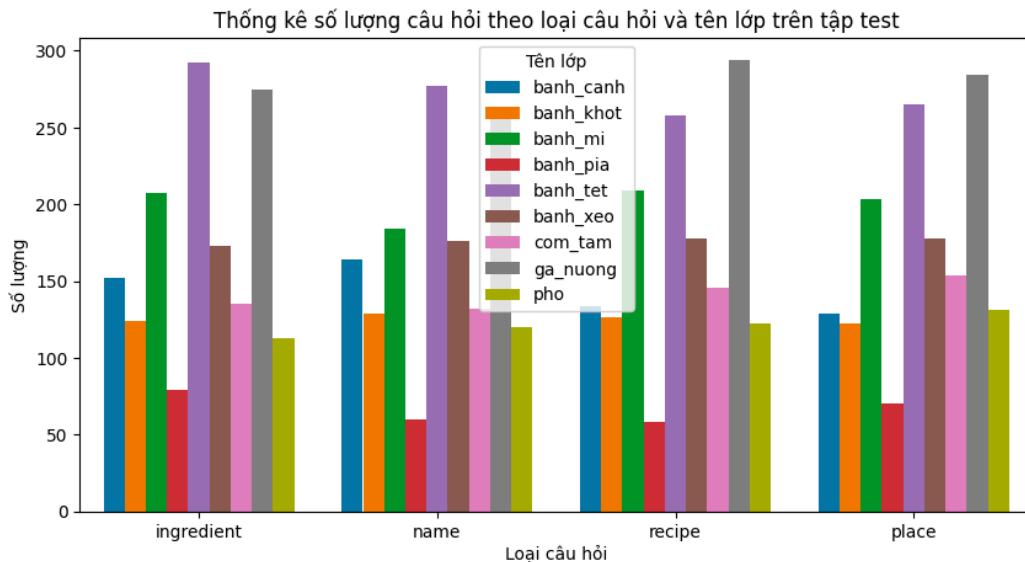
Dữ liệu được sinh ra đảm bảo sự cân bằng giữa các loại câu hỏi của mỗi lớp.

Ví dụ, lớp bánh canh có xấp xỉ 500 mẫu dữ liệu liên quan đến câu hỏi về nguyên liệu thì cũng phải có xấp xỉ 500 mẫu dữ liệu trong tập dữ liệu liên quan đến câu hỏi về cách chế biến, tên món ăn, địa phương. Hình 92 và Hình 93 thống kê số lượng câu hỏi theo loại câu hỏi trên từng lớp.

Thống kê số lượng câu hỏi theo loại câu hỏi và tên lớp trên tập train



Hình 92. Thống kê số lượng câu hỏi theo lớp trên tập dữ liệu train



Hình 93. Thống kê số lượng câu hỏi theo lớp trên tập dữ liệu test

2. Huấn luyện mô hình

Quá trình huấn luyện mô hình, tôi sử dụng Pytorch kết hợp với thư viện Huggingface Transformers nhằm tận dụng các mô hình pre-trained có sẵn trên nền tảng này.

Môi trường huấn luyện: tôi sử dụng Google Colab phiên bản PRO để huấn luyện cả ba mô hình. Với mô hình 1 và 2 được huấn luyện trên GPU V100, và GPU A100 với mô hình 3.

Xử lý dữ liệu đầu vào: sau khi tạo ra các tệp tin *.csv (mục 1), tôi sử dụng thư viện Huggingface Datasets¹⁴ để tiến hành đọc dữ liệu từ tệp tin *.csv thành DatasetDict¹⁵. Tiếp theo, để mô hình hiểu được đầu vào, chúng ta cần chuyển đổi các văn bản và hình ảnh sang các vector số hay ma trận số tương ứng. Do đó, tôi đã sử dụng các Tokenizer¹⁶ để chuyển đổi văn bản thành các vector số (Hình 94), và sử dụng thư viện PIL.Image¹⁷ để đọc hình ảnh thành ma trận số, sau đó sử dụng các Image-Processor¹⁸ để chuẩn hoá ma trận số (Hình 95).

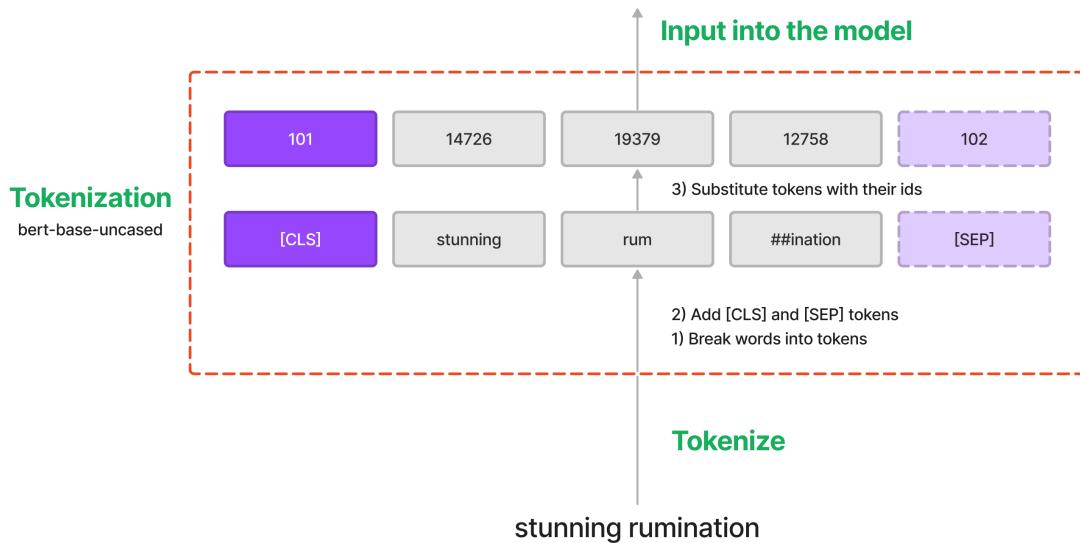
¹⁴ <https://huggingface.co/docs/datasets/quickstart>

¹⁵ https://huggingface.co/docs/datasets/package_reference/main_classes#datasets.DatasetDict

¹⁶ https://huggingface.co/docs/transformers/main_classes/tokenizer

¹⁷ <https://pillow.readthedocs.io/en/stable/reference/Image.html>

¹⁸ https://huggingface.co/docs/transformers/main_classes/image_processor



Hình 94. Sử dụng Tokenizer để chuyển đổi câu thành vector số



Hình 95. Sử dụng Image-Processor để chuẩn hoá ma trận số từ hình ảnh

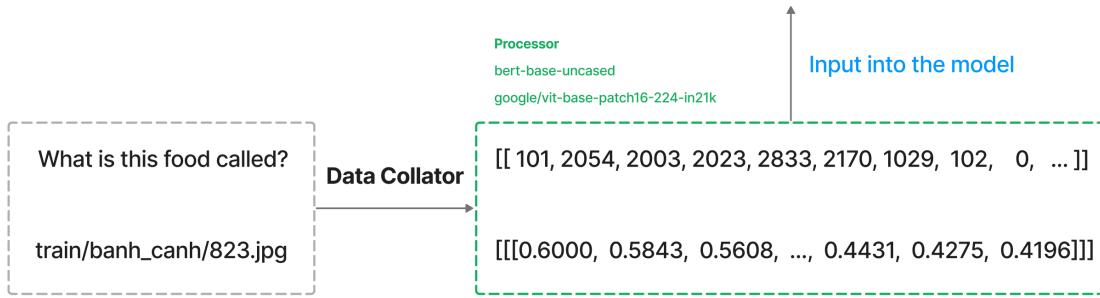
Mỗi mô hình đều sẽ có Tokenizer hay Image-Processor tương ứng. Chẳng hạn, BERT_{BASE} (bản uncased) sẽ sử dụng Tokenizer “bert-base-uncased¹⁹” hay ViLT [4] gộp Tokenizer và Image-Processor thành Processor²⁰, trong đó sẽ sử dụng Tokenizer của BERT [2] và Image-Processor của ViT [1].

Trong quá trình huấn luyện, để không phải chỉnh sửa các tập tin *.csv hay DatasetDict, tôi đã sử dụng Data Collator²¹ để tiền xử lý đầu vào trước khi được đưa vào mô hình (Hình 96). Riêng đối với Mô hình 3, tôi đã áp dụng Tokenizer và Image-Processor trên tập dữ liệu của Mô hình 3 lưu vào Google Drive dưới dạng DatasetDict trước khi tiến hành huấn luyện, do Mô hình 3 quá phức tạp nên không thể vừa huấn luyện vừa xử lý dữ liệu.

¹⁹ <https://huggingface.co/bert-base-uncased>

²⁰ https://huggingface.co/docs/transformers/main_classes/processors

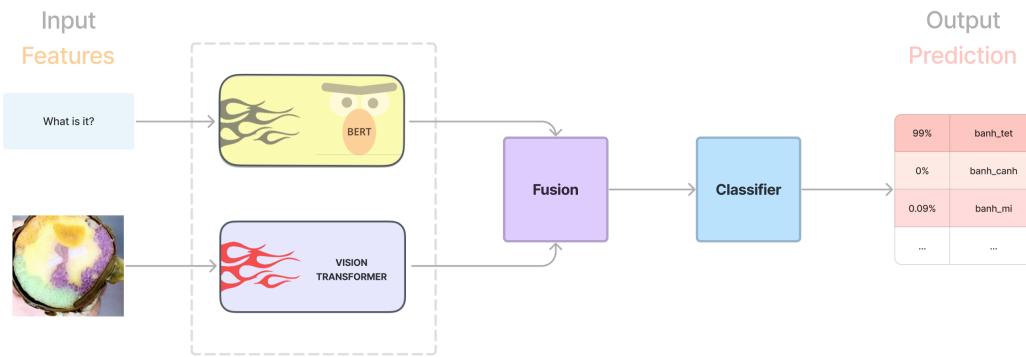
²¹ https://huggingface.co/docs/transformers/main_classes/data_collator



Hình 96. Mô tả cách hoạt động của Data Collator

Loss function được sử dụng để huấn luyện là `torch.nn.CrossEntropyLoss()`²². Optimizer là `transformers.AdamW`²³.

2.1. Mô hình 1: VQA sử dụng ViT [1] và BERT [2] / PhoBERT [3]



Hình 97. Nhắc lại kiến trúc của Mô hình 1

Đối với Mô hình 1, tôi đã sử dụng pre-trained “google/vit-base-patch16-224-in21k²⁴”, đối với mô hình ViT [1] và pre-trained “bert-base-uncased²⁵” cho mô hình BERT [2] để huấn luyện mô hình VQA (Hình 97) trên tập dữ liệu phân lớp dành cho tiếng Anh. Song song đó, tôi cũng đã sử dụng pre-trained “vinai/phobert-base-v2²⁶” cho mô hình PhoBERT [3] cũng kết hợp với ViT [1] huấn luyện mô hình VQA (có cùng kiến trúc như Hình 97 chỉ thay đổi BERT [2] thành PhoBERT [3]) trên tập dữ liệu phân lớp dành cho tiếng Việt.

²² <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>

²³ https://huggingface.co/docs/transformers/main_classes/optimizer_schedules#transformers.AdamW

²⁴ <https://huggingface.co/google/vit-base-patch16-224-in21k>

²⁵ <https://huggingface.co/bert-base-uncased>

²⁶ <https://huggingface.co/vinai/phobert-base-v2>

2.2. Mô hình 2: VQA sử dụng ViLT [4]

Mô hình 2 là mô hình ViLT [4] được tinh chỉnh lại bộ Classifier để phân lớp phù hợp với tập dữ liệu.

Trong quá trình huấn luyện, tôi đã sử dụng pre-trained “dandelin/vilt-b32-finetuned-vqa²⁷” để huấn luyện tiếp trên tập dữ liệu của mình.

2.3. Mô hình 3: Sinh câu trả lời với ViLT [4] và GPT-2 [5]

Để huấn luyện Mô hình 3, tôi đã tuỳ chỉnh lại lớp Vision Encoder-Decoder Model trong thư viện Huggingface Transformers để phù hợp với dữ liệu đầu vào bao gồm cả hình ảnh và văn bản, thay vì đầu vào chỉ có hình ảnh – mặc định của lớp Vision Encoder-Decoder Model.

Vision Encoder-Decoder Model là một mô hình Seq2Seq [17], do đó tôi đã sử dụng pre-trained “dandelin/vilt-b32-finetuned-vqa” của mô hình ViLT [4] cho Encoder và pre-trained “gpt2²⁸” của mô hình GPT-2 [5] cho Decoder.

3. Kết quả đánh giá sau khi chạy mô hình

3.1. Mô hình 1: VQA sử dụng ViT [1] và BERT [2] / PhoBERT [3]

Mô hình	Ngôn ngữ	Step	Train Loss	Test Loss	Accuracy
ViT và BERT	English	600	0.143200	0.232924	0.942276
ViT và PhoBERT	Tiếng Việt	600	0.164800	0.216926	0.948180

Bảng 1. Kết quả huấn luyện của Mô hình 1: VQA sử dụng ViT [1] và BERT [2] / PhoBERT [3]

Kết quả đánh giá độ chính xác của Mô hình 1 dựa trên chỉ số Accuracy sau khi huấn luyện được thể hiện ở Bảng 1.

Trong đó:

- Train Loss: giá trị của hàm mất mát trên tập dữ liệu train;
- Test Loss: giá trị của hàm mất mát trên tập dữ liệu test;
- Accuracy: giá trị Accuracy của Mô hình 1 trên tập dữ liệu test.

²⁷ <https://huggingface.co/dandelin/vilt-b32-finetuned-vqa>

²⁸ <https://huggingface.co/gpt2>

3.2. Mô hình 2: VQA sử dụng ViLT [4]

Mô hình	Ngôn ngữ	Step	Train Loss	Test Loss	Accuracy
ViLT	English	1200	0.014400	0.311519	0.928009

Bảng 2. Kết quả huấn luyện của Mô hình 2: VQA sử dụng ViLT [4]

Kết quả đánh giá độ chính xác của Mô hình 2 dựa trên chỉ số Accuracy sau khi huấn luyện được thể hiện ở Bảng 2.

Trong đó:

- Train Loss: giá trị của hàm mất mát trên tập dữ liệu train;
- Test Loss: giá trị của hàm mất mát trên tập dữ liệu test;
- Accuracy: giá trị Accuracy của Mô hình 2 trên tập dữ liệu test.

3.3. Mô hình 3: Sinh câu trả lời với ViLT [4] và GPT-2 [5]

Mô hình	Ngôn ngữ	Step	ROUGE-1	ROUGE-2	ROUGE-L
ViLT và GPT-2	English	1500	49.921200	39.263800	47.534200

Bảng 3. Kết quả huấn luyện của Mô hình 2: Sinh câu trả lời với ViLT [4] và GPT-2 [5]

Kết quả đánh giá độ chính xác của Mô hình 3 dựa trên phương pháp ROUGE [8] sau khi huấn luyện được thể hiện ở Bảng 3.

Trong đó:

- ROUGE-1: tỉ lệ khớp với 1-gram (unigram);
- ROUGE-2: tỉ lệ khớp với 2-gram (bigram);
- ROUGE-L: tỉ lệ khớp với Longest Common Subsequence (LCS).

4. Xây dựng ứng dụng chatbot đơn giản

Path of an image.

/Users/tqkhang/Desktop/LuanVan/datasets/processed/train/banh_xeo/1.jpg

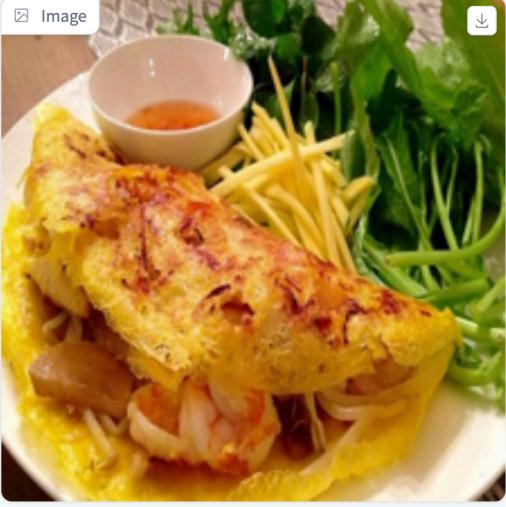
Question

What is this?

Language

en vi

Clear **Submit**

Image 

Answer

This food's name is Banh Xeo

Flag

Examples

Path of an image.	Question	Language
/Users/tqkhang/Desktop/LuanVan/datasets/processed/train/banh_xeo/1.jpg	What is this?	en
/Users/tqkhang/Desktop/LuanVan/datasets/processed/train/banh_xeo/1.jpg	Món ăn này là đặc sản ở tỉnh (thành phố) nào?	vi

Use via API 🔍 · Built with Gradio 🚀

Hình 98. Sử dụng Gradio tạo giao diện đơn giản tương tác với mô hình

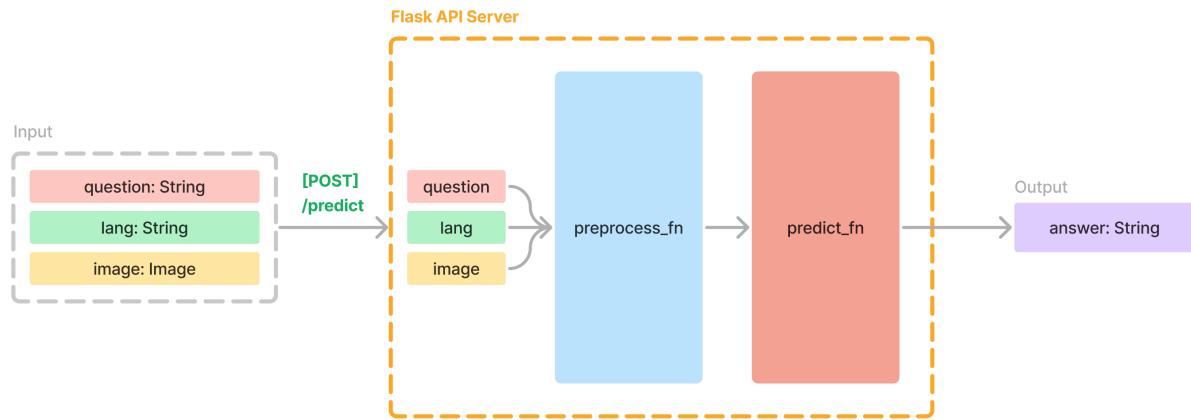
Hình 98, sử dụng Gradio²⁹ để tạo một giao diện đơn giản tương tác với mô hình trong quá trình kiểm thử. Mặc dù, Gradio đã tích hợp sẵn FastAPI³⁰ để cung cấp một API cho phép bên thứ ba tương tác với mô hình, tuy nhiên, để thuận tiện cho quá trình mở rộng hơn sau này, tôi đã chọn sử dụng Flask để tự xây dựng một server. Chi tiết hơn sẽ được trình bày ở phần tiếp theo.

²⁹ <https://gradio.app>

³⁰ <https://fastapi.tiangolo.com>

Phía server (Server-side)

Phía server, tôi đã sử dụng Flask để xây dựng một server cung cấp API /predict nhận dữ liệu đầu vào từ client, sau đó trả về câu trả lời phù hợp.



Hình 99. Quy trình tiếp nhận và xử lý dữ liệu trong server

Hình 99 mô tả các công đoạn được xử lý trong server, cụ thể:

- ① Nhận đầu vào từ client thông qua phương thức POST. Dữ liệu đầu vào bao gồm ba phần: câu hỏi về món ăn (ví dụ: “Đây là món gì?”); ngôn ngữ: “en” hoặc “vi”; hình ảnh món ăn.
- ② Hàm preprocess_fn: có nhiệm vụ chuyển dữ liệu đầu vào ở ① sang chuẩn đầu vào của mô hình.
- ③ Hàm predict_fn:

- Mô hình phân lớp: Nhận đầu vào từ ②, sau đó chọn ra lớp có xác suất cao nhất. Cuối cùng, lựa chọn ngẫu nhiên một đáp án trong tệp tin *data.json* để làm câu trả lời.
- Mô hình sinh câu trả lời: Nhận đầu vào từ ②, sau đó sinh ra câu trả lời sử dụng thuật toán Greedy Search hay Beam Search. Ở đây tôi đã sử dụng phương thức `.generate()`³¹ để sinh ra câu trả lời.

³¹ https://huggingface.co/docs/transformers/main_classes/text_generation

Phía client (Client-side)

Còn đối với phần client, tôi đã sử dụng React Native kết hợp với một số thư viện khác để tạo nên một ứng dụng chatbot trên di động có giao diện được trình bày ở các Hình 100, Hình 101 và Hình 102.

Để client có thể tương tác được với server, tôi đã sử dụng Ngrok³² để tạo đường dẫn chuẩn https công khai được trỏ đến server.

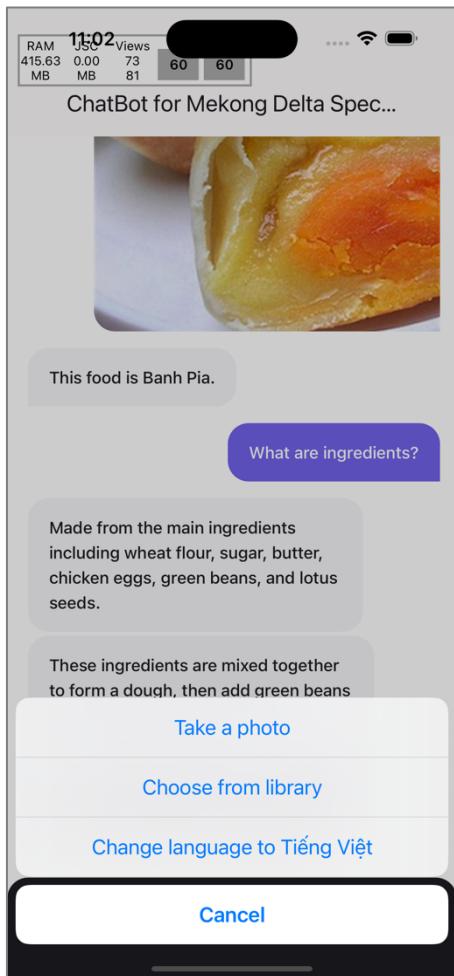


Hình 100. Giao diện chính của ứng dụng chatbot (Ảnh từ iPhone 14 Pro Max ảo)

Hình 100, là giao diện chính của ứng dụng, tại đây người sử dụng sẽ chọn hình ảnh món ăn từ thư viện (hoặc chụp ảnh món ăn) bằng cách nhấn vào nút bên trái khung nhập câu hỏi (Hình 101), sau khi người sử dụng đã chọn hình ảnh ứng dụng sẽ yêu cầu server trả về tên gọi của món ăn mà không cần người dùng đặt câu hỏi.

³² <https://ngrok.com>

Bên cạnh tiếng Anh, ứng dụng cũng hỗ trợ tiếng Việt (Hình 102), tuy nhiên mô hình trên tiếng Việt chỉ là mô hình phân lớp, không phải mô hình sinh câu trả lời như tiếng Anh.



Hình 101. Chuyển đổi ngôn ngữ sang tiếng Việt
(Ảnh từ iPhone 14 Pro Max ảo)



Hình 102. Ứng dụng chatbot hỗ trợ tiếng Việt
(Ảnh từ iPhone 12 thật)

C. PHẦN KẾT LUẬN

1. Kết quả đạt được

Mô hình VQA kết hợp ViT [1] và BERT [2] / PhoBERT [3] cho mục đích phân lớp đạt kết quả khá tốt (Accuracy đạt 94% với mô hình ViT và BERT trên tiếng Anh và gần 95% với mô hình ViT và PhoBERT trên tiếng Việt). Bên cạnh đó, mô hình VQA dựa trên ViLT trên tiếng Anh cũng đạt Accuracy xấp xỉ 93%.

Đối với mô hình sinh câu trả lời dựa trên ViLT và GPT-2 đạt kết quả dựa trên phương pháp đánh giá ROUGE, lần lượt là 49.92, 39.26, 47.53 tương ứng với ROUGE-1, ROUGE-2 và ROUGE-L.

Sau quá trình nghiên cứu và thực hiện đề tài, bài luận văn đã được một số kết quả như sau:

- Luận văn đã trình bày được các kiến thức liên quan để xây dựng các mô hình VQA kết hợp ViT [1] và BERT [2] hay PhoBERT [3]. Mô hình sinh câu trả lời với ViLT và GPT-2. Áp dụng các mô hình đã huấn luyện được, đặc biệt là mô hình sinh câu trả lời sử dụng ViLT [4] và GPT-2 [5] vào một ứng dụng chatbot đơn giản trên điện thoại di động.
- Ngoài ra, luận văn còn trình bày các bước để xây dựng tập dữ liệu, tiền xử lý tập dữ liệu, các phương pháp biểu diễn văn bản, và các phương pháp đánh giá mô hình.

2. Thảo luận và hướng phát triển

Bài toán trả lời câu hỏi từ hình ảnh (Visual Question Answering - VQA) là một chủ đề tuy không mới trong những năm gần đây, tuy nhiên nó vẫn được tiếp tục nghiên cứu nhiều trên thế giới. Việc sử dụng các mô hình Transformer-based như ViT, BERT, PhoBERT, VILT, GPT-2 vào bài toán VQA trên nhiều ngôn ngữ khác nhau đạt được nhiều kết quả khả quan.

Để cải thiện kết quả các mô hình được trình bày trong bài luận văn, chúng ta cần chuẩn bị một tập dữ liệu lớn hơn. Bên cạnh đó, trong tương lai, tôi sẽ phát triển thêm một số mô hình sinh câu trả lời trên tiếng Việt, chẳng hạn, sử dụng VILT với việc tùy chỉnh Tokenizer mặc định từ BERT sang PhoBERT, giữ nguyên Image-Processor là ViT cho phần Encoder, với Decoder tôi sẽ sử dụng pre-trained “NlpHUST/gpt2-vietnamese³³”, GPT-2 dành cho tiếng Việt.

³³ <https://huggingface.co/NlpHUST/gpt2-vietnamese>

TÀI LIỆU THAM KHẢO

- [1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," 2020.
- [2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2018.
- [3] Dat Quoc Nguyen and Anh Tuan Nguyen, "PhoBERT: Pre-trained language models for Vietnamese," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1037-1042.
- [4] Kim, Wonjae and Son, Bokyung and Kim, Ildoo, "ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision," in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, 2021, pp. 5583-5594.
- [5] Radford, Alec and Wu, Jeff and Child, Rewon and Luan, David and Amodei, Dario and Sutskever, Ilya, "Language Models are Unsupervised Multitask Learners," 2019.
- [6] Do, Trong-Hop and Nguyen, Duc-Duy-Anh and Dang, Hoang-Quan and Nguyen, Hoang-Nhan and Pham, Phu-Phuoc and Nguyen, Duc-Tri, "30VNFood: A Dataset for Vietnamese Foods Recognition," in *2021 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, 2021, pp. 311-315.
- [7] Thuan Trong Nguyen, Thuan Q. Nguyen, Dung Vo, Vi Nguyen, Ngoc Ho, Nguyen D. Vo, Kiet Van Nguyen, Khang Nguyen, "VinaFood21: A Novel Dataset for Evaluating Vietnamese Food Recognition," in *In Proceedings of the 15th IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, 2021.
- [8] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Text Summarization Branches Out*, Barcelona, Spain, Association for Computational Linguistics, 2004, p. 74–81.
- [9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin, "Attention Is All You Need," 2017.
- [10] Junnan Li, Dongxu Li, Caiming Xiong, Steven Hoi, "BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation," 2022.

- [11] Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, Lijuan Wang, "GIT: A Generative Image-to-text Transformer for Vision and Language," 2022.
- [12] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, "Efficient Estimation of Word Representations in Vector Space," 2013.
- [13] Rico Sennrich, Barry Haddow, Alexandra Birch, "Neural Machine Translation of Rare Words with Subword Units," 2015.
- [14] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," 2019.
- [15] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, H, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," 2016.
- [16] Schuster, Mike and Nakajima, Kaisuke, "Japanese and Korean voice search," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5149-5152.
- [17] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, "Sequence to Sequence Learning with Neural Networks," 2014.
- [18] Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton, "Layer Normalization," 2016.
- [19] Dan Hendrycks, Kevin Gimpel, "Gaussian Error Linear Units (GELUs)," 2016.
- [20] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, Noam Shazeer, "Generating Wikipedia by Summarizing Long Sequences," 2018.
- [21] Dat Quoc Nguyen and Dai Quoc Nguyen and Thanh Vu and Mark Dras and Mark Johnson, "A Fast and Accurate Vietnamese Word Segmenter," in *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*, 2018, pp. 2582-2587.
- [22] J. Kim. [Online]. Available: <https://gaussian37.github.io>.
- [23] J. Alammar. [Online]. Available: <http://jalammar.github.io>.
- [24] A. Kumar. [Online]. Available: <https://ankur3107.github.io>.

PHỤ LỤC

Nội dung tập tin data.json

```
{  
    "en_question": {  
        "name": [  
            "What is the name of this food?",  
            "What is the name of this dish?",  
            "What is the name of this meal?",  
            "What is this food called?",  
            "What is this dish called?",  
            "What is this meal called?",  
            "What do you call this food?",  
            "What do you call this dish?",  
            "What do you call this meal?",  
            "What is this food's name?",  
            "What is this?",  
            "What is this food?",  
            "What is this dish?",  
            "What is this meal?"  
        ],  
        "place": [  
            "Where is this food from?",  
            "Where is this dish from?",  
            "Where is this meal from?",  
            "What province (city) is this food from?",  
            "What province (city) is this dish from?",  
            "What province (city) is this meal from?",  
            "What is this food's origin?",  
            "What is this dish's origin?",  
            "What is this meal's origin?",  
            "Where can I get specialty food?",  
            "Where can I get specialty dish?",  
            "Where can I get specialty meal?",  
            "Which local specialty is this?",  
            "Which local food is this?",  
            "What regional speciality is this?"  
        ],  
        "ingredient": [  
            "What are the ingredients of this food?",  
            "What are the ingredients of this dish?",  
            "What are the ingredients of this meal?",  
            "What is this food made of?",  
            "What is this dish made of?",  
            "What is this meal made of?",  
            "What are the ingredients?",  
            "What are the ingredients of this?"  
        ]  
    }  
}
```

```

    "What is this made of?",  

    "To make this food, what ingredients do I need?",  

    "To make this dish, what ingredients do I need?",  

    "To make this meal, what ingredients do I need?",  

    "To make this, what ingredients do I need?",  

    "What ingredients do I need to make this food?"  

],  

"recipe": [  

    "How do you make this food?",  

    "How do you make this dish?",  

    "How do you make this meal?",  

    "How do you cook this food?",  

    "How do you cook this dish?",  

    "How do you cook this meal?",  

    "How do you recipe this food?",  

    "How do you recipe this dish?",  

    "How do you recipe this meal?",  

    "How do you make this?",  

    "How do you cook this?",  

    "How do you recipe this?"  

]  

},  

"vi_question": {  

    "name": [  

        "Tên món ăn này là gì?",  

        "Món ăn này được gọi là gì?",  

        "Đây là món gì?",  

        "Đây là món ăn gì?",  

        "Tên của đồ ăn này là gì?",  

        "Đây là gì?",  

        "Đây là món ăn gì?"  

    ],  

    "place": [  

        "Món ăn này là từ đâu?",  

        "Món ăn này là từ tỉnh (thành phố) nào?",  

        "Món ăn này nổi tiếng ở đâu?",  

        "Món ăn này là từ thành phố nào?",  

        "Món ăn này là từ tỉnh nào?",  

        "Món ăn này là đặc sản ở đâu?",  

        "Món ăn này là đặc sản ở tỉnh (thành phố) nào?"  

    ],  

    "ingredient": [  

        "Nguyên liệu của món ăn này là gì?",  

        "Món ăn này làm từ gì?",  

        "Món ăn này làm từ những nguyên liệu gì?",  

        "Món ăn này làm từ những gì?",  

        "Nguyên liệu chính?",  

        "Nguyên liệu của món ăn này là gì?",  

        "Món ăn này được làm ra từ những nguyên liệu gì?"  

]
}

```

```

        ],
        "recipe": [
            "Cách làm món ăn này là gì?", 
            "Cách nấu món ăn này là gì?", 
            "Cách làm món ăn này?", 
            "Cách nấu món ăn này?", 
            "Cách chế biến món ăn này là gì?", 
            "Cách chế biến món ăn này?", 
            "Chế biến món ăn này như thế nào?", 
            "Bạn có thể chỉ cho tôi cách ăn món này không?"
        ]
    },
    "banh_tet": {
        "name": {
            "en_answer": [
                "Banh Tet La Cam",
                "This is Banh Tet La Cam",
                "This food is Banh Tet La Cam",
                "This food's name is Banh Tet La Cam",
                "The name of this food is Banh Tet La Cam",
                "This food is called Banh Tet La Cam",
                "Banh Tet La Cam is the name of this food"
            ],
            "vi_answer": [
                "Bánh tết lá cẩm",
                "Đây là bánh tết lá cẩm",
                "Đây là món bánh tết lá cẩm",
                "Tên của món ăn này là bánh tết lá cẩm"
            ]
        },
        "place": {
            "en_answer": [
                "Can Tho",
                "This food is from Can Tho",
                "This food is from Can Tho city",
                "Can Tho, where this food is from, is a city in Vietnam",
                "This food is from Can Tho, a city in Vietnam,"
            ],
            "vi_answer": [
                "Cần Thơ",
                "Món ăn này bạn có thể tìm thấy ở Cần Thơ",
                "Món ăn này nổi tiếng ở thành phố Cần Thơ",
                "Cần Thơ, là một thành phố ở Việt Nam",
                "Cần Thơ, nơi món ăn này nổi tiếng, là một thành phố ở Đồng bằng sông
                Cửu Long, Việt Nam"
            ]
        },
        "ingredient": {
            "en_answer": [

```

"To make Banh Tet La Cam, you need to prepare ingredients such as glutinous rice, green beans, pork belly, grated coconut, purple leaves, pandan leaves, banana leaves, spring onion, shallots, salt, sugar, cooking oil, and string. All of these ingredients are used to create the distinctive flavor and beautiful color of Banh Tet La Cam.",

"Prepare ingredients such glutinous rice, green beans, pork belly, shredded coconut, purple leaves, pandan leaves, banana leaves, spring onions, shallots, salt, sugar, cooking oil, and thread in advance to create Banh Tet La Cam. These components combine to give Banh Tet La Cam its distinct flavor and stunning hue.",

"To make Banh Tet La Cam, prepare the ingredients in advance, including sticky rice, green beans, pork belly, shredded coconut, purple leaves, pandan leaves, banana leaves, spring onions, shallots, salt, sugar, frying oil, and thread. These elements work together to give Banh Tet La Cam its unique flavor and gorgeous color."

],

"vi_answer": [

"Để làm bánh tết lá cẩm, bạn cần chuẩn bị các nguyên liệu như gạo nếp, đậu xanh, thịt ba chỉ, dừa nạo, lá cẩm, lá dứa, lá chuối hột, hành lá, hành tím, mői, đường, dầu ăn và dây lát. Tất cả những nguyên liệu này được sử dụng để tạo nên hương vị đặc trưng và màu sắc đẹp mắt của bánh tết lá cẩm.",

"Để thực hiện việc làm bánh tết lá cẩm, bạn cần sẵn sàng những nguyên liệu như gạo nếp, đậu xanh, thịt ba chỉ, dừa bào, lá cẩm, lá dứa, lá chuối hột, hành lá, hành tím, mői, đường, dầu ăn và dây lát. Các thành phần này được sử dụng để tạo ra hương vị đặc trưng và màu sắc hấp dẫn cho bánh tết lá cẩm."

]

},

"recipe": {

"en_answer": [

"To make Banh Tet La Cam, people mix glutinous rice, green beans, and grated coconut with seasoning, then roll them around a thin slice of pork belly. The cake is wrapped in banana leaves, pandan leaves, and purple leaves, then steamed for about 8 hours. When cooked, Banh Tet La Cam has a distinctive aroma, beautiful color, and rich flavor, representing the tradition of the Mekong Delta region.",

"Glutinous rice, green beans, and grated coconut are combined with spice to form Banh Tet La Cam, which is then rolled around a thin piece of pork belly. The cake is cooked for roughly 8 hours after being covered in banana leaves, pandan leaves, and purple leaves. Banh Tet La Cam is a traditional dish from the Mekong Delta with a distinct scent, stunning color, and flavor when it is prepared.",

"Banh Tet La Cam, which is made of glutinous rice, green beans, shredded coconut, and spices, is rolled around a thin slice of pig belly. After being wrapped in purple, pandan, and banana leaves, the cake is baked for around 8 hours. The Mekong Delta's Banh Tet La Cam is a traditional meal with a distinctive aroma, beautiful color, and flavor when it is cooked."

],

"vi_answer": [

"Để làm bánh tết lá cẩm, người ta trộn gạo nếp, đậu xanh và dừa nạo với gia vị, sau đó cuốn vào miếng thịt ba chỉ đã thái mỏng. Bánh được bọc trong lá chuối hột, lá dứa và lá cẩm, sau đó hấp trong khoảng 8 giờ đồng hồ. Sau khi chín, bánh tết lá

cẩm có mùi thơm đặc trưng, màu sắc tuyệt đẹp và hương vị đậm đà, truyền thống của miền Tây sông nước.",

"Để tạo ra bánh tét lá cẩm, người ta pha trộn gạo nếp, đậu xanh và dừa bào với gia vị, sau đó cuốn chúng vào miếng thịt ba chỉ đã được cắt mỏng. Sau đó, bánh được bọc trong lá chuối hột, lá dứa và lá cẩm trước khi hấp trong khoảng 8 giờ đồng hồ. Khi đã chín, bánh tét lá cẩm mang mùi thơm đặc trưng, màu sắc đẹp mắt và hương vị đậm đà, mang tính truyền thống của miền Tây sông nước."

]

}

,

"pho": {

 "name": {

 "en_answer": [

 "Pho",

 "This is Pho",

 "This food is Pho",

 "This food's name is Pho",

 "The name of this food is Pho"

],

 "vi_answer": [

 "Phở",

 "Tên gọi của món ăn này là phở",

 "Món ăn này được gọi là phở",

 "Đây là phở"

]

 },

 "place": {

 "en_answer": [

 "Not only in the Mekong Delta, but also throughout Vietnam",

 "This food is available throughout Vietnam, not just in the Mekong Delta.",

 "This is a Vietnamese specialty that is available throughout Vietnam, not just in the Mekong Delta.",

 "Pho, a Vietnamese specialty, can be found throughout Vietnam, not just in the Mekong Delta."

],

 "vi_answer": [

 "Không chỉ ở đồng bằng sông Cửu Long, mà còn khắp cả nước",

 "Món ăn này có thể tìm thấy khắp cả nước, không chỉ ở đồng bằng sông Cửu Long",

 "Phở là món ăn truyền thống Việt Nam, có thể tìm thấy khắp cả nước, không chỉ ở đồng bằng sông Cửu Long"

]

 },

 "ingredient": {

 "vi_answer": [

 "Phở, món ăn truyền thống của Việt Nam, và nguyên liệu chính của nó bao gồm bánh phở (từ bột gạo), thịt bò, hoặc gà, hành tây, hành hoa, gừng, lá chanh, và

nước dùng được nấu từ xương và thịt. Các loại rau như rau mùi, rau ngò, giá đỗ, và hành phi được dùng làm gia vị trang trí và tăng thêm hương vị cho món ăn."

"Phở là một món ăn truyền thống của Việt Nam, với các thành phần chính bao gồm bánh phở được làm từ bột gạo, thịt bò hoặc gà, hành tây, hành hoa, gừng, lá chanh và nước dùng được nấu từ xương và thịt. Các loại rau như rau mùi, rau ngò, giá đỗ và hành phi được sử dụng như gia vị trang trí và cũng để tăng thêm hương vị cho món ăn."

],

"en_answer": [

"Pho is a traditional dish of Vietnam, and its main ingredients include pho noodles (made from rice flour), beef or chicken, onions, scallions, ginger, lime leaves, and broth made from bones and meat. Various herbs such as cilantro, basil, bean sprouts, and fried shallots are used as garnishes to enhance the flavor of the dish."

"The essential components of pho, a classic Vietnamese cuisine, are rice-flour-based pho noodles, beef or chicken, onions, scallions, ginger, lime leaves, and a broth prepared from bones and meat. Bean sprouts, fried shallots, cilantro, basil, and other herbs are added as garnishes to improve the flavor of the dish.",

"Noodles made of rice flour, beef or chicken, onions, scallions, ginger, lime leaves, and a broth made of bones and meat are the basic ingredients of pho, a traditional Vietnamese dish. To enhance the flavor of the meal, bean sprouts, fried shallots, cilantro, basil, and other herbs are used as garnishes."

]

},

"recipe": {

"vi_answer": [

"Để làm phở, thịt và xương được đun sôi trong nước khoảng 6–8 tiếng để tạo nước dùng đậm đà. Sau đó, nước dùng được nêm gia vị bao gồm đường, nước mắm và hạt tiêu. Bánh phở được ngâm nước cho mềm và sau đó đun trong nước sôi. Thịt và bánh phở được cho vào tô và trang trí với các loại rau thơm. Nước dùng sôi hối được đổ vào tô trực tiếp, tạo nên hương vị thơm ngon và bắt mắt của món phở Việt Nam.",

"Phở được làm bằng cách ninh thịt và xương trong nước từ 6 đến 8 tiếng để tạo thành nước súp sền sệt, sau đó cho các loại gia vị như đường, mắm, tiêu vào nước dùng rồi đun sôi nước dùng. Bày ra tô, trang trí với các loại rau thơm và chan nước dùng đang sôi trực tiếp vào tô để tạo nên độ ngon và hương vị ấn tượng của món phở Việt Nam."

],

"en_answer": [

"To make Pho, the meat and bones are boiled in water for about 6–8 hours to create a flavorful broth. Then, the broth is seasoned with spices including sugar, fish sauce, and peppercorns. The Pho noodles are soaked in water to soften and then boiled in the broth. The meat and Pho noodles are placed in a bowl and garnished with herbs. The hot broth is poured directly into the bowl, creating the delicious and visually appealing taste of Vietnamese pho.",

"To generate a tasty broth for the pho, the meat and bones are cooked in water for roughly 6 to 8 hours. After that, the soup is spiced with peppercorns, honey, and fish sauce. The Pho noodles are cooked in the broth after being softened with water. The herbs are added to a dish with the pork and Pho noodles. Vietnamese

```

pho's flavor and aesthetic appeal are created by the direct pouring of the hot soup
into the bowl."],
    "The meat and bones are cooked in water for 6 to 8 hours to prepare a
delicious broth for pho. After that, sugar, fish sauce, and peppercorns are added to
the soup as seasonings. The Pho noodles are softened by soaking them in water, and
they are then cooked in the broth. The pork, Pho noodles, and herbs are all combined
in a dish. Vietnamese pho has a pleasant and aesthetically pleasing taste since the
hot soup is poured straight into the bowl."
]
},
},
"com_tam": {
  "name": {
    "en_answer": [
      "Com Tam Long Xuyen",
      "This is Com Tam Long Xuyen",
      "This food is Com Tam Long Xuyen",
      "This food's name is Com Tam Long Xuyen",
      "The name of this food is Com Tam Long Xuyen"
    ],
    "vi_answer": [
      "Cơm tấm Long Xuyên",
      "Đây là món cơm tấm Long Xuyên",
      "Món ăn này là cơm tấm Long Xuyên",
      "Tên của món ăn này là cơm tấm Long Xuyên"
    ]
  },
  "place": {
    "en_answer": [
      "Long Xuyen, which is city in An Giang province in Viet Nam",
      "From Long Xuyen, which is city in An Giang province in Viet Nam",
      "This food comes from Long Xuyen, a city in the Vietnamese province of
      An Giang.",
      "It comes from Long Xuyen, a city in An Giang province in Vietnam,
      where it was first known as Com Tam Long Xuyen.",
      "This food is from Long Xuyen, An Giang"
    ],
    "vi_answer": [
      "Thành phố Long Xuyên, tỉnh An Giang",
      "Long Xuyên, thành phố thuộc tỉnh An Giang",
      "Món ăn này bắt nguồn từ thành phố Long Xuyên, An Giang",
      "Món ăn này nổi tiếng ở Long Xuyên, An Giang"
    ]
  },
  "ingredient": {
    "vi_answer": [
      "Cơm tấm Long Xuyên là món ăn phổ biến ở miền Tây Nam Bộ, Việt Nam. Để
      làm nên món ăn này, người ta sử dụng các nguyên liệu như gạo tấm thơm, mỡ hành, tôm
      khô, thịt heo nướng (hoặc luộc), chả trứng, rau sống và dưa leo muối. Những nguyên liệu

```

này được sắp xếp đẹp mắt trên đĩa cơm, tạo nên một món ăn hấp dẫn, ngon miệng và đầy dinh dưỡng.",

"Là một trong những món ăn phổ biến ở vùng miền Tây Nam Bộ của Việt Nam. Để tạo nên món ăn này, người dân sử dụng nhiều loại nguyên liệu như gạo tấm thơm, mỡ hành, tôm khô, thịt heo đã qua nướng hoặc luộc, chả trứng, rau sống và dưa leo muối. Bằng cách sắp xếp những nguyên liệu này đều đặn và trang trí đẹp mắt trên một đĩa cơm, món ăn trở nên hấp dẫn, ngon miệng và cung cấp đầy đủ dinh dưỡng cho cơ thể."

],

"en_answer": [

"Com tam Long Xuyen is a popular dish in the Southwest region of Vietnam. To make this dish, people use ingredients such as fragrant broken rice, scallion oil, dried shrimp, grilled (or boiled) pork, steamed egg loaf, fresh vegetables, and pickled cucumber. These ingredients are arranged beautifully on a plate of rice, creating an attractive, delicious, and nutritious dish." ,

"In Vietnam's Southwest, Com Tam Long Xuyen is a well-liked dish. People use fresh vegetables, pickled cucumber, steamed egg loaf, aromatic broken rice, scallion oil, dried shrimp, grilled (or boiled) pork, and grilled (or boiled) pork when preparing this dish. This recipe is gorgeous, delectable, and wholesome thanks to the skillful arrangement of the components over a bed of rice."

]

},

"recipe": {

"vi_answer": [

"Để làm món Cơm tấm Long Xuyên, đầu tiên người ta sẽ chọn gạo tấm thơm và ngâm trong nước khoảng 4-5 giờ, sau đó đem nấu chín. Tiếp theo, thịt heo được nướng hoặc luộc, chả trứng và tôm khô được chiên giòn. Các nguyên liệu được sắp xếp đẹp mắt trên đĩa cơm, phủ lên một lớp mỡ hành và rau sống. Cuối cùng, dưa leo muối được thêm vào để tạo thêm vị giòn và mặn.",

"Đầu tiên phải chuẩn bị nguyên liệu bao gồm gạo tấm, mỡ hành, tôm khô, thịt heo nướng, chả trứng, rau sống và dưa leo muối. Gạo tấm được nấu chín, sau đó trộn đều với mỡ hành. Thịt heo nướng và tôm khô được xé nhỏ. Chả trứng được thái mỏng. Rau sống và dưa leo muối được rửa sạch, thái nhỏ. Cuối cùng, các nguyên liệu được sắp xếp đẹp mắt lên đĩa cơm. Món ăn này thường được ăn kèm với nước mắm chấm."

],

"en_answer": [

"To make Com tam Long Xuyen, first, people will choose fragrant broken rice and soak it in water for about 4-5 hours, then cook it until it is done. Next, the pork is grilled (or boiled), the egg loaf and dried shrimp are fried until crispy. The ingredients are arranged beautifully on a plate of rice, covered with a layer of scallion oil and fresh vegetables. Finally, pickled cucumber is added to create a crunchy and salty taste." ,

"First, prepare the ingredients, including broken rice, scallion oil, dried shrimp, grilled (or boiled) pork, steamed egg loaf, fresh vegetables, and pickled cucumber. The broken rice is cooked until done, then mixed with scallion oil. The grilled pork and dried shrimp are shredded. The egg loaf is sliced thinly. The fresh vegetables and pickled cucumber are washed and chopped. Finally, the ingredients are arranged beautifully on a plate of rice. This dish is often served with fish sauce."

]

```

        }
    },
    "ga_nuong": {
        "name": {
            "en_answer": [
                "Ga Nuong O Thum",
                "This is Ga Nuong O Thum",
                "This food is Ga Nuong O Thum",
                "This food's name is Ga Nuong O Thum",
                "The name of this food is Ga Nuong O Thum"
            ],
            "vi_answer": [
                "Gà nướng Ô Thum",
                "Món ăn này là Gà nướng Ô Thum",
                "Món ăn này được gọi là Gà nướng Ô Thum",
                "Tên của món ăn này là Gà nướng Ô Thum",
                "Đây là món Gà nướng Ô Thum"
            ]
        },
        "place": {
            "en_answer": [
                "Tri Ton, which is district in An Giang province in Viet Nam",
                "This food is from Tri Ton, which is a district in An Giang province in Viet Nam",
                "Tri Ton district, An Giang province",
                "One of the favorite foods in Tri Ton district, An Giang province"
            ],
            "vi_answer": [
                "Tri Tôn, An Giang",
                "Một trong những món ăn phổ biến ở vùng núi Tri Tôn, An Giang",
                "Món ăn này nổi tiếng ở Tri Tôn, An Giang"
            ]
        },
        "ingredient": {
            "vi_answer": [
                "Gà Nướng Ô Thum (Tri Tôn, An Giang) là món ăn đặc sản của vùng Tây Nam Bộ Việt Nam. Món ăn này được làm từ gà ta chọn lựa, tẩm ướp với nước mắm, tỏi, ớt, mật ong và nhiều loại gia vị khác. Gà sau đó được nướng trên lửa than cho đến khi da vàng ruộm và thịt chín đều. Khi ăn, thịt gà được thái miếng và phục vụ cùng với bánh tráng, bánh hỏi, rau sống, xà lách, dưa leo và nước chấm."
            ],
            "en_answer": [
                "Ga Nuong O Thum (Tri Ton, An Giang) is a specialty dish of the Southwest region of Vietnam. This dish is made from selected chicken, marinated with"
            ]
        }
    }
}

```

fish sauce, garlic, chili, honey, and other spices. The chicken is then grilled over charcoal until the skin is golden and the meat is evenly cooked. When eating, the chicken is sliced and served with rice paper, rice vermicelli, fresh vegetables, lettuce, pickles, and dipping sauce.",

"A specialty of Vietnam's Southwest is a dish called Ga Nuong O Thum (Tri Ton, An Giang). This dish is produced using chicken that has been carefully chosen and marinated in fish sauce, garlic, chile, honey, and other spices. Once the skin is golden and the meat is well cooked, the chicken is roasted over charcoal. When eating, the chicken is sliced and provided with rice vermicelli, lettuce, pickles, fresh veggies, and dipping sauce."

]

,

"recipe": {

"vi_answer": [

"Để làm món Gà Nướng Ô Thum, trước hết, gà ta được tẩm ướp với nước mắm, tỏi, ớt, mật ong và các gia vị khác trong vòng 3-4 tiếng. Sau đó, chúng được nướng trên lửa than cho đến khi da vàng ruộm và thịt chín đều. Thịt gà sau đó được thái miếng và phục vụ cùng với bánh tráng, bánh hỏi, rau sống, xà lách, dưa leo và nước chấm. Cách làm món ăn này cũng tùy theo khẩu vị và kinh nghiệm của người làm.",

"Để làm Gà Nướng Ô Thum, người ta chọn gà ta tươi và tẩm ướp với nước mắm, tỏi, ớt, mật ong và các gia vị khác. Sau đó, gà được nướng trên lửa than đến khi da vàng ruộm và thịt chín đều. Khi ăn, gà được thái miếng và phục vụ cùng với bánh tráng, bánh hỏi, rau sống, xà lách, dưa leo và nước chấm. Chế biến món ăn này yêu cầu sự chú ý đến độ nhiệt và thời gian nướng để đảm bảo thịt được chín mềm và giữ được vị ngon."

],

"en_answer": [

"First of all, the chicken is marinated with fish sauce, garlic, chili, honey, and other spices for 3-4 hours. Then, they are grilled over charcoal until the skin is golden and the meat is evenly cooked. The chicken is then sliced and served with rice paper, rice vermicelli, fresh vegetables, lettuce, pickles, and dipping sauce. The way to make this dish also depends on the taste and experience of the cook.",

"To make Ga Nuong O Thum, people choose fresh chicken and marinate it with fish sauce, garlic, chili, honey, and other spices. Then, the chicken is roasted over charcoal until the skin is golden and the meat is evenly cooked. When eating, the chicken is sliced and served with rice paper, rice vermicelli, fresh vegetables, lettuce, pickles, and dipping sauce. The processing of this dish requires attention to the temperature and grilling time to ensure that the meat is soft and retains its delicious taste."

]

}

},

"banh_xeo": {

"name": {

"en_answer": [

"Banh Xeo",

"This is Banh Xeo",

"This food is Banh Xeo",

```

        "This food's name is Banh Xeo",
        "The name of this food is Banh Xeo"
    ],
    "vi_answer": [
        "Bánh xèo",
        "Đây là món bánh xèo",
        "Món ăn này được gọi là bánh xèo",
        "Tên của món ăn này là bánh xèo"
    ]
},
"place": {
    "en_answer": [
        "Bac Lieu",
        "This food is from Bac Lieu",
        "It comes from Bac Lieu, which is a Vietnamese province in the Mekong Delta."
    ],
    "vi_answer": [
        "Bạc Liêu",
        "Nổi tiếng ở Bạc Liêu, tỉnh thuộc vùng Đồng bằng sông Cửu Long",
        "Món ăn này là một trong những món ăn nổi tiếng ở Bạc Liêu"
    ]
},
"ingredient": {
    "vi_answer": [
        "Bánh xèo là một món ăn đặc trưng của Bạc Liêu, miền Tây Nam Bộ Việt Nam. Nguyên liệu để làm bánh xèo gồm bột gạo, đậu xanh, nước cốt dừa, nước mắm, hành tím, rau thơm và tôm khô. Những nguyên liệu này được pha trộn với nước để tạo thành hỗn hợp bột chất lỏng, sau đó đổ lên chảo nóng để chiên cho đến khi bánh giòn vàng. Khi ăn, bánh xèo được cuộn với rau sống và dưa leo, sau đó chấm với nước mắm chua ngọt.",
        "Là món ăn đặc sản nổi tiếng của Bạc Liêu, Việt Nam. Để làm bánh xèo, người ta sử dụng các nguyên liệu như bột gạo, nước cốt dừa, đậu xanh, thịt ba chỉ, tôm, giá, hành tím, rau thơm và dưa leo. Những nguyên liệu này được trộn với nước để tạo thành hỗn hợp bột, sau đó được chiên giòn và phục vụ cùng với nước chấm và rau sống."
    ],
    "en_answer": [
        "Banh Xeo is a specialty dish of Bac Lieu, Vietnam's Southwest. The ingredients for making Banh Xeo include rice flour, mung beans, coconut milk, fish sauce, shallots, herbs, and dried shrimp. These ingredients are mixed with water to form a liquid batter, which is then poured onto a hot pan to fry until the cake is crispy and golden. When eating, the Banh Xeo is rolled with raw vegetables and pickles, then dipped in sweet and sour fish sauce.",
        "This is a famous specialty of Bac Lieu, Vietnam. To make Banh Xeo, people use ingredients such as rice flour, coconut milk, mung beans, pork belly, shrimp, bean sprouts, shallots, herbs, and pickles. These ingredients are mixed with water to form a batter, which is then fried until crispy and served with dipping sauce and raw vegetables."
    ]
}

```

```

        ],
    },
    "recipe": {
        "vi_answer": [
            "Để làm bánh xèo, đầu tiên, người ta phải pha bột từ gạo và nước dừa. Sau đó, cho nước cốt dừa, đậu xanh đã ngâm nở, hành lá, tôm, thịt nạc băm nhò vào chảo phi thơm. Tiếp theo, cho bột vào chảo, đảo đều và chờ đến khi bánh xèo chín vàng, giòn. Bánh xèo được ăn cùng với nhiều loại rau sống như rau diếp cá, rau thơm, giá đỗ, tía tô và nước chấm chua ngọt.",
            "Bánh xèo là món ăn đặc trưng của miền Nam Việt Nam. Để làm bánh xèo, người ta pha bột từ gạo, nước và nước cốt dừa. Rau thơm, tôm, thịt, đậu xanh và hành tây cũng được chuẩn bị sẵn. Sau đó, bánh được chiên trên một chảo nóng với ít dầu ăn cùng với các nguyên liệu đã chuẩn bị sẵn. Khi bánh chín vàng, người ta cho bánh ra đĩa và ăn cùng với nước chấm và rau sống."
        ],
        "en_answer": [
            "To make Banh Xeo, first, people have to make flour from rice and coconut milk. Then, add coconut milk, soaked mung beans, shallots, shrimp, and minced pork into the pan and stir-fry until fragrant. Next, add the flour to the pan, stir well and wait until the Banh Xeo is cooked and golden brown. Banh Xeo is eaten with many types of raw vegetables such as fish mint, herbs, bean sprouts, perilla, and sweet and sour dipping sauce.",
            "Banh Xeo is a specialty dish of Southern Vietnam. To make Banh Xeo, people make flour from rice, water, and coconut milk. Herbs, shrimp, meat, mung beans, and shallots are also prepared. Then, the cake is fried on a hot pan with a little cooking oil with the prepared ingredients. When the cake is cooked and golden brown, people take it out of the pan and eat it with dipping sauce and raw vegetables."
        ]
    },
    "banh_pia": {
        "name": {
            "en_answer": [
                "Banh Pia",
                "This is Banh Pia",
                "This food is Banh Pia",
                "This food's name is Banh Pia",
                "The name of this food is Banh Pia"
            ],
            "vi_answer": [
                "Bánh pía",
                "Đây là bánh pía",
                "Tên gọi của bánh này là bánh pía"
            ]
        },
        "place": {
            "en_answer": [
                "Soc Trang",
                "This food is from Soc Trang",
            ]
        }
    }
}

```

"This dish comes from Soc Trang and is known as Banh Pia.",
 "Soc Trang Province",
 "This food is from Soc Trang Province",
 "Soc Trang, is a province in the Mekong Delta in Vietnam"

],

"vi_answer": [
 "Sóc Trăng",
 "Một trong những món ăn bắt nguồn từ Sóc Trăng",
 "Sóc Trăng, là một tỉnh thuộc vùng Đồng bằng sông Cửu Long ở Việt Nam",
 "Món ăn này là một trong những món ăn nổi tiếng ở Sóc Trăng"
]

},

"recipe": {
 "vi_answer": [
 "Bánh Pía Sóc Trăng là món bánh truyền thống của miền Tây Nam Bộ, Việt Nam. Nguyên liệu chính của bánh là bột mì, đường, mỡ lợn, trứng và hạt đậu xanh. Để làm bánh, bột mì được nhào đều với mỡ lợn, trứng và nước, sau đó được nhồi và nấu chín với hạt đậu xanh. Sau khi nguội, bánh được bọc với vỏ bánh mỏng và nướng trong lò. Bánh Pía Sóc Trăng có vị thơm ngon, độ giòn vừa phải và là món quà đặc biệt của miền Tây.",
 "Bánh Pía (Sóc Trăng) – món ăn đặc sản của miền Tây Nam Bộ Việt Nam. Cách làm bánh bao gồm các bước như: pha bột, đồ nhân, xếp và bọc bánh, nướng và đóng gói. Nguyên liệu chính bao gồm bột mì, trứng gà, đường, dừa và hành tím. Bánh được làm thủ công và mang hương vị đặc trưng của đất Sóc Trăng."
],
 "en_answer": [
 "Banh Pia (Soc Trang), is a traditional cake of the Southwestern region of Vietnam. The main ingredients of the cake are wheat flour, sugar, lard, eggs, and green beans. To make the cake, wheat flour is kneaded with lard, eggs, and water, then kneaded and cooked with green beans. After cooling, the cake is wrapped with a thin cake crust and baked in the oven. Banh Pia Soc Trang has a delicious taste, moderate crispness, and is a special gift of the West."
],
 "en_answer": [
 "Banh Pia Soc Trang is a specialty dish of the Southwestern region of Vietnam. The way to make the cake includes steps such as: making flour, pouring filling, stacking and wrapping the cake, baking and packaging. The main ingredients include wheat flour, chicken eggs, sugar, coconut, and shallots. The cake is made by hand and has the characteristic flavor of Soc Trang."
]
 }

"ingredient": {
 "vi_answer": [
 "Bánh Pía Sóc Trăng có nguyên liệu chính là bột mì, đường, mỡ lợn, mứt hạt sen và vỏ bánh được làm từ trứng gà, bột mì, nước và màu thực phẩm. Những nguyên liệu này được trộn đều, nhồi nhanh và ủ trong một khoảng thời gian ngắn. Sau đó, bánh được nhồi thành từng viên, nướng trên bếp than cho đến khi vỏ bánh giòn và mùi thơm lan tỏa."
],
 "en_answer": [
 "Được làm từ những nguyên liệu chính bao gồm bột mì, đường, bơ, trứng gà, đậu xanh và hạt sen. Những nguyên liệu này được trộn với nhau để tạo thành bột nhão, sau đó cho nhân đậu xanh và hạt sen vào trộn đều. Bột sau đó được dập mỏng, cuộn nhân và nướng trong lò để tạo ra lớp vỏ giòn và nhân thơm ngon."
]
 }

```

        ],
        "en_answer": [
            "Banh Pia Soc Trang has the main ingredients of wheat flour, sugar, lard, candied lotus seeds, and the cake crust is made from chicken eggs, wheat flour, water, and food coloring. These ingredients are mixed evenly, kneaded quickly, and fermented for a short period of time. Then, the cake is kneaded into balls, baked on a charcoal stove until the crust is crispy and fragrant."
        ],
        "vi_answer": [
            "Bánh Pía Sóc Trăng có thành phần chính là bột mì, đường, lard, hạt sen kẹo, và lớp vỏ bánh làm từ trứng gà, bột mì, nước và màu thực phẩm. Các nguyên liệu này được trộn đều, nắn nhanh chóng, và lên men trong một thời gian ngắn. Sau đó, bánh được nắn thành viên, nướng trên bếp than củi cho đến khi lớp vỏ bánh giòn và thơm."
        ]
    },
    "banh_mi": {
        "name": {
            "en_answer": [
                "Banh Mi Thit",
                "This is Banh Mi Thit",
                "This food is Banh Mi Thit",
                "This food's name is Banh Mi Thit",
                "The name of this food is Banh Mi Thit"
            ],
            "vi_answer": [
                "Bánh mì thịt",
                "Đây là món bánh mì thịt",
                "Được gọi là bánh mì thịt",
                "Tên của đồ ăn này là bánh mì thịt"
            ]
        },
        "place": {
            "en_answer": [
                "Not only in the Mekong Delta in Viet Nam, but Banh Mi is also popular all over Viet Nam",
                "All over Viet Nam",
                "This food is from all over Viet Nam",
                "This food is from all over Viet Nam, where it is called Banh Mi Thit",
                "All over Viet Nam, is a country in Southeast Asia",
                "This food is from all over Viet Nam, where it is called Banh Mi Thit. Viet Nam is a country in Southeast Asia"
            ],
            "vi_answer": [
                "Không chỉ riêng các tỉnh đồng bằng sông Cửu Long, mà bánh mì thịt phổ biến trên toàn đất nước Việt Nam",
                "Bánh mì thịt phổ biến trên toàn Việt Nam, không chỉ riêng các tỉnh đồng bằng sông Cửu Long"
            ]
        }
    }
}

```

```

    "ingredient": {
        "vi_answer": [
            "Bánh mì thịt là món ăn phổ biến ở Việt Nam. Nguyên liệu chính của bánh mì thịt bao gồm bánh mì, thịt nướng hoặc xông khói, rau sống, hành tây, dưa leo, tương ớt và một số gia vị khác. Những nguyên liệu này được sắp xếp hài hòa trên bánh mì giòn tan, tạo nên một món ăn vừa ngon miệng và bổ dưỡng.",

            "Nguyên liệu chính của món ăn này bao gồm bánh mì, thịt heo hoặc thịt bò kết hợp với các loại rau sống. Ngoài ra, còn có các loại gia vị khác tùy theo khẩu vị của mỗi người."
        ],
        "en_answer": [
            "Banh Mi Thit is a popular dish in Vietnam. The main ingredients of Banh Mi Thit include bread, grilled or smoked meat, raw vegetables, onions, cucumbers, chili sauce and some other spices. These ingredients are arranged harmoniously on crispy bread, creating a delicious and nutritious dish.",

            "The main ingredients of this dish include bread, pork or beef combined with raw vegetables. In addition, there are other spices depending on the taste of each person."
        ]
    },
    "recipe": {
        "vi_answer": [
            "Để làm bánh mì thịt, trước hết cần chuẩn bị nguyên liệu như bánh mì, thịt bò hoặc thịt heo, rau sống và gia vị như tương ớt, dưa chua. Thịt sau đó được xào với hành tây, tỏi, nước tương, đường và gia vị. Bánh mì được cắt đôi, bỏ nhân thịt vào, thêm rau sống. Cuối cùng, thêm tương ớt và dưa chua vào bánh mì."
        ],
        "en_answer": [
            "To make Banh Mi Thit, first of all, you need to prepare ingredients such as bread, beef or pork, raw vegetables and spices such as chili sauce, pickles. The meat is then stir-fried with onions, garlic, soy sauce, sugar and spices. The bread is cut in half, the meat is put in, add raw vegetables. Finally, add chili sauce and pickles to the bread."
        ]
    }
},
"banh_khot": {
    "name": {
        "en_answer": [
            "Banh Khot",
            "This food is called Banh Khot",
            "This is Banh Khot"
        ],
        "vi_answer": [
            "Bánh khọt",
            "Món ăn này được gọi là bánh khọt",
            "Đây là món bánh khọt"
        ]
    }
},

```

```

"place": {
    "en_answer": [
        "One of city in Mekong Delta, where Banh Khot is popular, is Can Tho",
        "This food is from Can Tho",
        "This is from Can Tho",
        "This food is from Can Tho city",
        "This is from Can Tho city",
        "Can Tho city, is a city under the central government, the center of
the Mekong Delta region"
    ],
    "vi_answer": [
        "Thành phố Cần Thơ",
        "Cần Thơ, là một thành phố trực thuộc Trung ương, trung tâm của vùng
Đồng bằng sông Cửu Long",
        "Món ăn này là một trong những món ăn nổi tiếng của Cần Thơ"
    ]
},
"ingredient": {
    "en_answer": [
        "Banh Khot is a popular dish in Can Tho. The main ingredients of Banh
Khot include rice flour, shrimp, pork, bean sprouts, green onions, and coconut milk.
These ingredients are mixed together and fried in a mold. Banh Khot is served with raw
vegetables and sweet and sour fish sauce.",
        "The main ingredients of this dish include rice flour, shrimp, pork,
bean sprouts, green onions, and coconut milk. In addition, there are other spices
depending on the taste of each person."
    ],
    "vi_answer": [
        "Bánh khọt là một món ăn phổ biến ở Cần Thơ. Nguyên liệu chính của bánh
khọt bao gồm bột gạo, tôm, thịt heo, giá đỗ, hành lá và nước cốt dừa. Những nguyên liệu
này được trộn với nhau và chiên trong khuôn. Bánh khọt được phục vụ cùng với rau sống
và nước mắm chua ngọt."
    ]
},
"recipe": {
    "vi_answer": [
        "Bánh khọt (Cần Thơ), là món ăn đặc sản miền Tây Nam Bộ Việt Nam.
Nguyên liệu chính bao gồm bột gạo, tôm, thịt ba rọi, rau thơm và gia vị. Bột gạo được
trộn với nước dừa tươi, đổ vào chảo nướng, sau đó thêm tôm, thịt, đậu phộng và rau
thơm. Bánh được ăn kèm với nước mắm pha chua ngọt và rau sống."
    ],
    "en_answer": [
        "Để làm bánh khọt, trước hết cần chuẩn bị nguyên liệu như bột gạo, tôm,
thịt heo, giá đỗ, hành lá và nước cốt dừa. Bột gạo được trộn với nước dừa tươi, đổ vào
khuôn nướng, sau đó thêm tôm, thịt, đậu phộng và rau thơm. Bánh được ăn kèm với nước
mắm pha chua ngọt và rau sống."
    ]
}

```

```

        "Banh Khot (Can Tho), is a specialty food of the Southwestern Vietnam. The main ingredients include rice flour, shrimp, pork, herbs and spices. Rice flour is mixed with fresh coconut milk, poured into a mold, then add shrimp, meat, peanuts and herbs. Banh Khot is served with sweet and sour fish sauce and raw vegetables.",
        "To make Banh Khot, first of all, you need to prepare ingredients such as rice flour, shrimp, pork, bean sprouts, green onions, and coconut milk. Rice flour is mixed with fresh coconut milk, poured into a mold, then add shrimp, meat, peanuts and herbs. Banh Khot is served with sweet and sour fish sauce and raw vegetables."
    ]
},
{
    "name": {
        "en_answer": [
            "Banh Canh",
            "This food is called Banh Canh",
            "This is Banh Canh"
        ],
        "vi_answer": [
            "Bánh canh",
            "Món ăn này được gọi là bánh canh",
            "Đây là món bánh canh"
        ]
    },
    "place": {
        "en_answer": [
            "It is popular in the Mekong Delta, especially in Can Tho, Vinh Long, Tien Giang, and Long An.",
            "This food is popular in the Mekong Delta, especially in Can Tho, Vinh Long, Tien Giang, and Long An.",
            "This is popular in the Mekong Delta, especially in Can Tho, Vinh Long, Tien Giang, and Long An."
        ],
        "vi_answer": [
            "Bánh canh cũng là món ăn phổ biến ở miền Tây, và có một số địa điểm được biết đến với bánh canh ngon. Các địa điểm nổi tiếng với bánh canh miền Tây bao gồm: Cần Thơ, Vĩnh Long, Tiền Giang, Long An.",
            "Món ăn này nổi tiếng ở miền Tây, đặc biệt là ở Cần Thơ, Vĩnh Long, Tiền Giang và Long An."
        ]
    },
    "ingredient": {
        "vi_answer": [
            "Bánh canh là món ăn truyền thống của người Việt, nguyên liệu chính gồm bột, tôm khô hoặc tươi, thịt heo hoặc cá, nước dùng từ xương heo, hành tím, rau mùi, rau răm và các gia vị như muối, đường, tiêu, bột nǎng. Bánh canh thường được ăn kèm với rau sống, giá đỗ, hành phi, tương ớt và chanh."
        ]
    }
}

```

"Nguyên liệu chính để làm bánh canh gồm bột gạo, tinh bột khoai mì, nước lèo từ xương heo hoặc từ hải sản như tôm, cua hay cá. Thêm vào đó là các loại rau cải, hành, ngò gai, mỡ hành và gia vị như muối, đường, tiêu, nước mắm, tương ớt."

],

"en_answer": [

"Banh Canh is a traditional Vietnamese food, the main ingredients include flour, dried or fresh shrimp, pork or fish, broth from pork bones, purple onions, coriander, Vietnamese coriander and spices such as salt, sugar, pepper, starch. Banh Canh is usually served with raw vegetables, bean sprouts, fried onions, chili sauce and lemon.",

"The main ingredients to make Banh Canh include rice flour, potato starch, broth from pork bones or seafood such as shrimp, crab or fish. In addition, there are vegetables, onions, coriander, scallion oil and spices such as salt, sugar, pepper, fish sauce, chili sauce."

]

},

"recipe": {

"vi_answer": [

"Bánh canh là món ăn phổ biến và rất được ưa chuộng trong đời sống ẩm thực của người dân đồng bằng sông Cửu Long. Để làm món bánh canh, đầu tiên người ta phải chuẩn bị những nguyên liệu như bột bánh canh, tôm khô, thịt heo, nước dùng từ xương, hành, tỏi, tiêu và gia vị. Sau đó, bột bánh canh được nhào với nước để tạo thành bột nhão. Tiếp theo, người ta thái tôm khô và thịt heo thành từng miếng nhỏ rồi đem phi với hành, tỏi, tiêu cho thơm. Sau đó, nước dùng từ xương được đun sôi và cho vào hỗn hợp thịt heo và tôm khô đã phi. Cuối cùng, bột nhão được nhúng vào nước dùng cho tới khi chín. Bánh canh được cho vào tô, thêm một chút gia vị và rau thơm để tạo hương vị thơm ngon.",

"Để làm món bánh canh, đầu tiên người ta phải chuẩn bị những nguyên liệu như bột bánh canh, tôm khô, thịt heo, nước dùng từ xương, hành, tỏi, tiêu và gia vị. Sau đó, bột bánh canh được nhào với nước để tạo thành bột nhão. Tiếp theo, người ta thái tôm khô và thịt heo thành từng miếng nhỏ rồi đem phi với hành, tỏi, tiêu cho thơm. Sau đó, nước dùng từ xương được đun sôi và cho vào hỗn hợp thịt heo và tôm khô đã phi. Cuối cùng, bột nhão được nhúng vào nước dùng cho tới khi chín. Bánh canh được cho vào tô, thêm một chút gia vị và rau thơm để tạo hương vị thơm ngon."

],

"en_answer": [

"Banh Canh is a popular and very popular dish in the culinary life of people in the Mekong Delta. To make Banh Canh, first of all, people have to prepare ingredients such as Banh Canh flour, dried shrimp, pork, broth from bones, onions, garlic, pepper and spices. Then, the Banh Canh flour is kneaded with water to form a dough. Next, people cut dried shrimp and pork into small pieces and then fry them with onions, garlic, and pepper until fragrant. Then, the broth from the bones is boiled and added to the mixture of pork and dried shrimp. Finally, the dough is dipped in the broth until cooked. Banh Canh is put into a bowl, add a little spice and herbs to create a delicious flavor.",

"To make Banh Canh, first, people have to prepare ingredients such as Banh Canh flour, dried shrimp, pork, broth from bones, onions, garlic, pepper and spices. Then, the Banh Canh flour is kneaded with water to form a dough. Next, people cut dried shrimp and pork into small pieces and then fry them with onions, garlic, and

```

    pepper until fragrant. Then, the broth from the bones is boiled and added to the
    mixture of pork and dried shrimp. Finally, the dough is dipped in the broth until
    cooked. Banh Canh is put into a bowl, add a little spice and herbs to create a
    delicious flavor."
        ]
    }
}

```

Đoạn mã Data Collator của Mô hình 1: ViT và BERT / PhoBERT

```

@dataclass
class ViT_BERT_Collator:
    tokenizer: AutoTokenizer
    preprocessor: AutoImageProcessor

    def tokenize_text(self, texts: List[str]):
        encoded_text = self.tokenizer(
            text=texts,
            padding="longest",
            max_length=40,
            truncation=True,
            return_tensors="pt",
            return_token_type_ids=True,
            return_attention_mask=True,
        )
        return {
            "input_ids": encoded_text["input_ids"].squeeze(),
            "token_type_ids": encoded_text["token_type_ids"].squeeze(),
            "attention_mask": encoded_text["attention_mask"].squeeze(),
        }

    def preprocess_images(self, images: List[str]):
        processed_images = self.preprocessor(
            images=[
                Image.open(dataset_dir + "/" + image).convert("RGB")
                for image in images
            ],
            return_tensors="pt",
        )

        return {
            "pixel_values": processed_images["pixel_values"].squeeze(),
        }

    def __call__(self, raw_batch_dict):
        return {
            **self.tokenize_text(
                raw_batch_dict["question"]

```

```

        if isinstance(raw_batch_dict, dict) else
        [i["question"] for i in raw_batch_dict]
    ),
    **self.preprocess_images(
        raw_batch_dict["image"]
        if isinstance(raw_batch_dict, dict) else
        [i["image"] for i in raw_batch_dict]
    ),
    "labels": torch.tensor(
        raw_batch_dict["label"]
        if isinstance(raw_batch_dict, dict) else
        [i["label"] for i in raw_batch_dict],
        dtype=torch.int64
    ),
}

```

Đoạn mã Data Collator của Mô hình 2: ViLT

```

@dataclass
class ViLT_Collator:
    processor: AutoProcessor

    def preprocess_input(self, images: List[str], texts: List[str]):
        processed_input = self.processor(
            [
                Image.open(
                    dataset_dir + "/" + image
                ).convert("RGB").resize((224, 224))
                for image in images
            ],
            texts,
            max_length=40,
            padding="max_length",
            truncation=True,
            return_tensors="pt",
        )

        return {
            "pixel_values": processed_input["pixel_values"].squeeze(),
            "pixel_mask": processed_input["pixel_mask"].squeeze(),

            "input_ids": processed_input["input_ids"].squeeze(),
            "attention_mask": processed_input["attention_mask"].squeeze(),
            "token_type_ids": processed_input["token_type_ids"].squeeze(),
        }

    def __call__(self, raw_batch_dict):
        return {
            **self.preprocess_input(

```

```

        raw_batch_dict["image"]
        if isinstance(raw_batch_dict, dict) else
        [i["image"] for i in raw_batch_dict],

        raw_batch_dict["question"]
        if isinstance(raw_batch_dict, dict) else
        [i["question"] for i in raw_batch_dict]
    ),
    "labels": torch.tensor(
        raw_batch_dict["label"]
        if isinstance(raw_batch_dict, dict) else
        [i["label"] for i in raw_batch_dict],
        dtype=torch.int64
    ),
}

```

Đoạn mã chuyển đổi tập dữ liệu và lưu vào Google Drive của Mô hình 3: ViLT và GPT-2

```

enc_processor = AutoProcessor.from_pretrained("dandelin/vilt-b32-finetuned-vqa")

dec_processor = AutoProcessor.from_pretrained("gpt2")
dec_processor.pad_token = dec_processor.eos_token

def processor_fn(image_paths, questions, labels):
    images = [
        Image.open(
            dataset_dir + "/" + image_file
        ).convert("RGB").resize((224, 224))
        for image_file in image_paths
    ]

    encodings = enc_processor(
        images,
        questions,
        truncation=True,
        max_length=40,
        padding="max_length",
        return_tensors="pt"
    )
    decodings = dec_processor(
        labels,
        truncation=True,
        max_length=140,
        padding="max_length",
        return_tensors="pt"
    )

    return {
        "input_ids": encodings.input_ids,

```

```

    "attention_mask": encodings.attention_mask,
    "token_type_ids": encodings.token_type_ids,

    "pixel_values": encodings.pixel_values,
    "pixel_mask": encodings.pixel_mask,

    "labels": decodings.input_ids,
}

def preprocess_fn(examples):
    image_paths = examples["image_path"]
    questions = examples["question"]
    labels = examples["label"]

    model_inputs = processor_fn(image_paths, questions, labels)
    return model_inputs

dataset = load_dataset("csv", encoding="utf-8", data_files={
    "train": "./csv/train.gen.csv",
    "test": "./test.gen.csv",
})
processed_dataset = dataset.map(
    lambda examples: {
        "label": examples["label_en"],
        "question": examples["question_en"],
        "image_path": examples["image_id"],
    },
    remove_columns=dataset["train"].column_names,
    batched=True
)

processed_dataset = processed_dataset.map(
    function=preprocess_fn,
    batched=True,
    batch_size=16,
    fn_kwargs={},
    remove_columns=processed_dataset["train"].column_names
)

processed_dataset.save_to_disk(
    "./datasets/DatasetDict",
    max_shard_size="100MB"
)

```

Doạn mã Model của Mô hình 1: ViT và BERT / PhoBERT

```
class ViT_BERT_Model(nn.Module):
    def __init__(
        self,
        num_labels: int = len(answer_space),
        intermediate_dim: int = 512,
        pretrained_text_name: str = "bert-base-uncased",
        pretrained_image_name: str = "google/vit-base-patch16-224-in21k"):

        super(ViT_BERT_Model, self).__init__()
        self.num_labels = num_labels
        self.pretrained_text_name = pretrained_text_name
        self.pretrained_image_name = pretrained_image_name

        self.text_encoder = AutoModel.from_pretrained(
            self.pretrained_text_name,
        )

        self.image_encoder = AutoModel.from_pretrained(
            self.pretrained_image_name,
        )

        self.fusion = nn.Sequential(
            nn.Linear(
                self.text_encoder.config.hidden_size
                + self.image_encoder.config.hidden_size
                , intermediate_dim
            ),
            nn.ReLU(),
            nn.Dropout(0.5),
        )

        self.classifier = nn.Linear(intermediate_dim, self.num_labels)

        self.criterion = nn.CrossEntropyLoss()

    def forward(
        self,
        input_ids: torch.LongTensor,
        pixel_values: torch.FloatTensor,
        attention_mask: Optional[torch.LongTensor] = None,
        token_type_ids: Optional[torch.LongTensor] = None,
        labels: Optional[torch.LongTensor] = None):

        encoded_text = self.text_encoder(
            input_ids=input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
```

```

        return_dict=True,
    )

    encoded_image = self.image_encoder(
        pixel_values=pixel_values,
        return_dict=True,
    )

    fused_output = self.fusion(
        torch.cat(
            [
                encoded_text["pooler_output"],
                encoded_image["pooler_output"]
            ],
            dim=1
        )
    )

    logits = self.classifier(fused_output)

    out = {
        "logits": logits
    }

    if labels is not None:
        loss = self.criterion(logits, labels)
        out["loss"] = loss

    return out

```

Đoạn mã Model của Mô hình 2: ViLT

```

class ViLT_Model(nn.Module):
    def __init__(
        self,
        num_labels: int = len(answer_space),
        intermediate_dim: int = 512,
        pretrained_vqa_name: str = "dandelin/vilt-b32-finetuned-vqa"):

        super(ViLT_Model, self).__init__()
        self.num_labels = num_labels
        self.pretrained_vqa_name = pretrained_vqa_name

        self.vqa = AutoModel.from_pretrained(
            pretrained_vqa_name
        )

        self.classifier = nn.Sequential(
            nn.Linear(self.vqa.config.hidden_size, intermediate_dim),

```

```

        nn.LayerNorm(intermediate_dim),
        nn.GELU(),
        nn.Linear(intermediate_dim, num_labels)
    )

    self.criterion = nn.CrossEntropyLoss()

def forward(
    self,
    input_ids: torch.LongTensor,
    pixel_values: torch.FloatTensor,
    pixel_mask: Optional[torch.FloatTensor] = None,
    attention_mask: Optional[torch.LongTensor] = None,
    token_type_ids: Optional[torch.LongTensor] = None,
    labels: Optional[torch.LongTensor] = None):

    outputs = self.vqa(
        pixel_values=pixel_values,
        pixel_mask=pixel_mask,
        attention_mask=attention_mask,
        token_type_ids=token_type_ids,
        input_ids=input_ids,
        return_dict=True,
    )

    logits = self.classifier(
        outputs["pooler_output"]
    )

    out = {
        "logits": logits
    }

    if labels is not None:
        loss = self.criterion(logits, labels)
        out["loss"] = loss

return out

```

Doạn mã Model của Mô hình 3: ViLT và GPT-2

```
class ViLT_GPT_Model(VisionEncoderDecoderModel):
    def __init__(self, config, encoder, decoder):
        super().__init__(config, encoder, decoder)

    # Copy from VisionEncoderDecoderModel.forward
    # but add input_ids, attention_mask, token_type_ids into encoder's input
    def forward(
        self,
        pixel_values: Optional[torch.FloatTensor] = None,
        pixel_mask: Optional[torch.BoolTensor] = None,
        input_ids: Optional[torch.LongTensor] = None,
        attention_mask: Optional[torch.BoolTensor] = None,
        token_type_ids: Optional[torch.LongTensor] = None,
        decoder_input_ids: Optional[torch.LongTensor] = None,
        decoder_attention_mask: Optional[torch.BoolTensor] = None,
        encoder_outputs: Optional[Tuple[torch.FloatTensor]] = None,
        past_key_values: Optional[Tuple[Tuple[torch.FloatTensor]]] = None,
        decoder_inputs_embeds: Optional[torch.FloatTensor] = None,
        labels: Optional[torch.LongTensor] = None,
        use_cache: Optional[bool] = None,
        output_attentions: Optional[bool] = None,
        output_hidden_states: Optional[bool] = None,
        return_dict: Optional[bool] = None,
        **kwargs,
    ) -> Union[Tuple[torch.FloatTensor], Seq2SeqLMOutput]:
        return_dict = return_dict if return_dict is not None \
            else self.config.use_return_dict

        kwargs_encoder = {
            argument: value
            for argument, value in kwargs.items()
            if not argument.startswith("decoder_")
        }

        kwargs_decoder = {
            argument[len("decoder_") :]: value
            for argument, value in kwargs.items()
            if argument.startswith("decoder_")
        }

        # Encoder
        if encoder_outputs is None:
```

```

        if pixel_values is None:
            raise ValueError("You have to specify pixel_values")

        encoder_outputs = self.encoder(
            pixel_values=pixel_values,
            pixel_mask=pixel_mask,

            input_ids=input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,

            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
            **kwargs_encoder,
        )
    elif isinstance(encoder_outputs, tuple):
        encoder_outputs = BaseModelOutput(*encoder_outputs)

    encoder_hidden_states = encoder_outputs[0]

    # optionally project encoder_hidden_states
    if (
        self.encoder.config.hidden_size != self.decoder.config.hidden_size
        and self.decoder.config.cross_attention_hidden_size is None
    ):
        encoder_hidden_states = self.enc_to_dec_proj(encoder_hidden_states)

    # else:
    encoder_attention_mask = None

    if (labels is not None) \
    and (decoder_input_ids is None and decoder_inputs_embeds is None):
        decoder_input_ids = shift_tokens_right(
            labels, self.config.pad_token_id, self.config.decoder_start_token_id
        )

    # Decoder
    decoder_outputs = self.decoder(
        input_ids=decoder_input_ids,
        attention_mask=decoder_attention_mask,

        encoder_hidden_states=encoder_hidden_states,
        encoder_attention_mask=encoder_attention_mask,

        inputs_embeds=decoder_inputs_embeds,
        output_attentions=output_attentions,
        output_hidden_states=output_hidden_states,
        use_cache=use_cache,
    )

```

```

        past_key_values=past_key_values,
        return_dict=return_dict,
        **kwargs_decoder,
    )

# Compute loss independent from decoder (as some shift the logits inside them)
loss = None
if labels is not None:
    logits = decoder_outputs.logits if return_dict else decoder_outputs[0]
    loss_fct = nn.CrossEntropyLoss()
    loss = loss_fct(
        logits.reshape(-1, self.decoder.config.vocab_size),
        labels.reshape(-1)
    )

if not return_dict:
    if loss is not None:
        return (loss,) + decoder_outputs + encoder_outputs
    else:
        return decoder_outputs + encoder_outputs

return Seq2SeqLMOutput(
    loss=loss,
    logits=decoder_outputs.logits,

    past_key_values=decoder_outputs.past_key_values,
    decoder_hidden_states=decoder_outputs.hidden_states,
    decoder_attentions=decoder_outputs.attentions,
    cross_attentions=decoder_outputs.cross_attentions,

    encoder_last_hidden_state=encoder_outputs.last_hidden_state,
    encoder_hidden_states=encoder_outputs.hidden_states,
    encoder_attentions=encoder_outputs.attentions,
)

```

Đoạn mã Flask: phần server cho ứng dụng chatbot

```
app = Flask(__name__)
CORS(app)

app.config["UPLOAD_FOLDER"] = f"{ROOT_DIR}/uploads"
app.config["MAX_CONTENT_PATH"] = 16 * 1024 * 1024

@app.route("/predict", methods=["POST"])
def predict():
    if request.method == "POST":
        try:
            f = request.files["file"]
            f.save(
                os.path.join(
                    app.config["UPLOAD_FOLDER"],
                    secure_filename(f.filename)
                )
            )
            question = request.form["question"]
            language = request.form["language"]
            print("\033[94m[Question]", f"({language})", question, "\033[0m")
            image_path = os.path.join(
                app.config["UPLOAD_FOLDER"], secure_filename(f.filename)
            )
        )
        if language == "en":
            examples = {
                "image": [image_path],
                "question": [question]
            }
            model_inputs = preprocess_fn(examples, mode="gen")
            generate_answer = predict_fn(model_inputs, en_vilt_gpt, mode="gen")
            return generate_answer, 200
        elif language == "vi":
            examples = {
                "image": [image_path],
                "question": [vietnamese_word_segment(question)]
            }
            model_inputs = preprocess_fn(examples, mode="cls")
            for k, v in model_inputs.items():
                model_inputs[k] = v.unsqueeze(0).to(device)
            random_answer = predict_fn(model_inputs, vit_phobert, mode="cls")
            return random_answer, 200
        else:
            return "Something went wrong!", 500
    except Exception as e:
        print("Error: ", e)
        return "Something went wrong!", 500
```

```

def preprocess_fn(examples, mode = "gen"):
    model_inputs = {}
    if mode == "gen":
        model_inputs = vilt_gpt_processor(examples)
    elif mode == "cls":
        model_inputs = vit_phobert_collator(examples)
    else:
        raise ValueError("Invalid mode")
    return model_inputs

```

```

def predict_fn(inputs, model, mode = "gen"):
    with torch.inference_mode():
        model.eval()
        input_ids = inputs["input_ids"]
        token_type_ids = inputs["token_type_ids"]
        attention_mask = inputs["attention_mask"]
        pixel_values = inputs["pixel_values"]
        if mode == "gen":
            pixel_mask = inputs["pixel_mask"]
            generated_ids = model.generate(
                input_ids=input_ids,
                token_type_ids=token_type_ids,
                attention_mask=attention_mask,
                pixel_values=pixel_values,
                pixel_mask=pixel_mask,
                # num_beams=20,
                # no_repeat_ngram_size=2,
                early_stopping=True,
                max_length=140,
            )
            generated_answer = dec_processor.batch_decode(
                generated_ids,
                skip_special_tokens=True
            )[0]
            print("\033[92m[Answer]", generated_answer, "\033[0m")
            return generated_answer
        elif mode == "cls":
            outputs = vit_phobert(
                input_ids=input_ids,
                attention_mask=attention_mask,
                token_type_ids=token_type_ids,
                pixel_values=pixel_values,
            )
            class_id = outputs["logits"].argmax(axis=-1)
            class_name = answer_space[class_id]
            question_type = class_name.split("_")[0]
            dish_name = ("_").join(class_name.split("_")[1:])
            random_answer = np.random.choice(

```

```

        data[dish_name][question_type]["vi_answer"]
    )
    print("\033[92m[Answer]", question_type + ": " + dish_name, "\033[0m")
    print(random_answer)
    return random_answer
else:
    raise ValueError("Invalid mode")

```

Đoạn mã React Native: ứng dụng chatbot cơ bản

```

import { Platform, ActionSheetIOS } from 'react-native';
import React, { useState, useRef, useEffect } from 'react';
import * as ImagePicker from 'expo-image-picker';
import { StatusBar } from 'expo-status-bar';
import { Appbar } from 'react-native-paper';
import { SafeAreaProvider } from 'react-native-safe-area-context';
import { Chat } from '@flyerhq/react-native-chat-ui';
import DropdownAlert from 'react-native-dropdownalert';
import axios from 'axios';

const uuidv4 = () => {
    return 'xxxxxxxx-xxxx-4xxx-xxxx-xxxxxxxxxx'.replace(/[xy]/g, (c) => {
        const r = Math.floor(Math.random() * 16)
        const v = c === 'x' ? r : (r % 4) + 8
        return v.toString(16)
    })
}

export default function App() {
    let dropDownAlertRef = useRef();
    const [_messages, setMessages] = useState([]);
    const [_image, setImage] = useState(null);
    const [_lang, setLang] = useState('en');

    const user = { id: '06c33e8b-e835-4736-80f4-63f44b66666c' };
    const admin = { id: '006715e0-d899-4866-be7d-674fa7856b58' };

    const defaultQuestion = {
        'en': 'What is this food (dish) called?',
        'vi': 'Đây là món ăn gì?'
    };

    const addMessage = (newMessage) => {
        setMessages([newMessage, ..._messages]);
    }

    const addBatchMessage = (newMessages) => {
        setMessages([...newMessages, ..._messages]);
    }
}

```

```

const getLanguage = (lang, reverse = False) => {
  if (lang === 'en') {
    return reverse ? 'Tiếng Việt' : 'English';
  }
  return reverse ? 'English' : 'Tiếng Việt';
}

const handleAnswer = (question, answer, image) => {
  if (answer[answer.length - 1] !== '.') answer += '.';
  const questionObj = {
    author: user,
    createdAt: Date.now(),
    id: uuidv4(),
    text: question,
    type: 'text',
  };
  answer = answer.split('.');
  let answerArr = [];
  for (let i = 0; i < answer.length; i++) {
    if (answer[i].trim() !== '') {
      const normalize = (text) => {
        text = text.trim();
        if (text[text.length - 1] !== '.') {
          text += '.';
        }
        return text;
      }
      answerArr.push({
        author: admin,
        createdAt: Date.now(),
        id: uuidv4(),
        text: normalize(answer[i]),
        type: 'text',
      });
    }
  }
  answerArr = answerArr.reverse();
  const imageObj = {
    author: user,
    createdAt: Date.now(),
    height: image.height,
    id: uuidv4(),
    name: image.fileName ?? image.uri?.split('/').pop() ?? 'IMG',
    size: image.fileSize ?? 0,
    type: 'image',
    uri: `data:image/*;base64,${image.base64}`,
    width: image.width,
  };
}

```

```

        if (!question) {
            addBatchMessage([...answerArr, imageObj]);
        } else {
            addBatchMessage([...answerArr, questionObj]);
        }
    }

const getAnswer = async (question, image) => {
    const serverUrl = 'https://5027-125-235-236-84.ngrok-free.app';
    const data = new FormData();
    data.append('file', {
        uri: Platform.OS === 'ios' ? image.uri.replace('file://', '') : image.uri,
        type: 'image/jpeg',
        name: image.fileName ?? image.uri?.split('/').pop() ?? 'IMG'
    });
    data.append('question', question ?? defaultQuestion[_lang]);
    data.append('language', _lang);
    const result = await axios.post(serverUrl + '/predict', data, {
        headers: { 'Content-Type': 'multipart/form-data' }
    });
    if (result.status !== 200) {
        dropDownAlertRef.alertWithType(
            'error', 'Error',
            'Have a problem with the server, please try again later!'
        );
        return;
    }
    handleAnswer(question, result.data, image);
}

const addImageMessage = async (response, upload = true) => {
    if (response?.base64) {
        setImage(response);
        const imageMessage = {
            author: user,
            createdAt: Date.now(),
            height: response.height,
            id: uuidv4(),
            name: response.fileName ?? response.uri?.split('/').pop() ?? 'IMG',
            size: response.fileSize ?? 0,
            type: 'image',
            uri: `data:image/*;base64,${response.base64}`,
            width: response.width,
        }
        addMessage(imageMessage);
        if (upload) await getAnswer(null, response);
    }
}

```

```

const takePhoto = async () => {
  const { status } = await ImagePicker.requestCameraPermissionsAsync();
  if (status !== 'granted') {
    dropDownAlertRef.alertWithType(
      'error', 'Error',
      'Sorry, we need camera roll permissions to make this work!'
    );
  }
  const result = await ImagePicker.launchCameraAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    allowsEditing: true,
    quality: 1,
    base64: true,
    aspect: [4, 4],
    allowsMultipleSelection: false,
    selectionLimit: 1,
  });
  if (!result.canceled) {
    await addImageMessage(result.assets[0])
  }
}

const chooseFromLibrary = async () => {
  const result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    allowsEditing: true,
    quality: 1,
    base64: true,
    aspect: [4, 4],
    allowsMultipleSelection: false,
    selectionLimit: 1,
  });
  if (!result.canceled) {
    addImageMessage(result.assets[0])
  }
}

const handleAttachmentPress = () => {
  ActionSheetIOS.showActionSheetWithOptions(
    {
      options: [
        'Cancel',
        'Take a photo',
        'Choose from library',
        'Change language to ' + getLanguage(_lang, true)
      ],
      cancelButtonIndex: 0,
    }, async (buttonIndex) => {
      if (buttonIndex == 0) {

```

```

        } else if (buttonIndex == 1) {
            await takePhoto();
        } else if (buttonIndex == 2) {
            await chooseFromLibrary();
        } else if (buttonIndex == 3) {
            setMessages([]);
            const newLang = _lang === 'en' ? 'vi' : 'en';
            setLang(newLang);
        }
    }
}

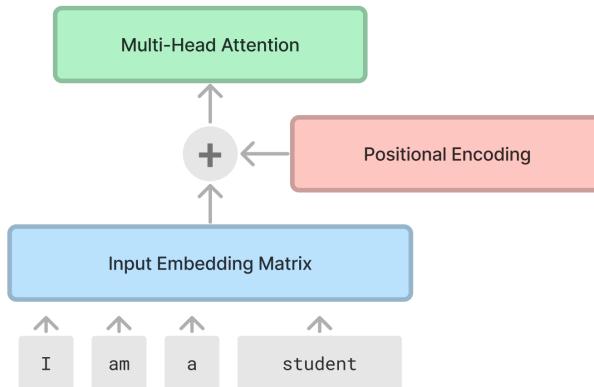
const handleSendPress = async (message) => {
    const textMessage = {
        author: user,
        createdAt: Date.now(),
        id: uuidv4(),
        text: message.text,
        type: 'text',
    };
    if (!_image) {
        dropDownAlertRef.alertWithType(
            'error', 'Error', 'Please upload an image first!'
        );
        return;
    }
    addMessage(textMessage);
    await getAnswer(message.text, _image);
}

useEffect(() => {
    const welcomeMessage = {
        en: 'Hello! I\'m a chatbot that can help you find the name of food in Mekong Delta. Please upload an image of the food you want to find the name.',
        vi: 'Xin chào! Tôi là chatbot có thể giúp bạn tìm tên món ăn ở Đồng bằng sông Cửu Long. Hãy tải lên ảnh món ăn bạn muốn tìm tên.'
    };
    addBatchMessage([
        {
            author: admin,
            createdAt: Date.now(),
            id: uuidv4(),
            text: welcomeMessage[_lang],
            type: 'text',
        }
    ]);
}, [_lang]);

```

```
return (
  <SafeAreaProvider>
    <Appbar.Header>
      <Appbar.Content
        title='ChatBot for Mekong Delta Speciality Dishes'
      />
    </Appbar.Header>
    <Chat
      messages={_messages}
      onSendPress={handleSendPress}
      onAttachmentPress={handleAttachmentPress}
      user={user}
      enableAnimation={false}
      locale={_lang}
      l10nOverride={(
        inputPlaceholder: (
          _image
            ? 'Type your question about the dish'
            : 'Please upload an image first!'
        ),
      )}
    />
    <DropdownAlert
      ref={(ref) => {
        if (ref) {
          dropDownAlertRef = ref;
        }
      }}
    />
    <StatusBar style='auto' />
  </SafeAreaProvider>
);
}
```

ⁱ Self-Attention được sử dụng để xác định mối quan hệ giữa mỗi đầu vào và các đầu vào khác.

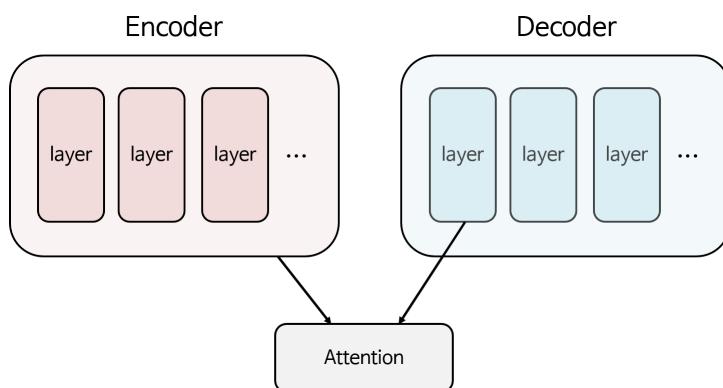


Hình 103. Mô tả câu “I am a student”

Ví dụ, trong một câu như “I am a student”, mỗi từ đều có mối quan hệ khác nhau với nhau. Self-Attention giúp học được các mối quan hệ này để hiểu được bối cảnh của câu.

Tuy nhiên, trong quá trình sử dụng Self-Attention để tìm kiếm các mối quan hệ, có thể xảy ra những kết nối không hợp lệ (illegal connection) giữa các phần tử. Để khắc phục điều này, ta có thể sử dụng một kỹ thuật gọi là *Mask* để loại bỏ các kết nối không hợp lệ này.

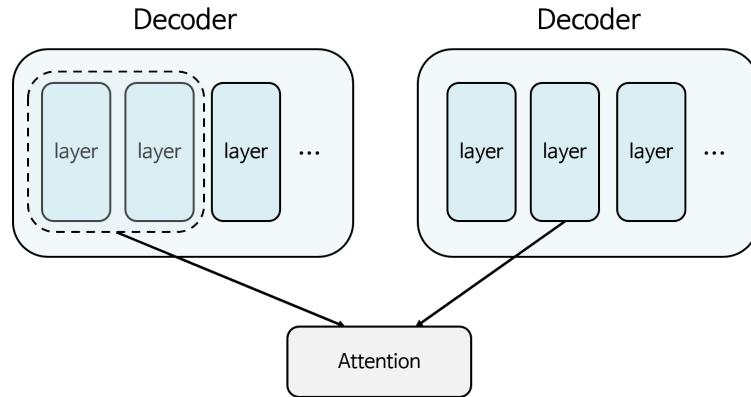
Đầu tiên, ta xem xét thế nào là kết nối không hợp lệ (illegal connection).



Hình 104. Mỗi Decoder sẽ kết nối với một hoặc nhiều Encoder trong quá trình tính toán Attention

Trong cơ chế Attention được sử dụng trong mô hình Seq2Seq, chúng sử dụng một số tầng (layer) trong Encoder cùng với một tầng trong Decoder để tính toán ra giá trị

Attention. Do đó, như Hình 104 phía trên, toàn bộ tầng (layer) của Decoder, tầng đầu tiên, tầng thứ hai, ... sẽ được kết hợp với các tầng tương ứng trong Encoder để tính toán giá trị Attention tuần tự một.



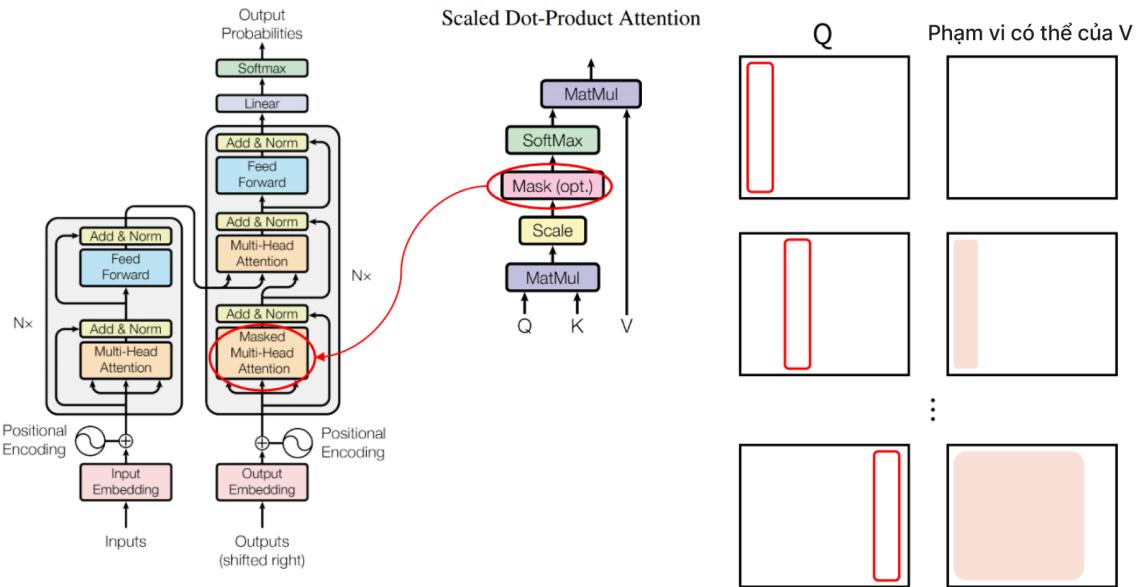
Hình 105. Phạm vi để chọn các Decoder trong quá trình tính toán Self-Attention

Trong Self-Attention, Decoder có thể thực hiện tính toán giá trị Attention với chính nó. Như Hình 105, để thực hiện tính toán giá trị Attention cho tầng thứ 2 bên phải, không thể sử dụng toàn bộ các tầng của Decoder, vì nếu coi Decoder như đầu ra xuất hiện theo thứ tự, các tầng 3 và 4 vẫn chưa có đầu ra. Do đó, chỉ có thể thực hiện tính toán giá trị Attention trên các tầng 1 và 2.

Theo đó, để thực hiện Self-Attention, chỉ có thể sử dụng các tầng (layer) ở trước để tính toán giá trị Attention.

Kết nối không hợp lệ (Illegal connection) trong Self-Attention có nghĩa là khi một tầng (layer) ở tương lai được dùng để tính toán giá trị Attention với một tầng (layer) ở hiện tại. Trong trường hợp này, khi thực hiện tính toán giá trị Attention trên tầng thứ 2 của Decoder, các tầng thứ 3 và thứ 4 cũng tham gia vào. Tuy nhiên, vì các tầng này đại diện cho kết quả được tính toán trong tương lai, việc sử dụng chúng sẽ gây ra vấn đề và phải được tránh.

Áp dụng vào mô hình Transformer



Hình 106. Phạm vi có thể chọn V của một Q cụ thể trong quá trình tính toán giá trị Attention

Như đã thấy ở bên phải trên Hình 106, với mỗi Query (Q) được chọn, thì phạm vi của Value (V) được tham gia vào việc tính toán giá trị Attention giới hạn từ vị trí đầu tiên đến vị trí ngay trước đó \Rightarrow Mask được sử dụng để cấm kết nối attention bất hợp pháp như vậy.

ii Trong NLP, từ thông thường được biểu diễn dưới dạng một vector số thực có số chiều cố định, với mỗi chiều đại diện cho một đặc trưng nhất định của từ đó như tần suất xuất hiện, từ loại, vị trí trong câu, ... Điều này có nghĩa là mỗi từ sẽ có một vector biểu diễn duy nhất, không thay đổi tùy thuộc vào ngữ cảnh của nó trong câu.

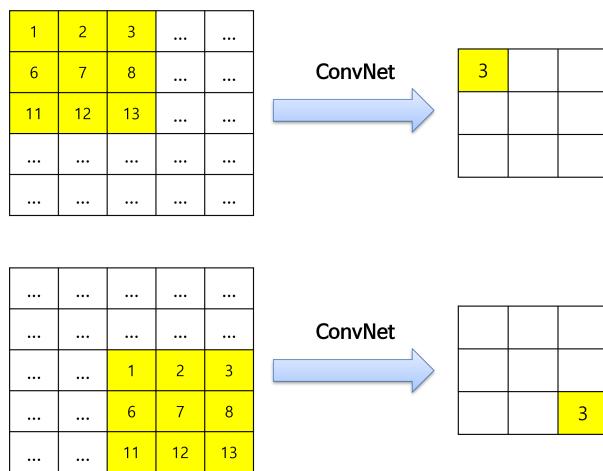
Tuy nhiên, các từ trong câu có thể có nhiều ý nghĩa khác nhau, phụ thuộc vào ngữ cảnh mà chúng xuất hiện. Do đó, việc biểu diễn các từ dưới dạng vector có thể cải thiện bằng cách tạo ra các embedding phụ thuộc vào ngữ cảnh, được gọi là contextualized word embeddings. Các embedding này được tính toán bằng cách sử dụng mô hình học sâu, như một mạng neuron, để đưa ra dự đoán về từ tiếp theo trong câu dựa trên toàn bộ ngữ cảnh trước đó.

Vì vậy, mỗi từ trong câu sẽ có một vector biểu diễn độc lập và phụ thuộc vào ngữ cảnh của nó trong câu. Contextualized word embedding được sử dụng rộng rãi trong các nhiệm vụ xử lý ngôn ngữ tự nhiên như phân loại văn bản, dịch máy và nhiều nhiệm vụ khác.

iii Inductive bias có nghĩa là những giả định và hạn chế mà mô hình đặt ra trong quá trình học. Đây có thể được coi là những giả định được sử dụng để dự đoán đầu ra cho các đầu vào mà mô hình chưa từng gặp.

Ví dụ, CNN giả định về tính tương đương dịch chuyển (translation equivariance) và tính cục bộ (locality) trong xử lý hình ảnh. Điều này có nghĩa là các mẫu hoặc cấu trúc giống nhau trên cùng một hình ảnh phải được nhận dạng theo cùng một cách, bất kể vị trí của chúng. Giả định này là một trong những lý do giải thích tại sao CNN có thể đạt được hiệu suất cao với số lượng dữ liệu thấp.

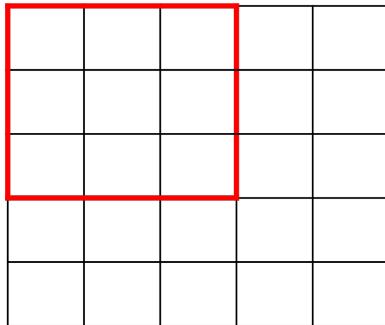
Translation equivariance là giả thiết rằng khi vị trí đầu vào thay đổi, đầu ra cũng sẽ thay đổi tại cùng một vị trí. Nói cách khác, khi mô hình nhận dạng đối tượng trong ảnh, nó phải có khả năng nhận dạng đối tượng đó theo cùng một cách, dù đối tượng đó xuất hiện ở vị trí nào trong ảnh. Đây là một trong những giả thiết quan trọng trong xử lý ảnh.



Hình 107. Translation equivariance trong CNN

Theo Hình 107, chúng ta có thể thấy rằng CNN giả định rằng vị trí của đầu vào thay đổi thì vị trí của đầu ra cũng thay đổi theo. Nghĩa là, trong khi nhận diện đối tượng trong hình ảnh, dù vị trí của đối tượng có thay đổi thì CNN vẫn giữ nguyên được các đặc trưng (feature) quan trọng được trích xuất từ đối tượng đó.

Locality là một trong những inductive bias khác được sử dụng trong CNN. Điều này có nghĩa là trong quá trình Convolution, bộ lọc chỉ xem một phần của hình ảnh trên toàn bộ hình ảnh. Nói cách khác, giả định rằng mỗi bộ lọc Convolution có thể trích xuất đặc trưng trong một khu vực nhỏ cụ thể của nó và bỏ qua các phần còn lại.



Hình 108. Locality trong CNN

Như vậy, ngay cả khi chỉ xem xét một khu vực nhỏ hơn của hình ảnh, ví dụ: 3×3 (hình vuông màu đỏ), CNN vẫn có thể trích xuất các đặc trưng.

Đối với mô hình Transformer, nó huấn luyện mà không cần phải nhìn vào *local receptive field*^(*) như CNN. Thay vào đó, mô hình sử dụng cơ chế Attention để xác định vị trí cần chú ý trong dữ liệu toàn bộ, do đó mô hình Transformer yêu cầu lượng dữ liệu lớn hơn so với CNN. Vì vậy, nếu huấn luyện mô hình với lượng dữ liệu không đủ, hiệu suất tổng quát sẽ giảm.

Ví dụ, nếu sử dụng tập dữ liệu ImageNet để huấn luyện, chúng ta sẽ thấy rằng hiệu suất của Transformer thấp hơn so với ResNet có kích thước tương tự.

Tuy nhiên, Transformer đã được chứng minh có hiệu suất cao hơn so với CNN. Do đó, phương pháp hiệu quả để sử dụng Transformer là sử dụng *Large-scale dataset*^(#) để huấn luyện hoặc sử dụng *Transfer Learning*.

Trong bài báo [1], các tác giả đã chứng minh rằng khi sử dụng một số tập dữ liệu lớn (*Large-scale dataset*) như ImageNet-21K hoặc JFT-300M để huấn luyện mô hình, sau đó sử dụng Transfer Learning trên tập dữ liệu nhỏ hơn như CIFAR-10, mô hình đạt được độ chính xác cao hơn so với sử dụng các mô hình khác như ResNet.

^(*)*Local receptive field* là một khái niệm trong mạng neural tích chập (CNN). Nó thể hiện khả năng của CNN trong việc xác định và sử dụng các đặc trưng cục bộ của dữ liệu đầu vào và được xác định bằng kích thước của bộ lọc tích chập được áp dụng trên đầu vào. Ví dụ, nếu kích thước của bộ lọc là 3×3 , thì receptive field sẽ là 3×3 . Mỗi neuron trong lớp tích chập sử dụng chỉ một phần của receptive field để tính toán đầu ra của nó. Do đó, các neuron trong lớp tích chập chỉ tính toán các đặc trưng cục bộ của đầu vào, chứ không phụ thuộc vào toàn bộ dữ liệu đầu vào.

^(#)*Large-scale dataset* là một thuật ngữ được sử dụng để chỉ các tập dữ liệu lớn với số lượng mẫu và đa dạng cao, thường được sử dụng để huấn luyện các mô hình học máy

phức tạp như Deep Learning. Các ví dụ về *Large-scale dataset* bao gồm ImageNet, COCO, Open Images, Common Crawl, và các tập dữ liệu lớn về ngôn ngữ như Wikipedia.