BÁO CÁO CUỐI KỲ
NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Đề tài: Fine-tuning mô hình ngôn ngữ lớn cải thiện hiệu suất tóm tắt văn bản

Nguyễn Hữu Quyết - 23001554
Trần Thị Mai Anh - 23001497
Nguyễn Lê Tùng Dương - 23001512

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Hà Nội - 2025

Thành viên nhóm

Phân công

Công việc	**Người phụ trách**
Lý thuyết học sâu và xử lý ngôn ngữ tự nhiên, đánh giá mô hình, làm web demo, làm slide	Nguyễn Lê Tùng Dương
Thực hiện tiền xử lý dữ liệu, kiến thức về Transformer, làm web demo, làm báo cáo	Trần Thị Mai Anh
Trưởng nhóm, tìm hiểu về fine-tuning, thực hiện train mô hình, thực nghiệm, làm web demo, làm slide	Nguyễn Hữu Quyết

Nội dung trình bày

Xử Lý Ngôn Ngữ Tự Nhiên (NLP)

Định nghĩa

Xử lý ngôn ngữ tự nhiên (NLP) là lĩnh vực kết hợp giữa **ngôn ngữ học** và **khoa học máy tính**, nhằm giúp máy tính hiểu, phân tích và phản hồi ngôn ngữ của con người.

Đặc điểm phương pháp truyền thống:

- Dựa trên các quy tắc (Rule-based).
- Hoặc dựa trên các mô hình thống kê (Statistical models).

Ứng dụng phổ biến:

- Dịch máy, Tóm tắt văn bản.
- Hệ thống sinh văn bản, Phân tích cảm xúc.

Bài toán Tóm tắt văn bản tự động

- **Định nghĩa:** Là quá trình sử dụng máy tính để rút gọn tài liệu đầu vào thành phiên bản ngắn hơn, cô đọng hơn.
- **Yêu cầu cốt lõi:**
  - Bảo toàn nội dung chính và ý nghĩa quan trọng nhất.
  - Giữ được tính mạch lạc và chính xác về mặt ngữ nghĩa.

Phân loại

Tóm tắt văn bản được chia làm hai hướng tiếp cận chính:

1. Tóm tắt Trích rút (Extractive).
2. Tóm tắt Trừu tượng (Abstractive).

1. Tóm tắt Trích rút (Extractive Summarization)

Định nghĩa

Hệ thống lựa chọn và trích xuất **nguyên văn** các câu hoặc cụm từ quan trọng nhất từ văn bản gốc để ghép lại thành bản tóm tắt.

- **Cơ chế:** Dựa trên đặc trưng thống kê:
  - Tần suất từ (TF-IDF).
  - Vị trí câu trong đoạn văn.
  - Các từ khóa chủ chốt.
- **Ưu điểm:** Đơn giản, đảm bảo đúng ngữ pháp gốc và độ chính xác thông tin cao.
- **Nhược điểm:** Bản tóm tắt thường rời rạc, thiếu tính liên kết mạch lạc giữa các câu và không thể tổng hợp ý mới.

2. Tóm tắt Trừu tượng (Abstractive Summarization)

Định nghĩa

Máy tính "đọc hiểu"nội dung và **viết lại** bản tóm tắt mới bằng ngôn ngữ của chính nó (tương tự con người).

- **Cơ chế:** Sử dụng mô hình Học sâu (Deep Learning) kiến trúc **Encoder-Decoder** để mã hóa ý nghĩa và sinh văn bản đích.
- **Ưu điểm:**
  - Tự nhiên, cô đọng.
  - Có tính tổng quát cao và mạch lạc.
- **Nhược điểm:**
  - Phức tạp, tốn tài nguyên.
  - Đòi hỏi dữ liệu huấn luyện lớn.
  - Dễ gặp vấn đề sai lệch thông tin (Hallucination).

Quy trình xử lý dữ liệu cho mô hình tóm tắt

Vai trò của Tiền xử lý (Preprocessing)

Là bước tiên quyết để chuyển đổi văn bản thô thành dạng vector số học mà mô hình học sâu có thể xử lý được.

Các bước chính:

1. Thu thập và chuẩn bị dữ liệu.
2. Tách từ và xây dựng từ điển (Tokenization).
3. Tạo vector nhúng (Word Embedding).

1. Thu thập và Tách từ

Bước 1: Thu thập và chuẩn bị dữ liệu

- Dữ liệu đầu vào thường là các cặp: **(Văn bản gốc - Văn bản tóm tắt)**.
- *Ví dụ:* Các bài báo (Content) và Tiêu đề/Sapo (Summary) tương ứng.

Bước 2: Tách từ (Tokenization)

- Quá trình chia câu thành các đơn vị từ riêng biệt (Tokens).
- Mỗi từ sau đó được ánh xạ thành một ID số trong từ điển.
- Giúp chuyển đổi văn bản thành các đơn vị dễ xử lý cho mô hình học máy.

2. Tạo vector nhúng (Word Embedding)

- **Kỹ thuật phổ biến:** Word2Vec, GloVe.
- **Mục đích:** Chuyển đổi từ ngữ thành các vector số trong không gian nhiều chiều.

Nguyên lý hoạt động

Các từ thường xuyên xuất hiện cùng nhau trong một ngữ cảnh hoặc có ý nghĩa tương đồng sẽ có vector biểu diễn nằm gần nhau trong không gian vector.

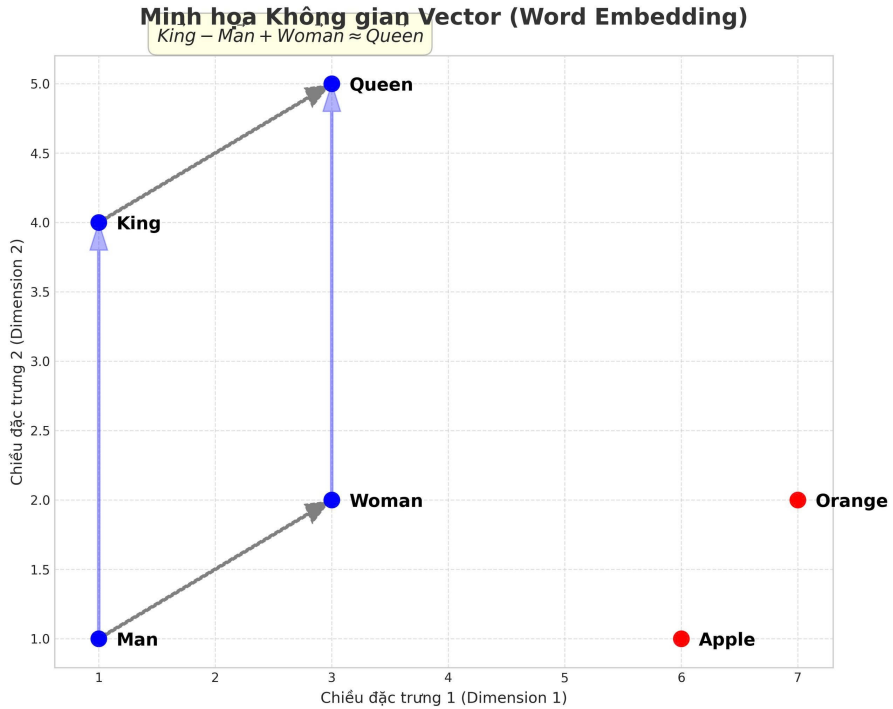
Vai trò

Giúp mô hình hiểu được mối quan hệ **ngữ nghĩa** giữa các từ thay vì chỉ so khớp ký tự đơn thuần (như One-hot encoding).

Các bước thực hiện Word Embedding

1. **Khởi tạo vector nhúng:**
   - Mỗi từ được ánh xạ thành một vector số d chiều.
   - Ban đầu, các vector này được khởi tạo ngẫu nhiên hoặc bằng giá trị nhỏ.
2. **Tối ưu hóa:**
   - Quá trình huấn luyện sẽ điều chỉnh các vector.
   - Mục tiêu: Đưa các từ có ý nghĩa tương tự về gần nhau trong không gian.
3. **Đầu ra (Ma trận nhúng W):**
   - Số hàng = Kích thước tập từ vựng.
   - Số cột = Số chiều vector nhúng.
   - Đây là đầu vào cho các lớp tiếp theo của mô hình học sâu.

Minh họa Không gian Vector (Word Embedding)



Hình 1: Minh hoạ Không gian Vector

Mạng Nơ-ron Nhân Tạo (ANN)

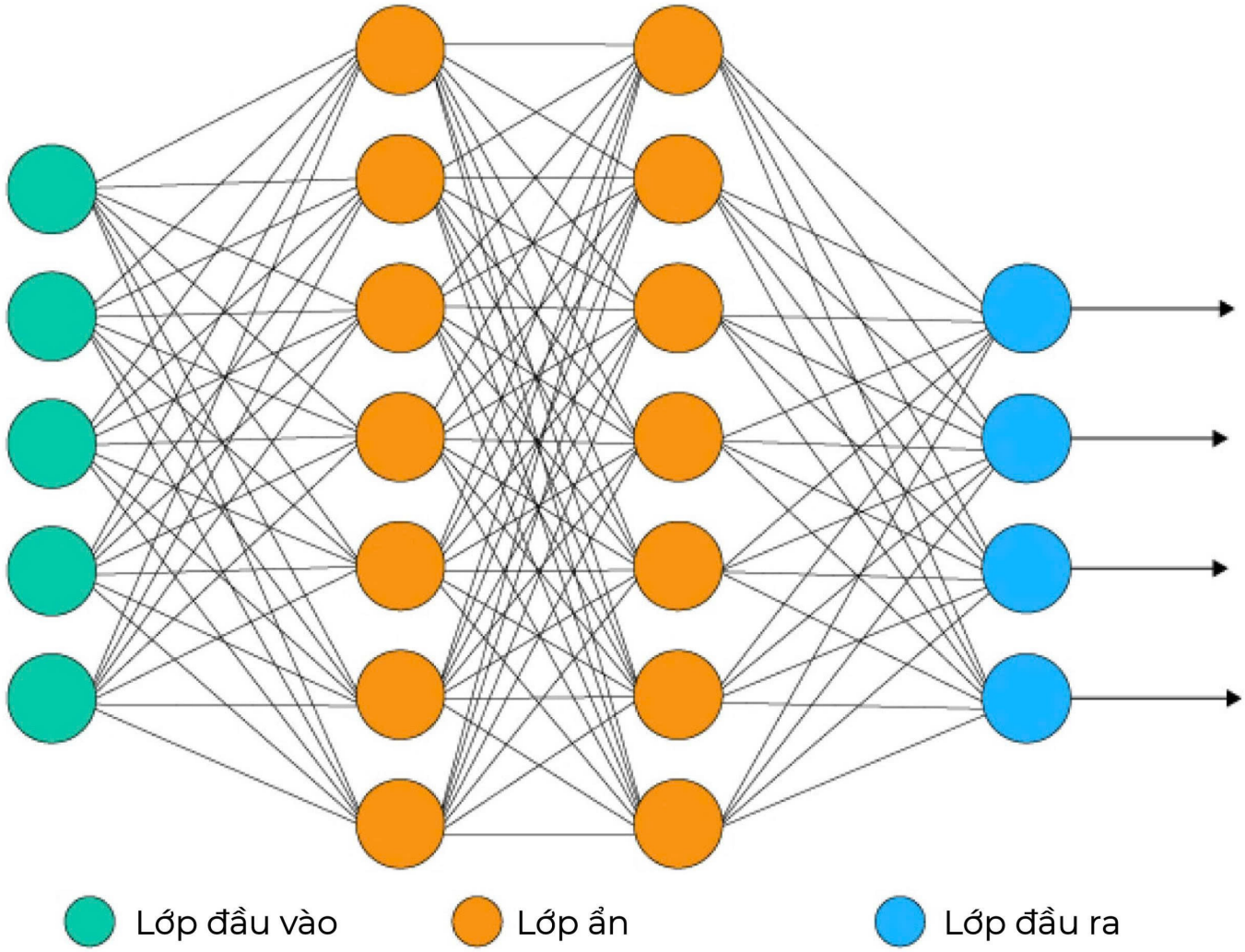
Khái niệm

Mạng nơ-ron nhân tạo là mô hình tính toán được xây dựng dựa trên hoạt động của não bộ con người. Cấu trúc cơ bản bao gồm các nút (nơ-ron) được kết nối qua các **trọng số** (w_i).

Công thức tính đầu ra của một nơ-ron

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \quad (3.1)$$

- x_i: Các tín hiệu đầu vào.
- w_i: Các trọng số tương ứng.
- b: Hệ số bù (bias).
- f: Hàm kích hoạt phi tuyến.



Hình 2: Minh họa Mạng Nơ-ron nhân tạo

Cơ chế hoạt động của Mạng nơ-ron

Quá trình xử lý thông tin diễn ra theo hai chiều:

1. **Lan truyền tiến (Forward Propagation):**
   - Dữ liệu được đưa vào → qua các lớp → lớp đầu ra để dự đoán kết quả.
2. **Lan truyền ngược (Backward Propagation):**
   - Điều chỉnh trọng số và hệ số bù (w_i, b) dựa trên lỗi của đầu ra.
   - Mạng học qua quá trình **huấn luyện** để giảm thiểu sai số.

Các Hàm Kích Hoạt Phi Tuyến

Các Hàm Kích Hoạt:

- **Sigmoid (3.2):** Chuyển đổi đầu ra về khoảng $(0, 1)$, thường dùng cho phân loại nhị phân.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

- **ReLU (Rectified Linear Unit) (3.3):** Giúp giảm vấn đề **biến mất đạo hàm** và tăng tốc độ tính toán.

$$ReLU(x) = \begin{cases} 0 & \text{nếu } x < 0 \\ x & \text{nếu } x \geq 0 \end{cases}$$

- **Softmax (3.4):** Biến đổi đầu ra thành **phân phối xác suất**, dùng để dự đoán từ tiếp theo.

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

3.2.3. Kiến trúc Encoder-Decoder

Kiến trúc Xương Sống

Là kiến trúc cốt lõi cho các mô hình **tóm tắt trừu tượng** (Abstractive).

Hai phần chính:

1. **Encoder (Bộ mã hóa):**
   - Đọc toàn bộ văn bản gốc dài.
   - Nén thông tin thành một "**vector ngữ cảnh**"chứa ý nghĩa tổng quát.
2. **Decoder (Bộ giải mã):**
   - Dựa trên vector ẩn này để sinh ra câu đích một cách tuần tự (dự đoán từng từ).

(Hình 3.3: Mô hình Encoder-Decoder xử lý chuỗi đầu vào và sinh tóm tắt)

3.2.4. Các mô hình xử lý chuỗi (Phần 1)

Hạn chế của RNN

Mạng nơ-ron hồi quy (RNN) gặp khó khăn với văn bản dài do vấn đề **biến mất đạo hàm** (*Vanishing Gradient*), khiến mô hình quên mất thông tin đầu câu.

Giải pháp: LSTM (Long Short-Term Memory)

- Khắc phục bằng cơ chế các **cổng** (Cổng quên, Cổng vào, Cổng ra).
- Giúp mô hình "nhớ"các thông tin quan trọng cần tóm tắt và "quên"các chi tiết thừa.

figureCấu trúc tế bào RNN và LSTM (Hình 3.4)

Đột phá của Transformer

Transformer loại bỏ sự phụ thuộc tuần tự của RNN.

Cơ chế Self-Attention

- Cho phép mô hình khi sinh một từ có thể "**nhìn**"lại toàn bộ văn bản gốc.
- Đánh trọng số xem phần nào là quan trọng nhất, bất kể khoảng cách xa gần.
- Là mô hình hiện đại nhất hiện nay (nền tảng của Llama 3, GPT).

figureMinh họa cơ chế hoạt động bên trong của Self-Attention (Hình 3.5)

3.3.1. Quá trình huấn luyện và tối ưu hóa (Phần 1)

Hàm mất mát (Loss Function) - Cross Entropy Loss

Đo độ chênh lệch giữa phân phối xác suất dự đoán của mô hình và từ thực tế (Ground Truth).

$$L = -\sum_{i=1}^{C} y_{true}^{(i)} \cdot \log(y_{predicted}^{(i)}) \quad (3.5)$$

- C: Kích thước từ vựng (Số lớp).
- $y_{true}^{(i)}$: Xác suất thực tế (thường là 1 cho từ đúng).

Regularization (Chống Overfitting)

Tập hợp các kỹ thuật thêm ràng buộc vào mô hình để giảm độ phức tạp và tăng khả năng tổng quát hóa trên dữ liệu mới.

L1 và L2 Regularization

L1 Regularization (Lasso)

Phạt tổng giá trị tuyệt đối của trọng số. Có xu hướng đưa các trọng số nhỏ về 0 (*Feature Selection*).

$$L_{new} = L_{old} + \lambda \sum_i |w_i| \quad (3.6)$$

L2 Regularization (Ridge)

Phạt tổng bình phương của trọng số. Giúp giữ các trọng số nhỏ nhưng khác 0.

$$L_{new} = L_{old} + \lambda \sum_i w_i^2 \quad (3.7)$$

Trong đó λ là hệ số regularization, kiểm soát mức độ phạt.

Dropout và Layer Normalization

Dropout

Kỹ thuật giúp tránh overfitting bằng cách bỏ đi ngẫu nhiên $p\%$ nơ-ron của lớp trong quá trình huấn luyện ($p \in [0.2, 0.5]$). Buộc các nơ-ron phải học độc lập hơn.

Layer Normalization (Chuẩn hóa Lớp)

Kỹ thuật chuẩn hóa được sử dụng để ổn định và tăng tốc quá trình huấn luyện các mạng nơ-ron sâu (đặc biệt là Transformer).

$$LayerNorm(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta \quad (3.8)$$

Trong đó μ là trung bình, σ^2 là phương sai của vector đầu vào x.

Learning Rate Schedule và Adam Optimizer

Learning Rate Schedule

Kỹ thuật điều chỉnh tốc độ học (α) trong quá trình huấn luyện để đạt được sự hội tụ tối ưu:

- **Warm-up:** Tăng dần tốc độ học ban đầu.
- **Decay:** Giảm dần tốc độ học sau giai đoạn huấn luyện chính.

Thuật toán tối ưu hóa Adam

Adam (Adaptive Moment Estimation) là thuật toán phổ biến nhất trong NLP, kết hợp Momentum và RMSProp để tự động điều chỉnh tốc độ học.

$$w_{t+1} = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (3.9)$$

($\hat{m}_t$: Trung bình động của gradient; $\hat{v}_t$: Trung bình động bình phương của gradient.)

Gradient Clipping (Cắt bớt đạo hàm)

Mục đích

Để tránh lỗi **bùng nổ đạo hàm** (*Exploding Gradient*) thường gặp trong mạng RNN/LSTM sâu (khi đạo hàm trở nên quá lớn).

Công thức

Gradient g sẽ được cắt (clip) nếu chuẩn ($\|g\|$) của nó vượt quá ngưỡng tối đa c:

$$g' = g \cdot \frac{c}{\max(c, \|g\|)} \quad (3.10)$$

(g' là Gradient đã được cắt.)

Transformer I

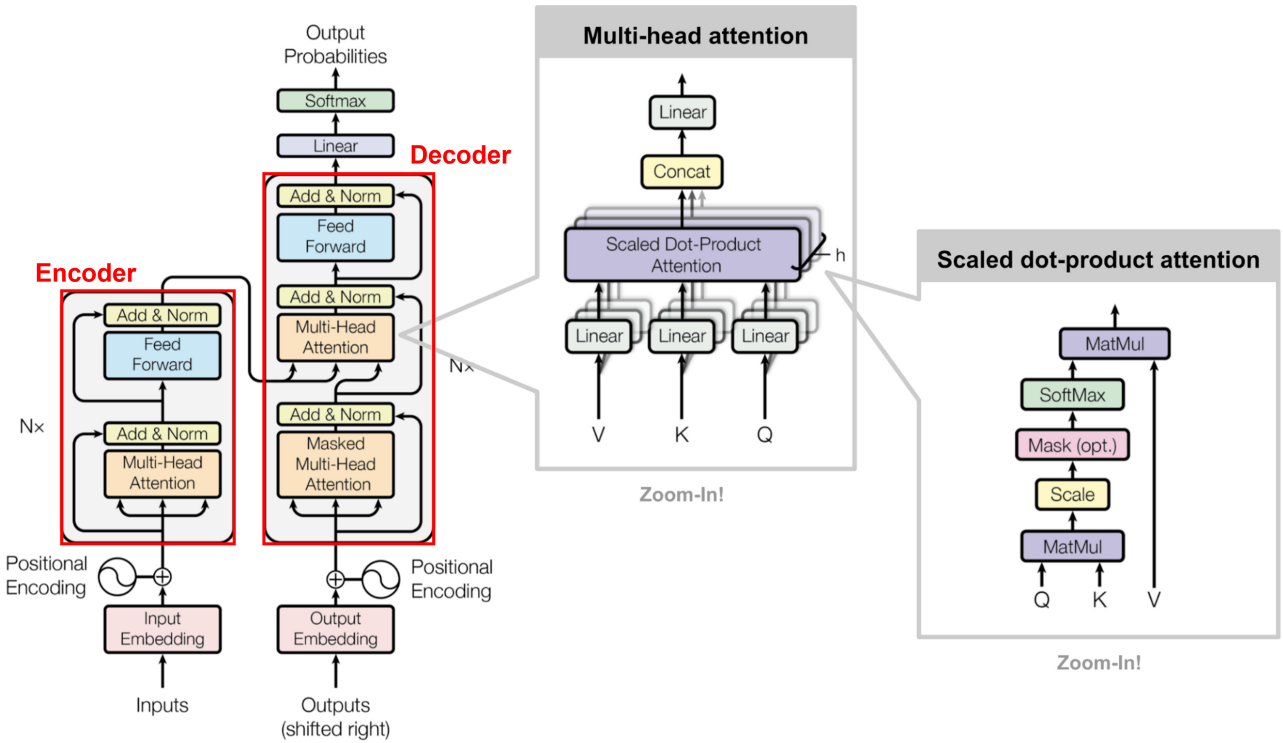
Định nghĩa

Transformer là một kiến trúc mạng nơ-ron được thiết kế để xử lý dữ liệu như văn bản. Nó đã thay thế các mạng tuần tự truyền thống như RNN và LSTM bằng cơ chế chú ý.

Mô hình chuyển đổi chuỗi bằng mạng nơ-ron đều áp dụng cấu trúc encoder-decoder:

- **Encoder** mã hóa: Sẽ chuyển đổi một chuỗi đầu vào gồm biểu diễn ký hiệu (x1, x2, x3, ..., xn) thành một chuỗi các biểu diễn liên tục z = (z1, z2, ..., zn)
- **Decoder** giải mã: Dựa trên z, decoder giải mã sẽ tạo ra một chuỗi đầu ra (y1, y2, ..., ym) của các ký hiệu

Transformer II



Hình 3: Kiến trúc mô hình Transformer

Tokenization I

Định nghĩa

Token hóa là bước tiền xử lý quan trọng, chuyển đổi văn bản thành các đơn vị nhỏ hơn mà mô hình có thể xử lý. Có ba phương pháp chính:

Token theo từ:

1. **Input before tokenization:** ["Machine Learning is fascinating"].
2. **Output when tokenized by words:** ["Machine", "learning", "is", "fascinating"].

Subword-level tokenization:

1. **Byte Pair Encoding (BPE):** Các bước thuật toán cơ bản gồm có:
   - Khởi tạo từ vựng ký hiệu bằng tất cả các ký tự.
   - Gộp cặp ký hiệu xuất hiện nhiều nhất thành một ký hiệu mới.
   - Thêm ký hiệu vừa gộp vào từ vựng.
   - Tiếp tục cho đến khi đạt kích thước từ vựng mục tiêu.

Tokenization II

2. **WordPiece:** Quy trình tổng quan gồm:
   - Khởi tạo từ vựng với từng ký tự riêng lẻ.
   - Tìm cặp ký hiệu tốt nhất sao cho việc gộp giúp tối đa hóa likelihood.
   - Gộp cặp ký hiệu thành một ký hiệu mới, thêm vào từ vựng.
   - Lặp lại cho đến khi đạt kích thước từ vựng mong muốn.
3. **Unigram:** Quy trình phân đoạn bằng Unigram trở nên rõ ràng như sau:
   - Lấy mẫu các đoạn văn bản ngẫu nhiên làm ứng viên của LM.
   - Đánh giá các ứng viên bằng độ đo perplexity – giá trị càng thấp càng tốt.
   - Phân đoạn từ thành tổ hợp ứng viên tối ưu.

Mã hóa (encoder) I

1. Nhúng đầu vào:

1. **Lý do cần embedding:** Máy tính không hiểu trực tiếp câu chữ, nên cần chuyển các từ thành các giá trị số, vector hoặc ma trận để có thể xử lý.
2. **Khái niệm:** Input embedding là quá trình biến mỗi từ trong câu thành một vector số học trong không gian nhiều chiều.
3. **Ý nghĩa:** Các từ có nghĩa tương tự được biểu diễn bằng những vector nằm gần nhau, giúp mô hình hiểu được quan hệ ngữ nghĩa giữa các từ.
4. **Các embedding có sẵn:** Hiện nay có nhiều bộ vector từ được huấn luyện sẵn như GloVe, FastText, Word2Vec, giúp tiết kiệm thời gian và tài nguyên khi làm NLP.
5. **Embedding trong encoder:** Tất cả các lớp encoder nhận đầu vào là các vector có kích thước dmodel = 512.

2. Mã hóa vị trí

Định nghĩa

Mã hóa vị trí là một kỹ thuật bổ sung thông tin về vị trí của các token trong chuỗi vào embedding.

Các mô hình Transformer, chẳng hạn như BERT hay GPT, không có cơ chế tuần tự như RNN, do đó không thể tự nhiên nhận ra thứ tự của các từ trong một câu.
Mã hóa vị trí sử dụng các hàm sin và cos với tần số khác nhau để biểu diễn vị trí:

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

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

3. Cơ chế tự chú ý

1. **Vấn đề của mô hình tuần tự:**
   - Khó ghi nhớ thông tin dài hạn.
   - Khi sinh đầu ra, mô hình chỉ dựa vào một vector trạng thái cuối, dễ mất mát thông tin quan trọng.
2. **Giải pháp:**
   - Cho phép mô hình truy cập trực tiếp vào tất cả trạng thái/tokens trước đó trong chuỗi.
   - Học trọng số chú ý để xác định phần nào của văn bản quan trọng hơn.
   - Giúp mô hình tập trung vào đúng vị trí chứa thông tin cần thiết trong đầu vào.
3. **Ứng dụng trong tóm tắt văn bản**
   - Câu trong bản tóm tắt có thể phụ thuộc vào các câu xa trong văn bản gốc.

- Attention giúp mô hình giữ được toàn bộ ngữ cảnh thay vì nén vào một vector duy nhất.
- Hiệu quả tóm tắt tăng rõ rệt khi kết hợp Attention + RNN.

4. **Transformer – Bước tiến mới**
   - Loại bỏ hoàn toàn RNN.
   - Xử lý văn bản song song, không tuần tự.
   - Tính toán chú ý giữa tất cả các cặp từ, giúp mô hình nắm bắt ngữ cảnh tốt hơn và huấn luyện nhanh hơn.

4. Cơ chế chú ý nhân vô hướng có tỷ lệ

1. **Attention trong Transformer**
   - Transformer dùng Dot-Product Attention nên chỉ là phép tích vô hướng giữa vector Q và K.
   - Tích vô hướng lớn dẫn đến hai vector liên quan mạnh.
   - Phép toán đơn giản, tính toán nhanh và song song được.
2. **Điểm chú ý**

Mã hóa (encoder) V

- Với từng từ ở vị trí i nên tính dot product giữa q_i và tất cả các k_i trong câu.
- Score càng cao nên mức độ liên quan càng mạnh.

3. **Chia cho $\sqrt{d_k}$ để ổn định gradient**
   - Tránh giá trị dot-product quá lớn.
   - Giúp softmax không bị bão hòa nên mô hình học ổn định hơn.
4. **Softmax tạo trọng số chú ý**
   - Chuyển score thành các trọng số dương, tổng = 1.
   - Trọng số cao nên mô hình tập trung nhiều vào từ đó.
   - Từ hiện tại thường có trọng số cao nhất (attending to itself).
5. **Weighted Values (Nhân trọng số với V)**
   - Mỗi vector value v_j được nhân với trọng số softmax tương ứng.
   - Từ liên quan → trọng số lớn → giữ nguyên ảnh hưởng.
   - Từ ít liên quan → trọng số nhỏ → giảm ảnh hưởng.
6. **Ma trận Attention**
   - Lặp cho mọi từ → thu được attention matrix.

- Mỗi hàng là vector chú ý của một từ.
- Ma trận biểu diễn mức độ liên hệ giữa các từ trong toàn câu.

5. Cơ chế chú ý đa đầu

Một từ trong câu có nhiều kiểu quan hệ với các từ khác nên cần nhiều góc nhìn. Một head chỉ học được một dạng quan hệ (ví dụ: ngữ pháp, vị trí, đối tượng...) nên dùng nhiều heads để mô hình học nhiều loại quan hệ song song.

Công thức Multi-Head Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O$$

Trong đó:

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

1. **Ý nghĩa của việc dùng nhiều đầu chú ý**
   - Mỗi head nhìn vào một không gian con biểu diễn khác nhau.

- Tạo ra nhiều ma trận chú ý: $Z_0, Z_1, .., Z_7$
- Sau đó concat lại rồi nhân với W^O để tạo một ma trận duy nhất phù hợp cho các lớp tiếp theo.

2. **Vai trò của Q, K, V trong Multi-Head Attention**
   - **Query (Q):** thứ cần đi tìm/so khớp (giống từ khóa tìm kiếm).
   - **Key (K):** thông tin để so sánh, quyết định mức độ liên quan (như tiêu đề trang web).
   - **Value (V):** nội dung được lấy ra sau khi tính mức độ liên quan (như nội dung trang web).
3. **Quy trình Multi-Head Attention**
   - Chiếu Q, K, V sang h không gian con $\rightarrow$ tạo Q_i, K_i, V_i.
   - Tính attention riêng cho từng head.
   - Nối các kết quả: Concat$(Z_1, ..., Z_h)$.
   - Nhân với ma trận W^O để đưa về kích thước d_{model}

6. Chuẩn hóa và kết nối phần dư

Kết quả được chuẩn hóa bằng LayerNorm trước khi đi đến lớp tiếp theo:

$$Output = LayerNorm(x + SubLayer(x))$$

- Sau khi cộng x vào SubLayer(x), ta áp dụng Layer Normalization để chuẩn hóa đầu ra của lớp.
- Các kết nối phân dư cùng với LayerNorm giúp mô hình tổng quát tốt hơn đối với dữ liệu mới, vì mạng không bị phụ thuộc quá nhiều vào các biến đổi trong từng lớp mà vẫn duy trì được thông tin từ đầu vào.

Mã hóa (encoder) IX

7. Mạng truyền thẳng Mạng Feed-Forward trong Transformer được mô tả bởi công thức sau:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

- Mạng FFN bao gồm hai lớp fully connected (tuyến tính) với trọng số W_1, W_2 tương ứng.
- Phép biến đổi tuyến tính thứ hai ánh xạ không gian trung gian này trở lại không gian đầu ra có kích thước d_{model}.
- Hàm ReLU có định nghĩa ReLU(x) = max(0,x), có tác dụng loại bỏ các giá trị âm và chỉ giữ lại các giá trị dương trong quá trình tính toán.
- Mạng FFN được áp dụng riêng biệt cho mỗi vị trí (token) trong chuỗi, nghĩa là mỗi từ trong chuỗi được xử lý độc lập với nhau.

Mã hóa (encoder) X

- Mặc dù mạng FFN áp dụng cho từng vị trí, nhưng các phép biến đổi này là giống nhau trên tất cả các vị trí, chỉ khác nhau ở các tham số (trọng số và bias) tại mỗi lớp.

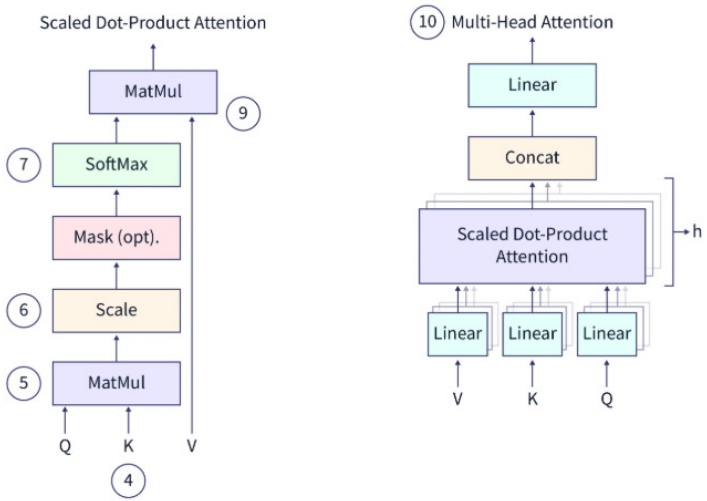
Giải mã (decoder) I

Định nghĩa

Decoder là một thành phần quan trọng của kiến trúc Transformer, đã cách mạng hóa lĩnh vực xử lý ngôn ngữ tự nhiên NLP trong những năm gần đây. Nó đặc biệt nổi tiếng với hiệu suất tiên tiến trong nhiều tác vụ khác nhau, bao gồm dịch máy, mô hình hóa ngôn ngữ và tóm tắt.

- **Cơ chế chú ý đa đầu với mặt nạ**
  - Bộ giải mã (decoder) của Transformer tạo chuỗi đầu ra bằng cách dùng trạng thái ẩn từ bộ mã hóa và các token đã sinh trước đó để dự đoán token tiếp theo.
  - Cơ chế chú ý đa đầu sử dụng ba vectơ có thể học được - Query, Key, Value - lấy ý tưởng từ hệ thống truy vấn-khóa-giá trị trong truy xuất thông tin.
  - Trọng số chú ý được tính bằng tích vô hướng giữa Query và Key tạo ra ma trận điểm, sau đó được scaled để tránh gradient bùng nổ.

Giải mã (decoder) II



- Ma trận điểm được đưa qua softmax để tạo phân phối xác suất; các trọng số này nhân với Value để làm nổi bật các phần tử quan trọng của đầu vào.
- Kết quả được đưa qua lớp tuyến tính, và quá trình tự chú ý được lặp lại theo kiểu đa đầu, cho phép mô hình song song hóa và học được nhiều kiểu quan hệ giữa các phần tử trong chuỗi.

- **Quá trình decoder**
  - **Lớp tuyến tính:** Đầu ra từ khối giải mã được truyền qua một lớp tuyến tính, áp dụng phép biến đổi tuyến tính cho đầu ra.

Giải mã (decoder) III

- **Hàm Softmax:** Nó đảm bảo rằng tổng các giá trị đầu ra bằng 1, và mỗi giá trị biểu thị xác suất của một lớp đầu ra cụ thể.
- **Xác suất đầu ra:** Sau khi đi qua hàm softmax, kết quả đầu ra là một phân phối xác suất trên các lớp đầu ra khả dĩ.
- **Dự đoán đầu ra:** Mã thông báo đầu ra được dự đoán này sau đó được đưa trở lại bộ giải mã làm đầu vào, cùng với các mã thông báo đầu ra đã tạo trước đó và các trạng thái ẩn của bộ mã hóa.

Khái niệm Fine-Tuning

Định nghĩa

Fine-tuning (Tinh chỉnh) là quá trình sử dụng một mô hình đã được huấn luyện trước (*pre-trained model*) làm nền tảng cơ sở (ví dụ: dòng GPT của OpenAI, Llama) để tiếp tục huấn luyện.

Đặc điểm và Quy trình:

- **Dữ liệu:** Huấn luyện thêm trên một tập dữ liệu *nhỏ hơn* và *đặc thù* cho một miền cụ thể.
- **Cơ chế:** Xây dựng dựa trên kiến thức có sẵn, thực hiện chuyển giao tri thức (*Transfer Learning*) từ mô hình gốc sang tác vụ mới.

Lợi ích và Ứng dụng của Fine-Tuning

Lợi ích cốt lõi

- **Hiệu suất cao:** Tận dụng tri thức từ mô hình lớn để đạt độ chính xác tốt hơn trên tác vụ hẹp.
- **Tiết kiệm tài nguyên:** Yêu cầu ít dữ liệu và thời gian tính toán hơn so với việc huấn luyện lại từ đầu (Pre-training from scratch).

Ứng dụng phổ biến trong NLP:

- **Phân loại văn bản:** Phân loại tin tức, thư rác (spam detection).
- **Phân tích cảm xúc:** Đánh giá ý kiến khách hàng (tích cực/tiêu cực).
- **Trả lời câu hỏi (QA):** Chatbot tư vấn, hệ thống hỗ trợ khách hàng.
- **Tóm tắt văn bản:** *(Đây là trọng tâm của đề tài báo cáo này).*

Phân loại các phương pháp Fine-Tuning

Có 3 phương pháp tiếp cận chính trong việc fine-tuning LLM:

1. **Unsupervised Fine-Tuning (Tinh chỉnh không giám sát):**
   - **Dữ liệu:** Sử dụng lượng lớn văn bản *chưa dán nhãn*.
   - **Mục tiêu:** Giúp mô hình hiểu ngôn ngữ của một miền mới (ví dụ: y tế, pháp lý).
   - *Hạn chế:* Kém chính xác hơn cho các tác vụ cụ thể.
2. **Supervised Fine-Tuning - SFT (Tinh chỉnh có giám sát):**
   - **Dữ liệu:** Sử dụng dữ liệu *được dán nhãn* phù hợp với tác vụ mục tiêu.
   - **Ưu điểm:** Hiệu quả cao cho các bài toán cụ thể (phân loại, tóm tắt).
   - *Hạn chế:* Tốn kém chi phí và thời gian gán nhãn dữ liệu.
3. **Instruction Fine-Tuning (Tinh chỉnh theo chỉ dẫn):**
   - **Cơ chế:** Dựa vào các hướng dẫn bằng ngôn ngữ tự nhiên (prompts).
   - **Ứng dụng:** Tạo ra các trợ lý ảo chuyên biệt, tuân thủ tốt hướng dẫn của người dùng.

So sánh Pre-training và Fine-tuning

Tiêu chí	**Pre-training (Tiền huấn luyện)**	**Fine-tuning (Tinh chỉnh)**
Mục tiêu	Xây dựng kiến thức ngôn ngữ chung	Chuyên môn hóa cho tác vụ cụ thể
Dữ liệu	Khổng lồ, đa dạng, chưa dán nhãn (Big Data)	Nhỏ hơn, có dán nhãn, đặc thù cho tác vụ
Phạm vi cập nhật	Huấn luyện toàn bộ tham số của mô hình	Thường chỉ tinh chỉnh các lớp cuối hoặc một phần tham số (PEFT)
Chi phí	Rất cao (Hàng tuần/tháng)	Thấp hơn (Hàng giờ/ngày)
Ví dụ	GPT-4, Llama 3	Fine-tuning Llama 3 cho tóm tắt văn bản

Bảng 1: Sự khác biệt giữa Pre-training và Fine-tuning

Tầm quan trọng của Fine-Tuning LLMs

Việc fine-tuning mang lại **7 lợi ích cốt lõi** giúp tối ưu hóa mô hình ngôn ngữ lớn:

1. **Transfer Learning:** Tận dụng kiến thức *pre-training*, thích ứng tác vụ cụ thể với thời gian và tài nguyên giảm.
2. **Giảm yêu cầu dữ liệu:** Cần ít dữ liệu dán nhãn hơn, tập trung điều chỉnh các tính năng cho tác vụ mục tiêu.
3. **Cải thiện khả năng tổng quát hóa:** Nâng cao khả năng xử lý các tác vụ hoặc miền chuyên biệt.
4. **Triển khai mô hình hiệu quả:** Tối ưu hóa về mặt tính toán cho các ứng dụng thực tế.
5. **Khả năng thích ứng:** Xử lý đa dạng tác vụ mà không cần thay đổi kiến trúc mô hình.
6. **Hiệu suất miền cụ thể:** Xuất sắc trong các lĩnh vực đặc thù nhờ nắm bắt được sắc thái và từ vựng chuyên ngành.
7. **Hội tụ nhanh hơn:** Bắt đầu với trọng số đã học $\rightarrow$ đạt điểm hội tụ nhanh hơn.

Retrieval Augmented Generation (RAG)

Tổng quan

RAG là phương pháp tận dụng dữ liệu riêng bằng cách đưa thông tin vào *prompt* khi truy vấn, thay vì chỉ dựa hoàn toàn vào kiến thức từ dữ liệu huấn luyện (Pre-trained data).

Lợi ích và Cơ chế:

- **Truy xuất thời gian thực:** Hệ thống tìm kiếm thông tin liên quan từ bên ngoài ngay tại thời điểm truy vấn.
- **Tối ưu chi phí:** Giúp các tổ chức sử dụng dữ liệu nội bộ để nâng cao kết quả mà **không tốn chi phí** cho quá trình Fine-tuning hay Pre-training lại từ đầu.

Cân nhắc lựa chọn: RAG vs Fine-Tuning

Việc lựa chọn phụ thuộc vào mục tiêu cụ thể của ứng dụng, dữ liệu sẵn có và yêu cầu về tính cập nhật:

Ưu tiên RAG khi:

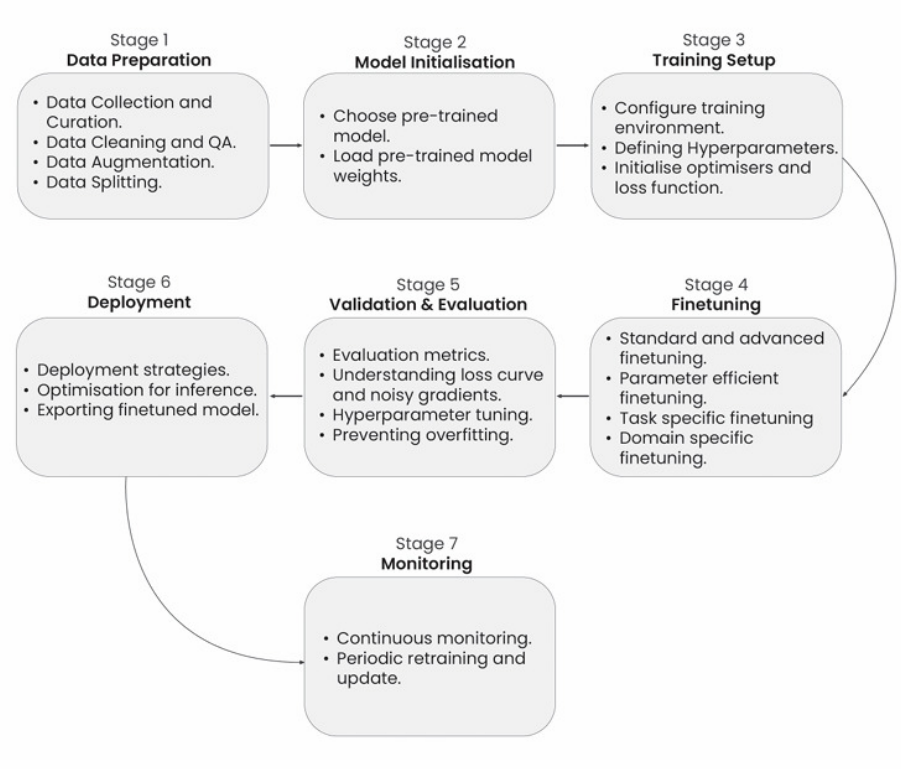
- **Dữ liệu động:** Cần truy cập dữ liệu thay đổi thường xuyên hoặc theo thời gian thực.
- **Minh bạch:** Yêu cầu trích dẫn nguồn gốc rõ ràng (giảm thiểu ảo giác).
- **Ngữ cảnh ngoài:** Cần truy cập nguồn dữ liệu bên ngoài mô hình gốc.

Ưu tiên Fine-Tuning khi:

- **Hành vi & Phong cách:** Cần điều chỉnh giọng văn, cấu trúc phản hồi cụ thể.
- **Kiến thức sâu:** Cần "nhúng"kiến thức miền đặc thù vào trọng số mô hình.
- **Dữ liệu có sẵn:** Có lượng lớn dữ liệu dán nhãn chất lượng cao để huấn luyện.

Quy trình Fine-Tuning: Tổng quan

Fine-tuning là một quy trình toàn diện gồm 7 giai đoạn riêng biệt để thích ứng mô hình pre-trained với tác vụ cụ thể.



Hình 4: Minh họa quy trình Fine-tuning

Quy trình Fine-Tuning (Giai đoạn 1 - 3)

- **Chuẩn bị Dữ liệu (Dataset Preparation)**
  - Làm sạch và định dạng dữ liệu (ví dụ: JSONL) khớp với tác vụ mục tiêu.
  - Cấu trúc cặp `<input, output>` phản ánh hành vi mong muốn.
  - *Ví dụ:* `###Human: <Query> ###Assistant: <Response>`

- **Khởi tạo Mô hình (Model Initialisation)**
  - Thiết lập tham số và cấu hình ban đầu từ Pre-trained Model.
  - Mục tiêu: Đảm bảo hiệu suất tối ưu, tránh bùng nổ gradient.

- **Thiết lập Môi trường (Training Environment Setup)**
  - Cấu hình cơ sở hạ tầng (GPU/TPU).
  - Xác định kiến trúc và siêu tham số (*hyperparameters*).

Quy trình Fine-Tuning (Giai đoạn 4)

- **Fine-Tuning (Partial or Full)**

  Thực hiện cập nhật tham số mô hình sử dụng dữ liệu đặc thù:

Các phương pháp chính

- **Full fine-tuning:** Cập nhật toàn bộ các lớp tham số của mô hình (tốn kém tài nguyên).
- **Partial / PEFT (Parameter-Efficient Fine-Tuning):** Chỉ cập nhật một phần nhỏ tham số hoặc thêm các lớp adapter (ví dụ: LoRA, QLoRA).

  212 *Ưu điểm:* Tiết kiệm tài nguyên tính toán và lưu trữ.

Quy trình Fine-Tuning (Giai đoạn 5 - 7)

- **Đánh giá và Kiểm định (Evaluation and Validation)**
  - Kiểm tra trên tập dữ liệu chưa từng thấy (*unseen data*).
  - Theo dõi chỉ số (cross-entropy, loss curves) để phát hiện *overfitting* hoặc *underfitting*.
- **Triển khai (Deployment)**
  - Tối ưu hóa mô hình cho phần cứng đích (Quantization, Pruning).
  - Thiết lập API, tích hợp hệ thống và bảo mật.
- **Giám sát và Bảo trì (Monitoring and Maintenance)**
  - Theo dõi độ trễ, chi phí và chất lượng câu trả lời thực tế.
  - Cập nhật lại mô hình khi dữ liệu hoặc yêu cầu thay đổi.

Nhu cầu của PEFT (Parameter-Efficient Fine-Tuning)

Trong bối cảnh LLM ngày càng lớn (ví dụ: Llama 3.1 hàng tỷ tham số), **Full Fine-tuning** gặp 2 rào cản lớn:

1. **Chi phí phần cứng:** Yêu cầu bộ nhớ VRAM khổng lồ để lưu trữ gradient và trạng thái tối ưu hóa.
2. **Lãng quên thảm khốc (Catastrophic Forgetting):** Mô hình quên mất kiến thức cũ khi học dữ liệu mới.

Giải pháp PEFT

Chỉ cập nhật một tập hợp nhỏ các tham số trong khi **đóng băng (freeze)** hầu hết trọng số của mô hình gốc.

LoRA: Nguyên lý hoạt động

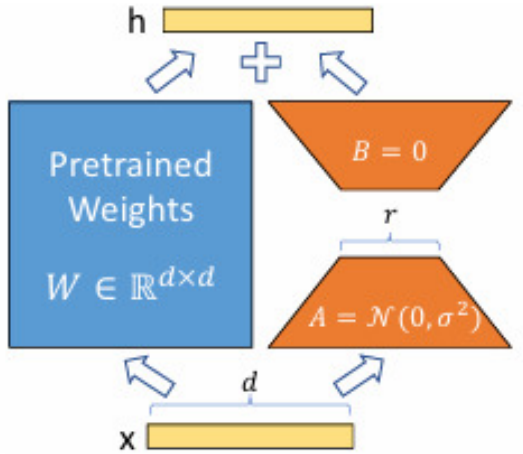
Giả thuyết: Sự thay đổi trọng số có "hạng nội tại thấp"(*low intrinsic rank*).

Phân rã Ma trận:

$$W = W_0 + \Delta W = W_0 + BA$$

Trong đó:

- $W_0 \in \mathbb{R}^{d \times k}$: Trọng số gốc (đóng băng).
- $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}$: Tham số huấn luyện với hạng $r \ll \min(d, k)$.



Hình 5: Sơ đồ LoRA

LoRA: Chiến lược và Triển khai

- **Chiến lược Khởi tạo (Initialization):** Để đảm bảo bắt đầu huấn luyện $\Delta W = 0$ (giữ nguyên hành vi gốc):
  - Ma trận A: Phân phối Gaussian ngẫu nhiên.
  - Ma trận B: Khởi tạo bằng số 0.

- **Hệ số tỷ lệ (Scaling Factor):** Áp dụng $\frac{\alpha}{r}$ cho kết quả $\Delta W x$ giúp giảm thiểu việc điều chỉnh lại siêu tham số khi thay đổi r.

- **Không độ trễ suy luận (Zero Inference Latency):**

Merge Weights

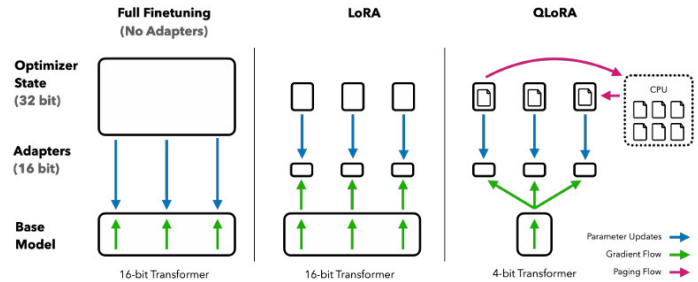
Khi triển khai, tính toán trước $W_{new} = W_0 + BA$. Mô hình chạy như bình thường, không tốn thêm chi phí tính toán phụ.

QLoRA: Tối ưu hóa bộ nhớ cực đại

Mục tiêu: Fine-tune mô hình 65 tỷ tham số trên một GPU 48GB duy nhất mà vẫn giữ hiệu suất tương đương 16-bit.

Cơ chế cốt lõi:

- Backpropagation gradient qua mô hình gốc đã bị **đóng băng** và **lượng tử hóa 4-bit**.
- Chỉ cập nhật các trọng số Adapter (LoRA).



QLoRA: 3 Cải tiến Kỹ thuật Đột phá

1. **4-bit NormalFloat (NF4):**
   - *Vấn đề:* Int4/Float4 không tối ưu cho trọng số phân phối chuẩn.
   - *Giải pháp:* NF4 tối ưu về lý thuyết thông tin, đảm bảo mỗi bin lượng tử hóa có lượng giá trị bằng nhau.
   - *Kết quả:* Độ chính xác cao hơn FP4/Int4.
2. **Lượng tử hóa Kép (Double Quantization):**
   - "Lượng tử hóa các hằng số lượng tử hóa"(từ 32-bit xuống 8-bit).
   - Tiết kiệm trung bình ~ 0.37 bits/tham số.
3. **Bộ tối ưu hóa Phân trang (Paged Optimizers):**
   - Sử dụng Unified Memory để tự động chuyển trạng thái optimizer sang **CPU RAM** khi GPU quá tải.
   - **Lợi ích:** Ngăn chặn lỗi Out-of-Memory (OOM) khi huấn luyện chuỗi dài.

So sánh: LoRA vs QLoRA

Bảng 2: Bảng so sánh kỹ thuật và hiệu quả

gray!10 **Đặc điểm**	**LoRA (2021)**	**QLoRA (2023)**
Trạng thái gốc	Đóng băng 16-bit (FP16/BF16).	Đóng băng & nén **4-bit** (NF4).
Cơ chế Adapter	Ma trận A, B (FP16/FP32).	Ma trận A, B (BF16).
Bộ nhớ (VRAM)	Giảm ~3 lần so với Full FT.	Giảm ~**18 lần** (96 $\rightarrow$ 5.2 bits/tham số).
Độ chính xác	Tương đương/tốt hơn Full FT.	Tương đương Full FT 16-bit.
Phần cứng	Cần GPU cao cấp (A100, V100).	Chạy tốt trên GPU phổ thông (RTX 3090, T4).
Llama 3	Tốt, nhưng tốn VRAM.	**Tối ưu nhất** cho máy cá nhân/Colab.

Kết luận

Kết luận lựa chọn

Nếu bạn có tài nguyên hạn chế (GPU $<$ 24GB VRAM) hoặc muốn thử nghiệm nhanh trên các mô hình lớn (70B+), **QLoRA** là lựa chọn bắt buộc. Nếu cần độ ổn định tuyệt đối trong môi trường sản xuất lớn, **LoRA** tiêu chuẩn vẫn là giải pháp an toàn.

MÔ HÌNH THỰC NGHIỆM

Thiết lập quy trình huấn luyện Llama 3.1 8B trên Kaggle với kỹ thuật QLoRA

Dữ liệu Huấn luyện: Vietnamese Text Summaries

Nguồn dữ liệu:

- Tập dữ liệu lớn từ báo điện tử VN (Kaggle).
- Đa dạng lĩnh vực: Kinh tế, Công nghệ, Y tế, Xã hội.
- *Vấn đề:* Nhiễu (HTML, URL), cấu trúc không đồng nhất.

Quy trình Làm sạch (Cleaning Pipeline):

1. **Định dạng:** Chuyển về mã hóa thống nhất (Unicode).
2. **Lọc nhiễu:** Xóa thẻ HTML, URL, ký tự đặc biệt.
3. **Chuẩn hóa:** Chuyển về chữ thường (lowercase), loại bỏ stopwords tiếng Việt.
4. **Tokenization:** Tách từ để chuẩn bị cho Embedding.

Kết quả: Dữ liệu sạch, nhất quán, sẵn sàng cho các thuật toán biểu diễn văn bản.

Mô hình và Môi trường Phần cứng

Mô hình Cơ sở (Base Model)

Llama 3.1 8B (`meta-llama/Meta-Llama-3.1-8B`): Cân bằng tốt giữa kích thước tham số và khả năng suy luận tóm tắt.

Hạ tầng Kaggle Thách thức:

- **GPU:** 2x NVIDIA Tesla T4 (16GB VRAM/card → Tổng 32GB).
- **Vấn đề:** Model Llama 3.1 8B (FP16) cần > 16GB chỉ để load trọng số (chưa tính Gradient). → Không thể load trên 1 card T4 đơn lẻ.
- **Giải pháp:** Bắt buộc dùng **QLoRA (4-bit)** để nén mô hình và chạy song song (Data Parallelism).

Triển khai Kỹ thuật QLoRA

Thiết lập nghiêm ngặt để tối ưu hóa trên NVIDIA T4:

1. **Lượng tử hóa 4-bit (Quantization):**
   - Định dạng: **NF4** (NormalFloat 4-bit) - Tối ưu lý thuyết thông tin.
   - Kích hoạt **Double Quantization** để tiết kiệm VRAM tối đa.
   - `bnb_4bit_compute_dtype`: Tối ưu cho kiến trúc Turing.

2. **Cấu hình Adapter (PEFT):**
   - **Rank (r): 8** (Cân bằng giữa tài nguyên và khả năng học).
   - **Alpha (α): 16** (Hệ số tỷ lệ $\alpha/r = 2$ giúp ổn định).
   - **Target Modules:** `[q_proj, k_proj, v_proj, o_proj]` - Áp dụng toàn diện cho các lớp Attention.

Chiến lược Huấn luyện (Training Strategy)

Với hạn chế bộ nhớ 16GB VRAM, chúng tôi áp dụng chiến lược:

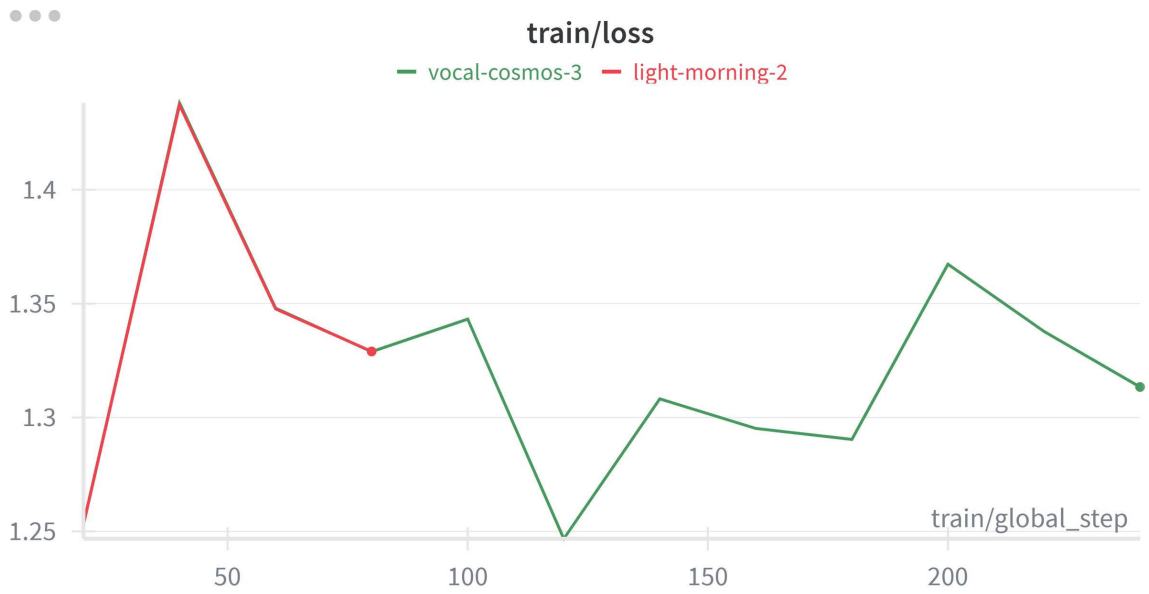
- **Micro-Batch Size:** `per_device = 1` (Mỗi GPU xử lý 1 mẫu/lần).
- **Gradient Accumulation:** `steps = 16`.

  212 *Effective Batch Size* $= 1 \times 2$ GPUs $\times 16 = 32$. Giúp hội tụ ổn định.
- **Paged Optimizers:** Sử dụng `paged_adamw_32bit` để đẩy trạng thái Optimizer sang CPU RAM khi GPU bị tràn (OOM protection).

Giám sát (WandB):

- Theo dõi *Training Loss*, *VRAM Usage*, và *GPU Temperature* thời gian thực.

Kết quả Thực nghiệm: Phân tích Loss



Hình 6: Diễn biến hàm mất mát (Training Loss)

Nhận xét:

- Xu hướng loss giảm rõ rệt (từ > 1.45 xuống ~ 1.31 ở bước 240).
- Biến động cục bộ là bình thường do Batch Size nhỏ, nhưng xu hướng tổng thể chứng tỏ mô hình đang hội tụ tốt.

Kết quả: So sánh ROUGE và BERTScore

Kết quả trên tập kiểm thử (Test Set) cho thấy sự cải thiện toàn diện:

Metric	**Base Model**	**Fine-tuned**	**Cải thiện**
ROUGE-1	0.5095	**0.5330**	+2.4%
ROUGE-2	0.3055	0.3140	+0.9%
ROUGE-L	0.3554	0.3657	+1.0%
BERTScore (F1)	0.7124	**0.7376**	**+2.5%**

Bảng 3: Bảng so sánh hiệu suất

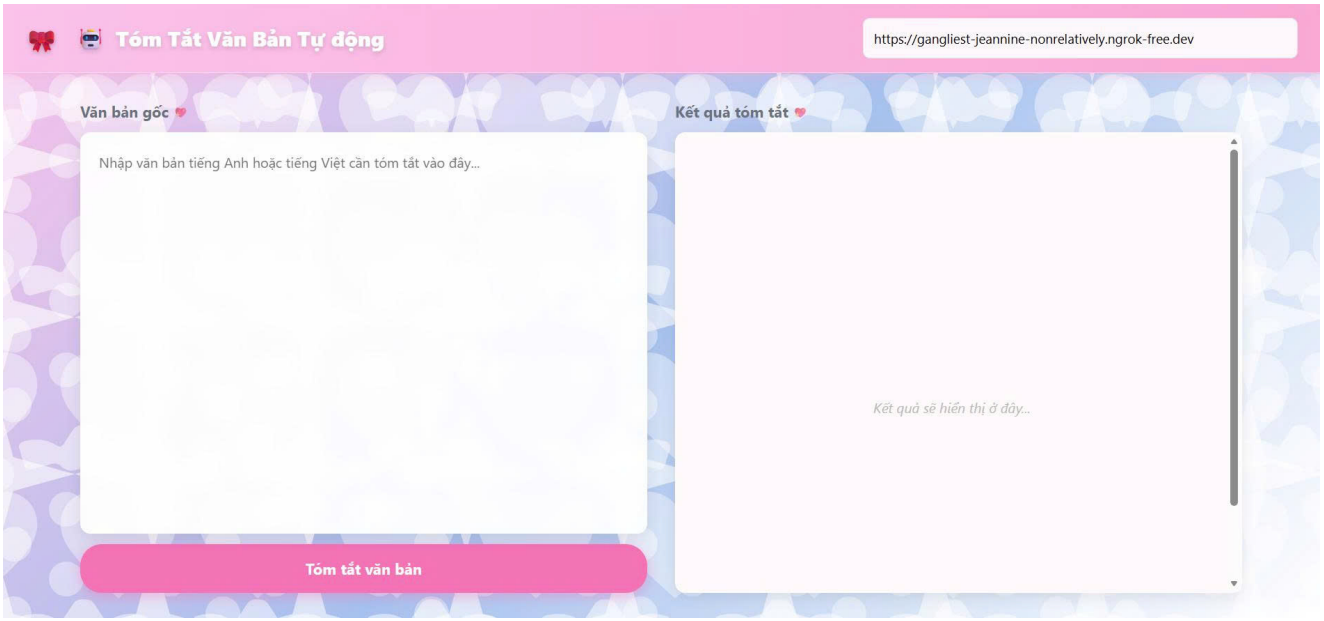
Kết luận

Phân tích:

- **ROUGE-1 (+2.4%):** Cải thiện đáng kể khả năng nắm bắt từ khóa.
- **BERTScore (+2.5%):** Mức tăng cao nhất. Chứng tỏ Fine-tuning giúp mô hình **hiểu ngữ nghĩa** và diễn đạt lại tốt hơn, thay vì chỉ sao chép từ vựng.

Giới thiệu Giao diện Ứng dụng

- **Tên ứng dụng:** "Tóm Tắt Văn Bản Giao diện tối giản, trực quan.
- **Chức năng chính:**
  - **Cấu hình API:** Ô nhập URL Ngrok để kết nối Server GPU (Colab).
  - **Nhập liệu:** Hỗ trợ dán văn bản tiếng Việt/Anh dài.
  - **Xử lý:** Nút bấm gọi API có hiệu ứng Loading (Spinner).



Hình 7: Giao diện Website

Đặc điểm kỹ thuật của Hệ thống

- **Mô hình nền tảng:** Sử dụng **Llama 3.1** (Hugging Face).
  → Phiên bản đã được tinh chỉnh (*Fine-tuned*) chuyên biệt cho nhiệm vụ tóm tắt trừu tượng (*Abstractive Summarization*).

- **Kiến trúc cốt lõi:** Dựa trên **Transformer**.
  - Cho phép xử lý song song dữ liệu.
  - Nắm bắt mối quan hệ phụ thuộc dài (*Long-Range Dependencies*) trong các văn bản phức tạp.

- **Cơ chế hoạt động:** Sử dụng **Self-Attention**.
  → Đánh trọng số cho từng từ trong văn bản gốc để xác định các ý chính quan trọng nhất.

Đặc điểm kỹ thuật của Hệ thống

Tối ưu hóa bộ nhớ (Quantization)

Mô hình được lượng tử hóa xuống **4-bit** sử dụng thư viện `bitsandbytes`.

- Giảm thiểu đáng kể yêu cầu bộ nhớ GPU (VRAM).
- Đảm bảo tốc độ suy luận nhanh mà vẫn giữ hiệu suất cao.

Cơ chế hoạt động của Hệ thống

Kiến trúc tổng quan: Client-Server kết nối qua API Gateway (Ngrok).

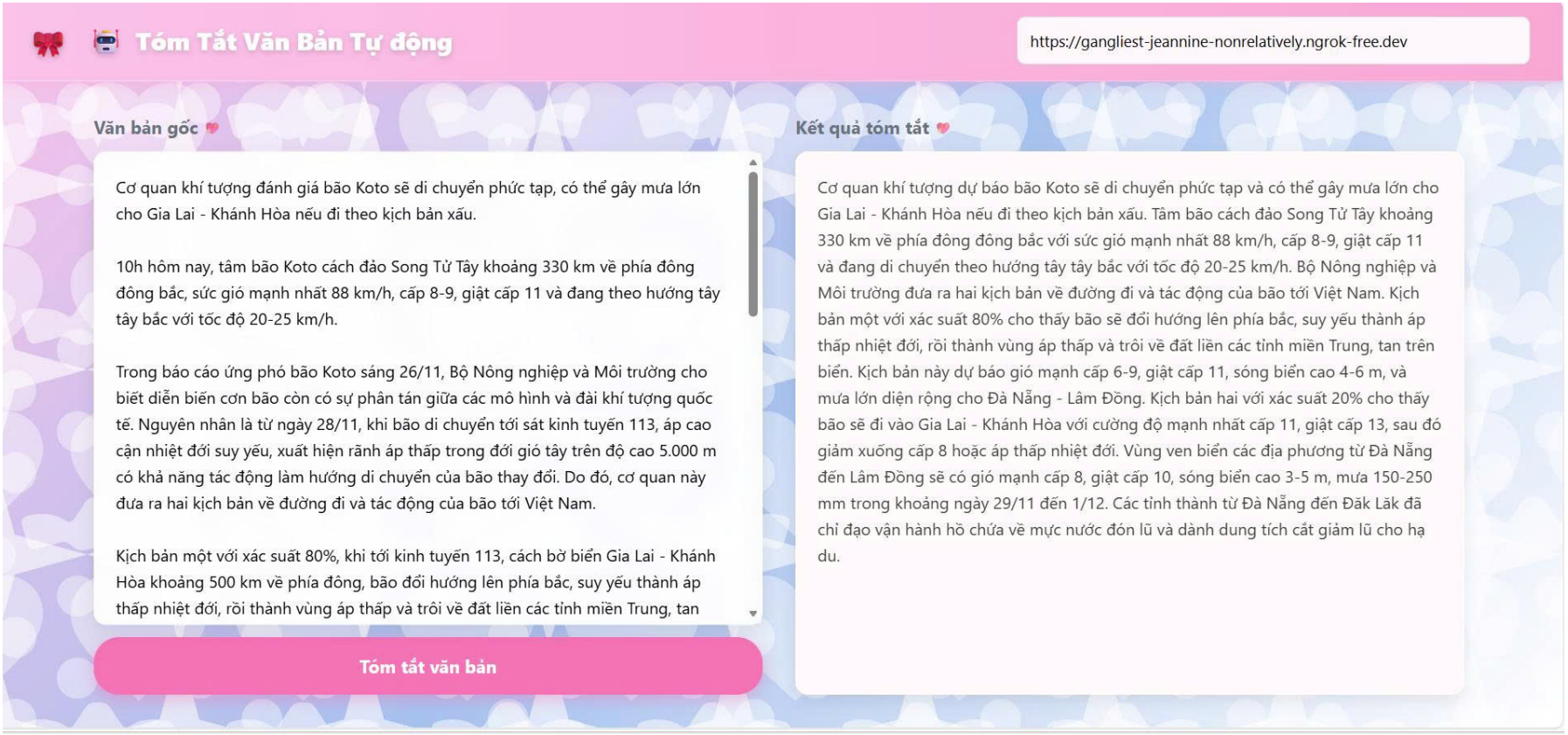
1. Phía Người dùng (Frontend)

1. **Kết nối API:** Người dùng nhập URL Ngrok để thiết lập cầu nối với Server.
2. **Gửi yêu cầu:** Người dùng nhấn "Tóm tắt"→ JS gọi hàm `summarizeText()` → Gửi `HTTP POST` đến endpoint `/summarize`.
3. **Cập nhật giao diện:** Nhận JSON phản hồi → Hiển thị kết quả vào khung `div#outputText` (Cơ chế Asynchronous).

2. Phía Máy chủ (Backend - Colab)

4. **Xử lý yêu cầu (Flask):** Nhận request → Trích xuất văn bản gốc → Đóng vai trò lớp trung gian (Middleware).
5. **Suy luận (Inference):** Tải model Llama 3.1 (4-bit) → Chạy hàm `model.generate` để sinh văn bản tóm tắt.
6. **Trả kết quả:** Đóng gói kết quả dạng JSON: `{"summary": "..."}` và phản hồi lại Client.

Demo Kết quả thực tế trên Website



Hình 8: Kết quả tóm tắt hiển thị trên giao diện người dùng