TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

---------- □□□ ----------

BÁO CÁO ĐỒ ÁN CUỐI KỲ

HỌC PHẦN: Xử Lý Ngôn Ngữ Tự Nhiên.

<u>ĐỀ TÀI:</u> ***Dịch máy Anh – Pháp / Anh – Đức Với Mô Hình ENCODER – DECODER LSTM.***

Giáo Viên: Thầy Nguyễn Tuấn Đăng.

Thành viên:

Tô Gia Huy - 3123410131.

Hồ Hoàng Khang - 3123410150.

TP HỒ CHÍ MINH, THÁNG 12 NĂM 2025.

Mục lục

Lời cảm ơn

Lời đầu tiên, nhóm chúng em xin trân trọng cảm ơn giảng viên Nguyễn Tuấn Đăng - người đã trực tiếp chỉ bảo, hướng dẫn nhóm trong quá trình hoàn thành bài tiểu luận này.

Bài tiểu luận về đề tài: *"Dịch máy Anh – Pháp/ Anh – Đức với mô hình ENCODER DECODER LSTM"* thuộc bộ môn Xử lý ngôn ngữ tự nhiên là kết quả của quá trình học tập, tiếp thu kiến thức tại trường, lớp và cả những tìm tòi, nghiên cứu riêng của bản thân chúng em và sự chỉ dạy tận tình của thầy - người đã trực tiếp hướng dẫn chúng em trong môn học này.

Thông qua bộ môn Xử lý ngôn ngữ tự nhiên nói chung, và đề tài này nói riêng, nhóm chúng em đã phần nào hiểu thêm được về các kiến thức liên quan đến bộ môn và lĩnh vực xử lý ngôn ngữ tự nhiên, đồng thời nhóm cũng đã có thể vận dụng những lý thuyết được học để giải quyết các vấn đề thực tế trong đời sống.

Mặc dù đã có những đầu tư nhất định trong quá trình làm bài song cũng khó có thể tránh khỏi những sai sót, nhóm chúng em kính mong nhận được ý kiến đóng góp của thầy để bài tiểu luận có thể được được hoàn thiện hơn.

Nhóm chúng em xin chân thành cảm ơn thầy vì sự nhiệt tình giảng dạy và giúp đỡ trong quá trình hoàn thành bài tiểu luận này!

Chương 1: Tổng quan đề tài.

1.1 Giới thiệu về xử lý ngôn ngữ tự nhiên.

Xử lý ngôn ngữ tự nhiên (Natural Language Processing – NLP) là một lĩnh vực đầy tiềm năng trong nghiên cứu và ứng dụng trí tuệ nhân tạo. Lĩnh vực này tập trung vào việc xây dựng các hệ thống có khả năng tương tác với con người thông qua ngôn ngữ tự nhiên. Mục tiêu chính của NLP là phát triển các mô hình giúp máy tính không chỉ hiểu và diễn giải, mà còn có thể tự động tạo ra ngôn ngữ tự nhiên, từ đó nâng cao trải nghiệm giao tiếp tự nhiên và hiệu quả cho người dùng.

NLP bao gồm nhiều yếu tố và công nghệ then chốt. Xử lý ngôn ngữ tự nhiên là quá trình phân tích và xử lý văn bản nhằm bóc tách từ, phân tích cú pháp, phân loại ngữ nghĩa, trích xuất thông tin và thực hiện truy vấn ngôn ngữ. Hiểu ngôn ngữ tự nhiên là một phần quan trọng của NLP, giúp hệ thống nhận diện và hiểu nội dung của câu, xác định ngữ cảnh, phân tích ý nghĩa cũng như nhận diện các thực thể như người, địa điểm hoặc tổ chức. Bên cạnh đó, tạo ngôn ngữ tự nhiên là quá trình mà máy tính sinh ra câu từ ngôn ngữ tự nhiên dựa trên dữ liệu, phục vụ cho việc trả lời câu hỏi, viết bài tự động hoặc tóm tắt nội dung.

Dịch máy, một ứng dụng phổ biến trong NLP, là quá trình tự động chuyển đổi văn bản từ một ngôn ngữ sang ngôn ngữ khác. Sự phát triển mạnh mẽ của các mô hình như Transformer đã mang lại những cải tiến đáng kể trong lĩnh vực này. Học máy (Machine Learning) trong NLP đóng vai trò quan trọng, với các phương pháp có giám sát và không giám sát, giúp hệ thống học hỏi và hiểu ngôn ngữ một cách linh hoạt, hiệu quả.

Ứng dụng của NLP hiện nay rất phong phú và đa dạng, từ hệ thống tìm kiếm thông tin, chatbot, đến các hệ thống phân tích cảm xúc và tóm tắt văn bản. Những tiến bộ vượt bậc trong NLP, đặc biệt nhờ vào học sâu (Deep Learning) và dữ liệu lớn, đã mở ra những khả năng mới cho lĩnh vực này, đồng thời mở rộng phạm vi ứng dụng trong nhiều ngành công nghiệp và dịch vụ.

Tuy nhiên, NLP cũng đang đối mặt với nhiều thách thức, đặc biệt là trong việc hiểu ngữ nghĩa sâu sắc, chuẩn hóa dữ liệu, và đảm bảo tính công bằng và đạo đức trong các ứng dụng. Các vấn đề như sự thiên vị trong dữ liệu, sự hiểu nhầm ngữ cảnh, và các vấn đề đạo đức liên quan đến việc sử dụng NLP (ví dụ như quyền riêng tư, bảo mật thông tin) cần được giải quyết để đảm bảo sự phát triển bền vững. Ngoài ra, tương lai của NLP hứa hẹn sẽ chứng kiến sự phát triển của các mô hình mới như BERT, GPT, hay các phương pháp học sâu khác, giúp cải thiện độ chính xác và tính linh hoạt trong việc xử lý và tạo ngôn ngữ tự nhiên.

Sự phát triển của NLP phụ thuộc vào khả năng giải quyết những vấn đề này, nhằm mang lại lợi ích tối ưu cho người dùng và xã hội.

1.2. Giới thiệu về đồ án.

1.2.1 Dịch máy là gì?

Dịch máy tự động (Machine Translation - MT) là một trong những ứng dụng quan trọng nhất của lĩnh vực Xử lý Ngôn ngữ Tự nhiên (NLP), có vai trò then chốt trong việc phá vỡ rào cản ngôn ngữ và thúc đẩy giao tiếp toàn cầu.

Trong lịch sử, MT đã trải qua nhiều giai đoạn phát triển, từ các hệ thống dựa trên quy tắc (Rule-Based MT) cho đến các hệ thống dựa trên thống kê (Statistical MT - SMT). Tuy nhiên, cả hai phương pháp này đều gặp phải những hạn chế nhất định, đặc biệt là trong việc xử lý ngữ cảnh câu dài và tạo ra bản dịch có độ trôi chảy, tự nhiên.

Sự ra đời của **Dịch máy Nơ-ron (Neural Machine Translation - NMT)** đã tạo ra bước đột phá. NMT sử dụng các mô hình học sâu để học cách biểu diễn ngữ nghĩa của toàn bộ câu (thay vì chỉ các cụm từ nhỏ), từ đó tạo ra bản dịch chất lượng cao và mạch lạc hơn đáng kể so với các phương pháp truyền thống.

1.2.2 Tại sao dịch thuật tự động quan trọng?

Trong kỷ nguyên toàn cầu hóa và bùng nổ thông tin kỹ thuật số, nhu cầu giao tiếp xuyên biên giới đã thúc đẩy sự cần thiết của các giải pháp dịch thuật nhanh chóng và chính xác. Dịch máy tự động (Machine Translation - MT) ra đời để giải quyết bài toán này, đóng vai trò then chốt vì những lý do sau:

- **Phá vỡ Rào cản Ngôn ngữ:** MT là công cụ thiết yếu để kết nối cộng đồng, cho phép mọi người tiếp cận kiến thức, văn hóa và dịch vụ từ khắp nơi trên thế giới mà không cần thành thạo ngôn ngữ nguồn.
- **Xử lý Khối lượng Lớn:** MT cung cấp giải pháp duy nhất để xử lý và dịch một lượng lớn dữ liệu văn bản (như tài liệu doanh nghiệp, email, và nội dung web) trong thời gian ngắn, vượt xa khả năng của dịch thuật viên con người.
- **Nâng cao Hiệu suất Kinh doanh:** Đối với các tập đoàn đa quốc gia, MT giúp tự động hóa quá trình bản địa hóa sản phẩm (localization) và hỗ trợ khách hàng, tiết kiệm chi phí và thời gian đáng kể.
- **Cải tiến Đổi mới Công nghệ:** Sự phát triển của MT, đặc biệt là sự chuyển dịch sang **Dịch máy Nơ-ron (NMT)**, đã chứng minh những tiến bộ vượt bậc trong lĩnh vực Trí tuệ Nhân tạo. NMT sử dụng các mô hình học sâu như **Encoder-Decoder LSTM** để học cách biểu diễn ngữ nghĩa của toàn bộ câu, từ đó tạo ra bản dịch có độ trôi chảy và ngữ cảnh chính xác hơn hẳn các phương pháp dựa trên quy tắc và thống kê truyền thống.

Việc nghiên cứu và triển khai mô hình **Encoder-Decoder LSTM** trong đồ án này không chỉ là một bài tập kỹ thuật mà còn là đóng góp vào việc hiểu sâu hơn về cơ chế hoạt động của các hệ thống NMT hiện đại.

1.2.3 Ứng dụng cụ thể của dịch tự động Anh -Pháp.

- Thương mại và Hợp tác Quốc tế

- **Bản địa hóa Nội dung (Content Localization):** Dịch tự động các tài liệu marketing, hướng dẫn sử dụng sản phẩm, và giao diện website/ứng dụng từ tiếng Anh sang tiếng Pháp (ngôn ngữ chính thức tại Pháp, Canada, và nhiều nước châu Phi) để tiếp cận thị trường lớn hơn.
- **Thương mại Điện tử:** Dịch mô tả sản phẩm, đánh giá của khách hàng, và điều khoản dịch vụ, giúp người bán và người mua ở hai khu vực ngôn ngữ giao dịch dễ dàng.
- **Tài liệu Pháp lý và Công nghệ:** Hỗ trợ dịch nhanh các hợp đồng, văn bản quy phạm pháp luật, và các tài liệu kỹ thuật chuyên ngành từ các tổ chức quốc tế (ví dụ: Liên hợp quốc, Tổ chức Thương mại Thế giới).

- Giáo dục và Nghiên cứu

- **Tiếp cận Tài nguyên Học thuật:** Dịch nhanh các bài báo khoa học, sách giáo khoa, và tài liệu nghiên cứu từ tiếng Anh (ngôn ngữ khoa học phổ biến) sang tiếng Pháp, giúp các nhà nghiên cứu và sinh viên francophone dễ dàng tiếp cận kiến thức.
- **Hỗ trợ Học tập Ngôn ngữ:** Cung cấp công cụ dịch tức thời để học sinh/sinh viên luyện tập và hiểu các cấu trúc câu phức tạp giữa hai ngôn ngữ.

- Du lịch và Giao tiếp Cá nhân

- **Dịch thời gian thực:** Cung cấp nền tảng cho các ứng dụng dịch trên thiết bị di động, hỗ trợ du khách giao tiếp tại các quốc gia nói tiếng Pháp.
- **Truyền thông Xã hội:** Hỗ trợ dịch các bài đăng, bình luận trên các nền tảng mạng xã hội, giúp người dùng Anh và Pháp dễ dàng tương tác.

- Phát triển Công nghệ

- **Xây dựng Công cụ NLP Khác:** Hệ thống dịch là nền tảng để phát triển các công cụ NLP phức tạp hơn như tóm tắt văn bản song ngữ, trích xuất thông tin đa ngôn ngữ, và phát triển các mô hình ngôn ngữ lớn (LLMs).

Tóm lại: Việc triển khai thành công mô hình Encoder-Decoder LSTM không chỉ chứng minh khả năng xử lý các bài toán chuỗi-sang-chuỗi (sequence-to-sequence)

mà còn tạo ra một công cụ có **giá trị thực tế cao**, góp phần vào sự lưu thông thông tin và tri thức giữa cộng đồng nói tiếng Anh và tiếng Pháp.

1.2.3 Thách thức trong dịch ngôn ngữ Anh – Pháp.

Mặc dù cả tiếng Anh và tiếng Pháp đều là các ngôn ngữ Ấn-Âu, việc dịch thuật giữa hai ngôn ngữ này vẫn đặt ra nhiều thách thức đáng kể đối với các mô hình Dịch máy Nơ-ron (NMT). Việc nhận diện các thách thức này là cơ sở để thực hiện phân tích lỗi và đề xuất cải tiến mô hình.

A. Thách thức về Cấu trúc Ngữ pháp và Cú pháp (Syntax)

Độ phức tạp trong cấu trúc cú pháp của tiếng Pháp thường đòi hỏi mô hình phải thực hiện sự sắp xếp lại từ ngữ (reordering) đáng kể so với tiếng Anh:

1. **Thứ tự Tính từ:** Tiếng Anh tuân theo quy tắc tính từ đứng trước danh từ, trong khi tiếng Pháp thường đặt tính từ sau danh từ (ví dụ: "a **black** cat" → "un chat **noir**").
2. **Thứ tự Đối tượng và Trạng từ:** Vị trí của các đại từ đối tượng trực tiếp và gián tiếp, cũng như các trạng từ ngắn, trong tiếng Pháp có xu hướng đứng trước động từ, tạo ra một cấu trúc câu khác biệt so với tiếng Anh (ví dụ: "I **gave him** the book" → "Je **lui ai donné** le livre").
3. **Hình thái học Phức tạp (Morphology):** Tiếng Pháp có hệ thống biến đổi từ vựng phức tạp, đặc biệt là sự tương hợp về **Giống (Gender)** và **Số (Number)** của danh từ, tính từ và mạo từ. Mô hình phải học cách biến đổi chính xác các hình thái này, điều mà tiếng Anh gần như không có.

B. Thách thức Kỹ thuật của Kiến trúc Encoder-Decoder LSTM

Mô hình nền tảng Encoder-Decoder LSTM (không có Attention) phải đối mặt với một hạn chế kỹ thuật lớn, trực tiếp ảnh hưởng đến chất lượng bản dịch:

1. **Vấn đề Context Vector Cố định (Information Bottleneck):** Kiến trúc này nén toàn bộ thông tin của câu nguồn (Input Sequence) thành một **Context Vector** duy nhất. Đối với các câu dài (vượt quá 50 tokens), Context Vector này có xu hướng **"quên"** các thông tin chi tiết hoặc ngữ cảnh quan trọng ở đầu câu. Đây là một điểm nghẽn (bottleneck) về thông tin, dẫn đến việc Decoder dịch sai hoặc bỏ sót chi tiết.
2. **Lỗi Từ vựng Ngoài (Out-Of-Vocabulary - OOV):** Do đồ án giới hạn từ vựng (ví dụ: 10.000 từ phổ biến nhất), những từ hiếm hoặc từ mới không có trong từ điển sẽ bị ánh xạ thành token **<unk>**. Mô hình không thể dịch được các từ này, làm giảm điểm BLEU và chất lượng bản dịch.

C. Thách thức về Ngữ nghĩa và Tính trôi chảy

1. **Đa nghĩa (Ambiguity):** Một từ tiếng Anh có thể có nhiều nghĩa khác nhau. Mô hình phải sử dụng Context Vector để xác định nghĩa phù hợp nhất trong ngữ cảnh cụ thể để dịch sang tiếng Pháp.
2. **Sử dụng Từ sai (Lexical Choice):** Mô hình có thể dự đoán từ dịch chính xác về ngữ nghĩa nhưng lại không phải là từ tự nhiên hoặc phổ biến nhất trong tiếng Pháp, làm giảm tính trôi chảy của câu dịch.

Kết luận: Việc giải quyết các thách thức kỹ thuật, đặc biệt là vấn đề *Context Vector Cố định*, là trọng tâm cho các đề xuất cải tiến (như Attention và Beam Search) nhằm nâng cao điểm BLEU của đồ án.

1.2.4 Giới Thiệu Mô Hình Encoder-Decoder LSTM (Seq2Seq).

Mô hình **Encoder-Decoder**, là kiến trúc nền tảng trong các bài toán **Sequence-to-Sequence (Seq2Seq)**, một kiến trúc kinh điển và nền tảng cho NMT, được đề xuất lần đầu bởi Sutskever et al. (2014). Kiến trúc này được chia thành hai khối chức năng chính:

a) Encoder (Bộ mã hóa)

- **Nhiệm vụ:** Đọc toàn bộ chuỗi đầu vào (câu tiếng Anh) và nén ý nghĩa của nó vào một vector.
- **Cơ chế:** Sử dụng *Mạng Nơ-ron Hồi quy (RNN),* cụ thể là *LSTM (Long Short-Term Memory),* để xử lý từng từ một theo thứ tự. LSTM giúp duy trì bộ nhớ dài hạn, đảm bảo thông tin quan trọng từ đầu câu không bị mất.
- **Đầu ra:** Trạng thái cuối cùng của Encoder (bao gồm *Hidden State* và *Cell State*) được gọi là *Context Vector (Vector Ngữ cảnh).* Vector này là bản tóm tắt cố định về ngữ nghĩa của câu nguồn.

b) Decoder (Bộ giải mã)

- **Nhiệm vụ:** Tạo ra chuỗi đầu ra (câu tiếng Pháp) từng từ một.
- **Cơ chế:** Decoder cũng sử dụng *LSTM*. Nó nhận *Context Vector* từ Encoder để khởi tạo trạng thái ẩn ban đầu của nó. Sau đó, nó thực hiện một vòng lặp, dự đoán từ tiếp theo dựa trên từ vừa dự đoán và trạng thái ẩn hiện tại.
- **Quá trình Dừng:** Quá trình dịch dừng lại khi mô hình dự đoán token **<eos>** (End of Sentence) hoặc đạt đến độ dài tối đa cho phép.

1.2.5 Mục tiêu và phạm vi đồ án.

Triển khai nền tảng: Xây dựng mô hình *Encoder-Decoder LSTM* hoàn chỉnh từ đầu bằng PyTorch, tuân thủ các yêu cầu kỹ thuật về *Padding, Packing* và cơ chế *Teacher Forcing* trong huấn luyện.

Đánh giá định lượng: Huấn luyện mô hình trên *Multi30K Dataset (Anh-Pháp)* và đánh giá hiệu suất bằng chỉ số *BLEU Score* (Bilingual Evaluation Understudy).

Phân tích và Cải tiến: Phân tích các lỗi dịch phổ biến (như lỗi OOV, lỗi ngữ pháp) và đề xuất các giải pháp nâng cao chất lượng như *Attention Mechanism* và *Beam Search*.

Chương 2: Thu thập dữ liệu.

2.1 Giới thiệu bộ dữ liệu.

Đồ án sử dụng bộ dữ liệu *Multi30K* làm nguồn dữ liệu chính.

Đặc điểm: Bao gồm các cặp câu tiếng Anh và tiếng Pháp, với độ dài câu thường *ngắn (khoảng 10–15 từ),* phù hợp cho việc huấn luyện trên CPU/GPU cơ bản.

Kích thước:

Train Set: 29,000 cặp.

Validation Set: 1,000 cặp.

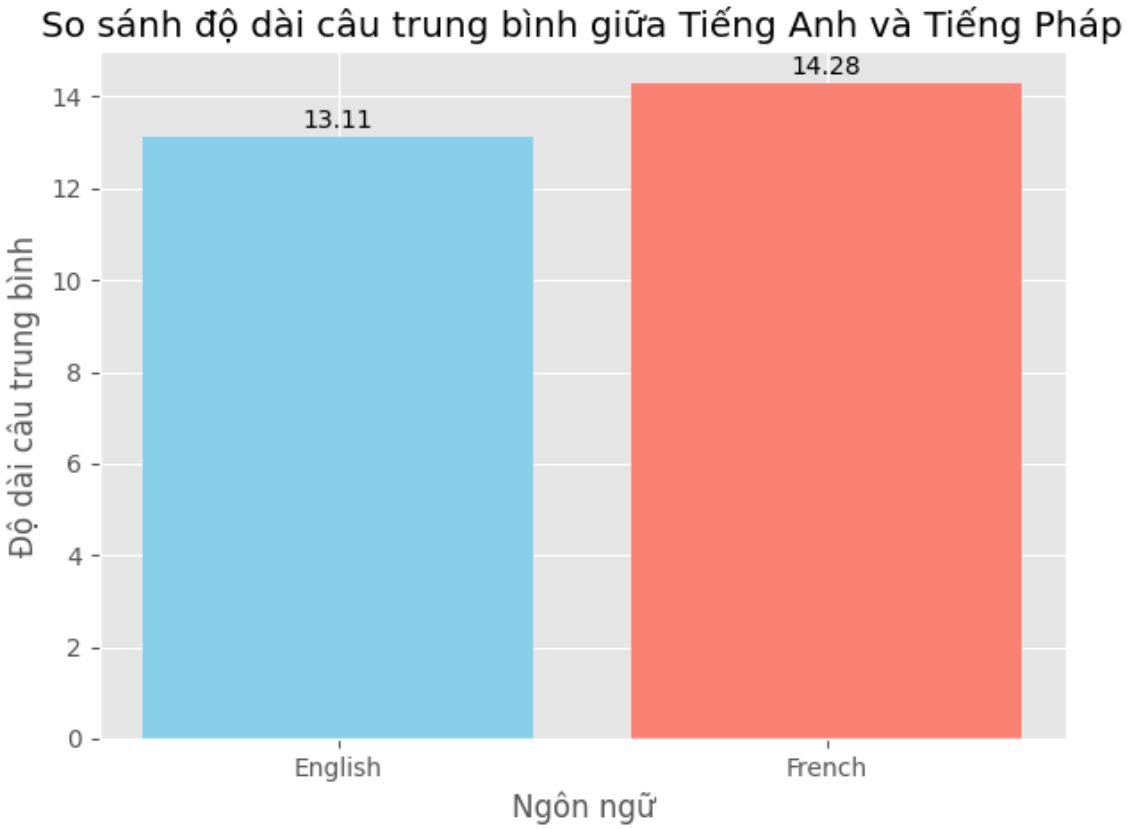
Test Set: 1,000 cặp.

Yêu cầu Xử lý: Do Multi30K trong các thư viện hỗ trợ chủ yếu là EN-DE, cần phải *tải các file raw EN-FR* và *tự xây dựng cơ chế xử lý Dataset*.

2.2 Phân tích và tiền xử lý.

2.2.1 Độ dài trung bình câu.

Để hiểu rõ hơn về tính chất của dữ liệu Multi30K (Anh-Pháp) và đưa ra quyết định về các tham số Padding, một phân tích về độ dài câu trung bình đã được thực hiện.



Độ dài Thấp: Độ dài câu trung bình của tiếng Anh là **13.11** tokens, và tiếng Pháp là **14.28** tokens. Giá trị thấp này khẳng định tính chất của bộ dữ liệu Multi30K: chứa các câu **ngắn (10–15 từ)**, chủ yếu là các mô tả hình ảnh đơn giản.

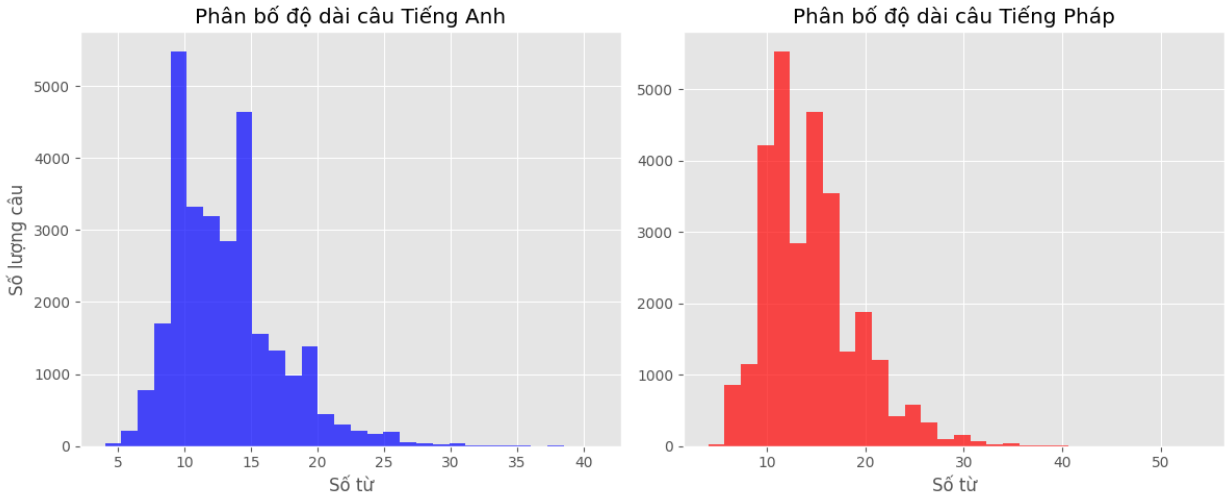
Chênh lệch Độ dài: Câu tiếng Pháp có độ dài trung bình lớn hơn tiếng Anh khoảng **1.17** tokens. Sự chênh lệch này không đáng kể nhưng phản ánh sự khác biệt về cấu trúc ngôn ngữ: Tiếng Pháp yêu cầu sử dụng nhiều từ chức năng hơn (như mạo từ, giới từ) để duy trì sự tương hợp về giống và số, dẫn đến câu đích có xu hướng dài hơn câu nguồn.

Ý nghĩa đối với Mô hình:

- Tính chất câu ngắn đồng đều là một lợi thế cho mô hình **Encoder-Decoder LSTM cơ bản** được sử dụng trong đồ án.
- Nó giúp giảm thiểu rủi ro **mất mát thông tin** (information bottleneck) thường gặp khi Context Vector cố định phải nén ý nghĩa của các câu quá dài.

2.2.2 Phân phối độ dài câu.

Để có cái nhìn sâu sắc hơn về tính chất của dữ liệu Multi30K (Anh-Pháp), đặc biệt là sự biến động về độ dài, biểu đồ phân bố độ dài câu đã được xây dựng và phân tích.



Phân bố Tập trung và Câu ngắn:

- Cả hai biểu đồ đều thể hiện sự phân bố tập trung mạnh mẽ (rất lệch về bên phải, có đuôi dài).
- Phần lớn các câu (số lượng câu cao nhất) nằm trong phạm vi hẹp *8 đến 16 tokens*. Điều này khẳng định Multi30K là một bộ dữ liệu chứa các *câu ngắn (10–15 từ),* phù hợp với mô tả hình ảnh đơn giản.

Đỉnh phân bố:

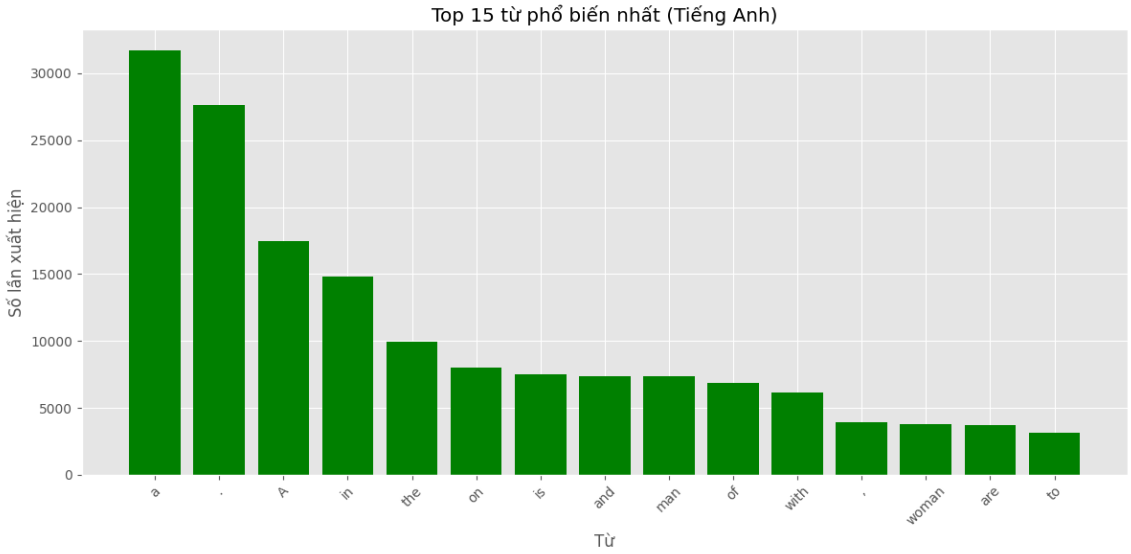
- Đỉnh của phân bố tiếng Anh nằm quanh *9–10 tokens*.
- Đỉnh của phân bố tiếng Pháp có xu hướng dịch chuyển nhẹ, nằm quanh *10–12 tokens*. Sự dịch chuyển này phù hợp với phân tích về độ dài trung bình, cho thấy câu tiếng Pháp có xu hướng dài hơn một chút do yêu cầu ngữ pháp.

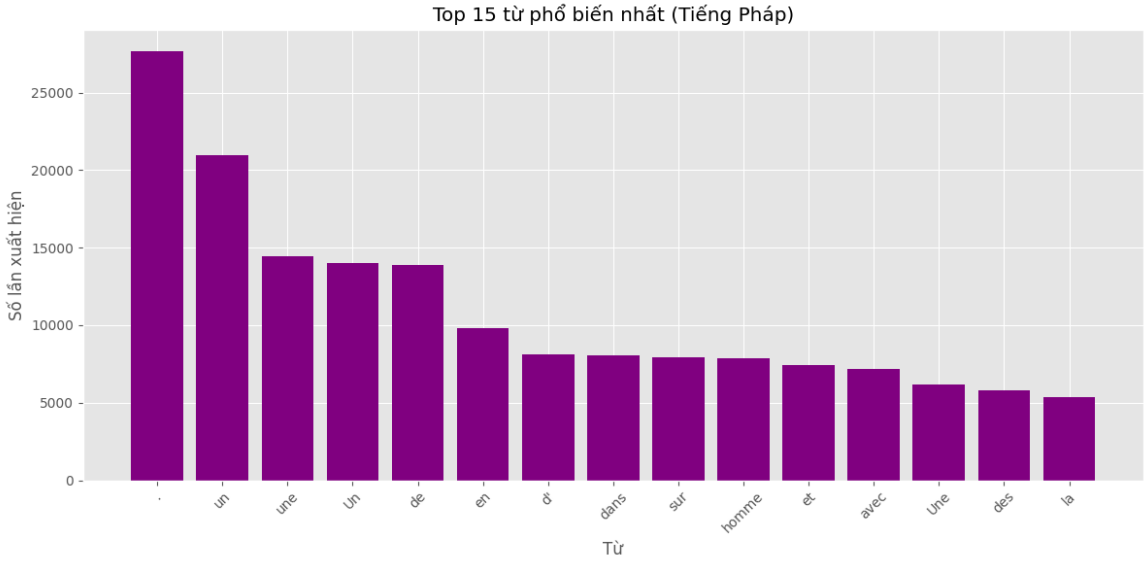
Tỷ lệ Câu dài:

- Số lượng câu có độ dài trên 25 tokens là *rất thấp* và gần như bằng không.
- **Ý nghĩa Kỹ thuật:** Phân tích này là cơ sở vững chắc cho quyết định giới hạn độ dài câu tối đa *(Max Length)* là *50 tokens*. Việc này bao phủ gần như toàn bộ dữ liệu, đồng thời đảm bảo rằng mô hình Encoder-Decoder LSTM cơ bản sẽ *giảm thiểu rủi ro mất mát thông tin* (information bottleneck) ở Context Vector, vốn là điểm yếu của nó khi xử lý chuỗi cực kỳ dài.

2.2.3 Từ phổ biến.

Phân tích tần suất xuất hiện của các từ là một bước thiết yếu để hiểu cấu trúc ngôn ngữ của bộ dữ liệu Multi30K và đưa ra quyết định tối ưu về tham số Từ vựng (Vocabulary).





Tính chất Chung và Stopwords:

- Ở cả hai ngôn ngữ, các từ có tần suất xuất hiện cao nhất đều là các từ chức năng như *mạo từ, giới từ, và dấu câu* (Stopwords). Điều này là đặc điểm chung của bất kỳ corpus ngôn ngữ tự nhiên nào.

Đặc điểm Ngôn ngữ:

- **Tiếng Anh:** Các từ phổ biến bao gồm các mạo từ cơ bản, giới từ ("in", "on", "of", "with") và các từ chỉ người/hành động ("is", "man", "woman").

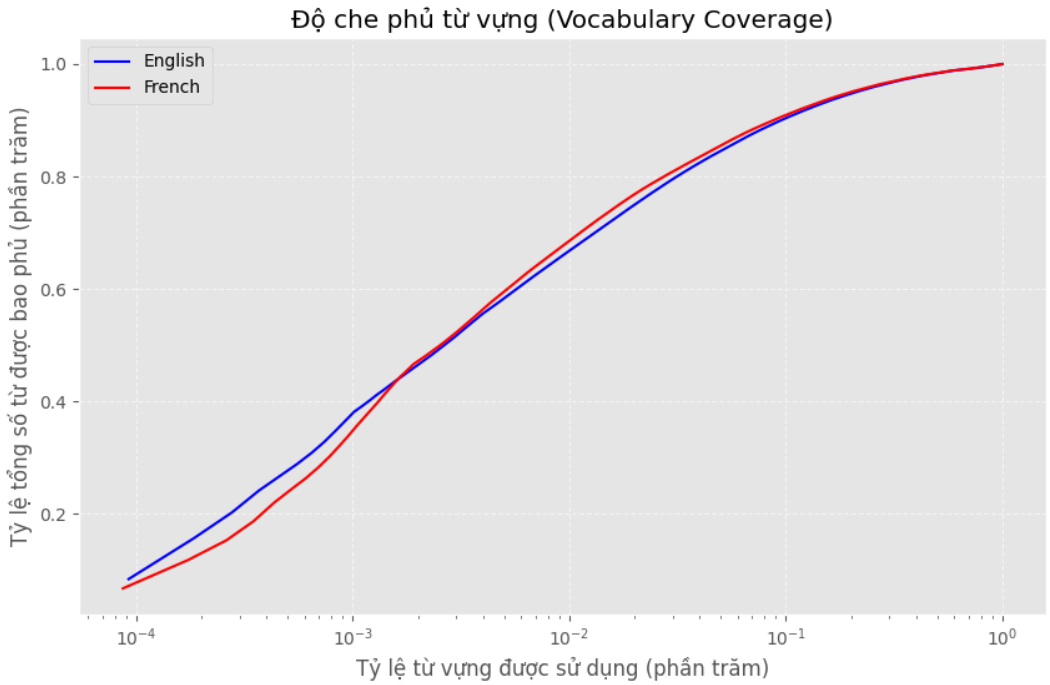
Sự hiện diện của các từ này phản ánh tính chất của Multi30K: mô tả ngắn về hình ảnh.

- **Tiếng Pháp:** Tần suất cao của các mạo từ chỉ giống (Gender) như "un", "une", "la", "des" là điểm nổi bật. Điều này xác nhận rằng mô hình cần học cách xử lý tương hợp giống và số — một thách thức về hình thái học của tiếng Pháp.

Ý nghĩa Kỹ thuật:

- **Phân bố tập trung:** Sự chênh lệch lớn về tần suất giữa các từ Top 1 và các từ còn lại chứng minh hiệu quả của việc *giới hạn từ vựng.*
- **Quyết định Từ vựng:** Phân tích này hỗ trợ quyết định giới hạn từ điển ở *10.000 từ phổ biến nhất.* Việc này nhằm tối ưu hóa tài nguyên bằng cách giảm kích thước của *Embedding Layer* và lớp *Softmax Output,* đồng thời đảm bảo mô hình tập trung học các từ có độ phủ cao nhất.

2.2.4 Độ che phủ từ vựng.



Phân bố Tập trung và Hiệu quả:

- Đường cong che phủ thể hiện sự phân bố tần suất từ vựng theo quy luật Zipf: một tỷ lệ **rất nhỏ** các từ vựng độc nhất (các từ phổ biến) chiếm một **tỷ lệ rất lớn** trong tổng số lần xuất hiện từ trong corpus.

- Ví dụ: Khi tỷ lệ từ vựng được sử dụng tăng lên nhanh chóng ở trục tung, điều đó chứng tỏ các từ phổ biến nhất có hiệu suất bao phủ ngữ nghĩa cao.

Sự Tương đồng Ngôn ngữ:

- Đường cong che phủ của tiếng Anh (màu xanh) và tiếng Pháp (màu đỏ) gần như **trùng khớp**. Điều này cho thấy cấu trúc phân bố tần suất từ vựng trong Multi30K là tương đồng giữa hai ngôn ngữ, mặc dù có sự khác biệt về hình thái học.

Ý nghĩa Kỹ thuật đối với Mô hình:

- **Tối ưu hóa Tài nguyên:** Phân tích này là cơ sở vững chắc để quyết định giới hạn từ vựng ở **10.000 từ phổ biến nhất**. Mục đích là để **giảm kích thước** của Embedding Layer và lớp Softmax Output, từ đó tăng tốc độ huấn luyện.
- **Thách thức OOV:** Mặc dù mang lại lợi ích về tính toán, việc loại bỏ các từ hiếm (phần cuối của đường cong) tạo ra lỗi **OOV (→ <unk>)**, một thách thức cốt lõi cần được phân tích chi tiết trong Chương V.

2.2.5 Tiền xử lý dữ liệu.

1. Tokenization (Tách từ)

- **Mục đích:** Chia câu văn bản thô thành các đơn vị nhỏ hơn gọi là **tokens** (thường là các từ hoặc dấu câu).
- **Thực hiện:** Sử dụng thư viện **Spacy** để tách từ cho cả tiếng Anh và tiếng Pháp (en_core_web_sm và fr_core_news_sm).

2. Xây dựng Từ điển (Vocabulary)

- **Mục đích:** Gán một số nguyên (ID) duy nhất cho mỗi token, giúp máy tính có thể xử lý văn bản dưới dạng vector số.
- **Token Đặc biệt:** Thêm các token cần thiết cho mô hình Sequence to Sequence:
  - **<unk>** (Unknown): Đại diện cho các từ không có trong từ điển.
  - **<pad>** (Padding): Dùng để đệm các câu ngắn.
  - **<sos>** (Start of Sentence): Đánh dấu sự bắt đầu của câu đích (đầu vào cho Decoder).
  - **<eos>** (End of Sentence): Đánh dấu sự kết thúc của câu.
- **Giới hạn Từ vựng:** Bắt buộc giới hạn từ điển ở **10.000 từ phổ biến nhất** cho mỗi ngôn ngữ.

3. Padding & Packing (Đệm và Đóng gói)

Đây là kỹ thuật cốt lõi để xử lý các chuỗi có độ dài khác nhau trong cùng một lô (batch).

- **Sắp xếp Batch:** Bắt buộc sắp xếp các câu trong batch theo **độ dài giảm dần** (từ dài nhất đến ngắn nhất).
- **Padding (Đệm):** Sử dụng torch.nn.utils.rnn.pad_sequence để thêm token <pad> vào cuối các câu ngắn, làm cho tất cả các tensor trong batch có cùng kích thước.
- **Packing (Đóng gói):** Sử dụng torch.nn.utils.rnn.pack_padded_sequence trước khi đưa vào Encoder LSTM. Việc này báo cho LSTM biết câu thực sự kết thúc ở đâu, **bỏ qua việc tính toán** cho các token <pad>, giúp tối ưu hóa hiệu suất và ngăn chặn gradient bị ảnh hưởng bởi phần đệm.

4. Tạo DataLoader

- **Mục đích:** Tạo ra các lô (batch) dữ liệu để đưa vào mô hình.
- **Hàm collate_fn:** Cần sử dụng hàm collate_fn tùy chỉnh để thực hiện các bước sắp xếp và Padding/Packing trên từng batch.
- **Batch Size:** Kích thước lô đề xuất từ 32 đến 128.

2.3 Chia dữ liệu huấn luyện và kiểm thử.

Bộ dữ liệu Multi30K được chia thành ba tập hợp riêng biệt để quản lý quá trình huấn luyện, tinh chỉnh tham số và đánh giá cuối cùng.

A. Tập Huấn luyện (Train Set)

- **Kích thước:** Tập huấn luyện bao gồm ≈ 29,000 cặp câu Anh-Pháp.
- **Mục đích:** Đây là tập dữ liệu chính được sử dụng để **cập nhật các tham số** (weights và biases) của mô hình Encoder-Decoder LSTM thông qua quá trình lan truyền ngược (backpropagation).

B. Tập Thẩm định (Validation Set)

- **Kích thước:** Tập thẩm định bao gồm ≈ 1,000 cặp câu.
- **Mục đích:** Tập Validation được sử dụng để theo dõi hiệu suất mô hình sau mỗi epoch huấn luyện, nhưng không được dùng để cập nhật tham số trực tiếp.
    - **Phòng chống Overfitting:** Theo dõi validation loss là cơ sở cho cơ chế **Dừng sớm (Early Stopping)**. Quá trình huấn luyện sẽ dừng lại nếu validation loss không giảm sau một số lượng epoch nhất định (ví dụ: 3 epoch).
    - **Lưu Checkpoint:** Tập Validation quyết định thời điểm lưu mô hình tốt nhất (best_model.pth) khi validation loss đạt mức thấp nhất.

C. Tập Kiểm thử (Test Set)

- **Kích thước:** Tập kiểm thử bao gồm ≈ 1,000 cặp câu.
- **Mục đích:** Tập dữ liệu này hoàn toàn tách biệt khỏi quá trình huấn luyện và thẩm định, và chỉ được sử dụng một lần duy nhất, ở cuối quá trình, để **đánh giá hiệu suất thực tế** của mô hình.
- **Chỉ số Đánh giá:** Dùng để tính toán chỉ số chất lượng chính là **BLEU Score**.

→ Quy trình chia dữ liệu này đảm bảo rằng mô hình được đánh giá một cách khách quan trên dữ liệu hoàn toàn mới.

Chương 3: Xây dựng mô hình.

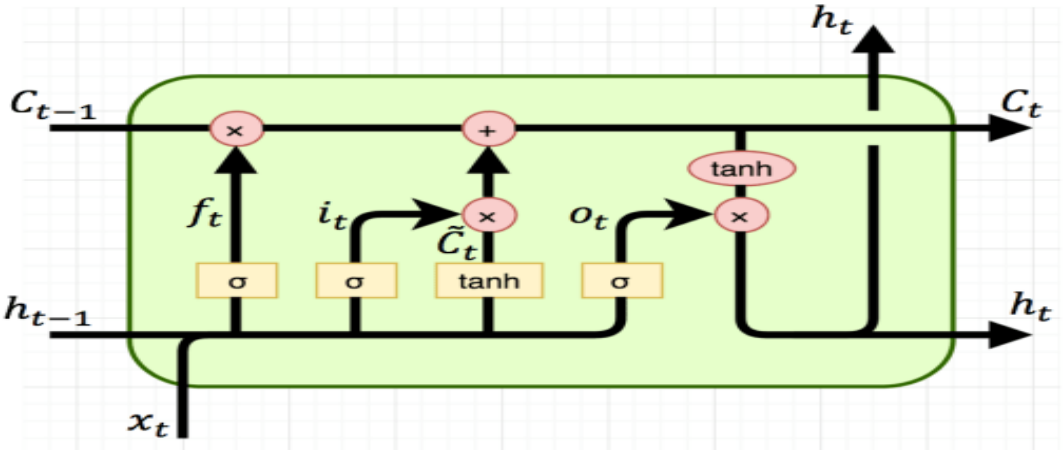
3.1 Cơ sở hướng tiếp cận và mô hình.

+ **Hướng tiếp cận:** Lựa chọn phương pháp *"Xây dựng từ nguyên lý" (Implementation from Scratch)* trên nền tảng PyTorch. Thay vì sử dụng các thư viện cấp cao có sẵn (như transformers hay seq2seq), hướng tiếp cận này yêu cầu tự xây dựng từng thành phần (Dataset, Model, Training Loop). Điều này nhằm đáp ứng yêu cầu nắm vững cơ chế hoạt động nội tại của kiến trúc Sequence-to-Sequence và đảm bảo khả năng tùy biến sâu vào luồng dữ liệu.

+ **Mô hình:** Mô hình Sequence-to-Sequence (Seq2Seq) dựa trên kiến trúc kiến trúc Encoder-Decoder sử dụng mạng LSTM (Long Short-Term Memory), Được huấn luyện trên thập dữ liệu Multi30k, tối ưu cho cặp ngôn ngữ Anh-Pháp

3.2 Mô tả chi tiết mô hình LSTM.

LSTM (Long Short-Term Memory) là một biến thể đặc biệt của mạng nơ-ron hồi quy (RNN), được thiết kế để giải quyết vấn đề "biến mất đạo hàm" (vanishing gradient) và tăng cường khả năng ghi nhớ các phụ thuộc xa trong chuỗi dữ liệu. Đối với bài toán Dịch máy (Machine Translation) từ tiếng Anh sang tiếng Pháp, nhóm đã lựa chọn LSTM làm nòng cốt cho kiến trúc Sequence-to-Sequence (Seq2Seq). Thay vì sử dụng các thư viện cấp cao có sẵn, nhóm đã xây dựng mô hình "từ đầu" (from scratch) sử dụng thư viện PyTorch để đảm bảo khả năng tùy biến sâu vào luồng dữ liệu.



Hình 3.2: Cấu trúc của một khối vanila LSTM block

Cấu trúc mô hình bao gồm các thành phần chính sau:

- **Kiến trúc Seq2Seq với Attention:** Mô hình được xây dựng dựa trên kiến trúc Encoder-Decoder kết hợp với cơ chế Attention (Chú ý). Khác với bài toán phân loại đơn thuần, bài toán dịch máy yêu cầu sinh ra một chuỗi từ mới dựa trên ngữ cảnh toàn cục. Do đó, mô hình sử dụng Encoder để nén thông tin và Decoder để sinh chuỗi, được kết nối bởi Attention để Decoder có thể "nhìn lại" toàn bộ câu nguồn tại mỗi bước dịch.
- **Lớp Embedding**: Trước khi đưa vào mạng nơ-ron, các từ vựng (được biểu diễn dưới dạng chỉ số) được chuyển đổi thành các vector số thực dày đặc (Dense Vectors) với kích thước cố định là 256 (Embedding Dimension). Lớp này giúp mô hình nắm bắt được sự tương đồng về ngữ nghĩa giữa các từ trong không gian vector hiệu quả hơn so với one-hot encoding.
- **Mạng LSTM (Encoder & Decoder):**

  + **Encoder:** Sử dụng mạng LSTM hai chiều (Bidirectional LSTM) để xử lý chuỗi đầu vào, cho phép mô hình học ngữ cảnh từ cả hai hướng (xuôi và ngược).

  + **Decoder**: Sử dụng mạng LSTM đơn chiều nhưng nhận đầu vào phức hợp bao gồm: từ hiện tại, trạng thái ẩn trước đó và vector bối cảnh (Context Vector) được tính toán từ cơ chế Attention.

- **Cơ chế Attention:** Mô hình tích hợp module Attention để tính toán trọng số quan trọng cho từng từ trong câu nguồn tại mỗi bước giải mã. Thay vì chỉ dựa vào một vector trạng thái cuối cùng (dễ gây mất mát thông tin ở câu dài), Attention cho phép Decoder tập trung vào các phần khác nhau của câu nguồn phù hợp nhất với từ đang được dịch.

- **Dropout:** Kỹ thuật Dropout được áp dụng với tỷ lệ 0.5. Lưu ý rằng trong mô hình này, Dropout được áp dụng ngay sau lớp Embedding (trước khi vào LSTM) chứ không phải sau lớp LSTM. Mục tiêu là tắt ngẫu nhiên 50% các đơn vị trong vector nhúng trong quá trình huấn luyện để giảm thiểu hiện tượng Overfitting (quá khớp), buộc mô hình phải học các đặc trưng mạnh mẽ hơn.

3.3 Kiến trúc phương án:

3.3.1 Module Encoder (Bộ mã hóa)

Module Encoder đóng vai trò là khối tiếp nhận và trích xuất đặc trưng từ chuỗi văn bản nguồn (tiếng Anh). Quá trình xử lý bắt đầu bằng việc chuyển đổi các chỉ số từ vựng (integer indices) thành các vector biểu diễn dày đặc thông qua lớp nn.Embedding, giúp mô hình nắm bắt được mối quan hệ ngữ nghĩa giữa các từ trong không gian vector. Để tăng cường khả năng tổng quát hóa, một lớp nn.Dropout với tỷ lệ 0.5 được áp dụng ngay sau tầng nhúng.

Điểm kỹ thuật quan trọng trong thiết kế của Encoder là việc tích hợp hàm pack_padded_sequence. Kỹ thuật này đóng gói các tensor đầu vào dựa trên độ dài thực tế của câu, cho phép mạng LSTM bỏ qua việc tính toán lãng phí trên các token đệm (<pad>), qua đó tối ưu hóa hiệu suất.

Lõi của Encoder là mạng LSTM hai chiều (Bidirectional LSTM) thay vì đơn chiều. Kiến trúc này cho phép mô hình học ngữ cảnh của từ dựa trên cả thông tin quá khứ (chiều xuôi) và tương lai (chiều ngược). Khác với các kiến trúc Seq2Seq cổ điển chỉ nén câu vào một vector duy nhất, Encoder này trả về hai thành phần:

1. **outputs**: Chứa toàn bộ trạng thái ẩn tại mọi bước thời gian của cả hai chiều (sau khi đã được pad_packed_sequence). Thành phần này là nguyên liệu cốt lõi để tính toán trọng số trong cơ chế Attention ở Decoder.
2. **hidden**: Là trạng thái ẩn cuối cùng được xử lý đặc biệt. Hai vector trạng thái ẩn cuối cùng của chiều xuôi (forward) và chiều ngược (backward) được nối lại (concatenate), sau đó đi qua một lớp tuyến tính (nn.Linear) và hàm kích hoạt Tanh. Vector kết quả này đóng vai trò khởi tạo trạng thái ẩn đầu tiên cho Decoder, trong khi trạng thái tế bào (cell) được loại bỏ để đơn giản hóa kiến trúc Attention.

3.3.2 Module Decoder (Bộ giải mã)

Trái ngược với Encoder, Module Decoder hoạt động theo cơ chế sinh từ tuần tự (step-by-step) để tái tạo chuỗi văn bản đích (tiếng Pháp). Quá trình giải mã bắt đầu bằng việc khởi tạo trạng thái ẩn của Decoder bằng chính trạng thái ẩn cuối cùng được xử lý từ Encoder, đảm bảo sự liên kết ngữ cảnh ban đầu.

Điểm cải tiến quan trọng trong mô hình này là việc tích hợp **cơ chế Attention**. Tại mỗi bước thời gian t, Decoder không chỉ nhìn vào từ hiện tại mà còn "nhìn lại" toàn bộ câu nguồn thông qua encoder_outputs để tính toán một **Vector ngữ cảnh có trọng số (Weighted Context Vector)**. Vector này đại diện cho những phần thông tin quan trọng nhất từ Encoder liên quan đến từ đang được dịch.

Do đó, dòng dữ liệu trong Decoder diễn ra như sau:

1. **Đầu vào LSTM:** Mạng LSTM của Decoder không chỉ nhận vector nhúng (Embedding) của từ hiện tại. Nó tiếp nhận sự kết hợp *(concatenation)* giữa *vector nhúng* và *Weighted Context Vector* vừa tính toán được.
2. **Dự đoán từ (Output):** Kết quả đầu ra của LSTM chưa được dùng ngay. Lớp kết nối đầy đủ cuối cùng (fc_out) thực hiện phép chiếu tuyến tính trên một vector tổng hợp bao gồm ba thành phần: *đầu ra LSTM, Weighted Context Vector* và *vector nhúng.*

Kết quả cuối cùng là một vector phân phối xác suất (logits) trên toàn bộ từ vựng tiếng Pháp, được dùng để quyết định từ tiếp theo thông qua thuật toán tham lam hoặc tìm kiếm chùm (beam search).

3.3.3. Module Seq2Seq (Tích hợp hệ thống)

Module Seq2Seq đóng vai trò là kiến trúc tổng thể, chịu trách nhiệm khởi tạo và điều phối luồng dữ liệu giữa hai thành phần con là Encoder và Decoder. Đây là lớp cao nhất trong hệ thống, đảm bảo quá trình truyền tải thông tin từ ngôn ngữ nguồn sang ngôn ngữ đích diễn ra liền mạch.

Trong phương thức forward, quy trình xử lý được thực hiện qua ba giai đoạn chính:

1. Giai đoạn Mã hóa (Encoding): Chuỗi đầu vào (Source Sentence) được đưa qua Encoder. Khác với mô hình cổ điển trả về cặp (hidden, cell), Encoder này trả về hai thành phần:

- encoder_outputs: Chứa trạng thái ẩn tại mọi thời điểm của câu nguồn (dùng cho cơ chế Attention).
- hidden: Trạng thái ẩn tổng hợp (đã qua xử lý tuyến tính từ hai chiều thuận/nghịch) để khởi tạo trạng thái bắt đầu cho Decoder.

2. Giai đoạn Giải mã (Decoding): Hệ thống khởi tạo một tensor outputs để lưu trữ kết quả. Trong vòng lặp giải mã, tại mỗi bước thời gian t, Decoder nhận vào ba tham số: từ đầu vào hiện tại, trạng thái ẩn trước đó,

và đặc biệt là encoder_outputs. Việc truyền encoder_outputs vào Decoder là điểm mấu chốt, cho phép Decoder "nhìn lại" toàn bộ câu gốc để tính toán trọng số Attention tại mỗi bước, thay vì chỉ dựa vào một Vector ngữ cảnh tĩnh duy nhất.

3. Chiến lược Teacher Forcing: Điểm đặc biệt trong quá trình huấn luyện là việc áp dụng chiến lược Teacher Forcing. Với tỷ lệ teacher_forcing_ratio (mặc định 0.5), hệ thống sẽ ngẫu nhiên quyết định đầu vào cho bước tiếp theo: hoặc là từ chính xác thực tế (Ground Truth) từ tập dữ liệu, hoặc là từ do mô hình dự đoán (top1) ở bước trước. Cơ chế này giúp mô hình hội tụ nhanh hơn trong giai đoạn đầu đồng thời vẫn giữ được khả năng tự sinh từ.

Chương 4: Triển khai và đánh giá

4.1. Quy trình tiền xử lý dữ liệu

Để đảm bảo hiệu suất tối ưu cho mô hình Sequence-to-Sequence LSTM, đồ án đã thiết lập một quy trình xử lý dữ liệu (Data Pipeline) chặt chẽ, được hiện thực hóa thông qua lớp Multi30kDataset và Collate. Quy trình bao gồm các giai đoạn: Thu thập, Tokenization (Tách từ), Xây dựng từ điển (Vocabulary Building) và Đóng gói theo batch (Batching).

4.1.1. Chiến lược Mã hóa Văn bản (Tokenization Strategy)

Bước đầu tiên và quan trọng nhất trong quy trình xử lý là Tokenization. Thay vì sử dụng các phương pháp tách chuỗi sơ khai dựa trên khoảng trắng (Whitespace Tokenization) – vốn bộc lộ nhiều hạn chế khi xử lý các ký tự đặc biệt hoặc từ viết tắt – đồ án đã lựa chọn giải pháp Tokenization dựa trên thư viện chuyên dụng Spacy.

A. Cơ sở lựa chọn công cụ (Tool Selection)

Hệ thống tích hợp thư viện Spacy với hai gói ngôn ngữ pre-trained hạng nhẹ (lightweight) là en_core_web_sm (cho tiếng Anh) và fr_core_news_sm (cho tiếng Pháp). Trong mã nguồn, cơ chế tự động tải (auto-download) cũng được thiết lập để đảm bảo môi trường thực thi luôn có sẵn các gói này. Sự lựa chọn này dựa trên các ưu điểm kỹ thuật sau:

- **Khả năng tách từ thông minh:**

Code sử dụng trực tiếp trình tách từ của Spacy (spacy_en.tokenizer và spacy_fr.tokenizer) để chuyển đổi văn bản thành danh sách các token (tok.text). Công cụ này vượt trội hơn hàm split() thông thường ở chỗ nó xử lý chính xác các trường hợp ngôn ngữ đặc thù như:

+ Tách dấu câu dính liền với từ (ví dụ: "hello." => "hello", ".").

+ Tách các từ viết tắt trong tiếng Anh (ví dụ: "don't" => "do", "n't").

+ Xử lý các quy tắc tokenization phức tạp của tiếng Pháp (như "l'homme" => "l'", "homme").

- **Hiệu suất tối ưu:**

Trong data_utils.py, hệ thống chỉ trích xuất phần văn bản (tok.text) từ bộ tokenizer mà không kích hoạt toàn bộ pipeline phân tích cú pháp (như gắn nhãn từ loại hay phân tích phụ thuộc). Điều này giúp tận dụng tốc độ xử lý nhanh của Cython trong Spacy, đảm bảo không tạo ra nút thắt cổ chai (bottleneck) khi xử lý tập dữ liệu lớn.

- **Tính nhất quán (Consistency):**

Hai hàm en_tokenizer và fr_tokenizer được định nghĩa cố định và sử dụng xuyên suốt toàn bộ dự án: từ việc xây dựng bộ từ điển (Vocabulary), xử lý dữ liệu huấn luyện trong Collate, cho đến việc xử lý câu input đầu vào trong quá trình Demo thực tế (beam_search_translate). Điều này đảm bảo tính đồng nhất tuyệt đối về định dạng dữ liệu đầu vào.

B. Cơ chế Xử lý Ngôn ngữ chi tiết

Quy trình Tokenization được thực hiện dựa trên các mô hình ngôn ngữ đã được huấn luyện sẵn (en_core_web_sm và fr_core_news_sm), được thiết kế để xử lý các đặc thù của từng ngôn ngữ nhằm tối ưu hóa đầu vào cho mạng LSTM:

- **Đối với Tiếng Anh (Source Language):**
    - **Xử lý dạng rút gọn (Contractions):** Mô hình en_core_web_sm tự động nhận diện và tách các từ như "don't" thành hai token riêng biệt là "do" và "n't" (đại diện cho "not"). Điều này giúp mô hình học được ý nghĩa phủ định một cách tường minh thay vì coi "don't" là một từ vựng riêng biệt chưa từng gặp.
    - **Xử lý dấu câu:** Các dấu câu dính liền (như dấu chấm cuối câu, dấu phẩy) được tách thành các token độc lập, giúp mô hình nhận biết ranh giới câu và ngữ điệu.
- **Đối với Tiếng Pháp (Target Language):**
    - **Xử lý hiện tượng lược âm (Elision):** Tiếng Pháp thường xuyên sử dụng dấu nháy đơn để lược âm (ví dụ: *l'homme*, *c'est*, *d'accord*). Tokenizer của Spacy (fr_core_news_sm) được cấu hình mặc định để tách các mạo từ hoặc giới từ bị lược bỏ này (ví dụ: "l'", "homme") thành token riêng. Việc này cực kỳ quan trọng giúp giảm kích thước

bộ từ điển (Vocabulary Size) và tăng khả năng tổng quát hóa của mô hình khi gặp các danh từ khác nhau đi sau mạo từ.

C. Kỹ thuật Triển khai (Implementation)

Để đảm bảo quyền kiểm soát chi tiết và hiệu suất cao, thay vì phụ thuộc vào các hàm tiện ích có sẵn của thư viện bên thứ 3 (như torchtext cũ), hệ thống tự định nghĩa các hàm wrapper (bao đóng) trực tiếp trong module data_utils.py:

- **Hàm Wrapper Tùy chỉnh:** Hệ thống triển khai hai hàm en_tokenizer(text) và fr_tokenizer(text). Các hàm này đóng vai trò trung gian, gọi trực tiếp vào đối tượng tokenizer của Spacy (spacy_en.tokenizer và spacy_fr.tokenizer) để thực hiện tách từ.
- **Trích xuất Token thô:** Kết quả trả về từ Spacy là các đối tượng phức tạp chứa nhiều thông tin ngữ pháp. Tuy nhiên, hàm wrapper sử dụng kỹ thuật *list comprehension* để chỉ trích xuất thuộc tính văn bản (tok.text). Đầu ra cuối cùng là một danh sách các chuỗi ký tự (List of Strings) thuần túy, loại bỏ các meta-data không cần thiết để tiết kiệm bộ nhớ trước khi chuyển sang bước ánh xạ số (Numericalization).

4.1.2 Quy trình Xây dựng Từ điển (Vocabulary Construction)

Sau khi văn bản đã được mã hóa thành các token riêng biệt, bước tiếp theo là xây dựng bộ từ điển (Vocabulary) để ánh xạ các token này sang dạng số nguyên (Indices). Quy trình này được thực hiện thủ công thông qua lớp Vocabulary với các chiến lược tối ưu cụ thể như sau:

A. Cơ chế Ánh xạ Hai chiều (Bi-directional Mapping) Hệ thống duy trì hai cấu trúc dữ liệu ánh xạ song song:

- String-to-Index (stoi): Dùng cho giai đoạn tiền xử lý đầu vào, chuyển từ vựng sang số hiệu để đưa vào lớp Embedding.
- Index-to-String (itos): Dùng cho giai đoạn hậu xử lý đầu ra (Inference), chuyển kết quả dự đoán (số hiệu) ngược lại thành văn bản con người đọc được.

B. Tích hợp Token Điều khiển (Special Control Tokens) Hệ thống dành riêng 4 chỉ số đầu tiên (0-3) cho các token đặc biệt để điều phối luồng dữ liệu:

1. <pad> (Padding Token - Index 0):

- Vai trò: Được chèn vào cuối các câu ngắn để đảm bảo ma trận đầu vào có kích thước cố định trong một Batch.

- Tối ưu: Chỉ số này được truyền vào tham số ignore_index của hàm mất mát CrossEntropyLoss trong main.py. Điều này đảm bảo mô hình không tính toán lỗi (loss) và không cập nhật trọng số dựa trên các vị trí đệm vô nghĩa này.

2. <sos> (Start of Sentence - Index 1):

- Vai trò: Tín hiệu kích hoạt bắt buộc cho Decoder. Trong quá trình huấn luyện và dịch (Inference), token này luôn được chèn vào đầu chuỗi đích để báo hiệu điểm bắt đầu.

3. <eos> (End of Sentence - Index 2):

- Vai trò: Đánh dấu điểm kết thúc của một câu. Trong thuật toán tìm kiếm chùm (beam_search_translate), khi một nhánh gặp token này, nó sẽ dừng mở rộng.

4. <unk> (Unknown Token - Index 3):

- Vai trò: Đại diện cho tất cả các từ nằm ngoài từ điển (Out-of-Vocabulary). Khi gặp từ lạ, hệ thống tự động ánh xạ về chỉ số 3 thay vì gây lỗi chương trình (KeyError), giúp hệ thống hoạt động bền vững (robust).

C. Chiến lược Kiểm soát Kích thước Từ điển (Optimization Strategy) Một thách thức lớn trong NLP là bùng nổ không gian chiều dữ liệu. Đồ án áp dụng đồng thời hai kỹ thuật lọc để giải quyết vấn đề này:

- **Ngưỡng tần suất tối thiểu** (freq_threshold): Hệ thống sử dụng lớp Counter để đếm tần suất xuất hiện của mọi từ. Chỉ những từ xuất hiện tối thiểu 2 lần (freq_threshold=2) mới được xem xét đưa vào từ điển. Việc loại bỏ các từ xuất hiện 1 lần (thường là lỗi chính tả hoặc tên riêng quá hiếm) giúp loại bỏ nhiễu và cải thiện khả năng học của Embedding.
- **Giới hạn số lượng từ tối đa** (max_size): (Bổ sung mới) Đây là chốt chặn quan trọng để kiểm soát bộ nhớ. Dù có bao nhiêu từ thỏa mãn điều kiện tần suất, hệ thống cũng chỉ giữ lại tối đa 10.000 từ phổ biến nhất cho mỗi ngôn ngữ.

    + **Cơ chế:** Code sử dụng hàm frequencies.most_common (max_size - 4) để chọn lọc top từ vựng (trừ đi 4 token đặc biệt).

    + **Kết quả:** Kích thước từ điển được cố định ở mức an toàn (~10.000), ngăn chặn hiện tượng bùng nổ kích thước mô hình (Model size explosion) tại lớp Linear cuối cùng của Decoder.

4.1.3. Chiến lược Tối ưu hóa Batching và Đóng gói dữ liệu (Advanced Batching & Packing Strategy)

Trong các mô hình Deep Learning xử lý chuỗi (Sequence Models), hiệu suất huấn luyện chịu ảnh hưởng lớn bởi cách tổ chức dữ liệu đầu vào. Đồ án đã triển khai chiến lược Batching tùy chỉnh thông qua lớp Collate để giải quyết bài toán độ dài biến thiên của ngôn ngữ tự nhiên, tập trung vào ba cơ chế tối ưu cốt lõi:

A. Cơ chế Sắp xếp theo Batch (In-Batch Sorting) Mặc định, DataLoader lấy ngẫu nhiên các mẫu dữ liệu để đảm bảo tính ngẫu nhiên (SGD). Tuy nhiên, để tối ưu hóa tính toán cho LSTM, các câu trong cùng một batch cần được sắp xếp.

- **Triển khai:** Trong hàm __call__ của lớp Collate, trước khi tạo tensor, toàn bộ các cặp câu (nguồn, đích) trong một Batch được sắp xếp lại dựa trên độ dài của câu nguồn (Source Sentence) theo thứ tự giảm dần (reverse=True).

- **Ý nghĩa:** Việc sắp xếp này là điều kiện tiên quyết để sử dụng hàm pack_padded_sequence trong Encoder với tham số enforce_sorted=True. Nó đảm bảo rằng PyTorch có thể xử lý luồng dữ liệu một cách tuyến tính và hiệu quả nhất.

B. Kỹ thuật Đệm động (Dynamic Padding) Thay vì sử dụng một độ dài cố định (Global Max Length) gây lãng phí bộ nhớ, đồ án áp dụng kỹ thuật đệm động thông qua hàm pad_sequence.

- **Cơ chế:** Hệ thống tự động xác định độ dài lớn nhất *chỉ trong batch hiện tại* và thêm token <pad> vào các câu ngắn hơn để đạt độ dài đó. Ví dụ: Nếu câu dài nhất trong batch là 15 từ, toàn bộ batch sẽ có chiều dài 15, thay vì 100 (độ dài tối đa của tập dữ liệu).

- Cấu hình Tensor: Các tensor đầu ra được định dạng với chiều thời gian đi trước (batch_first=False), tạo ra kích thước [Sequence Length, Batch Size]. Đây là định dạng tối ưu cho tính toán tuần tự của mạng RNN/LSTM trong PyTorch.

C. Chuẩn bị cho Kỹ thuật Packing (Packing Preparation) Đây là kỹ thuật nâng cao được áp dụng để giải quyết triệt để vấn đề lãng phí tính toán trên các token đệm vô nghĩa.

- Nguyên lý: Mạng LSTM thông thường sẽ thực hiện phép nhân ma trận trên mọi token, bao gồm cả <pad> (vốn có giá trị 0 hoặc vector rỗng). Điều này gây lãng phí tài nguyên tính toán và có thể gây nhiễu nếu không mask kỹ.

- Giải pháp: Lớp Collate tính toán và trả về một tensor src_lens chứa độ dài thực tế của từng câu nguồn. Thông tin này được chuyển xuống lớp Encoder,

tại đây hàm pack_padded_sequence sẽ "đóng gói" dữ liệu, loại bỏ hoàn toàn các bước thời gian đệm (padding steps) khỏi quá trình tính toán của LSTM, giúp tăng tốc độ huấn luyện đáng kể.

4.1.4 So sánh Seq2Seq cổ điển (No Attention) và Seq2Seq có Attention.

1. Hình ảnh so sánh (Analogy)

Seq2Seq (Không Attention)	**Seq2Seq (Có Attention)**
"Học vẹt & Trả bài"	"Vừa nhìn vừa dịch"
Bạn đọc hết câu tiếng Anh, gấp sách lại, cố gắng nhớ toàn bộ ý nghĩa trong đầu (nén vào 1 vector duy nhất), rồi viết ra câu tiếng Pháp.	Bạn đọc câu tiếng Anh, nhưng khi viết câu tiếng Pháp, bạn mở sách ra và ngón tay bạn chỉ vào từ tiếng Anh nào quan trọng nhất để dịch từ hiện tại.
Rủi ro: Nếu câu quá dài, bạn sẽ quên mất đoạn đầu (đầu voi đuôi chuột).	Lợi ích: Bạn không bao giờ quên thông tin, dù câu dài đến đâu.

2. Bảng so sánh kỹ thuật

Tiêu chí	Seq2Seq (LSTM thường)	Seq2Seq + Attention
Cơ chế truyền tin	Static Context Vector: Toàn bộ câu nguồn bị nén vào 1 vector duy nhất (trạng thái ẩn cuối cùng của Encoder).	Dynamic Context Vector: Tại mỗi bước dịch, Decoder nhìn lại toàn bộ các trạng thái của Encoder và chọn ra phần cần thiết.
Nút thắt cổ chai (Bottleneck)	Có. Mọi thông tin (ngữ nghĩa, ngữ pháp, màu sắc...) phải nhét vừa vào vector kích thước cố định (VD: 512 chiều).	Không. Decoder có thể truy cập trực tiếp vào bất kỳ từ nào trong câu nguồn.
Hiệu suất câu dài	Kém. Câu càng dài, chất lượng dịch càng giảm mạnh (do quên thông tin đầu câu).	Tốt. Ổn định với câu dài vì nó có thể "nhìn lại" từ đầu câu bất cứ lúc nào.
Khả năng giải thích	Hộp đen (Black Box). Khó biết model đang nghĩ gì.	Minh bạch. Có thể vẽ Attention Map để biết khi dịch từ "Noir", model đang nhìn vào từ "Black".
Tốc độ tính toán	Nhanh hơn (phép tính đơn giản).	Chậm hơn một chút (phải tính điểm số Attention tại mỗi bước).

4.2. Quá trình Huấn luyện

4.2.1. Thiết lập Siêu tham số:

Các tham số được lựa chọn dựa trên thực nghiệm và khuyến nghị cho kiến trúc Seq2Seq trên dữ liệu kích thước trung bình:

Tham số	Giá trị	Giải thích kỹ thuật
Optimizer	Adam	Tự động điều chỉnh Learning Rate, hội tụ nhanh hơn SGD.
Learning Rate	0.001	Tốc độ học khởi tạo.
Batch Size	128	Tận dụng khả năng tính toán song song của GPU.
Embedding Dim	256	Kích thước không gian vector biểu diễn từ.
Hidden Dim	512	Kích thước bộ nhớ ngữ cảnh của LSTM.
Dropout	0.5	Regularization để giảm Overfitting.

4.2.2. Chiến lược Tối ưu hóa và Kiểm soát lỗi (Optimization & Error Control Strategy)

Để đảm bảo mô hình Sequence-to-Sequence hội tụ ổn định và khắc phục các nhược điểm cố hữu của mạng nơ-ron hồi quy (RNN), đồ án đã triển khai đồng bộ bốn chiến lược kỹ thuật sau:

A. Hàm mất mát với Cơ chế Masking (Loss Function with Masking)

Hệ thống sử dụng hàm mất mát CrossEntropyLoss để tính toán độ sai lệch giữa phân phối xác suất dự đoán và từ đích thực tế. Tuy nhiên, do việc sử dụng Token đệm (<pad>) để đồng nhất kích thước Batch, nếu tính toán cả lỗi trên các token này, Gradient sẽ bị nhiễu và mô hình sẽ "học" **c**ách dự đoán khoảng trắng vô nghĩa.

- **Giải pháp:** Thiết lập tham số ignore_index trong hàm loss trỏ đến chỉ số của token <pad> trong từ điển.
- **Ý nghĩa:** Cơ chế này hoạt động như một lớp mặt nạ (Masking), chỉ đạo cho hàm Loss bỏ qua hoàn toàn các vị trí đệm. Kết quả là trọng số mô hình chỉ được cập nhật dựa trên các từ vựng có nội dung thực tế.

B. Thuật toán Tối ưu hóa & Lịch trình học (Optimizer & Scheduler)

Đồ án lựa chọn thuật toán Adam với tốc độ học (Learning Rate) khởi tạo là 0.001, kết hợp với cơ chế giảm tốc độ học tự động để tinh chỉnh quá trình hội tụ.

- **Adam**: Được lựa chọn nhờ khả năng tự điều chỉnh tốc độ học thích ứng (adaptive learning rate) cho từng tham số, giúp mô hình thoát khỏi các điểm cực tiểu địa phương nhanh hơn so với SGD truyền thống.
- **ReduceLROnPlateau**: Hệ thống tích hợp lr_scheduler. Nếu Loss trên tập Validation không giảm sau 1 epoch (patience=1), tốc độ học sẽ tự động giảm đi 10 lần (factor=0.1). Điều này giúp mô hình "tinh chỉnh" (fine-tune) chính xác hơn khi đã tiến gần đến điểm cực tiểu toàn cục.

C. Chiến lược Teacher Forcing (Hỗ trợ giải mã)

Để khắc phục hiện tượng "sai lầm dây chuyền" (Error Propagation) khi huấn luyện Decoder, đồ án áp dụng kỹ thuật Teacher Forcing với tỷ lệ 0.5.

- **Cơ chế:** Tại mỗi bước thời gian t, hệ thống tung một đồng xu (xác suất 50%) để quyết định đầu vào cho bước t+1:

  + **Trường hợp 1 (Teacher Forcing):** Sử dụng từ chính xác thực tế (Ground Truth). Việc này giúp "nắn" lại đường đi của Decoder ngay cả khi nó vừa dự đoán sai.

  + **Trường hợp 2 (Free Running):** Sử dụng từ dự đoán Y_t (top1) của chính mô hình. Điều này giúp mô hình rèn luyện khả năng tự sinh từ và tránh bị phụ thuộc hoàn toàn vào dữ liệu mẫu (Exposure Bias).

D. Kỹ thuật Cắt Gradient (Gradient Clipping)

Đây là chiến lược kiểm soát lỗi quan trọng thứ tư, được áp dụng đặc biệt cho mạng LSTM để giải quyết vấn đề Bùng nổ Gradient (Exploding Gradient) – nơi đạo hàm tích lũy qua nhiều bước thời gian trở nên quá lớn gây lỗi NaN.

- **Giải pháp**: Áp dụng kỹ thuật Gradient Clipping thông qua hàm torch.nn.utils.clip_grad_norm_ ngay trước bước cập nhật trọng số.
- **Thiết lập:** Ngưỡng cắt (Clip Value) được đặt là 1.0.
- **Cơ chế:** Hệ thống kiểm tra chuẩn L2 (Norm) của toàn bộ vector Gradient. Nếu giá trị này vượt quá 1.0, vector Gradient sẽ được co lại (rescaled) để giữ nguyên hướng nhưng giảm độ lớn về mức an toàn. Điều này đảm bảo quá trình huấn luyện diễn ra ổn định, tránh việc trọng số bị cập nhật quá mạnh gây văng khỏi vùng tối ưu.

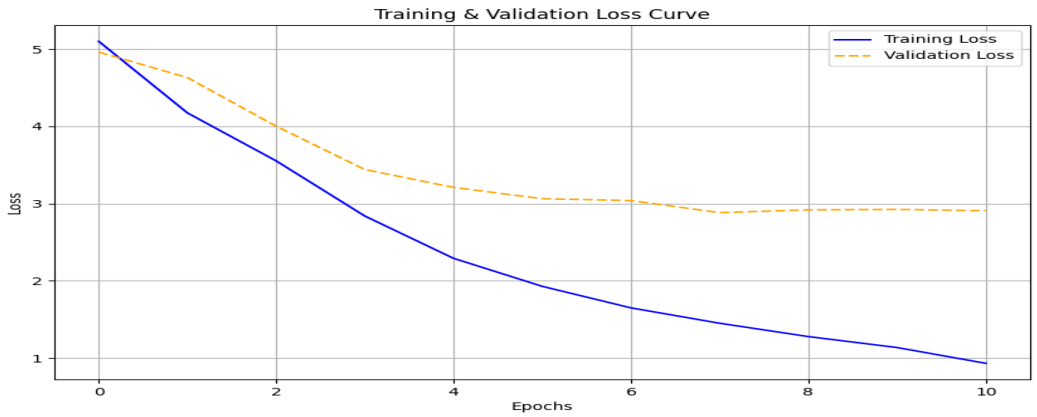
4.3. Kết quả và Đánh giá

4.3.1. Phương pháp Đánh giá:

- **Định lượng (Quantitative):** Sử dụng độ đo **BLEU Score** (Bilingual Evaluation Understudy) thông qua thư viện NLTK. Đây là tiêu chuẩn vàng trong dịch máy để đo lường độ trùng khớp n-gram giữa văn bản máy dịch và văn bản tham chiếu.

- **Định tính (Qualitative):** Kiểm tra khả năng dịch thực tế thông qua các trường hợp kiểm thử (Test Cases) cụ thể.

4.3.2. Phân tích Kết quả Thực nghiệm



Hình 4.1 Biểu đồ Train/val loss

-Nhận xét biểu đồ:

A. Tổng quan quá trình hội tụ Biểu đồ hể hiện sự biến thiên của hàm mất mát (Loss Function) trên tập huấn luyện (Training Set) và tập kiểm thử (Validation Set) qua các epoch đối với mô hình Seq2Seq sử dụng cơ chế Attention. Quá trình huấn luyện được thực hiện với kích thước batch là 128 và sử dụng bộ tối ưu hóa Adam với Learning Rate khởi tạo là 0.001.

B. Phân tích chi tiết Quan sát biểu đồ cho thấy hai giai đoạn rõ rệt của quá trình học:

- **Giai đoạn giảm nhanh (Epoch 0 - 5):** Trong 5 epoch đầu tiên, cả *Training Loss* (đường màu xanh) và *Validation Loss* (đường màu cam) đều giảm mạnh và đồng biến. Cụ thể, Training Loss giảm từ mức trên 5.0 xuống còn khoảng 2.0, và Validation Loss giảm từ 5.0 xuống 3.0. Điều này chứng tỏ kiến trúc Attention hoạt động hiệu quả, giúp mô hình nhanh chóng nắm bắt được các đặc trưng ngữ nghĩa và cấu trúc câu để thực hiện tác vụ dịch thuật.

- **Giai đoạn phân tách và bão hòa (Epoch 6 - 10):** Từ epoch thứ 6 trở đi, xu hướng của hai đường bắt đầu có sự phân tách rõ rệt:

  + **Training Loss** tiếp tục giảm sâu một cách đều đặn, đạt mức thấp nhất xấp xỉ **0.9** tại epoch cuối cùng. Điều này cho thấy mô hình có khả năng học và ghi nhớ dữ liệu huấn luyện rất tốt.

  + **Validation Loss** có xu hướng đi ngang (bão hòa) dao động quanh ngưỡng **2.9** và không còn dấu hiệu giảm thêm đáng kể.

C. Đánh giá hiện tượng Overfitting và cơ chế Early Stopping Khoảng cách giữa đường *Training Loss* và *Validation Loss* bắt đầu nới rộng từ sau epoch thứ 5 là dấu hiệu điển hình của hiện tượng *quá khớp (Overfitting)*. Mặc dù mô hình đã được áp dụng Dropout với tỉ lệ 0.5 tại cả Encoder và Decoder, việc Training Loss giảm xuống dưới 1.0 trong khi Validation Loss dậm chân tại chỗ cho thấy mô hình đang bắt đầu "học vẹt" các mẫu trong tập huấn luyện thay vì tổng quát hóa tốt hơn trên dữ liệu mới.

Tuy nhiên, cơ chế *Early Stopping* (dừng sớm) đã hoạt động chính xác. Theo thiết lập trong mã nguồn với tham số patience=3, quá trình huấn luyện đã tự động kết thúc tại epoch thứ 11 (trên biểu đồ là mốc 10) khi nhận thấy Validation Loss không được cải thiện sau 3 chu kỳ liên tiếp. Điều này giúp ngăn chặn việc lãng phí tài nguyên tính toán và giữ lại được trọng số mô hình tốt nhất (tại thời điểm Validation Loss thấp nhất trước khi bão hòa).

D. Kết luận Mô hình đạt trạng thái tối ưu trên tập Validation với mức Loss xấp xỉ *2.9.* So với các kiến trúc Seq2Seq cơ bản (thường có Loss > 3.3), sự cải thiện này khẳng định vai trò quan trọng của cơ chế Attention trong việc giải quyết vấn đề nút thắt cổ chai thông tin (Information Bottleneck), dù vẫn còn tồn tại xu hướng overfitting nhẹ cần khắc phục bằng các kỹ thuật điều chuẩn (regularization) mạnh hơn trong tương lai.

- Đánh giá kết quả BLEU SCORE:

```
3. BẮT ĐẦU HUẤN LUYỆN...
--------------------------------------------------------------------------
 Epoch  |   Time    | Train Loss  |   Val Loss   | Status
--------------------------------------------------------------------------
   1    |  01m 10s  |    5.0625   |    4.8455    | Lưu Model
   2    |  01m 09s  |    4.1010   |    4.5321    | Lưu Model
   3    |  01m 14s  |    3.3501   |    3.6539    | Lưu Model
   4    |  01m 15s  |    2.6179   |    3.2607    | Lưu Model
   5    |  01m 14s  |    2.1201   |    3.0847    | Lưu Model
   6    |  01m 15s  |    1.7993   |    2.9809    | Lưu Model
   7    |  01m 14s  |    1.5572   |    2.9058    | Lưu Model
   8    |  01m 15s  |    1.3876   |    2.8919    | Lưu Model
   9    |  01m 16s  |    1.2337   |    2.8983    | Wait (1/3)
  10    |  01m 15s  |    1.1033   |    2.9955    | Wait (2/3)
  11    |  01m 16s  |    0.9165   |    2.9266    | Wait (3/3)
--------------------------------------------------------------------------
Early Stopping kích hoạt! Val loss không giảm sau 3 epochs.
--------------------------------------------------------------------------

4. KẾT QUẢ CUỐI CÙNG

------------------------------------------
 ĐANG TẠO BÁO CÁO ĐÁNH GIÁ ...
 BLEU Score trung bình (Test): 0.2240
```

Hình 4.2 Hình ảnh kết quả BLEU SCORE

Sau quá trình huấn luyện, mô hình được đánh giá trên tập Test độc lập. Kết quả ghi nhận điểm *BLEU đạt 0.224 (tương đương 22.4%).*

Theo thang đo chuẩn của BLEU, mức điểm này thường nằm trong ngưỡng "dịch được ý chính" (understandable/fair interpretation). Điều này chứng tỏ kiến trúc mô hình hiện tại đã học được các mối liên kết từ vựng giữa nguồn và đích (alignment). Mặc dù vậy, với mức điểm 0.224, các bản dịch đầu ra có thể vẫn còn tồn tại các lỗi về ngữ pháp hoặc lựa chọn từ chưa hoàn toàn tự nhiên so với người bản xứ.

4.3.3 Các lỗi thường gặp và đề xuất cải tiến

1. Vấn đề "Out-of-Vocabulary" (Quá nhiều từ <unk>)

Input: "Obama is visiting Hanoi."

Output: <unk> is visiting <unk>.

(Mất hoàn toàn thông tin quan trọng).

- Nguyên nhân

1. **Từ chưa thấy (Unseen):** Từ ngữ (tên riêng "Obama", "Hanoi") chưa từng xuất hiện trong tập dữ liệu huấn luyện.
2. **Bị lọc bỏ (Filtered):** Từ xuất hiện quá ít (không đạt ngưỡng min_freq) hoặc nằm ngoài giới hạn kích thước từ điển (max_vocab_size), nên bị gán nhãn chung là <unk>.

- Đề xuất

1. **Sử dụng Subword Tokenization (BPE) (Khuyên dùng):** Chia từ lạ thành các đơn vị nhỏ hơn đã biết để dịch (Ví dụ: Hanoi $\rightarrow$ Ha + noi). Đây là chuẩn của các mô hình hiện đại (như GPT, Google Translate).
2. **Cơ chế Copy (Pointer Network):** Cho phép mô hình "sao chép" trực tiếp các thực thể (tên riêng, số) từ câu nguồn sang câu đích thay vì cố gắng dịch.
3. **Mở rộng từ điển:** Tăng max_vocab_size và giảm ngưỡng lọc từ hiếm min_freq.

2. Vấn đề Hallucination (Ảo giác)

Input: "i love you."

Output: "Un plongeur apprenant à l' école" (Một thợ lặn đang học ở trường).

Nguyên nhân:

1. Lệch dữ liệu (Chính): Mô hình được huấn luyện trên bộ dữ liệu Multi30k (chuyên mô tả hình ảnh: người, chó, chạy nhảy...) nên hoàn toàn không biết cách dịch các câu giao tiếp đời thường như "yêu", "chào".
2. Mù từ vựng (OOV): Các từ "love", "you" có thể không có trong từ điển (bị chuyển thành <unk>), khiến mô hình không hiểu đầu vào.
3. Học vẹt (Overfitting): Khi không hiểu đầu vào, mô hình tự động sinh ra một câu nó đã "học thuộc lòng" kỹ nhất trong quá trình huấn luyện để lấp chỗ trống.

Đề xuất:

1. Test đúng cách: Thử lại bằng câu mô tả hành động (VD: *"A man is running"*).
2. Đổi dữ liệu: Nếu muốn dịch giao tiếp, cần huấn luyện lại trên bộ dữ liệu hội thoại (như OpenSubtitles hoặc TED Talks).
3. Cải thiện Tokenizer: Sử dụng BPE (Byte-Pair Encoding) để mô hình xử lý được các từ lạ thay vì biến hết thành <unk>.

3. Lặp từ (Repetition Problem) / Vòng lặp vô tận (Infinite Loop).

Input: "i love girl"

Output: "Une petite fille fille fille fille fille fille"

Nguyên nhân:

1. Thuật toán Decoding thiếu kiểm soát: Chưa cài đặt cơ chế N-gram Blocking (chặn lặp từ). Khi mô hình LSTM dự đoán từ "fille" có xác suất cao nhất, trạng thái ẩn (hidden state) cập nhật không đủ lớn để chuyển sang từ khác, khiến Decoder cứ thế chọn lại từ này mãi mãi.
2. Attention bị kẹt: Cơ chế Attention trong model.py có thể đang tập trung (attend) vào cùng một từ nguồn (ví dụ: "girl") lặp đi lặp lại qua các bước thời gian.
3. Lệch miền dữ liệu (Domain Mismatch): Bộ dữ liệu Multi30k dùng để huấn luyện là về mô tả ảnh, không có các câu biểu lộ cảm xúc như "I love...". Mô hình gặp Input lạ, chỉ nhận ra từ "girl" và bám víu vào đó để sinh từ, dẫn đến lỗi lặp.

Đề xuất:

1. Sửa thuật toán (Nhanh nhất): Thêm logic cấm lặp (Block N-gram) vào hàm beam_search_translate trong main.py. Nếu từ dự đoán trùng với từ vừa sinh ra trước đó, hãy gán điểm xác suất cực thấp cho nó.
2. Coverage Penalty: Cài đặt thêm cơ chế phạt nếu Attention tập trung vào một từ quá nhiều lần.
3. Tăng Alpha: Trong beam_search_translate, thử giảm tham số alpha (Length Penalty) hoặc giới hạn độ dài câu chặt chẽ hơn để cắt ngắn vòng lặp.

4. Lỗi thừa từ (Insertion Error)

Input: " Two men are talking."

Output: " Des deux hommes parlent'.

- **Nguyên nhân**

  Model bị "lưỡng lự":

  1 Nó thấy trong data hầu hết các câu số nhiều đều bắt đầu bằng **"Des"**.

  2 Nó cũng biết **"Two"** phải dịch là **"Deux"**.

  3 **Kết quả:** Nó gộp cả hai thành **"Des deux..."** để "cho chắc ăn".

- Đề xuất:

1.tăng beam_width lên **10** (mặc định là 3). Model sẽ đủ thông minh để nhận ra cụm "Des deux" có xác suất thấp hơn "Deux" khi xét cả câu.

2. Train thêm **10-20 epochs** nữa. Model hiện tại chưa "hội tụ" đủ để phân biệt rạch ròi ngữ pháp.

3. Sử dụng khởi tạo trọng số **Xavier** (thay vì Normal) trong hàm init_weights. Nó giúp model tránh bị kẹt ở các từ phổ biến như "Des".

5. Lỗi Tổng Quát Hóa Chủ Ngữ (Subject Generalization Error)

Input: "He listens to music".

Output: "Quelqu'un écoute de la musique".

- **Nguyên nhân**

  Do học vẹt dữ liệu: Bộ dữ liệu Multi30k (mô tả ảnh) chứa rất nhiều câu bắt đầu bằng *"Someone..."* hoặc *"A person..."* (dịch là *Quelqu'un*). Model thấy từ này an toàn và xác suất đúng cao nên dùng bừa, bất chấp từ gốc là "He".

- **Đề xuất**
  1. Thêm dữ liệu (Data Augmentation): Bổ sung vào tập train các câu ngắn có chủ ngữ rõ ràng (*He runs, She eats*) để model học lại.
  2. Vẽ Attention: Kiểm tra xem model có "nhìn" vào từ "He" không. Nếu có mà vẫn dịch sai to lỗi do dữ liệu. Nếu không nhìn to lỗi do Encoder yếu.

Tài liệu tham khảo

- **Sutskever et al. (2014).** Sequence to Sequence Learning with Neural Networks.
- **PyTorch Documentation:** torch.nn.LSTM.
- **Multi30K Dataset:** https://github.com/multi30k/dataset.

Link github dự án:

https://github.com/Huy25-kudo/-NLP-_DoAn_Translate_En_Fr.git

Phụ lục

Phụ lục 1: Code dự án

Model.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
import random

# 1. ENCODER
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, enc_hid_dim, dec_hid_dim, dropout):
        super().__init__()
        self.embedding = nn.Embedding(input_dim, emb_dim)

        # Bidirectional = True
        self.rnn = nn.LSTM(emb_dim, enc_hid_dim, bidirectional=True)

        # Linear layer để gộp 2 chiều (Forward + Backward) thành 1 vector cho Decoder
        self.fc = nn.Linear(enc_hid_dim * 2, dec_hid_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, src, src_len):
        # src: [src len, batch size]
        embedded = self.dropout(self.embedding(src))

        # Pack sequence
        packed_embedded = pack_padded_sequence(embedded, src_len.to('cpu'),
enforce_sorted=True)

        packed_outputs, (hidden, cell) = self.rnn(packed_embedded)


        # outputs: [src len, batch size, enc hid dim * 2]
        outputs, _ = pad_packed_sequence(packed_outputs)

        # Xử lý Hidden state ban đầu cho Decoder
        hidden = torch.tanh(self.fc(torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)))

        return outputs, hidden
```

```
# 2. ATTENTION
class Attention(nn.Module):
    def __init__(self, enc_hid_dim, dec_hid_dim):
        super().__init__()
        # Attention tính điểm dựa trên Hidden state của Decoder và Outputs của Encoder
        self.attn = nn.Linear((enc_hid_dim * 2) + dec_hid_dim, dec_hid_dim)
        self.v = nn.Linear(dec_hid_dim, 1, bias=False)

    def forward(self, hidden, encoder_outputs):
        # hidden: [batch size, dec hid dim]
        # encoder_outputs: [src len, batch size, enc hid dim * 2]

        batch_size = encoder_outputs.shape[1]
        src_len = encoder_outputs.shape[0]

        hidden = hidden.unsqueeze(1).repeat(1, src_len, 1)
        encoder_outputs = encoder_outputs.permute(1, 0, 2)
        # Tính Energy: E = tanh(Wx + Uy)
        energy = torch.tanh(self.attn(torch.cat((hidden, encoder_outputs), dim=2)))

        # Tính Attention score
        attention = self.v(energy).squeeze(2)

        return F.softmax(attention, dim=1)

# 3. DECODER
class Decoder(nn.Module):
    def __init__(self, output_dim, emb_dim, enc_hid_dim, dec_hid_dim, dropout,
attention):
        super().__init__()
        self.output_dim = output_dim
        self.attention = attention
        self.embedding = nn.Embedding(output_dim, emb_dim)

        self.rnn = nn.LSTM((enc_hid_dim * 2) + emb_dim, dec_hid_dim)

        self.fc_out = nn.Linear((enc_hid_dim * 2) + dec_hid_dim + emb_dim,
output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, encoder_outputs):
```

```
        input = input.unsqueeze(0)
        embedded = self.dropout(self.embedding(input))

        # Tính attention weights [batch, src len]
        a = self.attention(hidden, encoder_outputs)
        a = a.unsqueeze(1) # [batch, 1, src len]

        encoder_outputs = encoder_outputs.permute(1, 0, 2)

        # Tính Weighted Context Vector: Tổng trọng số các từ nguồn
        weighted = torch.bmm(a, encoder_outputs)
        weighted = weighted.permute(1, 0, 2)

        # Đưa vào LSTM: [Embedding; Context]
        rnn_input = torch.cat((embedded, weighted), dim=2)

        output, (hidden, cell) = self.rnn(rnn_input, (hidden.unsqueeze(0),
torch.zeros_like(hidden.unsqueeze(0))))

        # Dự đoán từ tiếp theo
        assert (output == hidden).all()
        embedded = embedded.squeeze(0)
        output = output.squeeze(0)
        weighted = weighted.squeeze(0)

        prediction = self.fc_out(torch.cat((output, weighted, embedded), dim=1))

        return prediction, hidden.squeeze(0)

# 4. SEQ2SEQ
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.device = device

    def forward(self, src, src_len, trg, teacher_forcing_ratio=0.5):
        batch_size = src.shape[1]
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.output_dim

        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(self.device)
```

```
    # Encoder trả về outputs (cho Attention) và hidden (cho khởi tạo Decoder)
    encoder_outputs, hidden = self.encoder(src, src_len)

    input = trg[0, :] # <sos>

    for t in range(1, trg_len):
        # Decoder nhận thêm encoder_outputs
        output, hidden = self.decoder(input, hidden, encoder_outputs)
        outputs[t] = output

        teacher_force = random.random() < teacher_forcing_ratio
        top1 = output.argmax(1)
        input = trg[t] if teacher_force else top1

    return outputs
```

Data_ultils

```
import torch
from torch.utils.data import Dataset
from torch.nn.utils.rnn import pad_sequence
from collections import Counter
import os
import sys
import spacy



# 1. TẢI MODEL NGÔN NGỮ
def load_spacy_model(model_name):
    try:
        return spacy.load(model_name)
    except OSError:
        print(f"🔄 Đang tải gói ngôn ngữ {model_name}...")
        os.system(f"{sys.executable} -m spacy download {model_name}")
        return spacy.load(model_name)

spacy_en = load_spacy_model('en_core_web_sm')
spacy_fr = load_spacy_model('fr_core_news_sm')
```

```
# 3. TOKENIZER
def en_tokenizer(text):
    return [tok.text for tok in spacy_en.tokenizer(text)]

def fr_tokenizer(text):
    return [tok.text for tok in spacy_fr.tokenizer(text)]

# 4. CLASS VOCABULARY
class Vocabulary:
    def __init__(self, freq_threshold=2, max_size=10000):
        self.itos = {0: "<pad>", 1: "<sos>", 2: "<eos>", 3: "<unk>"}
        self.stoi = {"<pad>": 0, "<sos>": 1, "<eos>": 2, "<unk>": 3}
        self.freq_threshold = freq_threshold
        self.max_size = max_size

    def __len__(self):
        return len(self.itos)

    def build_vocabulary(self, sentence_list, tokenizer):
        frequencies = Counter()
        idx = 4

        # Đếm tần suất tất cả các từ
        for sentence in sentence_list:
            for word in tokenizer(sentence):
                frequencies[word] += 1

        # Lấy danh sách các từ phổ biến nhất theo max_size trừ đi 4 token đặc biệt
        common_words = frequencies.most_common(self.max_size - 4)

        for word, freq in common_words:
            if freq >= self.freq_threshold:
                self.stoi[word] = idx
                self.itos[idx] = word
                idx += 1

    def __call__(self, tokens):
        return [self.stoi.get(token, self.stoi["<unk>"]) for token in tokens]

    def lookup_token(self, index):
        return self.itos.get(index, "<unk>")

def build_vocab(filepath, tokenizer):
```

```
    print(f"   - Đang đọc file {os.path.basename(filepath)} để xây từ điển (Max 10k)...")
    if not os.path.exists(filepath):
        raise FileNotFoundError(f"Không tìm thấy file: {filepath}")

    vocab = Vocabulary(freq_threshold=2, max_size=10000)
    sentences = [line.strip() for line in open(filepath, encoding='utf-8')]
    vocab.build_vocabulary(sentences, tokenizer)
    return vocab

# 5. DATASET & COLLATE
class Multi30kDataset(Dataset):
    def __init__(self, src_file, trg_file):
        if not os.path.exists(src_file):
            raise FileNotFoundError(f"❌ KHÔNG TÌM THẤY FILE: {src_file}")
        self.src_data = [line.strip() for line in open(src_file, 'r', encoding='utf-8')]
        self.trg_data = [line.strip() for line in open(trg_file, 'r', encoding='utf-8')]

    def __len__(self):
        return len(self.src_data)

    def __getitem__(self, index):
        return self.src_data[index], self.trg_data[index]

class Collate:
    def __init__(self, src_vocab, trg_vocab, device):
        self.src_vocab = src_vocab
        self.trg_vocab = trg_vocab
        self.device = device
        self.pad_idx = src_vocab.stoi["<pad>"]
        self.sos_idx = src_vocab.stoi["<sos>"]
        self.eos_idx = src_vocab.stoi["<eos>"]

    def text_transform(self, text, tokenizer, vocab):
        tokens = tokenizer(text)
        indices = vocab(tokens)
        return torch.tensor([self.sos_idx] + indices + [self.eos_idx], dtype=torch.long)

    def __call__(self, batch):
        src_batch, trg_batch = [], []
        for src_text, trg_text in batch:
            src_batch.append(self.text_transform(src_text, en_tokenizer, self.src_vocab))
            trg_batch.append(self.text_transform(trg_text, fr_tokenizer, self.trg_vocab))
```

```
        batch_zipped = sorted(zip(src_batch, trg_batch), key=lambda x: len(x[0]),
reverse=True)
        src_batch, trg_batch = zip(*batch_zipped)

        src_lens = torch.tensor([len(x) for x in src_batch], dtype=torch.long)
        src_padded = pad_sequence(src_batch, padding_value=self.pad_idx,
batch_first=False)
        trg_padded = pad_sequence(trg_batch, padding_value=self.pad_idx,
batch_first=False)

        return src_padded, trg_padded, src_lens
```

Main.py

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import os
import sys
import time
import random
import matplotlib.pyplot as plt
import warnings
from nltk.translate.bleu_score import sentence_bleu
from model import Encoder, Decoder, Attention, Seq2Seq
warnings.filterwarnings("ignore")
from data_utils import Multi30kDataset, Collate, build_vocab, en_tokenizer,
fr_tokenizer
from model import Encoder, Decoder, Seq2Seq
import torch
import torch.nn.functional as F

def init_weights(m):
    for name, param in m.named_parameters():
        nn.init.uniform_(param.data, -0.08, 0.08)

def count_parameters(model):
```

```
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

def train_epoch(model, iterator, optimizer, criterion, clip, device, scaler):
    model.train()
    epoch_loss = 0
    use_amp = (device.type == 'cuda')

    for i, (src, trg, src_len) in enumerate(iterator):
        src, trg = src.to(device), trg.to(device)
        optimizer.zero_grad()

        if use_amp:
            with torch.cuda.amp.autocast():
                output = model(src, src_len, trg)
                output_dim = output.shape[-1]
                output = output[1:].view(-1, output_dim)
                trg = trg[1:].view(-1)
                loss = criterion(output, trg)

            scaler.scale(loss).backward()
            scaler.unscale_(optimizer)
            torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
            scaler.step(optimizer)
            scaler.update()
        else:
            output = model(src, src_len, trg)
            output_dim = output.shape[-1]
            output = output[1:].view(-1, output_dim)
            trg = trg[1:].view(-1)
            loss = criterion(output, trg)
            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
            optimizer.step()

        epoch_loss += loss.item()

    return epoch_loss / len(iterator)

def evaluate(model, iterator, criterion, device):
    model.eval()
    epoch_loss = 0
    with torch.no_grad():
        for src, trg, src_len in iterator:
```

```
            src, trg = src.to(device), trg.to(device)
            output = model(src, src_len, trg, 0) # Tắt teacher forcing khi eval
            output = output[1:].view(-1, output.shape[-1])
            trg = trg[1:].view(-1)
            loss = criterion(output, trg)
            epoch_loss += loss.item()
    return epoch_loss / len(iterator)

def evaluate_and_report(model, iterator, trg_vocab, device):
    print("\n----------------------------------------")
    print(" ĐANG TẠO BÁO CÁO ĐÁNH GIÁ ...")
    model.eval()

    total_bleu = 0
    examples = []
    trg_itos = trg_vocab.itos

    with torch.no_grad():
        for src, trg, src_len in iterator:
            src = src.to(device)
            trg = trg.to(device)

            output = model(src, src_len, trg, 0)
            output = output.argmax(dim=2)

            trg = trg.transpose(0, 1).cpu().numpy()
            output = output.transpose(0, 1).cpu().numpy()

            for i in range(trg.shape[0]):
                ref_tokens = []
                for idx in trg[i]:
                    if idx in [0, 1]: continue
                    if idx == 2: break
                    ref_tokens.append(trg_itos[idx])

                pred_tokens = []
                for idx in output[i]:
                    if idx == 2: break
                    if idx not in [0, 1]:
                        pred_tokens.append(trg_itos[idx])

                #  Tính BLEU
                score = sentence_bleu([ref_tokens], pred_tokens)
```

```
            total_bleu += score

            examples.append({
                'ref': " ".join(ref_tokens),
                'pred': " ".join(pred_tokens),
                'bleu': score
            })

    avg_bleu = total_bleu / len(examples)
    print(f" BLEU Score trung bình (Test): {avg_bleu:.4f}")
    return avg_bleu

# 1. HÀM DỊCH BEAM SEARCH
def beam_search_translate(sentence, model, src_vocab, trg_vocab, device,
beam_width=3, max_len=50, alpha=1.0):
    model.eval()

    # Bước 1: Xử lý đầu vào
    tokens = en_tokenizer(sentence.lower())
    if not tokens: return ""
    src_indices = src_vocab(tokens)
    # Thêm <sos> và <eos>
    src_tensor = torch.tensor([src_vocab.stoi["<sos>"]] + src_indices +
[src_vocab.stoi["<eos>"]], dtype=torch.long).unsqueeze(1).to(device)
    src_len = torch.tensor([len(src_tensor)], dtype=torch.long)

    with torch.no_grad():
        encoder_outputs, hidden = model.encoder(src_tensor, src_len)

    # Bước 2: Khởi tạo Beam
    beam = [(0, trg_vocab.stoi["<sos>"], hidden, [])]

    # Bước 3: Vòng lặp Decoding
    for _ in range(max_len):
        candidates = []

        # Duyệt qua các nhánh hiện tại trong beam
        for score, token_idx, hid, seq in beam:
            # Nếu nhánh đã gặp <eos>, giữ nguyên nhánh đó
            if token_idx == trg_vocab.stoi["<eos>"]:
                candidates.append((score, token_idx, hid, seq))
                continue
```

```
            # Chuẩn bị input cho decoder
            input_tensor = torch.tensor([token_idx], dtype=torch.long).to(device)

            with torch.no_grad():
                # Decoder cần: input, hidden cũ, và encoder_outputs
                output, new_hidden = model.decoder(input_tensor, hid, encoder_outputs)

            # Tính Log Softmax
            probs = F.log_softmax(output, dim=1)
            topk_probs, topk_ids = probs.topk(beam_width)

            # Mở rộng nhánh
            for k in range(beam_width):
                new_score = score + topk_probs[0][k].item()
                new_token = topk_ids[0][k].item()
                candidates.append((new_score, new_token, new_hidden, seq +
[new_token]))

        # Bước 4: Sắp xếp và Cắt tỉa (Pruning)
        ordered = sorted(candidates, key=lambda x: x[0] / ((len(x[3]) + 1) ** alpha),
reverse=True)

        # Lấy top k nhánh tốt nhất
        beam = ordered[:beam_width]

        # Điều kiện dừng sớm: Nếu tất cả các nhánh trong beam đều đã kết thúc bằng
<eos>
        if all(x[1] == trg_vocab.stoi["<eos>"] for x in beam):
            break

    # --- Bước 5: Chọn kết quả tốt nhất ---
    # Tìm nhánh có điểm normalized cao nhất
    best_score = -float('inf')
    best_seq = None

    for score, token_idx, _, seq in beam:
        # Length Normalization: score / (len^alpha)
        norm_score = score / (len(seq) ** alpha)
        if norm_score > best_score:
            best_score = norm_score
            best_seq = seq
    # Chuyển index về từ vựng
```

```
    tokens = [trg_vocab.lookup_token(idx) for idx in best_seq if idx !=
trg_vocab.stoi["<eos>"]]
    return " ".join(tokens)

# 2. HÀM DỊCH GREEDY
def translate(sentence, model, src_vocab, trg_vocab, device):
    model.eval()
    try:
        tokens = en_tokenizer(sentence.lower())
        if not tokens: return ""
        src_indices = src_vocab(tokens)
    except:
        return "<error>"

    src_tensor = torch.tensor([src_vocab.stoi["<sos>"]] + src_indices +
[src_vocab.stoi["<eos>"]], dtype=torch.long).unsqueeze(1).to(device)
    src_len = torch.tensor([len(src_tensor)], dtype=torch.long)

    with torch.no_grad():
        # Lấy encoder_outputs cho Attention
        encoder_outputs, hidden = model.encoder(src_tensor, src_len)

    trg_indices = [trg_vocab.stoi["<sos>"]]

    for _ in range(50):
        trg_tensor = torch.tensor([trg_indices[-1]], dtype=torch.long).to(device)

        with torch.no_grad():
            output, hidden = model.decoder(trg_tensor, hidden, encoder_outputs)

        pred_token = output.argmax(1).item()

        if pred_token == trg_vocab.stoi["<eos>"]:
            break
        trg_indices.append(pred_token)
    return " ".join([trg_vocab.lookup_token(idx) for idx in trg_indices[1:]]
```

Phụ lục 2:Phân công nhiệm vụ

Họ và tên	MSSV	Nhiệm vụ	Hoàn thành
Tô Gia Huy	3123410131	Code: Model.py,Main.py Báo cáo :chương 3,4	100%
Hồ hoàng khang	3123410150	Code:EDA, data_ultils.py Báo cáo:Chương 1,2	100%