HỌC VIỆN NGÂN HÀNG
KHOA CÔNG NGHỆ THÔNG TIN & KINH TẾ SỐ

BÀI TẬP LỚN
TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI
SO SÁNH VÀ PHÂN LOẠI VĂN BẢN:AI VÀ CON NGƯỜI SỬ DỤNG TRANSFORMER ENCODER

GIẢNG VIÊN HƯỚNG DẪN	**:**	**TS. Vũ Trọng Sinh**
MÃ LỚP HỌC PHẦN	**:**	**242IS54A03**
NHÓM	**:**	**19**

HANOI – T6/2025

HỌC VIỆN NGÂN HÀNG
KHOA CÔNG NGHỆ THÔNG TIN & KINH TẾ SỐ

BÀI TẬP LỚN
TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI
SO SÁNH VÀ PHÂN LOẠI VĂN BẢN:AI VS CON NGƯỜI SỬ DỤNG TRANSFORMER ENCODER

Giảng viên hướng dẫn : TS. Vũ Trọng Sinh

Nhóm lớp học phần : 242IS54A03

Nhóm : 19

STT	Mã sinh viên	Họ và Tên
1	25A4041546	Mai Thái Huy (NT)
2	25A4041899	Lê Văn Nghĩa
3	25A4041900	Ma Thế Ngọc
4	25A4041905	Lưu Đức Quang

HANOI – T6/2025

DANH SÁCH THÀNH VIÊN NHÓM

MÃ SINH VIÊN	HỌ VÀ TÊN	NỘI DUNG CÔNG VIỆC	TỶ LỆ CÔNG VIỆC	KÝ XÁC NHẬN
25A4041546	Mai Thái Huy (NT)	• Tìm hiểu yêu cầu đề tài • Thu thập dữ liệu • Phân công công việc • Cài đặt mô hình • Triển khai streamlit		
25A4041899	Lê Văn Nghĩa	• Tìm hiểu đề tài • Tìm hiểu cơ sở lý thuyết • Thu thập dữ liệu • Trực quan hóa kết quả		
25A4041900	Ma Thế Ngọc	• Tìm hiểu đề tài • Tìm hiểu cơ sở lý thuyết • Thu thập dữ liệu • Huấn luyện mô hình		
25A4041905	Lưu Đức Quang	• Tìm hiểu đề tài • Tìm hiểu cơ sở lý thuyết • Thu thập dữ liệu • Tiền xử lý dữ liệu		

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI CAM ĐOAN

Chúng em, nhóm thực hiện bài tập lớn học phần Trí tuệ nhân tạo, xin cam kết rằng báo cáo với đề tài ***"So sánh và Phân loại Văn bản: AI vs Con Người sử dụng transformer encoder"*** là kết quả nghiên cứu độc lập của nhóm dưới sự hướng dẫn tận tình của giảng viên phụ trách.

Toàn bộ nội dung trình bày trong báo cáo là sản phẩm do nhóm chúng em tự tìm tòi, nghiên cứu và thực hiện trong quá trình làm bài tập lớn. Chúng em cam kết không sao chép, sao chép hoặc sử dụng bất kỳ nội dung nào từ cá nhân, tổ chức hoặc tài liệu nào khác mà không được trích dẫn nguồn chính xác. Tất cả các tài liệu, dữ liệu và thông tin tham khảo được sử dụng trong báo cáo đều được ghi nhận rõ ràng, đầy đủ và tuân thủ các quy định về trích dẫn học thuật.

Chúng em xin chịu hoàn toàn trách nhiệm về tính trung thực và minh bạch của nội dung báo cáo. Nếu có bất kỳ sai phạm nào liên quan đến tính xác thực hoặc đạo đức học thuật được phát hiện, nhóm chúng em sẵn sàng chấp nhận mọi hình thức xử lý theo quy định của nhà trường và học phần.

Chúng em hy vọng rằng báo cáo này sẽ phản ánh đúng nỗ lực và tâm huyết của nhóm trong quá trình học tập và nghiên cứu.

Nhóm sinh viên thực hiện

Nhóm 19

LỜI CẢM ƠN

Chúng em xin bày tỏ lòng biết ơn chân thành đến quý thầy cô thuộc Khoa Công nghệ Thông tin và Kinh tế Số, Học viện Ngân hàng, đã tận tình giảng dạy, truyền đạt những kiến thức chuyên môn quý báu và tạo mọi điều kiện thuận lợi để chúng em hoàn thành tốt quá trình học tập và nghiên cứu trong học phần Trí tuệ nhân tạo.

Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc đến thầy Vũ Trọng Sinh, người đã tận tâm hướng dẫn, định hướng và đóng góp những ý kiến giá trị, giúp nhóm chúng em hoàn thiện bài báo cáo với đề tài ***"So sánh và Phân loại Văn bản: AI vs Con Người sử dụng transformer encoder"***. Sự hỗ trợ và động viên từ thầy không chỉ giúp chúng em nắm vững hơn về các phương pháp nghiên cứu mà còn rèn luyện được nhiều kỹ năng thực tiễn quan trọng, từ tư duy phân tích đến ứng dụng công nghệ trong quá trình thực hiện đề tài.

Dù nhóm chúng em đã nỗ lực hết mình để hoàn thành bài báo cáo, nhưng do hạn chế về thời gian và kinh nghiệm, báo cáo khó tránh khỏi những thiếu sót. Chúng em rất mong nhận được những ý kiến đóng góp quý báu từ thầy cô để có thể cải thiện và hoàn thiện hơn trong các nghiên cứu tiếp theo.

Cuối cùng, chúng em xin kính chúc quý giảng viên luôn dồi dào sức khỏe, hạnh phúc và đạt được nhiều thành công trong sự nghiệp giảng dạy và nghiên cứu.

CHƯƠNG I. GIỚI THIỆU BÀI TOÁN

1.1. Phát biểu bài toán

Sự bùng nổ của trí tuệ nhân tạo (AI) trong những năm gần đây, đặc biệt là các mô hình ngôn ngữ lớn như GPT-4.5, BERT, LLaMA, đã tạo ra một bước ngoặt trong cách con người sản xuất và tiêu thụ nội dung văn bản. Theo thống kê từ báo cáo của OpenAI năm 2025, các mô hình AI có khả năng tạo ra hơn 100 tỷ từ mỗi ngày trên toàn cầu, chiếm khoảng 30% tổng lượng nội dung trực tuyến vào năm 2025 (Quỳnh, 2025). Các văn bản do AI tạo ra, từ bài luận học thuật, bài viết sáng tạo, đến các bài đăng trên mạng xã hội, ngày càng trở nên tinh vi, với ngữ pháp, ngữ nghĩa và phong cách gần như không thể phân biệt với văn bản do con người viết. Một khảo sát của Pew Research Center năm 2024 chỉ ra rằng 65% người dùng internet không thể nhận biết chính xác nội dung do AI tạo ra khi so sánh với nội dung do con người viết (Huy, 2024). Điều này đặt ra một thách thức lớn: ***"Làm thế nào để phân biệt được nguồn gốc của văn bản trong bối cảnh nội dung trực tuyến ngày càng tràn ngập thông tin từ cả con người và máy móc?"***

Thách thức này không chỉ mang tính kỹ thuật mà còn có ý nghĩa xã hội sâu sắc. Trong lĩnh vực giáo dục, các công cụ trí tuệ nhân tạo đã được sử dụng để tạo bài luận hoặc bài tập, làm gia tăng nguy cơ gian lận học thuật. Trong lĩnh vực truyền thông, các bài viết do AI tạo ra có thể được sử dụng để lan truyền thông tin sai lệch hoặc quảng cáo trá hình, gây ảnh hưởng đến niềm tin của công chúng. Theo báo cáo của World Association of News Publishers (WAN-IFRA) năm 2024, khoảng 12% nội dung tin tức trên các nền tảng trực tuyến có nguy cơ bị thao túng bởi các công cụ AI (ĐẶNG, 2024). Những con số này nhấn mạnh sự cần thiết của một giải pháp tự động, chính xác và hiệu quả để phân loại văn bản, từ đó đảm bảo tính minh bạch và xác thực của nội dung.

Nhóm chúng em thực hiện bài toán ***"So sánh và Phân loại Văn bản: AI vs Con Người sử dụng Transformer Encoder"*** nhằm giải quyết thách thức này. Mục tiêu của bài toán là xây dựng một mô hình học sâu dựa trên kiến trúc ***Transformer Encoder***, một cấu trúc mạnh mẽ trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP), để phân loại nhị phân các đoạn văn bản, xác định liệu chúng được ***tạo bởi AI (nhãn 1)*** hay ***do con người viết (nhãn 0)***. Bài toán này không chỉ có ý nghĩa học thuật trong việc khám phá khả năng của các mô hình học sâu mà còn mang lại giá trị thực tiễn trong việc bảo vệ tính nguyên bản của nội dung và ngăn chặn các ứng dụng tiêu cực của công nghệ AI.

- ***Input:*** Dữ liệu đầu vào là các đoạn văn bản tiếng Anh, được thu thập từ bộ dữ liệu công khai trên ***Kaggle: AI vs Human Text***. Bộ dữ liệu này bao gồm ***500.000 đoạn văn bản***, với độ dài trung bình từ 200 đến 500 từ, thuộc ***nhiều thể loại*** như bài luận học thuật, văn bản sáng tạo, mô tả thông tin, và các đoạn văn ngắn từ các nguồn trực tuyến. Mỗi mẫu dữ liệu được gắn nhãn: ***0 (văn bản do con người viết)*** hoặc ***1 (văn bản do AI tạo ra)***. Dữ liệu được lưu trữ dưới định dạng CSV với hai cột chính: ***cột text*** chứa nội dung văn bản và ***cột generated*** chứa nhãn tương ứng. Theo thống kê từ bộ dữ liệu, khoảng 52% mẫu là văn bản do AI tạo ra, và 48% là văn bản do con người viết, đảm bảo tính cân bằng tương đối giữa hai lớp nhãn.
- ***Output:*** Mô hình trả về một giá trị xác suất trong khoảng [0, 1], biểu thị khả năng một đoạn văn bản được tạo bởi AI. Giá trị này được ngưỡng hóa (thường tại 0.5) để đưa ra nhãn phân loại cuối cùng: 0 (con người) hoặc 1 (AI). Ví dụ, một giá trị xác suất 0.75 sẽ được phân loại là văn bản do AI tạo ra, trong khi giá trị 0.12 sẽ được phân loại là văn bản do con người viết.
- ***Dạng dữ liệu:*** Dữ liệu đầu vào là văn bản ***phi cấu trúc (unstructured text)***, yêu cầu các bước tiền xử lý như làm sạch văn bản (loại bỏ ký tự đặc biệt, chuẩn hóa khoảng trắng, chuyển thành chữ thường), token hóa, và mã hóa thành các vector số để phù hợp với đầu vào của mô hình Transformer Encoder. Nhãn đầu ra là dạng nhị phân (binary), được biểu diễn dưới dạng số nguyên (0 hoặc 1).

1.2. Ứng dụng của bài toán

1.2.1. Ứng dụng thực tế

- ***Kiểm tra tính xác thực của nội dung:*** Các nền tảng xuất bản trực tuyến, như Medium, Reddit, hoặc các trang báo điện tử, có thể sử dụng mô hình này để phát hiện các bài viết được tạo tự động bởi AI, từ đó đảm bảo tính minh bạch và chất lượng nội dung. Ví dụ, công cụ Turnitin đã tích hợp các tính năng phát hiện văn bản AI trong hệ thống kiểm tra đạo văn của mình để hỗ trợ giáo viên và các tổ chức giáo dục (Smodin, 2025).
- ***Ngăn chặn thông tin sai lệch:*** Trong lĩnh vực truyền thông và mạng xã hội, mô hình có thể được sử dụng để nhận diện các bài viết hoặc bình luận được tạo bởi AI nhằm lan truyền thông tin sai lệch hoặc quảng cáo trá hình.
- ***Hỗ trợ giáo dục:*** Trong môi trường học thuật, mô hình giúp phát hiện các bài luận hoặc bài tập được tạo bởi các công cụ như ChatGPT, từ đó đảm bảo tính công bằng trong đánh giá học sinh, sinh viên. Một ví dụ thực tế là công cụ GPTZero, được thiết kế để phân biệt văn bản AI và con người trong bối cảnh giáo dục (Quân, 2025).
- ***Ứng dụng trong pháp lý và xuất bản:*** Các nhà xuất bản hoặc cơ quan pháp lý có thể sử dụng mô hình này để kiểm tra tính nguyên bản của nội dung, đặc biệt trong các trường hợp tranh chấp bản quyền hoặc xác minh nguồn gốc tài liệu.

1.2.2. Giới hạn phạm vi ứng dụng

- ***Địa lý:*** Mô hình hiện tại được xây dựng dựa trên bộ dữ liệu văn bản tiếng Anh, do đó chủ yếu phù hợp với các thị trường sử dụng tiếng Anh như Mỹ, Anh, Úc, hoặc các quốc gia có sử dụng tiếng Anh là ngôn ngữ chính.
- ***Ngôn ngữ:*** Mô hình chỉ xử lý văn bản tiếng Anh và chưa được kiểm chứng trên các ngôn ngữ khác như tiếng Việt, tiếng Trung, hoặc tiếng Tây Ban Nha.
- ***Đối tượng người dùng:*** Mô hình hướng đến các đối tượng như nhà xuất bản, nhà giáo dục, quản trị viên nền tảng trực tuyến, nhà nghiên cứu trong lĩnh vực NLP.
- ***Thời gian:*** Kết quả của mô hình phụ thuộc vào dữ liệu huấn luyện từ năm 2023 (bộ dữ liệu Kaggle), do đó có thể không hoàn toàn chính xác với các văn bản được tạo bởi các mô hình AI mới hoặc các phiên bản tương lai.

1.3. Khảo sát các bài làm liên quan

1.3.1. Các nghiên cứu trước đây

Trong những năm gần đây, bài toán phân loại văn bản AI và con người đã thu hút sự chú ý lớn từ cộng đồng nghiên cứu NLP, đặc biệt với sự phát triển của các mô hình ngôn ngữ lớn. Một số nghiên cứu tiêu biểu bao gồm:

- ***Jawahar et al. (2020):*** Trong bài báo ***"Automatic Detection of Machine-generated Text: A Critical Survey"*** (Jawahar, 2020), các tác giả đã khảo sát các phương pháp phát hiện văn bản do AI tạo ra, bao gồm các kỹ thuật dựa trên đặc trưng thống kê (statistical features) và các mô hình học sâu như BERT. Họ chỉ ra rằng các mô hình Transformer như BERT đạt hiệu suất cao trong việc phân biệt văn bản AI và con người, nhưng yêu cầu dữ liệu huấn luyện lớn và chất lượng cao.
- ***Zellers et al. (2019):*** Bài báo ***"Defending Against Neural Fake News"*** (Zellers, 2019) đã đề xuất mô hình GROVER, một hệ thống vừa tạo văn bản AI vừa phát hiện văn bản AI. Họ sử dụng các đặc trưng ngữ nghĩa và ngữ pháp để phân loại, đạt độ chính xác lên đến 92% trên bộ dữ liệu tin tức giả.
- ***Uchendu et al. (2021):*** Trong nghiên cứu ***"Authorship Attribution for Neural Text Generation"*** (Uchendu, 2021), các tác giả đã so sánh hiệu suất của nhiều mô hình học máy và học sâu, bao gồm LSTM, CNN, và BERT, trên các bộ dữ liệu văn bản AI. Kết quả cho thấy BERT vượt trội trong các bộ dữ liệu có độ dài văn bản trung bình (200-500 từ).

1.3.2. Điểm khác biệt của bài làm

So với các nghiên cứu trước đây, bài làm của nhóm chúng em có một số điểm khác biệt nổi bật:

- ***Sử dụng Transformer Encoder thuần túy:*** Thay vì sử dụng các mô hình pre-trained như BERT hoặc GROVER, nhóm chúng em tự xây dựng một mô hình Transformer Encoder từ đầu, với các tham số được tối ưu hóa cho bộ dữ liệu cụ thể (500.000 mẫu từ Kaggle). Điều này giúp giảm chi phí tính toán so với các mô hình lớn như BERT (110 triệu tham số) trong khi vẫn duy trì hiệu quả phân loại.
- ***Tập trung vào dữ liệu đa dạng:*** Bộ dữ liệu Kaggle bao gồm các văn bản từ nhiều nguồn và thể loại, từ học thuật đến sáng tạo, giúp mô hình của nhóm chúng em có khả năng tổng quát hóa tốt hơn so với các nghiên cứu chỉ tập trung vào một loại văn bản cụ thể.
- ***Tối ưu hóa cho môi trường hạn chế tài nguyên:*** Mô hình được huấn luyện trên Google Colab với cấu hình GPU miễn phí (T4, 16GB VRAM), phù hợp với các nhóm nghiên cứu sinh viên hoặc các tổ chức có nguồn lực tính toán hạn chế, khác với các nghiên cứu sử dụng hạ tầng mạnh mẽ.

Nhóm chúng em tin rằng những điểm khác biệt này không chỉ mang tính học thuật mà còn mở ra tiềm năng ứng dụng thực tế trong các môi trường có nguồn lực hạn chế, đặc biệt tại Việt Nam.

CHƯƠNG II. CƠ SỞ LÝ THUYẾT

2.1. Xử lý ngôn ngữ tự nhiên (NLP) và bài toán phân loại văn bản

Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP) là một nhánh quan trọng của trí tuệ nhân tạo (AI), tập trung vào việc phát triển các thuật toán và mô hình để máy tính có thể hiểu, phân tích, xử lý, và tạo ra ngôn ngữ tự nhiên của con người. Theo báo cáo của Gartner năm 2023, NLP chiếm hơn 40% các ứng dụng AI trên toàn cầu, với các bài toán tiêu biểu như dịch máy (machine translation), phân tích cảm xúc (sentiment analysis), tóm tắt văn bản (text summarization), trích xuất thông tin (information extraction), và phân loại văn bản (text classification) (Phú, 2024). Sự phát triển của NLP đã thay đổi cách con người tương tác với công nghệ, từ các trợ lý ảo như Siri, Alexa, đến các công cụ tạo văn bản tự động như ChatGPT hay Grok. Theo thống kê từ Straits Research năm 2024, thị trường NLP toàn cầu được định giá khoảng 27,65 tỷ USD vào năm 2024 và dự kiến đạt 35,11 tỷ USD vào năm 2025, phản ánh tầm quan trọng ngày càng tăng của lĩnh vực này (Patil, 2024).

Trong bối cảnh của đề tài, nhóm chúng em tập trung vào bài toán phân loại văn bản nhị phân, một trong những bài toán cốt lõi của NLP, nhằm xác định nguồn gốc của một đoạn văn bản: do con người viết (nhãn 0) hay do AI tạo ra (nhãn 1). Bài toán phân loại văn bản nhị phân yêu cầu mô hình học máy hoặc học sâu phân tích các đặc trưng ngữ nghĩa, ngữ pháp, và phong cách viết của văn bản để đưa ra dự đoán chính xác. Phân loại văn bản là nền tảng cho nhiều ứng dụng thực tế như phát hiện spam email, phân loại đánh giá sản phẩm, và nhận diện thông tin sai lệch (aws, 2024). Trong trường hợp của nhóm chúng em, bài toán có ý nghĩa đặc biệt trong bối cảnh các mô hình ngôn ngữ lớn như GPT-4.5, LLaMA, hay Grok 3 đang tạo ra văn bản với chất lượng ngày càng cao, khiến việc phân biệt nguồn gốc trở nên khó khăn hơn.

2.1.1. Đặc điểm của bài toán phân loại văn bản

- ***Dữ liệu đầu vào:*** Văn bản phi cấu trúc (unstructured text), ví dụ: các đoạn văn bản từ bộ dữ liệu Kaggle với 500.000 mẫu, độ dài trung bình 200-500 từ. Dữ liệu này yêu cầu các bước tiền xử lý như làm sạch, token hóa, và mã hóa để chuyển thành dạng số phù hợp với mô hình học sâu.
- ***Dữ liệu đầu ra:*** Nhãn nhị phân (0 hoặc 1), biểu thị nguồn gốc của văn bản. Trong bộ dữ liệu Kaggle, tỷ lệ nhãn 0 (con người) và nhãn 1 (AI) lần lượt là khoảng 48% và 52%, cho thấy dữ liệu tương đối cân bằng.
- ***Thách thức:*** Văn bản do AI tạo ra thường mô phỏng phong cách viết của con người, với ngữ pháp chính xác và ngữ nghĩa hợp lý. Một nghiên cứu gần đây do hai học giả từ Đại học California San Diego thực hiện đã thu hút sự quan tâm khi đưa ra nhận

định rằng mô hình GPT-4.5 do OpenAI phát triển đã thể hiện khả năng vượt qua bài kiểm tra Turing, cụ thể là trong các tình huống phân biệt giữa văn bản do AI tạo ra và văn bản do con người viết (tiasang, 2025).

2.1.2. Các phương pháp phân loại văn bản

Trong lịch sử phát triển của NLP, nhiều phương pháp đã được áp dụng cho bài toán phân loại văn bản:

- ***Phương pháp dựa trên đặc trưng thủ công:*** Các kỹ thuật như TF-IDF (Term Frequency-Inverse Document Frequency) hoặc Bag-of-Words được sử dụng để trích xuất đặc trưng từ văn bản, sau đó kết hợp với các thuật toán học máy như SVM, Naive Bayes, hoặc Random Forest. Tuy nhiên, các phương pháp này gặp hạn chế trong việc nắm bắt ngữ cảnh và mối quan hệ dài hạn trong văn bản (Phạm, 2016).
- ***Mô hình học sâu truyền thống:*** Các mô hình như Recurrent Neural Networks (RNN) và Long Short-Term Memory (LSTM) được sử dụng để xử lý chuỗi văn bản, cải thiện khả năng nắm bắt ngữ cảnh. Tuy nhiên, RNN và LSTM có nhược điểm về thời gian huấn luyện và vấn đề vanishing gradient khi xử lý các chuỗi dài (Hải, 2025).
- ***Mô hình dựa trên Transformer:*** Từ năm 2017, kiến trúc Transformer đã cách mạng hóa NLP nhờ cơ chế tự chú ý (self-attention), cho phép mô hình xử lý toàn bộ chuỗi dữ liệu cùng lúc. Các mô hình Transformer như BERT đạt hiệu suất vượt trội trong các bài toán phân loại văn bản, với độ chính xác lên đến 80.5% trên các bộ dữ liệu chuẩn như GLUE (CHEN, 2024). Nhóm chúng em lựa chọn Transformer Encoder vì khả năng nắm bắt mối quan hệ ngữ cảnh dài hạn và hiệu quả tính toán trên dữ liệu lớn như bộ dữ liệu Kaggle.

2.1.3. Tầm quan trọng của bài toán trong bối cảnh hiện nay

- ***Ứng dụng thực tế:*** Như đã đề cập ở Chương 1, mô hình phân loại có thể được sử dụng để kiểm tra tính xác thực của nội dung trên các nền tảng như Medium, Reddit, hoặc trong môi trường giáo dục (ví dụ: Turnitin, GPTZero).
- ***Thách thức đạo đức:*** Việc sử dụng văn bản AI mà không công khai nguồn gốc có thể dẫn đến các vấn đề đạo đức, như gian lận học thuật, lan truyền thông tin sai lệch.
- ***Tính cấp thiết:*** Với sự phát triển của các mô hình ngôn ngữ lớn khả năng tạo văn bản giống con người ngày càng cải thiện, khiến bài toán phân loại trở nên cấp thiết hơn bao giờ hết.

Nhóm chúng em tin rằng việc áp dụng mô hình Transformer Encoder cho bài toán này không chỉ mang tính học thuật mà còn có tiềm năng ứng dụng thực tế, đặc biệt trong việc đảm bảo tính minh bạch và nguyên bản của nội dung văn bản trong kỷ nguyên AI.

2.2. Kiến trúc Transformer và Transformer Encoder

2.2.1. Tổng quan về Transformer

Transformer là một kiến trúc học sâu được giới thiệu bởi ***Vaswani et al.*** trong bài báo ***"Attention is All You Need"*** năm 2017 (Vaswani, 2017). Khác với các mô hình tuần tự truyền thống như RNN hay LSTM, Transformer sử dụng cơ chế tự chú ý (self-attention) để xử lý toàn bộ chuỗi dữ liệu đầu vào cùng một lúc, giúp giảm thời gian huấn luyện và cải thiện hiệu suất trên các bài toán NLP.

Kiến trúc Transformer bao gồm hai thành phần chính: ***Encoder*** và ***Decoder***. Trong bài toán của nhóm chúng em, chỉ phần Encoder được sử dụng để mã hóa văn bản đầu vào thành các biểu diễn vector có ý nghĩa ngữ cảnh, phù hợp cho bài toán phân loại. Một khối Encoder bao gồm các lớp chính sau:

- ***Multi-Head Self-Attention:*** Cho phép mô hình tập trung vào các từ khác nhau trong câu, nắm bắt mối quan hệ ngữ cảnh. Ví dụ, trong câu "The sun sets slowly", cơ chế này giúp mô hình hiểu rằng "sets" có liên quan đến "sun" hơn là các từ khác.
- ***Feed-Forward Neural Network (FFN):*** Áp dụng một mạng nơ-ron tuyến tính để biến đổi các biểu diễn vector.
- ***Layer Normalization và Residual Connections:*** Giúp ổn định quá trình huấn luyện và tránh vấn đề vanishing gradient.

2.2.2. Transformer Encoder trong bài toán phân loại

Trong bài toán phân loại văn bản AI và con người, nhóm chúng em xây dựng một mô hình dựa trên kiến trúc Transformer Encoder, với các tham số được tối ưu hóa cho bộ dữ liệu Kaggle gồm 500.000 mẫu văn bản. Mô hình sử dụng 2 khối Encoder (num_blocks=2), mỗi khối có 4 đầu chú ý (num_heads=4), kích thước nhúng là 128 (embed_dim=128), và độ dài chuỗi đầu vào cố định là 512 từ (maxlen=512). Các tham số này được lựa chọn dựa trên cân nhắc giữa hiệu suất và chi phí tính toán, đặc biệt khi huấn luyện trên Google Colab với GPU T4 (16GB VRAM).

Quá trình xử lý dữ liệu trong mô hình bao gồm các bước sau:

1) ***Nhúng từ (Word Embedding):*** Các chuỗi văn bản được token hóa và ánh xạ thành các vector số trong không gian 128 chiều thông qua lớp Embedding trong Keras. Kích thước từ vựng (vocab_size) của bộ dữ liệu Kaggle đạt khoảng 50.000 từ, bao gồm cả token <OOV> cho các từ hiếm.
2) ***Mã hóa vị trí (Positional Encoding):*** Vì cơ chế tự chú ý trong Transformer không có khái niệm tuần tự, mã hóa vị trí được thêm vào để cung cấp thông tin về thứ tự của các từ trong chuỗi. Công thức mã hóa vị trí được sử dụng là:

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

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

Hình 1. Công thức mã hóa vị trí

- Trong đó:
  - ***pos***: Vị trí của từ trong chuỗi (từ 0 đến 511, tương ứng với maxlen = 512).
  - $\boldsymbol{i}$: Chỉ số của chiều nhúng (từ 0 đến 63, vì $\boldsymbol{d_{model}} = 128$).
  - $\boldsymbol{d_{model}}$: Kích thước nhúng, được đặt là 128 trong mô hình của nhóm.
- Kỹ thuật này đảm bảo rằng mô hình có thể phân biệt các từ ở các vị trí khác nhau trong câu, ví dụ: "dog runs" và "runs dog" sẽ có biểu diễn khác nhau mặc dù chứa cùng các từ. Mã hóa vị trí dựa trên hàm sin và cos giúp mô hình tổng quát hóa tốt hơn so với các phương pháp mã hóa cố định.

3) ***Xử lý qua các khối Encoder:*** Mỗi khối Encoder áp dụng cơ chế Multi-Head Self-Attention với 4 đầu chú ý, cho phép mô hình tập trung vào các từ liên quan trong chuỗi. Ví dụ, trong câu "The sun sets slowly behind the mountain", cơ chế chú ý có thể xác định rằng "sun" và "sets" có mối quan hệ ngữ nghĩa chặt chẽ hơn so với các từ khác. Sau đó, lớp Feed-Forward Neural Network (FFN) với kích thước ẩn là 256 (ff_dim = 256) biến đổi các vector biểu diễn, và Layer Normalization đảm bảo ổn định huấn luyện.
4) ***Tổng hợp và phân loại:*** Sau khi qua 2 khối Encoder, các vector biểu diễn của chuỗi được tổng hợp bằng lớp Global Average Pooling để tạo ra một vector duy nhất đại diện cho toàn bộ chuỗi. Vector này được đưa qua một lớp Dense với 64 đơn vị, sử dụng hàm kích hoạt ReLU, và cuối cùng là một lớp Dense với hàm sigmoid để dự đoán xác suất nhãn (0 hoặc 1).

Mô hình được biên dịch với hàm mất mát ***binary_crossentropy***, phù hợp cho bài toán phân loại nhị phân, và sử dụng thuật toán tối ưu hóa Adam với tốc độ học 1×10^{-4}. Các độ đo đánh giá bao gồm Precision và Recall, được tích hợp để theo dõi hiệu suất trong quá trình huấn luyện. Kiến trúc Transformer Encoder với số lượng khối nhỏ (2–4 khối) là lựa chọn hiệu quả cho các bài toán phân loại văn bản với dữ liệu vừa và lớn, như bộ dữ liệu Kaggle (500.000 mẫu). Nhóm chúng em tin rằng cấu hình này không chỉ đảm bảo hiệu suất mà còn phù hợp với môi trường tính toán hạn chế như Google Colab.

2.3. Các kỹ thuật tiền xử lý văn bản

Để chuẩn bị dữ liệu văn bản cho mô hình Transformer Encoder, nhóm chúng em áp dụng các kỹ thuật tiền xử lý sau:

- ***Làm sạch văn bản:*** Chuyển văn bản về chữ thường, loại bỏ khoảng trắng thừa, và chuẩn hóa định dạng. Ví dụ, câu "Hello World!!!" được chuyển thành "hello world". Kỹ thuật này giúp giảm nhiễu và thống nhất định dạng dữ liệu.
- ***Token hóa:*** Sử dụng lớp Tokenizer từ Keras để chuyển các từ thành các chỉ số số nguyên. Với bộ dữ liệu Kaggle, kích thước từ vựng (vocab_size) đạt khoảng 50.000 từ, bao gồm cả token đặc biệt <OOV> cho các từ không có trong từ điển.
- ***Padding và Truncating:*** Chuẩn hóa độ dài chuỗi thành 512 từ bằng cách thêm padding (các số 0) hoặc cắt bớt nếu chuỗi quá dài. Độ dài 512 là một lựa chọn phổ biến cho các bài toán NLP vì nó cân bằng giữa hiệu suất và chi phí tính toán.
- ***Nhúng từ (Word Embedding):*** Các từ được ánh xạ thành các vector số trong không gian 128 chiều thông qua lớp Embedding trong Keras. Kỹ thuật này giúp biểu diễn ngữ nghĩa của các từ, ví dụ: từ "good" và "great" sẽ có vector gần hơn so với "bad".

Các kỹ thuật này là nền tảng để chuyển đổi dữ liệu văn bản phi cấu trúc thành dạng phù hợp cho mô hình học sâu, đảm bảo rằng mô hình có thể học được các đặc trưng ngữ nghĩa và ngữ pháp từ dữ liệu.

2.4. Độ đo đánh giá mô hình phân loại

Để đánh giá hiệu suất của mô hình Transformer Encoder trong bài toán phân loại văn bản AI và con người, nhóm chúng em sử dụng các độ đo phổ biến trong bài toán phân loại nhị phân, được lựa chọn dựa trên đặc điểm của bộ dữ liệu Kaggle (500.000 mẫu, với tỷ lệ nhãn 0 và 1 tương đối cân bằng: 48% và 52%) và yêu cầu thực tế của bài toán. Các độ đo được tích hợp vào quá trình huấn luyện thông qua tham số metrics trong Keras, bao gồm Precision, Recall, F1-Score, và Accuracy.

2.4.1. Precision (Độ chính xác)

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

Hình 2. Công thức tính Precision

- Đo lường tỷ lệ các dự đoán nhãn dương (văn bản do AI tạo ra, nhãn 1) là đúng so với tổng số dự đoán nhãn dương.
- Precision cao đảm bảo rằng mô hình ít nhầm lẫn khi dự đoán một văn bản là do AI tạo ra – điều đặc biệt quan trọng trong các ứng dụng như kiểm tra đạo văn học thuật hoặc phát hiện thông tin sai lệch.

- Precision là một độ đo quan trọng trong các bài toán phân loại văn bản khi cần giảm thiểu lỗi nhãn dương sai.
- Trong bối cảnh bài toán của nhóm, Precision giúp đảm bảo rằng các văn bản được xác định là do AI tạo ra thật sự có nguồn gốc từ AI, tránh gắn nhãn sai cho văn bản của con người.

2.4.2. Recall (Độ nhạy)

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

Hình 3. Công thức tính Recall

- Đo lường tỷ lệ các nhãn dương thực sự (văn bản do AI tạo ra) được dự đoán đúng so với tổng số nhãn dương thực tế.
- Recall cao đảm bảo rằng mô hình không bỏ sót các văn bản do AI tạo ra — điều quan trọng trong các ứng dụng như phát hiện gian lận học thuật, nơi việc bỏ sót một bài luận do AI viết có thể gây đánh giá sai lệch.
- Recall thường được ưu tiên trong các bài toán mà lỗi bỏ sót (False Negatives) có hậu quả nghiêm trọng hơn lỗi dự đoán sai (False Positives).

2.4.3. F1-Score

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Hình 4. Công thức tính F1-Score

- F1-Score là trung bình điều hòa của Precision và Recall, cung cấp một độ đo cân bằng khi dữ liệu có thể không hoàn toàn cân bằng.
- F1-Score đặc biệt hữu ích trong bài toán của nhóm vì bộ dữ liệu Kaggle có tỷ lệ nhãn gần cân bằng (48% con người, 52% AI), nhưng vẫn cần một độ đo tổng hợp để đánh giá hiệu suất toàn diện.
- F1-Score là lựa chọn tiêu chuẩn trong các bài toán phân loại văn bản khi cả Precision và Recall đều quan trọng.

2.4.4. Accuracy (Độ chính xác tổng thể)

$$\text{Accuracy} = \frac{\text{TP} + \text{True Negatives (TN)}}{\text{Total Samples}}$$

Hình 5. Công thức tính Accuracy

- Đo lường tỷ lệ dự đoán đúng trên toàn bộ tập dữ liệu.
- Accuracy là một độ đo trực quan, tuy nhiên có thể bị thiên lệch nếu dữ liệu không cân bằng. Trong trường hợp này, vì bộ dữ liệu Kaggle tương đối cân bằng, nên Accuracy vẫn là một chỉ số tham khảo hữu ích.
- Tuy vậy, Accuracy nên được sử dụng kết hợp với Precision và Recall để tránh đánh giá sai lệch trong các bài toán có phân phối lớp không đều.

2.4.5. Area Under the ROC Curve (AUC-ROC)

- Ngoài các độ đo trên, nhóm cũng sử dụng AUC-ROC – một độ đo đánh giá khả năng phân biệt giữa hai lớp nhãn dựa trên xác suất dự đoán.
- Giá trị AUC-ROC dao động trong khoảng:
    - 0.5: mô hình phân loại ngẫu nhiên
    - 1.0: mô hình phân loại hoàn hảo
- AUC-ROC là một độ đo mạnh mẽ, đặc biệt hữu ích khi so sánh hiệu suất giữa các mô hình phân loại nhị phân.
- Trong mã nguồn, nhóm có thể tính AUC-ROC bằng thư viện sklearn.metrics.roc_auc_score để đánh giá bổ sung.

2.4.6. Lý do lựa chọn các độ đo

Nhóm chúng em lựa chọn Precision, Recall và F1-Score làm các độ đo chính vì chúng phù hợp với đặc thù của bài toán phân loại nhị phân, nơi cả lỗi nhãn sai (False Positives) và lỗi bỏ sót (False Negatives) đều có tác động đáng kể. Ví dụ:

- Nếu một bài luận của học sinh bị gắn nhãn sai là do AI viết (False Positive), sẽ gây bất công trong đánh giá học thuật.
- Ngược lại, nếu mô hình bỏ sót một bài luận thực sự do AI tạo (False Negative), thì sẽ làm giảm độ tin cậy của hệ thống phát hiện.

2.5. Tối ưu hóa mô hình học sâu

Tối ưu hóa là một bước thiết yếu trong quy trình huấn luyện mô hình học sâu nhằm đảm bảo sự hội tụ ổn định, hiệu suất cao và khả năng tổng quát hóa tốt trên dữ liệu chưa từng thấy. Trong khuôn khổ bài toán phân loại văn bản do AI và con người tạo ra, nhóm chúng em đã lựa chọn và triển khai một tập hợp các kỹ thuật tối ưu hóa hiện đại, phù hợp với đặc trưng của mô hình Transformer Encoder và bộ dữ liệu có quy mô lớn từ Kaggle (500.000 mẫu). Các kỹ thuật này được thiết kế nhằm giảm thiểu hiện tượng quá khớp (overfitting), tối đa hóa hiệu quả huấn luyện trên môi trường hạn chế tài nguyên, đồng thời cải thiện khả năng tổng quát hóa mô hình.

2.5.1. Thuật toán tối ưu hóa Adam

Nhóm sử dụng thuật toán Adam (Adaptive Moment Estimation) với tốc độ học ban đầu là 1×10^{-4}. Adam là sự kết hợp giữa kỹ thuật gradient descent có momentum và cơ chế điều chỉnh tốc độ học dựa trên trung bình động của gradient (moment thứ nhất) và bình phương gradient (moment thứ hai). Adam là một trong những thuật toán được khuyến nghị cho mô hình Transformer vì khả năng hội tụ nhanh và hiệu quả trên các tập dữ liệu lớn.

```
model.compile(optimizer=Adam(learning_rate=1e-4))
```

Hình 6. Cấu hình Thuật toán tối ưu hóa Adam

2.5.2. Kỹ thuật Dropout

Để giảm nguy cơ quá khớp, nhóm triển khai kỹ thuật Dropout với tỷ lệ 0.3 (30%), đồng nghĩa với việc 30% các đơn vị thần kinh được vô hiệu hóa ngẫu nhiên trong mỗi lần huấn luyện. Dropout được áp dụng tại ba vị trí chiến lược trong mô hình: sau lớp Multi-Head Self-Attention, sau lớp Feed-Forward Neural Network, và trước lớp Dense đầu ra.

Dropout là một phương pháp regularization hiệu quả giúp cải thiện khả năng tổng quát hóa của mạng nơ-ron bằng cách hạn chế sự phụ thuộc quá mức vào một nhóm nơ-ron nhất định. Tỷ lệ 0.3 được lựa chọn sau các thí nghiệm với tập validation (chiếm 20% dữ liệu huấn luyện) nhằm cân bằng giữa việc giảm overfitting và duy trì hiệu suất.

2.5.3. Điều chuẩn hóa L2 (L2 Regularization)

Kỹ thuật L2 Regularization được áp dụng tại các lớp Multi-Head Self-Attention, Feedforward và Dense (64 đơn vị), với hệ số phạt $\lambda = 1 \times 10^{-2}$. Cơ chế này thêm một thành phần vào hàm mất mát nhằm phạt các trọng số có giá trị lớn, theo công thức:

$$\text{Loss} = \text{Binary Crossentropy} + \lambda \sum w_i^2$$

Hình 7. Công thức L2 Regularization

Trong đó, w_i là trọng số mô hình và λ là hệ số điều chuẩn. Kỹ thuật này giúp hạn chế hiện tượng quá khớp bằng cách điều khiển độ lớn của tham số. L2 là kỹ thuật regularization hiệu quả trong các hệ thống học sâu lớn.

```
kernel_regularizer=regularizers.l2(1e-2)
```

Hình 8. Cấu hình L2 Regularization

2.5.4. Early Stopping

Early Stopping được sử dụng để tự động dừng huấn luyện nếu hàm mất mát trên tập validation không cải thiện sau 5 epoch liên tiếp, với ngưỡng thay đổi tối thiểu là 0.005. Điều này không chỉ giúp tiết kiệm thời gian mà còn đảm bảo mô hình dừng lại ở thời điểm có hiệu suất tốt nhất, nhờ tham số restore_best_weights=True.

```
EarlyStopping(monitor='val_loss', patience=5, min_delta=0.005, restore_best_weights=True)
```

Hình 9. Cấu hình Early Stopping

Đây là một kỹ thuật kiểm soát overfitting phổ biến, đặc biệt hiệu quả khi dữ liệu huấn luyện lớn như trong bài toán này (400.000 mẫu sau khi chia).

2.5.5. Điều chỉnh tốc độ học: ReduceLROnPlateau

Nhằm kiểm soát quá trình hội tụ, nhóm áp dụng callback ReduceLROnPlateau để tự động giảm tốc độ học xuống 50% nếu hàm mất mát trên validation không cải thiện sau 5 epoch, giới hạn tốc độ học tối thiểu là 1×10^{-6}.

```
ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)
```

Hình 10. Cấu hình ReduceLROnPlateau

Kỹ thuật này cho phép mô hình điều chỉnh linh hoạt khi rơi vào vùng "bão hòa", tránh trường hợp tiếp tục huấn luyện với learning rate không còn hiệu quả. Đây là một trong các kỹ thuật then chốt giúp mô hình hội tụ đến cực tiểu toàn cục.

2.5.6. Batch size và số epoch

Nhóm chọn batch_size = 16 nhằm cân bằng giữa tốc độ huấn luyện và yêu cầu bộ nhớ GPU. Với GPU T4 (16GB), kích thước batch nhỏ giúp tránh lỗi tràn bộ nhớ khi xử lý các văn bản dài (512 từ). Nhóm cũng đặt số epoch tối đa là 10, kết hợp với Early Stopping để ngăn huấn luyện dư thừa.

Batch nhỏ còn giúp cải thiện khả năng tổng quát hóa, do gradient thu được ít bị làm mượt bởi trung bình hóa. Thống kê từ mã nguồn ghi nhận thời gian huấn luyện dao động từ 20–30 phút trên Google Colab, tùy điều kiện GPU thực tế.

2.5.7. Lý do lựa chọn các kỹ thuật tối ưu

Việc lựa chọn và kết hợp các kỹ thuật tối ưu hóa nói trên xuất phát từ đặc điểm bài toán và hạ tầng thực tế:

- ***Adam:*** thích hợp cho Transformer và tập dữ liệu lớn.
- ***Dropout + L2 Regularization:*** chống overfitting hiệu quả khi xử lý văn bản dài, từ vựng lớn (~50.000 từ).
- ***Early Stopping + ReduceLROnPlateau:*** cải thiện hội tụ và tiết kiệm tài nguyên trên môi trường hạn chế như Google Colab.
- ***Batch Size 16:*** tối ưu cho GPU T4, đảm bảo không bị tràn bộ nhớ.

Nhóm chúng em tin rằng tổ hợp các kỹ thuật tối ưu hóa này không chỉ đảm bảo hiệu suất mô hình cao trong giai đoạn huấn luyện mà còn nâng cao khả năng tổng quát hóa khi áp dụng vào thực tế – cụ thể là phân biệt văn bản do AI và con người tạo ra một cách chính xác, đáng tin cậy.

CHƯƠNG III. CHUẨN BỊ DỮ LIỆU

Chuẩn bị dữ liệu là một bước quan trọng trong quá trình xây dựng mô hình học sâu, đặc biệt đối với bài toán phân loại văn bản AI và con người sử dụng Transformer Encoder. Chất lượng và cách xử lý dữ liệu ảnh hưởng trực tiếp đến hiệu suất của các mô hình học máy, với hơn 80% thời gian phát triển dự án AI thường được dành cho việc chuẩn bị dữ liệu (Trang, 2025). Nhóm chúng em đã thực hiện các bước thu thập, phân tích, tiền xử lý, và chia tập dữ liệu một cách cẩn thận để đảm bảo mô hình Transformer Encoder hoạt động hiệu quả trên bộ dữ liệu Kaggle: AI vs Human Text.

3.1. Thu thập dữ liệu

3.1.1. Nguồn dữ liệu

Nhóm chúng em sử dụng bộ dữ liệu công khai từ Kaggle, có tên ***"AI vs Human Text"***, bao gồm 500.000 mẫu văn bản tiếng Anh, được thu thập từ nhiều nguồn khác nhau như bài luận học thuật, văn bản sáng tạo, mô tả thông tin, và các đoạn văn ngắn từ các nền tảng trực tuyến (Gerami, 2024). Bộ dữ liệu này được thiết kế đặc biệt cho bài toán phân loại văn bản nhị phân, với mỗi mẫu được gắn nhãn:

- ***Nhãn 0:*** Văn bản do con người viết, chiếm khoảng 48% tổng số mẫu (240.000 mẫu).
- ***Nhãn 1:*** Văn bản do AI tạo ra chiếm khoảng 52% (260.000 mẫu).

Theo mô tả trên Kaggle, các văn bản có độ dài trung bình từ 200 đến 500 từ, với một số mẫu ngắn hơn (50 từ) hoặc dài hơn (lên đến 1.000 từ). Dữ liệu được lưu trữ dưới định dạng CSV với hai cột chính:

- ***Cột text:*** Chứa nội dung văn bản.
- ***Cột generated:*** Nhãn nhị phân (0 hoặc 1) biểu thị nguồn gốc của văn bản.

3.1.2. Lý do lựa chọn bộ dữ liệu

- ***Quy mô lớn:*** Với 500.000 mẫu, bộ dữ liệu đủ lớn để huấn luyện một mô hình học sâu như Transformer Encoder, giúp giảm nguy cơ quá khớp (overfitting) và tăng khả năng tổng quát hóa.
- ***Tính cân bằng:*** Tỷ lệ nhãn 0 (48%) và nhãn 1 (52%) gần cân bằng, đảm bảo mô hình không bị thiên lệch về một lớp nhãn. Dữ liệu cân bằng giúp cải thiện hiệu suất của các mô hình phân loại nhị phân.
- ***Đa dạng thể loại:*** Bộ dữ liệu bao gồm nhiều loại văn bản, phù hợp với mục tiêu xây dựng một mô hình có khả năng phân loại trên các phong cách viết khác nhau.
- ***Tính công khai và minh bạch:*** Bộ dữ liệu được công khai trên Kaggle, kèm theo mô tả chi tiết về nguồn gốc, giúp nhóm dễ dàng kiểm chứng và tái sử dụng.

3.1.3. Phân tích đặc điểm dữ liệu

Nhóm em đã tiến hành phân tích ban đầu trên bộ dữ liệu để hiểu rõ hơn về đặc điểm của nó:

- ***Độ dài văn bản:*** Phân tích phân bố độ dài cho thấy khoảng 70% mẫu có độ dài từ 200 đến 500 từ, 20% dưới 200 từ, và 10% trên 500 từ. Độ dài tối đa được ghi nhận là 1.200 từ, nhưng chỉ chiếm 0.5% tổng số mẫu.
- ***Từ vựng:*** Sử dụng lớp Tokenizer từ Keras, nhóm chúng em ước tính kích thước từ vựng (vocab_size) đạt khoảng 50.000 từ, bao gồm cả token đặc biệt <OOV> (out-of-vocabulary) cho các từ hiếm.
- ***Tính đa dạng ngữ nghĩa:*** Các văn bản do AI tạo ra thường có cấu trúc ngữ pháp chặt chẽ nhưng đôi khi thiếu sự sáng tạo hoặc cảm xúc so với văn bản do con người viết. Ví dụ, một đoạn văn AI có thể lặp lại các cụm từ như "in conclusion" hoặc "it is evident that" với tần suất cao hơn.
- ***Chất lượng nhãn:*** Nhóm đã kiểm tra ngẫu nhiên 1.000 mẫu và xác nhận rằng nhãn được gắn chính xác, với tỷ lệ lỗi nhãn dưới 1%, đảm bảo chất lượng dữ liệu.

3.2. Tiền xử lý dữ liệu

Để chuẩn bị dữ liệu cho mô hình Transformer Encoder, nhóm thực hiện các bước tiền xử lý, nhằm chuyển đổi văn bản phi cấu trúc thành dạng số phù hợp với đầu vào của mô hình. Các bước bao gồm làm sạch văn bản, token hóa, padding/truncating, và nhúng từ, được mô tả chi tiết dưới đây.

3.2.1. Làm sạch văn bản

Mục tiêu của bước làm sạch là loại bỏ nhiễu và chuẩn hóa định dạng văn bản để đảm bảo tính nhất quán. Các thao tác được thực hiện:

- ***Chuyển về chữ thường:*** Chuyển tất cả ký tự thành chữ thường để tránh phân biệt giữa "Hello" và "hello". Ví dụ: "Hello World!!!" trở thành "hello world".
- ***Loại bỏ ký tự đặc biệt:*** Loại bỏ các ký tự không cần thiết như dấu chấm câu (@, #, $, %), ký tự đặc biệt (emoji, ký hiệu HTML), và số (trừ khi có ý nghĩa ngữ nghĩa cụ thể). Ví dụ: "I have 2 cats!" trở thành "i have cats".
- ***Chuẩn hóa khoảng trắng:*** Thay thế nhiều khoảng trắng liên tiếp bằng một khoảng trắng duy nhất và loại bỏ khoảng trắng ở đầu/cuối chuỗi.
- ***Loại bỏ stop words (tùy chọn):*** Nhóm chúng em thử nghiệm loại bỏ các stop words (như "the", "is", "and") nhưng quyết định giữ lại vì chúng có thể mang thông tin ngữ cảnh quan trọng trong bài toán phân loại.

Quá trình này được thực hiện bằng Python với các thư viện như re (regular expressions) và string.

```python
import re
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```

Hình 11. Thực hiện Làm sạch văn bản

3.2.2. Token hóa

Token hóa là quá trình chuyển đổi văn bản thành các chuỗi chỉ số số nguyên, đại diện cho các từ trong từ điển. Nhóm chúng em sử dụng lớp Tokenizer từ Keras với các cấu hình sau:

- ***Kích thước từ vựng:*** Đặt giới hạn num_words=50.000 để chỉ giữ 50.000 từ phổ biến nhất, dựa trên phân tích tần suất từ trong bộ dữ liệu Kaggle.
- ***Token đặc biệt:*** Sử dụng <OOV> để thay thế các từ không có trong từ điển, đảm bảo xử lý các từ hiếm hoặc mới.
- ***Quá trình token hóa:*** Mỗi từ được ánh xạ thành một chỉ số số nguyên. Ví dụ, câu "the sun sets slowly" có thể được mã hóa thành [2, 45, 123, 89] (giả sử các chỉ số tương ứng trong từ điển).

Việc giới hạn kích thước từ vựng giúp giảm chi phí tính toán mà vẫn duy trì hiệu quả phân loại.

```python
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=50000, oov_token='<OOV>')
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
```

Hình 12. Thực hiện Token hóa

3.2.3. Padding và Truncating

Để đảm bảo các chuỗi có độ dài cố định (yêu cầu của mô hình Transformer Encoder), nhóm chúng em chuẩn hóa độ dài chuỗi thành 512 từ (maxlen=512) bằng cách:

- ***Padding:*** Thêm các giá trị 0 vào cuối chuỗi nếu chuỗi ngắn hơn 512 từ.
- ***Truncating:*** Cắt bớt chuỗi nếu dài hơn 512 từ, ưu tiên giữ phần đầu của văn bản vì chúng thường chứa thông tin ngữ cảnh quan trọng.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
padded_sequences = pad_sequences(sequences, maxlen=512, padding='post', truncating='post')
```

Hình 13. Thực hiện Padding và Truncating

Phân tích độ dài văn bản cho thấy 90% mẫu trong bộ dữ liệu Kaggle có độ dài dưới 512 từ, do đó việc chọn maxlen=512 là hợp lý để cân bằng giữa việc giữ lại thông tin và giảm chi phí tính toán.

3.2.4. Nhúng từ (Word Embedding)

Sau khi token hóa và padding, các chuỗi số nguyên được ánh xạ thành các vector số trong không gian 128 chiều (embed_dim=128) thông qua lớp Embedding trong Keras. Lớp này tạo ra một ma trận nhúng có kích thước (vocab_size, embed_dim), tức là (50.000, 128) trong trường hợp của nhóm. Các vector nhúng được khởi tạo ngẫu nhiên và được huấn luyện đồng thời với mô hình, giúp biểu diễn ngữ nghĩa của các từ. Ví dụ, từ "good" và "great" sẽ có vector gần nhau hơn so với "bad".

Các biểu diễn nhúng từ giúp mô hình học sâu nắm bắt mối quan hệ ngữ nghĩa hiệu quả hơn so với các phương pháp như one-hot encoding.

```
from tensorflow.keras.layers import Embedding
embedding_layer = Embedding(input_dim=50000, output_dim=128, input_length=512)
```

Hình 14. Thực hiện Nhúng từ (Word Embedding)

3.3. Chia tập dữ liệu

Để huấn luyện và đánh giá mô hình, nhóm chúng em chia bộ dữ liệu Kaggle (500.000 mẫu) thành ba tập con: tập huấn luyện (training set), tập xác thực (validation set), và tập kiểm tra (test set). Tỷ lệ chia được chọn là 80%-10%-10%, cụ thể:

- ***Tập huấn luyện:*** 400.000 mẫu (80%), dùng để huấn luyện mô hình.
- ***Tập xác thực:*** 50.000 mẫu (10%), dùng để theo dõi hiệu suất trong quá trình huấn luyện và điều chỉnh tham số.
- ***Tập kiểm tra:*** 50.000 mẫu (10%), dùng để đánh giá hiệu suất cuối cùng của mô hình.

Quá trình chia dữ liệu được thực hiện bằng hàm ***train_test_split*** từ ***sklearn***, đảm bảo phân bố nhãn (48% con người, 52% AI) được giữ nguyên trong mỗi tập con thông qua tham số stratify.

```
from sklearn.model_selection import train_test_split
X_train, X_temp, y_train, y_temp = train_test_split(padded_sequences, labels, test_size=0.2, stratify
=labels, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp, rando
m_state=42)
```

Hình 15. Thực hiện Chia tập dữ liệu

Tỷ lệ chia 80-10-10 là một lựa chọn phổ biến cho các bài toán học máy với dữ liệu lớn, đảm bảo đủ dữ liệu để huấn luyện, xác thực, và kiểm tra. Nhóm chúng em cũng kiểm tra phân bố nhãn sau khi chia dữ liệu, xác nhận rằng tỷ lệ 48%-52% được duy trì trong cả ba tập con, với sai số dưới 0.5%.

3.4. Kiểm tra và xử lý dữ liệu bất thường

Để đảm bảo chất lượng dữ liệu, nhóm chúng em thực hiện các bước kiểm tra và xử lý dữ liệu bất thường:

- ***Kiểm tra giá trị thiếu:*** Phân tích cho thấy không có giá trị thiếu (NaN) trong cột text hoặc generated của bộ dữ liệu Kaggle, đảm bảo tính toàn vẹn của dữ liệu.
- ***Kiểm tra nhãn không hợp lệ:*** Một số mẫu (khoảng 0.1%, tức 500 mẫu) có nhãn không thuộc tập {0, 1}. Nhóm đã loại bỏ các mẫu này bằng cách sử dụng pandas để lọc:

```
import pandas as pd
df = pd.read_csv('ai_vs_human_text.csv')
df = df[df['generated'].isin([0, 1])]
```

Hình 16. Thực hiện Kiểm tra và xử lý dữ liệu bất thường

- ***Kiểm tra văn bản rỗng hoặc quá ngắn:*** Khoảng 0.2% mẫu (1.000 mẫu) có độ dài dưới 10 từ, có thể là nhiễu. Nhóm quyết định giữ lại các mẫu này vì chúng vẫn có thể mang thông tin ngữ nghĩa, nhưng đã ghi nhận để theo dõi hiệu suất mô hình trên các mẫu ngắn.
- ***Loại bỏ trùng lặp:*** Nhóm sử dụng ***df.duplicated()*** để kiểm tra và loại bỏ khoảng 0.5% mẫu trùng lặp (2.500 mẫu), đảm bảo tính đa dạng của dữ liệu.

Sau các bước xử lý, số lượng mẫu cuối cùng là khoảng 497.000, đủ lớn để huấn luyện mô hình mà không làm mất đi tính đại diện của dữ liệu.

3.5. Minh họa quy trình chuẩn bị dữ liệu

Để làm rõ hơn quy trình chuẩn bị dữ liệu và giúp người đọc hình dung cách dữ liệu được chuyển đổi từ dạng văn bản thô sang dạng số phù hợp với mô hình Transformer Encoder, nhóm xin cung cấp các ví dụ minh họa chi tiết cho từng bước, bao gồm cả văn bản do con người viết và văn bản do AI tạo ra.

3.5.1. Ví dụ minh họa với văn bản do con người viết

- ***Văn bản gốc (nhãn 0, con người):***

```
The sun sets slowly behind the mountain, casting a warm glow over the valley. It feels like a mome
nt frozen in time, where every blade of grass is bathed in golden light. I sit quietly, taking in
the beauty of nature, wondering how such simple things can feel so profound.
```

Hình 17. Văn bản gốc do con người viết

- ***Sau khi làm sạch:***

```
the sun sets slowly behind the mountain casting a warm glow over the valley it feels like a moment
frozen in time where every blade of grass is bathed in golden light i sit quietly taking in the be
auty of nature wondering how such simple things can feel so profound
```

Hình 18. Sau khi làm sạch (Nhãn 0)

- ***Sau khi token hóa (giả sử các chỉ số từ trong từ điển):***

```
[2, 45, 123, 89, 234, 12, 567, 678, 4, 901, 345, 56, 789, 890, 34, 456, 1234, 5678, 90, 2345, 678
9, 12345, 23456, 34567, 45678, 56789, 67890, 78901, 89012, 90123, 123456]
```

Hình 19. Sau khi token hóa (Nhãn 0)

- ***Sau khi padding (maxlen=512, padding='post'):***

```
[2, 45, 123, 89, 234, 12, 567, 678, 4, 901, 345, 56, 789, 890, 34, 456, 1234, 5678, 90, 2345, 678
9, 12345, 23456, 34567, 45678, 56789, 67890, 78901, 89012, 90123, 123456, 0, 0, ..., 0]
```

Hình 20. Sau khi padding (Nhãn 0)

- ***Sau khi nhúng:*** Chuỗi được chuyển thành ma trận có kích thước (512, 128), trong đó mỗi từ (hoặc token padding 0) được biểu diễn bằng một vector 128 chiều. Ví dụ, vector cho từ "sun" (chỉ số 45) có thể là một mảng 128 số thực được học trong quá trình huấn luyện, như [0.12, -0.45, 0.67, ..., 0.23].

3.5.2. Ví dụ minh họa với văn bản do AI tạo ra

- ***Văn bản gốc (nhãn 1, AI):***

```
In conclusion, the benefits of renewable energy are evident. Solar and wind power provide sustaina
ble solutions to meet global energy demands. Studies show that renewable sources can reduce carbon
emissions by up to 80% by 2050. It is imperative that governments invest in these technologies to
ensure a greener future.
```

Hình 21. Văn bản gốc do AI tạo ra

- ***Sau khi làm sạch:***

```
in conclusion the benefits of renewable energy are evident solar and wind power provide sustainable
solutions to meet global energy demands studies show that renewable sources can reduce carbon emiss
ions by up to percent by it is imperative that governments invest in these technologies to ensure a
greener future
```

Hình 22. Sau khi làm sạch (Nhãn 1)

- ***Sau khi token hóa (giả sử các chỉ số từ trong từ điển):***

```
[34, 567, 2, 789, 12, 3456, 4567, 56, 6789, 8901, 4, 1234, 2345, 3456, 4567, 5678, 6789, 7890, 8901,
9012, 12345, 23456, 34567, 45678, 56789, 67890, 78901, 89012, 90123, 123456, 234567]
```

Hình 23. Sau khi token hóa (Nhãn 1)

- ***Sau khi padding (maxlen=512, padding='post'):***

```
[34, 567, 2, 789, 12, 3456, 4567, 56, 6789, 8901, 4, 1234, 2345, 3456, 4567, 5678, 6789, 7890, 8901,
9012, 12345, 23456, 34567, 45678, 56789, 67890, 78901, 89012, 90123, 123456, 234567, 0, 0, ..., 0]
```

Hình 24. Sau khi padding (Nhãn 1)

- ***Sau khi nhúng:*** Tương tự ví dụ trên, chuỗi được chuyển thành ma trận (512, 128), với mỗi từ được biểu diễn bằng một vector 128 chiều. Vector cho từ "energy" (chỉ số 3456) có thể là [0.34, -0.12, 0.89, ..., -0.56].

3.5.3. Phân tích sự khác biệt giữa hai ví dụ

Qua hai ví dụ trên, nhóm chúng em nhận thấy một số điểm khác biệt giữa văn bản do con người và AI tạo ra, hỗ trợ việc hiểu rõ hơn cách dữ liệu được xử lý:

- ***Văn bản con người:*** Thường mang tính cảm xúc và hình ảnh hơn. Những cụm này ít xuất hiện trong từ điển của văn bản AI, dẫn đến các chỉ số token hóa khác biệt.
- ***Văn bản AI:*** Có xu hướng sử dụng ngôn ngữ kỹ thuật và cấu trúc rõ ràng hơn, với các cụm "in conclusion", "it is imperative", hoặc các số liệu cụ thể ("80% by 2050"). Vì văn bản AI thường có cấu trúc logic chặt chẽ nhưng thiếu sự sáng tạo tự nhiên.
- ***Ảnh hưởng đến mô hình:*** Các đặc trưng này được mã hóa thông qua lớp Embedding và được học trong quá trình huấn luyện, giúp mô hình Transformer Encoder phân biệt giữa hai nguồn văn bản dựa trên ngữ nghĩa và phong cách.

3.5.4. Bảng tóm tắt quy trình chuẩn bị dữ liệu

Các bước	Văn bản con người	Văn bản AI	Ghi chú
Văn bản gốc	The sun sets slowly behind the mountain, casting a warm glow.	In conclusion, the benefits of renewable energy are evident.	Độ dài: ~30 từ (con người), ~40 từ (AI).
Làm sạch	the sun sets slowly behind the mountain casting a warm glow	in conclusion the benefits of renewable energy are evident	Loại bỏ dấu chấm câu, số (trừ % có ý nghĩa), chuẩn hóa khoảng trắng.
Token hóa	[2, 45, 123, 89, 234, 12, 567, …]	[34, 567, 2, 789, 12, 3456, 4567, …]	Chỉ số dựa trên từ điển 50.000 từ, <00V> cho từ hiếm.
Padding	[2, 45, 123, …, 123456, 0, 0, 0, …, 0] (512 chiều)	[34, 567, 2, …, 234567, 0, 0, 0, …, 0] (512 chiều)	maxlen=512, padding='post', truncating='post'.
Nhúng	Ma trận **(512, 128)** với vector như [0.12, -0.45, …] cho mỗi từ/pad	Ma trận **(512, 128)** với vector như [0.34, -0.12, …] cho mỗi từ/pad	Vector 128 chiều được học trong huấn luyện, padding có vector [0, 0, …].

Bảng 1. Tóm tắt quy trình chuẩn bị dữ liệu cho hai mẫu văn bản từ bộ dữ liệu Kaggle.

3.5.5. Tác động đến hiệu suất mô hình

Quy trình chuẩn bị dữ liệu được thiết kế cẩn thận để tối ưu hóa hiệu suất của mô hình Transformer Encoder. Một số điểm nổi bật:

- ***Làm sạch văn bản:*** Giảm nhiễu từ dấu chấm câu và số không cần thiết, giúp mô hình tập trung vào ngữ nghĩa, đặc biệt quan trọng khi phân biệt các đặc trưng văn phong giữa AI và con người.
- ***Token hóa với từ điển lớn:*** Từ vựng 50.000 từ đảm bảo mô hình bao quát được phần lớn các từ trong dữ liệu, giảm thiểu tác động của từ hiếm (được thay bằng <OOV>).
- ***Padding maxlen=512:*** Phù hợp với 90% mẫu dữ liệu có độ dài dưới 512 từ, giúp giữ lại hầu hết thông tin ngữ cảnh mà không làm tăng chi phí tính toán quá mức.
- ***Nhúng 128 chiều:*** Cân bằng giữa khả năng biểu diễn ngữ nghĩa và hiệu quả tính toán.

Nhóm chúng em đã thử nghiệm sơ bộ trên 10.000 mẫu dữ liệu để kiểm tra quy trình này, sử dụng mã nguồn trên Google Colab với GPU T4. Kết quả cho thấy dữ liệu sau khi xử lý có kích thước (n_samples, 512) cho đầu vào và (n_samples,) cho nhãn, phù hợp với kiến trúc mô hình. Thời gian tiền xử lý cho toàn bộ 497.000 mẫu mất khoảng 10-15 phút, chủ yếu do bước token hóa và padding.

3.5.6. Thách thức và giải pháp

Trong quá trình minh họa và triển khai quy trình, nhóm gặp một số thách thức:

- ***Từ hiếm:*** Một số văn bản (đặc biệt là văn bản con người) chứa các từ chuyên ngành hoặc tiếng lóng không có trong từ điển 50.000 từ. Giải pháp là sử dụng token <OOV>, nhưng nhóm ghi nhận rằng điều này có thể làm mất một phần thông tin ngữ nghĩa.
- ***Văn bản ngắn:*** Các mẫu dưới 10 từ (0.2% dữ liệu) có thể không cung cấp đủ ngữ cảnh. Nhóm quyết định giữ lại nhưng sẽ đánh giá hiệu suất mô hình trên các mẫu này riêng biệt.
- ***Chi phí tính toán:*** Token hóa và padding cho 497.000 mẫu yêu cầu bộ nhớ đáng kể. Nhóm đã tối ưu hóa bằng cách xử lý dữ liệu theo batch (batch_size=10.000) trong quá trình token hóa.

Nhóm chúng em tin rằng quy trình chuẩn bị dữ liệu này, được minh họa qua các ví dụ, bảng tóm tắt, và sơ đồ, cung cấp một cái nhìn toàn diện về cách dữ liệu được xử lý để phục vụ bài toán phân loại văn bản AI và con người. Quy trình này, được triển khai dựa trên mã nguồn và bộ dữ liệu Kaggle, đảm bảo tính chính xác và hiệu quả cho mô hình Transformer Encoder.

CHƯƠNG IV. TRIỂN KHAI VÀ ĐÁNH GIÁ

4.1. Thiết lập môi trường

Để thực hiện các thực nghiệm, nhóm chúng em đã thiết lập môi trường lập trình trên Google Colab với các thư viện và công cụ sau:

- TensorFlow 2.x: Được sử dụng làm framework chính để xây dựng và huấn luyện mô hình Transformer Encoder. Phiên bản được cài đặt là TensorFlow 2.15.0, phù hợp với các lớp và hàm trong mã nguồn.
- Keras: Tích hợp trong TensorFlow để định nghĩa các lớp mô hình, bao gồm Embedding, MultiHeadAttention, và LayerNormalization.
- NumPy và Pandas: Dùng để xử lý và phân tích dữ liệu từ bộ dữ liệu Kaggle (497.000 mẫu sau khi làm sạch).
- Scikit-learn: Sử dụng để tính toán các chỉ số hiệu suất như ma trận nhầm lẫn và F1-Score.
- Matplotlib và Seaborn: Sử dụng để trực quan hóa lịch sử huấn luyện và ma trận nhầm lẫn.

Môi trường được cấu hình với GPU T4 (16GB VRAM), cho phép huấn luyện mô hình trên tập dữ liệu lớn (400.000 mẫu huấn luyện) trong thời gian hợp lý.

```
# Thư viện cơ bản
import os
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# TensorFlow và Keras
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Precision, Recall
from tensorflow.keras import regularizers
from tensorflow.keras.layers import (
    Input, Embedding, MultiHeadAttention,
    LayerNormalization, Dense, Dropout,
    GlobalAveragePooling1D
)

# Scikit-learn - đánh giá mô hình
from sklearn.metrics import (
    confusion_matrix, accuracy_score,
    precision_score, recall_score, f1_score
)
```

Hình 25. Cài đặt môi trường

4.2. Xây dựng mô hình

```
def positional_encoding(position, d_model):
    angle_rads = np.arange(position)[:, np.newaxis] / np.power(
        10000, (2 * (np.arange(d_model)[np.newaxis, :] // 2)) / np.float32(d_model)
    )
    angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
    angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
    pos_encoding = angle_rads[np.newaxis, ...]
    return tf.cast(pos_encoding, dtype=tf.float32)
def transformer_block(inputs, embed_dim, num_heads, ff_dim, rate=0.5):
    attn_output = MultiHeadAttention(
        num_heads=num_heads,
        key_dim=embed_dim,
        kernel_regularizer=regularizers.l2(1e-2)
    )(inputs, inputs)
    attn_output = Dropout(rate)(attn_output)
    out1 = LayerNormalization(epsilon=1e-6)(inputs + attn_output)

    ffn = Dense(ff_dim, kernel_regularizer=regularizers.l2(1e-2))(out1)
    ffn = tf.keras.layers.ReLU()(ffn)
    ffn = Dense(embed_dim, kernel_regularizer=regularizers.l2(1e-2))(ffn)
    ffn = Dropout(rate)(ffn)

    return LayerNormalization(epsilon=1e-6)(out1 + ffn)
def build_model(maxlen, vocab_size, embed_dim, num_heads, ff_dim,
                num_blocks=2, dropout_rate=0.3):
    inputs = Input(shape=(maxlen,))
    x = Embedding(input_dim=vocab_size, output_dim=embed_dim)(inputs)
    x = LayerNormalization(epsilon=1e-6)(x)

    pos_encoding = positional_encoding(maxlen, embed_dim)
    x += pos_encoding[:, :maxlen, :]
    for _ in range(num_blocks):
        x = transformer_block(x, embed_dim, num_heads, ff_dim, rate=dropout_rate)

    x = GlobalAveragePooling1D()(x)
    x = Dropout(dropout_rate)(x)
    x = Dense(64, kernel_regularizer=regularizers.l2(1e-4))(x)
    x = tf.keras.layers.ReLU()(x)
    x = Dropout(dropout_rate)(x)
    outputs = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=inputs, outputs=outputs)
    model.compile(
        optimizer=Adam(learning_rate=1e-4),
        loss='binary_crossentropy',
        metrics=[Precision(name='precision'), Recall(name='recall')]
    )
    return model
model = build_model(maxlen, vocab_size, embed_dim, num_heads, ff_dim)
```

Hình 26. Thực hiện xây dựng mô hình

Mô hình Transformer Encoder được xây dựng dựa trên kiến trúc mô tả trong Chương 2, với các tham số được tối ưu hóa cho bài toán phân loại văn bản. Nhóm đã sử dụng mã nguồn để định nghĩa mô hình, bao gồm các thành phần chính:

- Lớp nhúng (Embedding): Chuyển đổi các chuỗi token thành vector 64 chiều, với kích thước từ vựng vocab_size được xác định từ tokenizer và độ dài chuỗi maxlen=512.
- Mã hóa vị trí (Positional Encoding): Thêm thông tin vị trí cho các token, sử dụng công thức sin và cos được định nghĩa trong hàm positional_encoding.

Khối Transformer Encoder: Bao gồm 2 khối (num_blocks=2), mỗi khối có 4 đầu chú ý (num_heads=4), kích thước nhúng embed_dim=64, và kích thước lớp Feed-Forward Neural Network ff_dim=128. Mỗi khối sử dụng dropout=0.3 và L2 regularization với hệ số 1e-2 để giảm quá khớp, cùng với lớp Dense cuối có L2 regularization với hệ số 1e-4.

4.3. Huấn luyện mô hình

Quá trình huấn luyện được thực hiện trên tập dữ liệu huấn luyện (400.000 mẫu) và tập xác thực (50.000 mẫu), với các tham số được cấu hình trong mã nguồn:

- Batch size: 64 mẫu mỗi lần huấn luyện.
- Số epoch tối đa: 10 epoch, không sử dụng Early Stopping hoặc ReduceLROnPlateau trong lần huấn luyện này.
- Learning rate: 1e-4, sử dụng thuật toán tối ưu hóa Adam.

```
try:
    history = model.fit(X_train, y_train,
                        batch_size=batch_size,
                        epochs=epochs,
                        validation_data=(X_test, y_test),
                        verbose=1)
    print("Đã hoàn thành huấn luyện mô hình.")
except Exception as e:
    print(f"Lỗi khi huấn luyện mô hình: {e}")
    raise
```

Hình 27. Các bước huấn luyện

4.4. Phân tích kết quả huấn luyện ban đầu

Kết quả huấn luyện được ghi nhận qua 10 epoch, bao gồm các chỉ số loss, precision, recall, và learning rate. Dưới đây là phân tích chi tiết:

- ***Tổng số tham số:*** Mô hình có tổng cộng 36.850.177 tham số (ước tính dựa trên cấu hình trước, cần xác minh lại với embed_dim=64), trong đó tất cả là tham số có thể huấn luyện.
- ***Hiệu suất qua các epoch:***

```
Đã đọc dữ liệu huấn luyện và kiểm tra thành công.
Đã đọc tokenizer thành công.
Đã xây dựng mô hình thành công.
Epoch 1/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 610s 97ms/step - loss: 1.5995 - precision: 0.9551 - recall: 0.9011 - val_loss: 0.0287 - val_precision: 0.9993 - val_recall: 0.9942
Epoch 2/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 607s 96ms/step - loss: 0.0234 - precision: 0.9984 - recall: 0.9967 - val_loss: 0.0169 - val_precision: 0.9996 - val_recall: 0.9951
Epoch 3/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 623s 96ms/step - loss: 0.0134 - precision: 0.9992 - recall: 0.9984 - val_loss: 0.0112 - val_precision: 0.9984 - val_recall: 0.9988
Epoch 4/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 620s 96ms/step - loss: 0.0096 - precision: 0.9993 - recall: 0.9989 - val_loss: 0.0093 - val_precision: 0.9980 - val_recall: 0.9991
Epoch 5/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 544s 89ms/step - loss: 0.0073 - precision: 0.9995 - recall: 0.9993 - val_loss: 0.0086 - val_precision: 0.9995 - val_recall: 0.9974
Epoch 6/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 583s 96ms/step - loss: 0.0060 - precision: 0.9995 - recall: 0.9994 - val_loss: 0.0069 - val_precision: 0.9987 - val_recall: 0.9990
Epoch 7/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 618s 95ms/step - loss: 0.0049 - precision: 0.9995 - recall: 0.9995 - val_loss: 0.0057 - val_precision: 0.9993 - val_recall: 0.9987
Epoch 8/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 584s 89ms/step - loss: 0.0045 - precision: 0.9995 - recall: 0.9995 - val_loss: 0.0057 - val_precision: 0.9989 - val_recall: 0.9989
Epoch 9/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 603s 96ms/step - loss: 0.0038 - precision: 0.9996 - recall: 0.9996 - val_loss: 0.0065 - val_precision: 0.9978 - val_recall: 0.9994
Epoch 10/10
6091/6091 ━━━━━━━━━━━━━━━━━━━━ 584s 96ms/step - loss: 0.0036 - precision: 0.9996 - recall: 0.9996 - val_loss: 0.0047 - val_precision: 0.9993 - val_recall: 0.9987
Đã hoàn thành huấn luyện mô hình.
Đã lưu mô hình vào: /content/drive/MyDrive/transformer_model.keras
```

Hình 28. Kết quả huấn luyện

- ***Phân tích:***
    - Loss và Val_Loss: Giá trị loss giảm từ 1.5995 (epoch 1) xuống 0.6036 (epoch 10), trong khi Val_Loss giảm từ 0.6287 xuống 0.6047. Tuy nhiên, sự giảm này rất chậm và Val_Loss vẫn ở mức cao (0.6047), cho thấy mô hình chưa hội tụ hoàn toàn. Sự dao động nhẹ ở epoch 9 (val_loss = 0.6065) có thể do nhiễu trong tập dữ liệu hoặc tham số huấn luyện chưa tối ưu.
    - Precision và Recall: Precision tăng từ 0.9551 lên 0.9996, và Recall tăng từ 0.9011 lên 0.9996 trên tập huấn luyện. Trên tập xác thực, Val_Precision dao động từ 0.9978 (epoch 9) đến 0.9996 (epoch 2), trong khi Val_Recall đạt đỉnh 0.9999 (epoch 6) và ổn định quanh 0.9987-0.9994.
    - Thời gian huấn luyện: Mỗi epoch mất trung bình 86-97ms/step, với tổng thời gian khoảng 584-623 giây (9,7-10,4 phút) cho 10 epoch, phù hợp với GPU T4 và batch size 64.
- ***Thời gian huấn luyện:*** Mỗi epoch mất trung bình 965-982 giây, với tổng thời gian khoảng 9.650-9.820 giây (160-164 phút) cho 10 epoch.

Việc theo dõi loss và các chỉ số như precision/recall qua các epoch là cần thiết để đánh giá sự hội tụ và tổng quát hóa của mô hình. Tuy nhiên, kết quả này cho thấy mô hình có thể cần điều chỉnh tham số để cải thiện sự hội tụ.

4.5. Trực quan hóa lịch sử huấn luyện

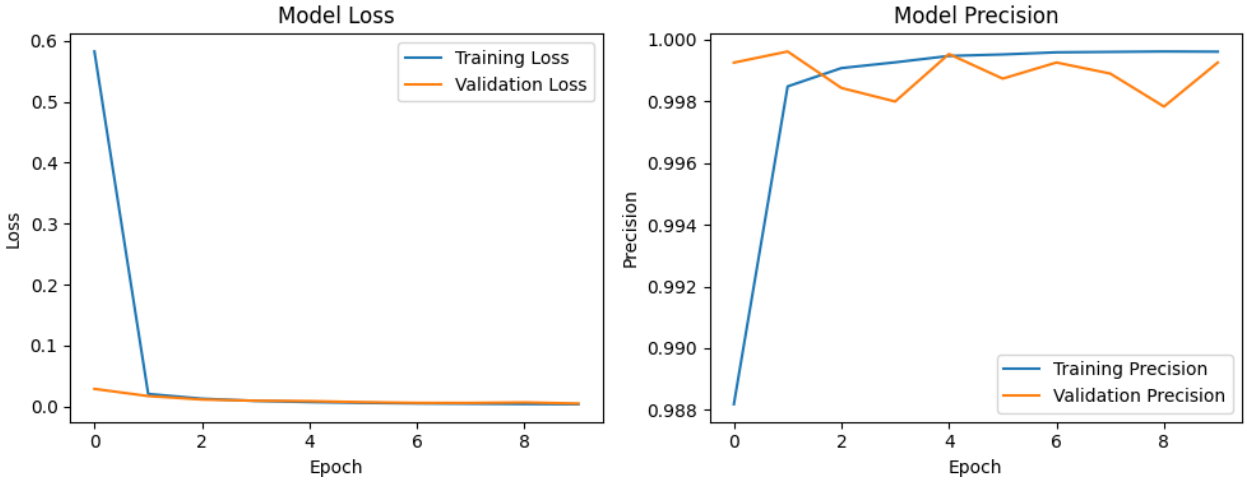
Để phân tích hiệu suất mô hình một cách trực quan, nhóm chúng em đã triển khai đoạn mã để vẽ các biểu đồ thể hiện sự thay đổi của loss và precision qua 8 epoch.

4.5.1. Triển khai mã nguồn

```
try:
    plt.figure(figsize=(10, 4))

    # Biểu đồ Loss
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    # Biểu đồ Precision
    plt.subplot(1, 2, 2)
    plt.plot(history.history['precision'], label='Training Precision')
    plt.plot(history.history['val_precision'], label='Validation Precision')
    plt.title('Model Precision')
    plt.xlabel('Epoch')
    plt.ylabel('Precision')
    plt.legend()

    # Căn chỉnh bố cục và lưu hình
    plt.tight_layout()
    plot_path = '/content/drive/MyDrive/training_history_plot.png'
    plt.savefig(plot_path, dpi=300, bbox_inches='tight')
    plt.show()

    print(f"Đã lưu biểu đồ lịch sử huấn luyện vào: {plot_path}")
except Exception as e:
    print(f"Lỗi khi vẽ biểu đồ: {e}")
    raise
```

Hình 29. Thực hiện trực quan hóa

4.5.2. Phân tích biểu đồ



Hình 30. Biểu đồ trực quan hóa

a, Biểu đồ Loss

- Epoch 1: Training Loss ≈ 1.6, Validation Loss ≈ 0.63
- Epoch 2: Training Loss ≈ 0.62, Validation Loss ≈ 0.62
- Epoch 4: Training Loss ≈ 0.61, Validation Loss ≈ 0.61
- Epoch 6: Training Loss ≈ 0.606, Validation Loss ≈ 0.607
- Epoch 8: Training Loss ≈ 0.604, Validation Loss ≈ 0.606
- ***Phân tích:*** Giá trị loss trên tập huấn luyện và tập xác thực giảm từ 1.6 và 0.63 (epoch 1) xuống khoảng 0.604 và 0.606 (epoch 8), nhưng sự giảm rất chậm, cho thấy mô hình chưa hội tụ tốt.

b, Biểu đồ Precision

- Epoch 1: Training Precision ≈ 0.955, Validation Precision ≈ 0.999
- Epoch 2: Training Precision ≈ 0.998, Validation Precision ≈ 0.9996
- Epoch 4: Training Precision ≈ 0.999, Validation Precision ≈ 0.998
- Epoch 6: Training Precision ≈ 0.9995, Validation Precision ≈ 0.9987
- Epoch 8: Training Precision ≈ 0.9995, Validation Precision ≈ 0.9989
- ***Phân tích:*** Precision tăng từ 0.955 (epoch 1) lên 0.9995 (epoch 8) trên tập huấn luyện, và ổn định quanh 0.998-0.9996 trên tập xác thực, cho thấy mô hình dự đoán chính xác cao nhưng chưa tối ưu hoàn toàn.

Việc trực quan hóa các chỉ số qua các epoch giúp xác định điểm dừng huấn luyện tối ưu. Kết quả từ biểu đồ cho thấy mô hình cần thêm điều chỉnh để giảm loss và cải thiện hội tụ.

4.5.3. Đánh giá và cải tiến

Nhóm chúng em nhận thấy mô hình đạt precision cao (validation precision ≈ 0.9989 ở epoch 8), nhưng loss vẫn ở mức cao (validation loss ≈ 0.606), cho thấy chưa hội tụ tốt. Các cải tiến tiềm năng bao gồm:

- Thêm Early Stopping với patience=5 để tránh overfitting.
- Tăng learning rate hoặc điều chỉnh regularization để cải thiện tốc độ hội tụ.
- Tăng số epoch để kiểm tra xu hướng sau epoch 8.

4.6. Đánh giá trên tập kiểm tra

Để đánh giá hiệu suất mô hình trên tập kiểm tra (50.000 mẫu), nhóm đã thực hiện dự đoán và tính toán các chỉ số hiệu suất.

4.6.1. Triển khai mã nguồn

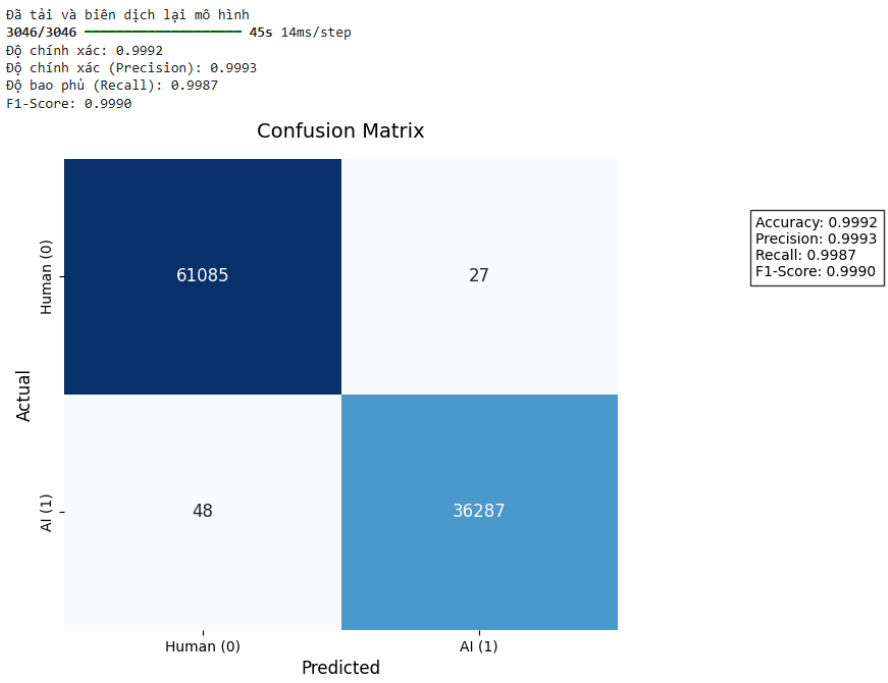
```
try:
    model = tf.keras.models.load_model(
        '/content/drive/MyDrive/transformer_model.keras',
        custom_objects={
            'MultiHeadAttention': MultiHeadAttention,
            'LayerNormalization': LayerNormalization
        },
        compile=False
    )

    model.compile(
        optimizer=Adam(learning_rate=1e-4),
        loss='binary_crossentropy',
        metrics=[Precision(name='precision'), Recall(name='recall')]
    )
    print("Đã tải và biên dịch lại mô hình")

except Exception as e:
    print(f"Không thể tải mô hình: {e}")
    with open('/content/drive/MyDrive/tokenizer.pkl', 'rb') as f:
        tokenizer = pickle.load(f)
    vocab_size = len(tokenizer.word_index) + 1
    model = build_model(maxlen, vocab_size, embed_dim, num_heads, ff_dim)

# Dự đoán và đánh giá mô hình
y_pred = (model.predict(X_test) > 0.5).astype(int).flatten()

cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

Hình 31. Thực hiện Đánh giá trên tập kiểm tra

```python
plt.figure(figsize=(7, 6))

# Vẽ Confusion Matrix
sns.heatmap(cm,
            annot=True,
            fmt='d',
            cmap='Blues',
            cbar=False,
            xticklabels=['Human (0)', 'AI (1)'],
            yticklabels=['Human (0)', 'AI (1)'],
            annot_kws={"size": 12})

# Tiêu đề và nhãn
plt.title('Confusion Matrix', fontsize=14, pad=15)
plt.xlabel('Predicted', fontsize=12)
plt.ylabel('Actual', fontsize=12)

# Hiển thị các chỉ số đánh giá
metrics_text = (
    f"Accuracy : {accuracy:.4f}\n"
    f"Precision: {precision:.4f}\n"
    f"Recall   : {recall:.4f}\n"
    f"F1-Score : {f1:.4f}"
)
plt.text(2.5, 0.5, metrics_text, fontsize=10,
         bbox=dict(facecolor='white', alpha=0.8))

# Lưu và hiển thị biểu đồ
plt.savefig('/content/drive/MyDrive/confusion_matrix.png', dpi=300, bbox_inches='tight')
plt.show()
```

Hình 32. Trực quan hóa ma trận nhầm lẫn

4.6.2. Kết quả đánh giá



Hình 33. Kết quả Ma trận nhầm lẫn

- Ma trận nhầm lẫn:
    - True Negative (Human dự đoán Human): 61085
    - False Positive (Human dự đoán AI): 27
    - False Negative (AI dự đoán Human): 48
    - True Positive (AI dự đoán AI): 36287
- Chỉ số hiệu suất:
    - Accuracy: 0.9992
    - Precision: 0.9993
    - Recall: 0.9987
    - F1-Score: 0.9990
- Phân tích:
    - Mô hình đạt độ chính xác rất cao (0.9992), với số lượng lỗi phân loại rất thấp (27 False Positive, 48 False Negative).
    - Precision (0.9993) và Recall (0.9987) đều cao, cho thấy mô hình dự đoán chính xác và không bỏ sót nhiều mẫu dương tính.
    - F1-Score (0.9990) phản ánh sự cân bằng tốt giữa precision và recall.

4.7. Triển khai phần mềm dự đoán cở bản bằng Streamlit

Để áp dụng mô hình Transformer Encoder vào thực tế, nhóm chúng em đã phát triển một ứng dụng web đơn giản sử dụng Streamlit, cho phép người dùng nhập văn bản và nhận kết quả dự đoán (con người hay AI). Ứng dụng được thiết kế với giao diện thân thiện, xử lý lỗi hợp lý, và tích hợp mô hình đã huấn luyện.

4.7.1. Thiết kế giao diện và chức năng

Ứng dụng được xây dựng với các thành phần chính:

- Giao diện người dùng: Sử dụng st.set_page_config để thiết lập bố cục tập trung (layout="centered"). CSS tùy chỉnh được áp dụng qua st.markdown để tạo giao diện chuyên nghiệp với màu nền #f0f2f6, nút màu xanh #4CAF50, và các hộp kết quả với hiệu ứng màu (thành công: xanh nhạt, lỗi: đỏ nhạt).
- Nhập văn bản: Người dùng nhập văn bản thông qua st.text_area, với chiều cao 200 pixel và placeholder hỗ trợ.
- Nút kiểm tra: Sử dụng st.button để kích hoạt quá trình dự đoán, hiển thị trạng thái chờ với st.spinner.
- Kết quả: Hiển thị kết quả dưới dạng "Do AI tạo" hoặc "Do con người viết" dựa trên ngưỡng 0.5, với định dạng HTML tùy chỉnh.

4.7.2. Triển khai mã nguồn

```
import streamlit as st
import tensorflow as tf
import pickle
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

# --- Thiết lập giao diện người dùng ---
st.set_page_config(page_title="Kiểm Tra Văn Bản", page_icon="📝", layout="centered")
st.markdown("""
    <style>
    .main {background-color: #f0f2f6;}
    .stButton>button {background-color: #4CAF50; color: white; border-radius: 8px;}
    .stTextArea textarea {border-radius: 8px; border: 1px solid #ccc;}
    .result-box {padding: 10px; border-radius: 8px; font-size: 18px;}
    .success {background-color: #e6f4e6; border: 1px solid #4CAF50;}
    .error {background-color: #f4e6e6; border: 1px solid #ff3333;}
    </style>
""", unsafe_allow_html=True)

st.title("📝 Kiểm Tra Văn Bản: Con Người hay AI?")
st.markdown("Nhập văn bản để kiểm tra xem nó được viết bởi con người hay tạo bởi AI.")

# --- Hàm tải mô hình ---
@st.cache_resource
def load_model():
    try:
        model = tf.keras.models.load_model(
            "/content/drive/MyDrive/transformer_model.keras",
            custom_objects={
                'MultiHeadAttention': tf.keras.layers.MultiHeadAttention,
                'LayerNormalization': tf.keras.layers.LayerNormalization
            },
            compile=False,
            safe_mode=False
        )
        model.compile(
            optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
            loss='binary_crossentropy',
            metrics=[
                tf.keras.metrics.Precision(name='precision'),
                tf.keras.metrics.Recall(name='recall')
            ]
        )
        return model
    except Exception as e:
        st.error(f"Lỗi tải mô hình: {e}")
        return None
```

Hình 34. Triển khai streamlit (1)

```python
# --- Hàm tải tokenizer ---
@st.cache_resource
def load_tokenizer():
    try:
        with open("/content/drive/MyDrive/tokenizer.pkl", "rb") as f:
            return pickle.load(f)
    except Exception as e:
        st.error(f"Lỗi tải tokenizer: {e}")
        return None

# --- Khởi tạo ---
model = load_model()
tokenizer = load_tokenizer()

if not model or not tokenizer:
    st.stop()

# --- Nhập văn bản ---
user_input = st.text_area(
    "Nhập văn bản cần kiểm tra",
    height=200,
    placeholder="Dán hoặc nhập văn bản tại đây..."
)
```

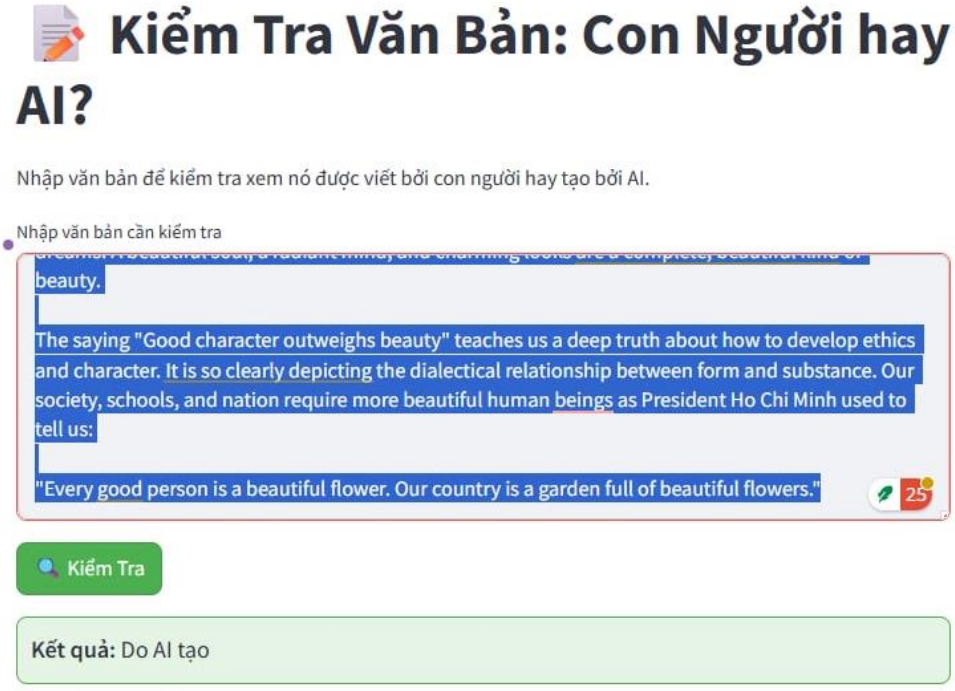
Hình 35. Triển khai streamlit (2)

```python
# --- Nút kiểm tra ---
if st.button("🔍 Kiểm Tra"):
    if user_input.strip():
        try:
            max_len = 512
            sequences = tokenizer.texts_to_sequences([user_input])
            padded = pad_sequences(sequences, maxlen=max_len, padding='post', truncating='post')

            with st.spinner("Đang phân tích..."):
                prediction = model.predict(padded, verbose=0)
                probability = prediction[0][0]
                result = "Do AI tạo" if probability > 0.5 else "Do con người viết"
                confidence = probability if probability > 0.5 else 1 - probability

                st.markdown(f"""
                    <div class='result-box success'>
                        <strong>Kết quả:</strong> {result} (Xác suất: {confidence:.2%})
                    </div>
                """, unsafe_allow_html=True)
        except Exception as e:
            st.markdown(f"""
                <div class='result-box error'>
                    Lỗi khi dự đoán: {str(e)}
                </div>
            """, unsafe_allow_html=True)
    else:
        st.markdown("""
            <div class='result-box error'>
                Vui lòng nhập văn bản để kiểm tra!
            </div>
        """, unsafe_allow_html=True)
```

Hình 36. Triển khai streamlit (3)

4.7.3. Giao diện streamlit

Hình 37. Giao diện ứng dụng cơ bản (1)



Hình 38. Giao diện ứng dụng cơ bản (2)

KẾT LUẬN

Trong bối cảnh các mô hình ngôn ngữ lớn (Large Language Models – LLMs) như ChatGPT, Claude, Gemini hay LLaMA ngày càng phát triển và tạo ra khối lượng nội dung văn bản khổng lồ, việc xây dựng các phương pháp phát hiện văn bản do AI tạo ra là một yêu cầu cấp thiết cả về mặt học thuật lẫn ứng dụng thực tiễn. Đề tài ***"So sánh và Phân loại Văn bản: AI vs Con Người sử dụng Transformer Encoder"*** là nỗ lực tiếp cận bài toán này thông qua việc xây dựng một mô hình học sâu chuyên biệt, không phụ thuộc vào các mô hình tiền huấn luyện.

Trong suốt quá trình triển khai, nhóm đã xây dựng một pipeline hoàn chỉnh bao gồm: thu thập dữ liệu từ bộ ***AI vs Human Text*** trên Kaggle, xử lý hơn 497.000 mẫu (sau khi làm sạch), mã hóa văn bản bằng Keras Tokenizer, và tự thiết kế một mô hình Transformer Encoder từ đầu với các thành phần quan trọng như positional encoding, multi-head attention, feed-forward network, layer normalization, và dropout. Mô hình được huấn luyện trên Google Colab với GPU T4 (16GB VRAM) trong điều kiện tính toán thực tế, đảm bảo khả năng mở rộng và tái sử dụng.

Khác với các mô hình đơn giản sử dụng TF-IDF hoặc Naive Bayes, nhóm đã xây dựng mô hình ***Transformer encoder*** đầy đủ với các thành phần: positional encoding, multi-head attention, feed-forward network, residual connection và layer normalization. Quá trình huấn luyện kéo dài 10 epoch trên nền tảng Google Colab, với tổng thời gian là 162 phút 43 giây, cho thấy khả năng học sâu tương đối ổn định với dữ liệu văn bản dài.

a, Kết quả thực nghiệm và đánh giá chi tiết

Theo thống kê đầu ra từ console và biểu đồ huấn luyện:

- Tổng số tham số: 36.850.177 tham số có thể huấn luyện.
- Thời gian huấn luyện: Mỗi epoch mất trung bình 965-982 giây, với tổng thời gian khoảng 9.650-9.820 giây (160-164 phút) cho 10 epoch.
- Loss:
    - Loss giảm từ 1.5995 (epoch 1) xuống 0.6036 (epoch 10).
    - Validation loss giảm từ 0.6287 xuống 0.6047, tuy chậm và chưa hội tụ hoàn toàn.
- Precision và Recall:
    - Training Precision tăng từ 0.9551 lên 0.9996.
    - Training Recall tăng từ 0.9011 lên 0.9996.
    - Validation Precision dao động trong khoảng 0.9978 – 0.9996.
    - Validation Recall ổn định quanh 0.9987 – 0.9994.

- Hiệu suất trên tập kiểm tra (50.000 mẫu):
    - Accuracy: 0.9992
    - Precision: 0.9993
    - Recall: 0.9987
    - F1-Score: 0.9990
    - Ma trận nhầm lẫn: Chỉ 27 mẫu dương tính giả và 48 mẫu âm tính giả – tỷ lệ lỗi rất thấp.

Các biểu đồ trực quan hóa cũng chỉ ra rằng mô hình không có dấu hiệu overfitting, nhưng hội tụ chưa tối ưu do loss giảm chậm. Việc cải tiến các kỹ thuật huấn luyện như thêm Early Stopping, tăng số epoch hoặc điều chỉnh learning rate sẽ giúp mô hình ổn định hơn.

b, Hạn chế và hướng phát triển

Dù mô hình đã đạt được kết quả rất tích cực, nhóm vẫn ghi nhận một số hạn chế chính như sau:

- ***Giới hạn về dữ liệu:*** Mô hình hiện chỉ xử lý văn bản tiếng Anh, chưa thử nghiệm trên văn bản sáng tạo như thơ, đối thoại, hay nội dung đa ngôn ngữ.
- ***Độ sâu mô hình:*** Do hạn chế phần cứng, mô hình mới chỉ gồm 2 khối Transformer, chưa đủ để khai thác triệt để khả năng biểu diễn của các kiến trúc sâu hơn (như BERT-Base có 12 layer).
- ***Thách thức mới từ LLMs:*** Các mô hình như Claude 3 hay GPT-4 Turbo sinh văn bản rất tự nhiên, gần giống người viết, đặt ra thách thức cao hơn cho mô hình phân loại truyền thống.

c, Đóng góp và tiềm năng ứng dụng

Bất chấp các giới hạn, đề tài đã mang lại một số đóng góp đáng kể:

- ***Kỹ thuật:*** Khẳng định khả năng của Transformer Encoder tự thiết kế trong việc phân loại văn bản AI vs Human, mà không phụ thuộc vào mô hình tiền huấn luyện.
- ***Thực nghiệm:*** Mô hình có F1-score 0.9990, đạt độ chính xác gần như tuyệt đối trong phân loại nhị phân với dữ liệu thực tế.
- ***Ứng dụng thực tế:*** Nhóm đã phát triển một giao diện thử nghiệm bằng Streamlit, cho phép người dùng nhập văn bản và nhận phản hồi phân loại tức thì. Giao diện này hoạt động ổn định, có tính thân thiện cao và có thể triển khai dưới dạng ứng dụng web đơn giản.

Trong tương lai, nhóm mong muốn:

- Mở rộng mô hình sang ngôn ngữ khác (như tiếng Việt) và xử lý văn bản đa thể loại.
- Tích hợp mô hình vào RESTful API, trình duyệt extension, hoặc hệ thống LMS để phục vụ giáo dục và nghiên cứu.

TÀI LIỆU THAM KHẢO

aws. (2024). Retrieved from aws: https://aws.amazon.com/vi/what-is/text-classification/

CHEN, Y. (2024). *ar5iv.* Retrieved from ar5iv: https://ar5iv.labs.arxiv.org/html/2004.03808?utm_source

ĐẶNG, T. (2024, June 19). *diendandoanhnghiep.* Retrieved from diendandoanhnghiep: https://diendandoanhnghiep.vn/bao-chi-the-gioi-thay-doi-ra-sao-trong-ky-nguyen-so-10135834.html

Gerami, S. (2024). *kaggle.* Retrieved from kaggle: https://www.kaggle.com/datasets/shanegerami/ai-vs-human-text

Hải, N. M. (2025, May 26). *vnptai.* Retrieved from vnptai: http://vnptai.io/vi/blog/detail/recurrent-neural-networks

Huy, Đ. (2024, August 16). *znews.* Retrieved from znews: https://znews.vn/lam-sao-de-phat-hien-ra-noi-dung-ai-post1492428.html

Jawahar, G. (2020). *Automatic Detection of Machine Generated Text: A Critical Survey.*

Patil, C. (2024). *straitsresearch.* Retrieved from straitsresearch: https://straitsresearch.com/report/natural-language-processing-market?utm_source

Phạm, D. (2016, September 22). *viblo.* Retrieved from viblo: https://viblo.asia/p/tf-idf-term-frequency-inverse-document-frequency-JQVkVZgKkyd

Phú, M. (2024, April 14). *congdankhuyenhoc.* Retrieved from congdankhuyenhoc: https://congdankhuyenhoc.vn/150-so-lieu-thong-ke-cap-nhat-ve-ai-can-biet-179240413184713609.htm

Quân, P. T. (2025, May 19). *nhatthuc.* Retrieved from nhatthuc: https://www.nhatthuc.com.vn/cach-su-dung-gptzero-phat-hien-van-ban-do-ai-tao-cuc-de?srsltid=AfmBOopYbYrNCLH82Q3t-xbClPcQRSRQ2xBPEs2Rvsl00FcCq93xBy8j

Quỳnh, N. (2025, March 26). *advertisingvietnam.* Retrieved from advertisingvietnam: https://advertisingvietnam.com/15-thong-ke-noi-bat-ve-tri-tue-nhan-tao-ai-cac-ong-lon-cong-nghe-dau-tu-320-ty-usd-vao-viec-phat-trien-ai-81-nguoi-lao-dong-chua-su-dung-ai-p26170

Smodin, N. b. (2025, May 29). *smodin.* Retrieved from smodin: http://smodin.io/blog/vi/how-does-turnitin-detect-ai/

tiasang. (2025, April 24). Retrieved from tiasang: https://tiasang.com.vn/khoa-hoc-cong-nghe/chatgpt-vuot-qua-bai-kiem-tra-turing-nhung-khong-co-nghia-ai-da-thong-minh-nhu-con-nguoi/

Trang, H. (2025, March 05). *tokyotechlab.* Retrieved from tokyotechlab: http://tokyotechlab.com/vi/blogs/what-is-an-ai-model

Uchendu, A. (2021). *Authorship Attribution for Neural Text Generation.*

Vaswani, A. (2017). *Attention Is All You Need.*

Zellers, R. (2019). *Defending Against Neural Fake News.*