BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN
HỌC THỐNG KÊ

# BÁO CÁO BÀI TẬP MÔN HỌC

# NỘI DUNG: TRANSFORMER ARCHITECTURE MEDICAL QUESTION ANSWERING


**Giảng viên hướng dẫn** :Lê Long Quốc

**Lớp** :CQ2022/22

**Sinh viên thực hiện** :Huỳnh Tấn Lộc-22120186

Đinh Viết Lợi- 22120188

Nguyễn Nhật Long-22120194


*Hồ Chí Minh, ngày 30 tháng 4 năm 2025*

# MỤC LỤC

# PHẦN 1: TỔNG QUAN ĐỒ ÁN

## I. Tổng quan đồ án

### 1. Giới thiệu đồ án

- Đồ án Transformer_based_NLP_Application là đồ án cuối kì phục vụ cho môn học Học thống kê của lớp CQ2022/22 năm học 2024-2025 của Đại học Khoa học Tự nhiên.
- Nội dung của đồ án tập trung vào việc tiếp cận và làm quen với các kiến thức liên quan đến các mô hình ngôn ngữ lớn (LLMs), lĩnh vực Xử lý Ngôn ngữ Tự nhiên (NLP) mà cụ thể hơn là tập trung vào việc tìm hiểu và xây dựng ứng dụng thực tế từ một mô hình có nền tảng kiến trúc Transformer.
- Bài toán được nhóm lựa chọn là 'Question Answering' phục vụ cho công việc tra cứu nhanh đơn thuốc và liều lượng bệnh nhân đang sử dụng từ bệnh án.

### 2. Yêu cầu đồ án

- Lựa chọn một bài toán trong lĩnh vực Xử lý Ngôn ngữ Tự nhiên để giải quyết. Một số bài toán có thể lựa chọn bao gồm: PoS Tagging, Text Classification, Named Entity Recognition, Machine Translation, Text Summarization, Question Answering…
- Sử dụng mô hình có kiến trúc Transformer để giải quyết bài toán được đặt ra.
- Tìm một tập dataset phù hợp và huấn luyện lại mô hình trên bộ data này. Thông tin về tập dữ liệu, mô hình sử dụng, phân chia tập các tập dữ liệu, kĩ thuật và độ đo đánh giá áp  dụng cho quá trình huấn luyện.
- Xây dựng giao diện web trình bày độ khả dụng và kết quả biểu diễn của mô hình.

## II. Kết quả đạt được

- Bộ dữ liệu sử dụng cho mô hình đảm bảo chất lượng lẫn số lượng, ít lỗi tiềm ẩn trong dữ liệu và phù hợp với mục đích sử dụng của nhóm.
- Mô hình được sử dụng phù hợp với bộ dữ liệu, đảm bảo có thể được sử dụng với lượng tài nguyên khan hiếm của nhóm.
- Áp dụng nhiều kỹ thuật cơ bản, kỹ thuật tối ưu, quy tắc huấn luyện và công cụ hỗ trợ bài bảng đã giúp việc huấn luyện mô hình đạt được kết quả tương đối tốt có thể được sử dụng trong thực tế.
- Xây dựng giao diện thân thiện với người dùng, hỗ trợ quan sát kết quả rõ ràng và thể hiện được toàn bộ tiềm năng của mô hình.
- Quá trình làm việc nhóm diễn ra bài bảng thuận lợi nhờ việc tổ chức không gian làm việc, trao đổi hiệu quả giữa các thành viên.
- Tất cả các thành viên hiểu được các khái niệm, quy trình quan trọng xuyên suốt quá trình làm việc, ý nghĩa và vai trò của từng giai đoạn cũng như công việc cụ thể.

# PHẦN 2: CƠ SỞ LÝ THUYẾT

## I. Natural Language Processing

- Natural Language Processing (NLP)- hay Xử lý Ngôn ngữ Tự Nhiên- là một lĩnh vực của trí tuệ nhân tạo chuyên nghiên cứu các phương thức để cải thiện trao đổi giữa con người và máy tính, giúp cho việc hiểu, diễn giải và phản hồi lại ngôn ngữ từ con người của máy tính trở nên tự nhiên nhất.
- Natural Language Understanding (NLU)- là thành phần quan trọng đầu tiên trong lĩnh vực này tập trung vào việc giúp máy tính hiểu "ý nghĩa" từ ngôn ngữ của con người. Biến đổi ngôn ngữ thành dữ liệu để xử lý.
- Natural Language Generation (NLG)- là thành phần quan trọng còn lại hướng tới việc giúp máy tính phản hồi, tạo ra thông tin mới cho con người từ việc tiếp nhận ngôn ngữ của con người. Biến đổi dữ liệu qua xử lý thành ngôn ngữ.
- Những khó khăn chính trong lĩnh vực này đến từ những nguyên nhân như ngữ pháp, cú pháp, ngữ cảnh, sự mơ hồ về ngữ nghĩa, vị trí của từ trong câu, cảm xúc, kiến thức nền tảng thế giới…
- Thông qua quá trình huấn luyện trên tập dữ liệu bằng các kỹ thuật học máy, chúng ta thu được một mô hình ngôn ngữ có khả năng xử lý các tác vụ cụ thể.

## II. Language Model

- Language Model (LM) là một mô hình đại diện cho những *kiến thức đã biết* về một ngôn ngữ, những *kiến thức* ấy có thể là những từ, chuỗi các từ **có thể có** hay **mức độ thường xuyên mà chúng xuất hiện**.
- Language Model được chia thành ba nhóm đó là Statistical Language Model (Count-based), Neural Network Language Model (Continuous-space) và Knowledge-based Language Model:
    - Statisical Language Model (Count-based): là mô hình ngôn ngữ được xây dựng dựa trên thống kê tần suất xuất hiện của các từ hoặc cụm từ trong văn bản. Mục tiêu là ước lượng xác suất một từ xuất hiện tiếp theo dựa vào các từ trước đó, thường sử dụng kỹ thuật N-gram.
    - Neural Network Language Model (Continuous-space): là những mô hình được xây dựng dựa trên mạng neural và biểu diễn từ dưới dạng vector liên tục (word embeddings). Khắc phục hạn chế của count-based bằng cách học được ngữ nghĩa và mối quan hệ giữa các từ, những phiên bản đầu tiên cho NNLM là RNN, LSTM…
    - Knowledge-based Language Model: mô hình ngôn ngữ được xây dựng dựa trên những kiến thức đã được con người (cụ thể là các chuyên gia ngôn ngữ) tích luỹ, phân tích từ cú pháp một câu, cách chia động từ. Mô hình được định nghĩa bằng các luật nên còn gọi là rule-based language model.
- Nhìn chung các mô hình ngôn ngữ này đều tồn tại nhiều khuyết điểm riêng biệt như

không hiểu ngữ nghĩa, mối quan hệ giữa các từ, cần nhiều tài nguyên tính toán phức tạp hoặc cần phải xây dựng thủ công, phụ thuộc nhiều vào tri thức chuyên ngành.

## III. Transformer

- **Transformer** là một kiến trúc mạng nơ-ron được thiết kế để xử lý các loại dữ liệu theo chuỗi (sequence), chẳng hạn như văn bản, âm thanh hoặc mã nguồn. Mô hình này được giới thiệu trong bài báo khoa học mang tên **"Attention is All You Need" (2017)** của các nhà nghiên cứu Google. Khác với các mô hình như RNN hay LSTM vốn xử lý từng phần tử theo thứ tự tuần tự, Transformer cho phép xử lý toàn bộ chuỗi dữ liệu cùng lúc, giúp tăng hiệu suất huấn luyện và dễ mở rộng lên các mô hình lớn hơn.
- Ngày nay, Transformer là nền tảng của nhiều mô hình ngôn ngữ mạnh nhất thế giới như GPT, BERT, T5, RoBERTa, và nhiều mô hình khác trong thị giác máy tính, xử lý âm thanh, thậm chí cả lập trình tự gen trong sinh học.
- Trước Transformer, các mô hình RNN và LSTM có khả năng ghi nhớ ngữ cảnh trước đó trong một chuỗi, nhưng có các nhược điểm sau:
  - Khó huấn luyện khi chuỗi quá dài do hiện tượng "quên dần" (**vanishing gradient**).
  - Không thể xử lý song song vì phải tuần tự hóa dữ liệu.
  - Tốn thời gian tính toán cho các chuỗi dài.
- Transformer giải quyết các vấn đề này thông qua các cơ chế:
  - Dùng cơ chế **attention** để xử lý toàn bộ chuỗi cùng lúc, thay vì từng phần tử.
  - Không có trạng thái tuần tự (stateless), cho phép song song hóa hoàn toàn trên GPU.
  - Hiểu được mối quan hệ giữa mọi cặp phần tử trong chuỗi, kể cả những phần tử cách xa nhau.
- Kiến trúc tổng thể của Transformer gồm hai thành phần chính là Encoder và Decoder:
  - **Encoder**: nhận đầu vào (ví dụ: một câu) và mã hóa thành các vector ngữ nghĩa.
  - **Decoder**: nhận vector từ encoder và sinh ra đầu ra (ví dụ: một câu dịch).
- Mỗi phần gồm nhiều lớp giống nhau xếp chồng lên nhau. Mỗi lớp gồm các thành phần sau:
  - **Embedding:** Các từ trong văn bản được biểu diễn thành các vector thực (số). Đây là bước đầu tiên để đưa dữ liệu văn bản vào mô hình toán học.
  - **Positional Encoding:** Vì mô hình không xử lý tuần tự nên cần thêm thông tin về vị trí từng từ. Positional encoding thêm các tín hiệu (dạng sóng sin/cos hoặc học được) vào các vector embedding để mô hình nhận biết thứ tự từ.
  - Multi-Head Attention: Đây là cơ chế cốt lõi giúp mô hình "chú ý" đến những phần

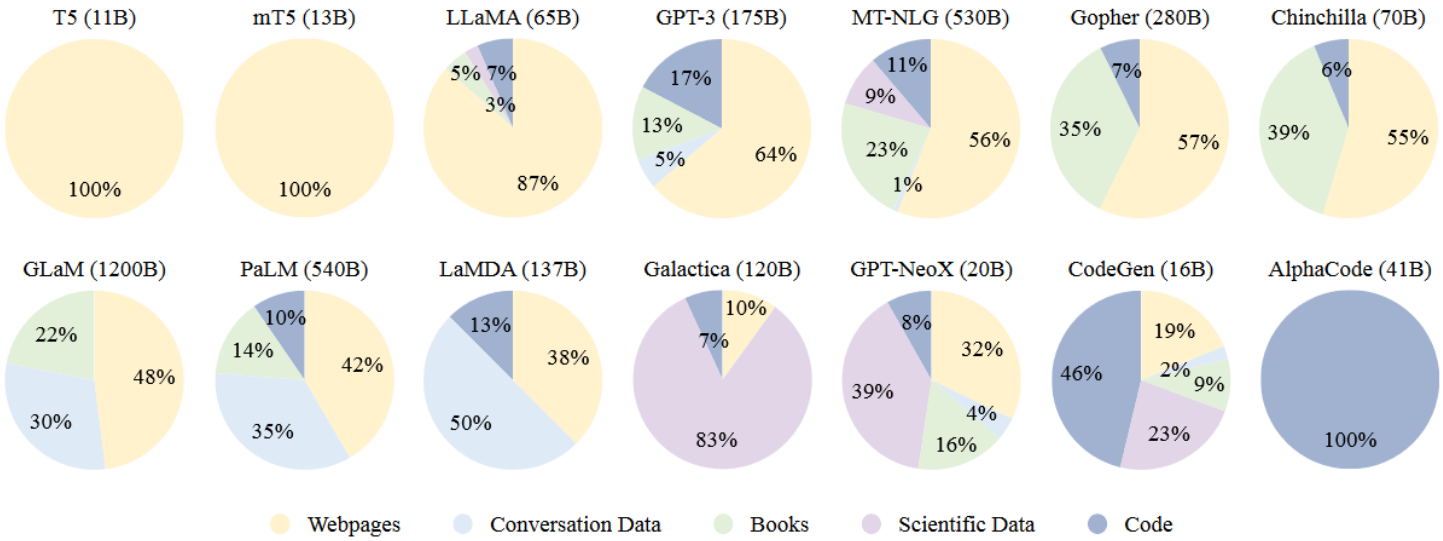
quan trọng trong chuỗi đầu vào. Một "đầu chú ý" học một khía cạnh của mối quan hệ giữa các từ, còn multi-head attention cho phép học nhiều khía cạnh cùng lúc. Điều này làm tăng khả năng hiểu ngữ cảnh của mô hình.

- **Residual Connection và Layer Normalization:** Các kết nối tắt (residual) giúp mô hình học nhanh hơn và ổn định hơn, còn Layer Normalization giúp điều chỉnh phân phối của các giá trị trong mô hình để tránh hiện tượng lệch chuẩn.
- **Feed-Forward Network (FFN):** Đây là thành phần rất quan trọng nhưng thường bị xem nhẹ vì tưởng đơn giản. FFN là một mạng nơ-ron con hoạt động độc lập trên từng vị trí trong chuỗi dữ liệu, có cấu trúc gồm hai lớp tuyến tính (dense layers) với một hàm kích hoạt phi tuyến (thường là ReLU) nằm ở giữa.

- Các mô hình kế thừa Transformer
  - **BERT**: Chuyên học ngữ cảnh hai chiều trong câu (từ cả trái và phải).
  - **GPT**: Tạo văn bản tự nhiên theo chiều từ trái sang phải.
  - **T5**: Biến mọi bài toán NLP thành bài toán "dịch" đầu vào sang đầu ra.
  - **RoBERTa, XLNet**: Các phiên bản cải tiến dựa trên Transformer.

## IV. Pre-trained models

- Việc huấn luyện các mô hình ngôn ngữ đặc biệt tiêu tốn rất nhiều tài nguyên tính toán, thời gian và yêu cầu lượng dữ liệu khủng lồ. Đặc biệt nếu tác vụ càng chuyên biệt và yêu cầu hiệu suất cao và không phải tổ chức nào cũng có thể đáp ứng lượng kinh phí này.
- May mắn cho các nhà phát triển, công đoạn trên hoàn toàn có thể lượt bỏ khi có rất nhiều mô hình ngôn ngữ có thể "lấy về dùng ngay" trên nhiều nền tảng mã nguồn mở, các công ty công nghệ được gọi là pre-trained models.
- Pre-trained models là các mô hình **đã được huấn luyện trước** trên các tập dữ liệu lớn và đa dạng thông tin như sách báo, hình ảnh, Wikipedia, truyền thông, hội thoại… Thông qua việc huấn luyện trước, các mô hình này đã được truyền sẵn bộ trọng số giúp cho mô hình có thể hiểu biết tổng quát về ngôn ngữ, có thể tái sử dụng cho nhiều tác vụ, thực hiện nhiều công việc đơn giản và biểu hiện việc hiểu và trao đổi tốt với con người.
- Các nhà phát triển dựa trên các pretrained model có thể dễ dàng tạo các ứng dụng AI nhanh hơn mà không phải lo lắng về việc cần lượng lớn dữ liệu nền tảng về ngôn ngữ và thế giới cũng như cách để cho mô hình có thể hiểu được ngôn ngữ.
- Việc sử dụng một pretrained model chất lượng cao với số lượng lớn trọng số đại diện chính xác sẽ mang lại cơ hội thành công cao hơn cho việc triển khai AI. Các trọng số có thể được sửa đổi và có thể bổ sung nhiều dữ liệu hơn vào mô hình để tùy chỉnh hoặc tinh chỉnh nó.




Hình 1: Cơ cấu bộ dữ liệu huấn luyện một số Language Model

## V. Fine-tuning

- Các pre-trained vốn đã thực hiện rất tốt các nhiệm vụ đơn giản, hiểu được tổng quát về ngôn ngữ tuy nhiên vì được huấn luyện bởi bộ General Data nên các câu trả lời được trả về cũng chỉ nằm ở mức phổ thông.
- Các nhà phát triển có nhu cầu khiến cho mô hình ngôn ngữ có khả năng thích nghi với nhiệm vụ cụ thể, giải quyết các nhiệm vụ cụ thể hơn thì cần phải tự fine-tuning mô hình.
- Fine-tune là quá trình tiếp tục huấn luyện một mô hình đã được pre-train nhưng với một tập dữ liệu nhỏ hơn và chuyên biệt hơn, nhằm thích nghi với một nhiệm vụ cụ thể gọi là Specialized Data.
- Quy trình cơ bản của fine-tune bao gồm lựa chọn pre-trained phù hợp (tức là bản thân nó vốn đã phù hợp cho nhiệm vụ chuyên biệt này), chuẩn bị tập dữ liệu chuyên biệt cho bài toán, cập nhật trọng số của mô hình dựa trên tập dữ liệu, đánh giá và tinh chỉnh mô hình.

## VI. Large Language Models

- Bắt đầu từ Statistical Language Models (SLM), sau đó phát triển thành Neural Language Models (NLM) và tiếp tục với Pretrained Language Models (PLM), mô hình ngôn ngữ đã chứng minh khả năng giải quyết hiệu quả nhiều tác vụ NLP. Thông thường, việc tăng kích thước mô hình hoặc mở rộng dữ liệu huấn luyện giúp cải thiện hiệu suất. Tuy nhiên, với PLM, các nghiên cứu chỉ ra rằng khi mô hình đạt đến quy mô đủ lớn, nó có thể xuất hiện những năng lực mới (emergent abilities) mà các mô hình nhỏ hơn không có. Những mô hình như vậy, với kích thước lớn và khả năng mạnh mẽ, được gọi là Large Language Models (LLMs), có thể xử lý các chuỗi tác vụ phức tạp vốn trước đây chỉ con người mới thực hiện được. – Tham khảo: [Từ Transformer Đến Language Model] Tổng quan về Large Language Model (phần 1)
- Thuật ngữ Large language model" thường dùng để chỉ các mô hình sử dụng kỹ thuật

học sâu và có số lượng tham số lớn, có thể từ hàng tỷ đến hàng nghìn tỷ. Những mô hình này có thể phát hiện các quy luật phức tạp trong ngôn ngữ và tạo ra các văn bản y hệt con người.

- LLMs là thế hệ mô hình ngôn ngữ mạnh mẽ, có khả năng hiểu và sinh ngôn ngữ tự nhiên với độ chính xác và đa năng vượt trội nhờ quy mô lớn.
- Việc lựa chọn các mô hình LLMs có thể giúp các tác vụ trở nên chính xác hơn nhiều so với các language model đời cũ.

## VII. Question Answering (QA) – Downstream Task

- Như đã đề cập bên trên, các nhà phát triển sẽ có mong muốn phát triển các ứng dụng AI có chức năng chuyên biệt giúp giải quyết các vấn đề cụ thể hơn bằng cách fine-tune trên những pre-trained models khi kết hợp với bộ dữ liệu liên quan, các tác vụ này gọi là downstream task- nhiệm vụ phía sau.
- Question Answering là tác vụ trong NLP nhằm trả lời một câu hỏi dựa trên một đoạn văn bản, tài liệu, hoặc kiến thức nền. Đây là một trong những downstream tasks (tác vụ ứng dụng) quan trọng, đòi hỏi mô hình không chỉ hiểu câu hỏi mà còn tìm ra và sinh ra câu trả lời phù hợp.
- Một số dạng phổ biến của Question Answering (QA) bao gồm:
  - Extractive QA: trích xuất thông tin trực tiếp từ văn bản được cung cấp.
  - Abstractive QA: Mô hình sinh ra câu trả lời mới không nhất thiết có trong văn bản.
  - Open-domain QA: không cung cấp đoạn văn bản, mô hình cần phải tìm câu trả lời từ web hoặc bộ dữ liệu.
  - Closed-domain QA: chỉ tìm câu trả lời trong một số nguồn cố định (nội bộ).

## VIII. BERT- RoBERTa

- BERT- **Bidirectional Encoder Representations from Transformers**)- là một mô hình ngôn ngữ được phát triển bởi Google AI và công bố vào năm 2018.
  - Kiến trúc: Dựa trên **Transformer**, nhưng chỉ dùng phần Encoder.
  - Mục tiêu: Học biểu diễn ngữ nghĩa của từ trong ngữ cảnh hai chiều (bidirectional).
  - Ứng dụng: Phân loại văn bản, trả lời câu hỏi, gán nhãn từ loại, nhận dạng thực thể (NER),...
- Một số cải tiến của BERT so với các mô hình trước đó:
  - **Bidirectional Contextual Understanding:** BERT đọc toàn bộ câu theo cả hai chiều cùng lúc, giúp hiểu ngữ nghĩa tốt hơn.
  - **Masked Language Modeling (MLM):** BERT học bằng cách **ẩn (mask)** một số từ trong câu và đoán chúng, giúp mô hình học được ý nghĩa ngữ cảnh từ hai chiều mà không bị "lộ" từ cần dự đoán.

  - **Next Sentence Prediction (NSP):** Giúp mô hình hiểu quan hệ giữa các câu để đoán câu B có là câu tiếp theo của câu A không.
  - BERT là một pre-trained model với 110M tham số.
- **RoBERTa** (Robustly Optimized BERT Approach) là một biến thể cải tiến của BER. Nó giữ nguyên kiến trúc Transformer Encoder giống BERT, nhưng được huấn luyện tốt hơn nhờ một số thay đổi quan trọng như:
  - Loại bỏ Next Sentence Prediction (NSP)
  - Kỹ thuật mask:  Dynamic masking mỗi epoch
  - Batchsize lớn hơn
- " Huấn luyện trước mô hình ngôn ngữ đã mang lại những cải thiện hiệu suất đáng kể, tuy nhiên việc so sánh cẩn thận giữa các phương pháp khác nhau là một thách thức. Quá trình huấn luyện tốn nhiều tài nguyên tính toán, thường được thực hiện trên các tập dữ liệu riêng tư với kích thước khác nhau, và — như chúng tôi sẽ chỉ ra — các lựa chọn siêu tham số (hyperparameter) có ảnh hưởng lớn đến kết quả cuối cùng.

  Chúng tôi thực hiện một nghiên cứu lặp lại (replication study) về huấn luyện trước của BERT (Devlin và cộng sự, 2019), nhằm đo lường một cách kỹ lưỡng tác động của nhiều siêu tham số quan trọng và kích thước dữ liệu huấn luyện.

  Chúng tôi phát hiện rằng BERT ban đầu đã được huấn luyện chưa đầy đủ, và hoàn toàn có thể đạt được hoặc vượt qua hiệu năng của mọi mô hình được công bố sau đó. Mô hình tốt nhất của chúng tôi đạt kết quả tốt nhất hiện tại (state-of-the-art) trên các bộ dữ liệu GLUE, RACE và SQuAD." - RoBERTa: ARobustly Optimized BERT Pretraining Approach . Đọc chi tiết bài báo tại : [1907.11692] RoBERTa: A Robustly Optimized BERT Pretraining Approach

# PHẦN 3: BÁO CÁO KẾT QUẢ

## I. Nguồn dữ liệu

- Trong lĩnh vực y tế, đơn thuốc thường chứa nhiều thuật ngữ chuyên ngành, nội dung dài dòng và phức tạp. Điều này gây khó khăn không chỉ cho bệnh nhân khi muốn hiểu thông tin trên đơn thuốc, mà ngay cả bác sĩ cũng mất thời gian khi cần tra cứu hoặc tóm tắt nhanh nội dung. Vì vậy, việc xây dựng một mô hình Question Answering (QA) tự động, có khả năng trích xuất nhanh và chính xác thông tin từ đơn thuốc theo câu hỏi người dùng đặt ra, sẽ hỗ trợ rất lớn trong việc tiết kiệm thời gian, nâng cao hiệu quả giao tiếp và đảm bảo tính chính xác trong điều trị.
- Medical Question Answer là ứng dụng từ lĩnh vực NLP giúp giải quyết vấn đề trên thông qua việc embedding vector từng từ vựng được đưa vào bởi ngữ cảnh (context), mô hình sẽ cố gắng dự đoán vị trí bắt đầu và vị trí kết thúc của câu trả lời trong ngữ cảnh để đưa về kết quả câu hỏi cho người dùng.
- Tập dữ liệu được sử dụng cho các tác vụ tương tự như trên được gọi là SQuAD (Standford Question Answering Dataset) được sử dụng chuyên biệt cho các việc huấn luyện và đánh giá các mô hình Question Answering. Cấu trúc của bộ dữ liệu SQuAD bao gồm:
  - **Context:** đoạn văn bản chứa thông tin, luôn chứa cả nội dung câu hỏi (SQuAD v1.1) là 1 trong 2 thành phần đầu vào quan trọng nhất.
  - **Question:** là câu hỏi được đặt ra nhằm mục đích lấy thông tin trong context, là thành phần đầu vào không thể thiếu còn lại.
  - **Answer:** kết quả được trả về cho câu hỏi, toàn bộ nội dung chắc chắn nằm trong context.
  - **Answer_start, end_start:** vị trí bắt đầu và kết thúc của answer
  - **Is_impossible:** cho biết câu trả lời có thể được trả lời hay không, bộ dữ liệu SQuAD v2.0 tồn tại những câu hỏi không có câu trả lời, giúp cho kết quả của mô hình đúng thực tế hơn.
- **emrQA-msquad** là một bộ dữ liệu mới được tái cấu trúc từ emrQA, gồm **163.695 câu hỏi** và **4.136 câu trả lời thủ công**, được thiết kế cho tác vụ trích xuất đoạn văn bản (span extraction) trong lĩnh vực y tế. Bộ dữ liệu này giúp giải quyết hiệu quả các thách thức về thuật ngữ chuyên ngành và tính mơ hồ trong câu hỏi. Khi fine-tune trên emrQA-msquad, các mô hình đạt được cải thiện rõ rệt về độ chính xác, với **F1-score tăng từ 10,1% lên 37,4% đối với BERT**, **từ 18,7% lên 44,7% với RoBERTa**, và **từ 16,0% lên 46,8% với Tiny RoBERTa**. Bộ dữ liệu này đóng vai trò quan trọng trong việc nâng cao hiệu suất của hệ thống hỏi đáp y khoa, hỗ trợ tốt hơn cho quyết định lâm sàng và nghiên cứu y học.[Nguồn dữ liệu: https://arxiv.org/abs/2404.12050]

## II. Tài nguyên & Công cụ

- Nguồn dữ liệu: emrQA-msquad v 2.0
- Công cụ quản lí mã nguồn dự án: Github
- Công cụ huấn luyện mô hình: PyTorch, TensorFlow, Hugging Face Transformers
- Công cụ tối ưu hoá và fintune: Gradient Accumulation, GradScaler.
- Tài nguyên phần cứng
  - Kaggle: Kaggle hỗ trợ cung cấp GPU mạnh mẽ như 2 GPU T4, GPU P100 với thời lượng sử dụng lớn lên đến 30 giờ/ tuần giúp tiết kiệm nhiều thời gian huấn luyện.
  - Google Colab: ứng dụng của Google cho phép người dùng tiếp cận miễn phí với tài nguyên huấn luyện mạnh mẽ với thời gian tương đương.
- Công cụ phân tích và đánh giá mô hình: wandb
- Công cụ triển khai mô hình: FastAPI, Transformers (Hugging Face), Starlette- CORS Middleware, Vue.js.

## III. Kỹ thuật

### 1. Kỹ thuật nền tảng

- *Inference Pretrained Transformer Model*
  - Nhóm đã thử nghiệm nhiều model với nhiều tham số khác nhau bao gồm: “bert-base-uncased", “deepset/roberta-large-squad2", “deepset/deberta-v3-squad2", “microsoft/deberta-v3-base".
  - Sử dụng AutoModelForQuestionAnswering từ thư viện Transformers (ở đây là deepset/roberta-base-squad2) để tận dụng model đã được finetune với Squad v2.0
  - Kết quả cho thấy Roberta base squad2 đã được finetune với squad2, và có kích thước vừa đủ, không bị overfit, nên đã trở thành lựa chọn phù hợp nhất.

| Mô hình | Số lượng tham số | Exact Match | Precison | Recall | F1-Score | BLEU-4 |
|---|---|---|---|---|---|---|
| deepset/roberta-base-squad2 | 125M | 0.0000 | 0.0078 | 0.0145 | 0.001 | 0.0013 |
| bert-base-uncased | 110M | 0.0000 | 0.0083 | 0.0093 | 0.00087 | 0.0008 |
| deepset/roberta-large-squad2 | 355M | 0.0003 | 0.0147 | 0.0233 | 0.018 | 0.0039 |

- Khởi tạo ngẫu nhiên và điều chỉnh môi trường:
  - Thiết lập seed cho torch, numpy và random để đảm bảo tính tái lập của kết quả.
  - Cấu hình biến môi trường

```
1. PYTORCH_CUDA_ALLOC_CONF="expandable_segments:True"
```

cho phép PyTorch quản lý bộ nhớ GPU một cách linh hoạt hơn, đặc biệt là với mixed-precision.

- Tận dụng GPU tăng tốc độ huấn luyện:

```
1.  (torch.device("cuda" if ... ))
```

- *Tokenization với AutoTokenizer*.
- *Sliding Window:*
  - Áp dụng cơ chế cắt chuỗi và trượt ngữ cảnh (max_length, stride, overflow_to_sample_mapping) để xử lý các ví dụ context quá dài.
  - Qua nhiều lần thử nghiệm lựa chọn, nhận thấy max_length = 512, stride = 256 là lựa chọn tối ưu nhất về thời gian và độ chính xác các thông số.
  - return_offsets_mapping=True để ánh xạ token trở lại vị trí ký tự gốc, nhằm tính toán start/end positions chính xác.
- *Offset Mapping* và *sequence_ids* để ánh xạ chính xác vị trí ký tự trong context về các vị trí token.
- *Data Collation***:** Collate_fn gom batch về dạng tensor đồng nhất, đảm bảo phù hợp với DataLoader.
- ***Chọn learning rate phù hợp***: Nhóm đã thử nhiều mức learning rate, kết quả thực tế là lr = 5e^-5 giúp valid loss giảm đi đáng kể, thấp hơn cả kết quả valid loss của tác giả.
- Thanh tiến trình (*Progress Bar*): Sử dụng tqdm để hiển thị tiến độ training, giúp theo dõi dễ dàng.
- eval_batch_size gấp 5 lần lần train_batch_size, vì ko phải tính toán gradient như train, nên có thể nâng eval_batch_size cao để giảm thời gian huấn luyện.

**2. Kỹ thuật tối ưu**

- *Mixed Precision*: Dùng torch.cuda.amp.autocast() và GradScaler để mixed-precision training, tăng tốc và giảm VRAM.
- *Data Parallel:*
  - Nn.DataParallel giúp chia mô hình chạy trên nhiều GPU. Mỗi GPU xử lý 1 phần batch song song. Trong 1 số trường hợp, nâng batch size lên gấp 2 giúp thời gian giảm đi ½.
  - Sử dụng num_proc cho Dataset.map để tiền xử lý song song nhiều tiến trình, giúp tăng tốc. Trong quá trình huấn luyện, nhóm đã thử nhiều giá trị và thấy num_proc = 4 là mức phù hợp.
  - Sử dụng num_workers cho DataLoader để tận dụng đa lõi CPU, giảm thời gian preprocessing và I/O. .
- *Gradient Accumulation*: accumulation_steps=10 để mô phỏng batch size lớn hơn, nhằm khắc phục hạn chế của GPU.
- *Optimizer & LR Scheduler*: Sử dụng get_scheduler("linear") để giảm learning rate

đều từ giá trị khởi đầu về 0 trong suốt quá trình training

- *Optimizer AdamW*: Dùng AdamW với các siêu tham số (lr=5e-5, betas=(0.9, 0.999), eps=1e-8) để tối ưu hoá trên các tham số của mô hình.
- *Checkpointing & Resume:*
  - Lưu model/tokenizer sau mỗi epoch (và epoch tốt nhất) để có thể resume hoặc phục hồi khi có sự cố.
  - Kiểm tra và tiếp tục từ checkpoint gần nhất, tránh mất công sức training.
- *Early Stopping:* Dừng training khi validation loss không cải thiện sau 1 epoch, tiết kiệm thời gian và tài nguyên.
- *Quản lý bộ nhớ:*
  - Thiết lập os.environ["PYTORCH_CUDA_ALLOC_CONF"] giúp PyTorch quản lý segment pool, giảm fragmentation.
  - Dọn optimizer.zero_grad() và model.train() ngay sau bước cập nhật gradient để giải phóng bộ nhớ.
  - import gc:tránh OOM trong các stage lưu checkpoint

## IV. Thông số đánh giá

- Các mô hình BERT cho bài toán Question Answering (QA) kiểu span extraction trả về chỉ số *start_token* và *end_token*, tương ứng với vị trí bắt đầu và kết thúc của câu trả lời trong đoạn văn bản đã được token hoá.
- Việc sử dụng *start_token* và *end_token* phù hợp cho quá trình huấn luyện vì tính trực tiếp loss (thường là CrossEntropyLoss) giúp mô hình học chính xác vị trí cần dự đoán.
- Tuy nhiên không phù hợp để dùng làm chỉ số đánh giá (benchmark) vì không dễ so sánh giữa các mô hình nếu dùng tokenizer khác nhau hoặc khớp văn bản không chính xác. Thay vào đó các benchmark như SQuAD, emrQA, hay mQAS thường sử dụng việc so sánh trực tiếp giữa câu trả lời dự đoán và câu trả lời đúng dưới dạng văn bản để đánh giá độ hiệu quả. Các chỉ số này bao gồm: Exact Math, Precision, Recall, F1 Score.

### 1. Exact Math (EM)

- Chỉ số **Exact Match (EM)** đánh giá tỷ lệ câu trả lời mà mô hình dự đoán **khớp hoàn toàn** với câu trả lời đúng (ground truth), sau khi chuẩn hóa.

```
EM = số câu trả lời dự đoán chính xác hoàn toàn / tổng số câu hỏi
```

- Ví dụ:

```
1. Ground truth: "Hồ Chí Minh"
2. Prediction: "hồ chí minh" →  EM = 1
3. Prediction: "thành phố Hồ Chí Minh" →  EM = 0
```

- Đây là chỉ số **khắt khe**, yêu cầu dự đoán phải **chính xác từng ký tự**

### 2. Precision

- Precision đo lường trong các từ mà mô hình dự đoán, có bao nhiêu từ thực sự xuất

hiện trong câu trả lời đúng.

```
1. Precision = Số từ trùng khớp / Tổng số từ trong câu trả lời dự đoán
```

- Ví dụ:

```
1. Ground truth: "Hồ Chí Minh"
2. Prediction: "thành phố Hồ Chí Minh"
3. Từ trùng: Hồ, Chí, Minh → 3 từ
4. Tổng số từ dự đoán: 5 → Precision = 3 / 5 = 0.6
```

### 3. Recall

- Recall đo lường trong các từ của câu trả lời đúng, có bao nhiêu từ được mô hình dự đoán chính xác hay nói cách khác là **độ bao phủ** của câu trả lời.

```
1. Recall = Số từ trùng khớp / Tổng số từ trong câu trả lời đúng
```

- Ví dụ:

```
1. Ground truth: "Hồ Chí Minh" → 3 từ
2. Prediction: "thành phố Hồ Chí Minh" → 3 từ trùng khớp
3. → Recall = 3 / 3 = 1.0
```

### 4. BLEU

- **BLEU** đo mức độ giống nhau giữa câu trả lời của mô hình và câu trả lời đúng (ground truth) dựa trên n-gram precision (độ chính xác của chuỗi n từ liên tiếp).
- **Ví dụ:**

```
1. Ground truth: "thành phố Hồ Chí Minh"
2. Prediction: "Hồ Chí Minh thành phố"
3. Bước 1: Các n-gram
4.         1-gram trùng khớp: "Hồ", "Chí", "Minh", "thành", "phố" → 5/5 → 1.0
5.         2-gram trùng khớp: Không có cặp nào đúng thứ tự → Precision 2-gram = 0
6.         3-gram trở lên: = 0
7.         →( Precision giảm khi n tăng, phản ánh câu đúng nhưng sai thứ tự từ.
8. Bước 2: BLEU-4
9.         BLEU-4 sẽ cho điểm thấp vì trật tự từ sai, mặc dù từ đúng đầy đủ.
```

### 5. F1-Score

- **F1 Score** là trung bình điều hòa của Precision và Recall, dùng để cân bằng giữa **độ chính xác** và **độ bao phủ**.

```
1. F1 = 2 × (Precision × Recall) / (Precision + Recall)
```

- Ví dụ:

```
1. Precision = 0.75
2. Recall = 1.0 → F1 = 2 × (0.75 × 1.0) / (0.75 + 1.0) ≈ 0.857
```

## V. Quy trình xây dựng mô hình

- Chuẩn bị môi trường

```
1. !pip install -q transformers
2. os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True"
```

  - expandable_segments:True giúp PyTorch tự động điều chỉnh phân bổ bộ nhớ GPU, tránh lỗi OOM (Out of Memory) khi xử lý batch lớn.
  - Cài đặt transformers – thư viện chính để sử dụng các mô hình pre-trained.

- Cấu hình mô hình và tham số huấn luyện

```
1. model_name = "deepset/roberta-base-squad2"
2. batch_size = 128
```

```
3. evaluate_batch = 80
4. max_length = 512
5. stride = 300
6. accumulation = 10
7. epochs = 3
8. lr = 5e-5
9. accumulation_steps = 10
10. seed = 42
```

- Tải dữ liệu SQuAD dạng JSON

```
1. raw_datasets = load_dataset("json", data_files={"train": train_path, "validation": valid_path})
```

  - Datasets hỗ trợ mapping dữ liệu linh hoạt, tiết kiệm RAM và có khả năng xử lý song song tốt.

- Khởi tạo tokenizer và mô hình

```
1. tokenizer = AutoTokenizer.from_pretrained(model_name)
2. model = AutoModelForQuestionAnswering.from_pretrained(model_name)
3. model.to(device)
```

- Tiền xử lý – Tokenization và gắn nhãn (label alignment)
- Biến dữ liệu đã tokenize thành Dataset và DataLoader

```
1. train_loader = DataLoader(tokenized_train, batch_size=batch_size, shuffle=True,
collate_fn=data_collator)
```

- Cấu hình optimizer, scheduler và mixed precision

```
1. optimizer = AdamW(model.parameters(), lr=lr) scaler = GradScaler() lr_scheduler =
get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0, num_training_steps=total_steps)
```

- Huấn luyện với Gradient Accumulation và Mixed Precision

```
1. for step, batch in enumerate(train_loader): with autocast():
2.   outputs = model(**batch) loss = outputs.loss / accumulation_steps
3.   scaler.scale(loss).backward()
4. if (step + 1) % accumulation_steps == 0:
5.   scaler.step(optimizer)
6.   scaler.update()
8.   lr_scheduler.step()
```

- Đánh giá mô hình (Validation)

```
1. model.eval()
2. with torch.no_grad():
3.     for batch in valid_loader:
4.         ...
5.
```

  - torch.no_grad() và model.eval() tắt dropout và không tính gradient → giảm bộ nhớ và tăng tốc độ suy luận.

- Checkpointing – Lưu mô hình

```
1. def save_checkpoint(model, tokenizer, epoch):
2.     os.makedirs(f"checkpoint-epoch-{epoch}", exist_ok=True)
3.     model.save_pretrained(...)
4.     tokenizer.save_pretrained(...)
```

- Lặp lại vòng lặp trên (nếu cần chạy tiếp epoch mới).
- Kết quả huấn luyện mô hình cho thấy

| |
|---|
| Epoch 1 Train Loss: 0.0551 \| Val Loss: 0.0884 |

| |
|---|
| Exact Match: 0.0010<br>Precision: 0.0933<br>Recall: 0.0998<br>F1 Score: 0.0862<br>BLEU Score: 0.0051 |
| Epoch 2 Train Loss: 0.0089 \| Val Loss: 0.0563<br>Exact Match: 0.0010<br>Precision: 0.0950<br>Recall: 0.1003<br>F1 Score: 0.0875<br>BLEU Score: 0.0052 → Thông số sau cùng của mô hình huấn luyện trên tập valid. |
| Epoch 3 Train Loss: 0.0083 \| Val Loss: 0.0639<br>Exact Match: 0.0010<br>Precision: 0.0935<br>Recall: 0.0984<br>F1 Score: 0.0862<br>BLEU Score: 0.0051 |

Sau Epoch 3 thì Val Loss đã tăng nên dừng huấn luyện với thiết lập early_stopping =1.

# PHẦN 4: TRIỂN KHAI ỨNG DỤNG

## I. Công nghệ sử dụng

### 1. Backend

- FastAPI
  - FastAPI là một framework hiện đại để xây dựng API với Python 3.7+ dựa trên các tính năng của **Starlette** (web microframework) và **Pydantic** (xử lý dữ liệu và validation).
  - Được dùng để định nghĩa và triển khai endpoint /qa nhận POST request và xử lý truy vấn đọc hiểu.
- Pydantic
  - Pydantic là thư viện Python chuyên dùng để xác thực và phân tích dữ liệu dựa trên các type annotation (kiểu dữ liệu) của Python.
  - Dùng để định nghĩa lớp QARequest cho dữ liệu đầu vào (question, context) của endpoint.
- Transformers (Hugging Face)
  - Transformers là thư viện mã nguồn mở của Hugging Face cho phép tải, huấn luyện, và sử dụng các mô hình Transformer (như BERT, RoBERTa, GPT, T5...) phục vụ các bài toán NLP
  - Dùng AutoModelForQuestionAnswering để xử lý bài toán đọc hiểu văn bản.
  - Load mô hình từ thư mục local (./best_model) → tăng tốc độ & đảm bảo offline inference.
- Starlette – CORS Middleware
  - Starlette là một web framework nền tảng (ASGI) mà FastAPI xây dựng dựa trên. Trong đó CORSMiddleware là một middleware dùng để cấu hình Cross-Origin Resource Sharing (CORS) — cho phép frontend ở domain khác truy cập API.
  - CORS được bật cho tất cả các domain (allow_origins=["*"])

### 2. Frontend- Vue.js

- Vue.js là một framework JavaScript dùng để xây dựng giao diện người dùng hiện đại. Vue hỗ trợ kiến trúc component, giúp chia nhỏ giao diện thành các phần tái sử dụng được. Hệ thống reactivity (tự động cập nhật khi dữ liệu thay đổi) và Virtual DOM (cập nhật hiệu quả vào giao diện) giúp Vue có hiệu năng cao và phát triển dễ dàng.
- Vue.js thường sử dụng thư viện Axios để gửi và nhận dữ liệu từ backend thông qua các API HTTP. Trong dự án này, frontend Vue gọi đến các endpoint của FastAPI (chạy ở backend) để thực hiện các chức năng như gửi câu hỏi và nhận câu trả lời từ mô hình AI. Axios hỗ trợ xử lý request/response linh hoạt, dễ cấu hình headers và hỗ trợ tốt các tác vụ như xác thực, xử lý lỗi, hoặc hiển thị tiến trình tải dữ liệu.

- Kết nối giữa frontend và backend được đảm bảo qua CORS (Cross-Origin Resource Sharing), cho phép ứng dụng Vue.js đang chạy ở một domain (hoặc cổng khác) có thể truy cập đến API của FastAPI một cách hợp lệ.

## II. Quy trình triển khai

- Chuẩn bị môi trường phát triển:
  - Frontend: Cài đặt Node.js và npm, khởi tạo dự án Vue.
  - Backend: Cài đặt FastAPI, Uvicorn, Transformers, Torch, và các thư viện liên quan, tải mô hình học sâu từ Hugging Face.
- Phát triển chức năng chính:
  - **Frontend (Vue.js)**: Xây dựng giao diện nhập liệu, kết nối API backend bằng Axios để gửi câu hỏi và context, hiển thị kết quả trả lời từ backend.
  - **Backend (FastAPI)**: Tải và nạp mô hình NLP phục vụ đọc hiểu văn bản, xây dựng API /QA nhận câu hỏi và ngữ cảnh để trả về câu trả lời, cấu hình CORS để cho phép frontend truy cập.
- Kiểm thử cục bộ (Local Testing): kiểm thử chức năng cơ bản, phản hồi từ backend và kiểm tra kỹ năng xử lý lỗi, rỗng, hoặc dữ liệu bất thường
- Hoàn thiện & kiểm tra tích hợp: Xử lý các lỗi liên quan đến CORS, domain, hoặc phiên bản API, kiểm tra hiệu năng trả lời, khả năng xử lý nhiều yêu cầu, tinh chỉnh giao diện.

## III. Cấu trúc thư mục

### 1. BE_AnswerAndQuestion (Backend - FastAPI)

- Thư mục này chứa mã nguồn backend sử dụng **FastAPI** để cung cấp API xử lý câu hỏi và trả lời.
- Cấu trúc:
  - *app.py:* Tập tin chính chạy ứng dụng FastAPI, khai báo API /qa, nạp mô hình và tokenizer từ thư mục best_model.
  - *best_model/:* Chứa mô hình học sâu đã fine-tune phục vụ cho bài toán đọc hiểu văn bản.
  - *README.md*: Mô tả cách chạy và sử dụng phần backend.
  - Medical_Question_Answering_Finetune: file huấn luyện mô hình.
  - Extract_data.ipynb: file trích xuất thông tin và định dạng lại tập dữ liệu

### 2. FE_AnswerAndQuestion (Frontend - Vue.js)

- Thư mục này chứa mã nguồn giao diện người dùng sử dụng Vue.js, nơi người dùng nhập câu hỏi, context và nhận lại câu trả lời từ API.
- Cấu trúc:

- *src/:* Chứa toàn bộ mã nguồn của ứng dụng Vue như component, logic gọi API, style, v.v.
- *public/:* Thư mục chứa các tài nguyên tĩnh (static assets), như favicon hoặc ảnh.
- *.vscode/:* Cấu hình dành cho VS Code (nếu có), không ảnh hưởng đến ứng dụng.
- *index.html:* File HTML gốc để Vue render ứng dụng.
- *package.json:* Khai báo thông tin project, các dependency, script liên quan.
- *vite.config.ts:* Cấu hình Vite - công cụ build và phát triển nhanh cho Vue.
- *tsconfig*.json:* Cấu hình TypeScript (do dùng Vue 3 + TypeScript).
- *README.md:* Hướng dẫn cài đặt và chạy frontend.
- Các file cấu hình như .gitignore, .editorconfig, .prettierrc.json,... dùng để chuẩn hóa và giữ code sạch.

**3. Dataset**

- Trained_dataset.json: Tập dữ liệu dành cho việc huấn luyện mô hình.
- Valid_dataset.json: Tập dữ liệu dành cho việc đánh giá chất lượng mô hình

**4. Docs**

- Report.pdf: báo cáo cụ thể của nhóm

# PHẦN 5: BÁO CÁO NHÓM

## I. Quy trình thực hiện

- Đồ án môn học được thực hiện trong  khoảng thời gian 29/3/2025-11/5/2025, quá trình thực hiện đồ án có thể được chia làm 3 giai đoạn chính bao gồm: **tiếp cận và ý tưởng, thử nghiệm và đánh giá, nâng cao và triển khai.**
- Quá trình làm việc yêu cầu sự trao đổi kiến thức và đánh giá liên tục của các thành viên nhằm hoàn thiện tốt sản phẩm. Sản phẩm liên tục được cập nhật theo tốc độ cập nhật kiến thức từ nhóm.
- Quy trình làm việc cụ thể của nhóm được mô tả bên dưới:

| Giai đoạn | Công việc | Mô tả chi tiết | Thời gian thực hiện |
|---|---|---|---|
| **Tiếp cận và ý tưởng** | Tìm hiểu về Transformer | Đọc tài liệu nhằm nắm bắt các khái niệm cơ bản của Transformer, các khác biệt chính của Transformer với các kiến trúc cũ hơn. | 29/3 - 1/4 |
| | Lựa chọn downstream-task | Downstream task sẽ quyết định tập dữ liệu, kiến trúc mô hình, loại mô hình được sử dụng cho sản phẩm. | 1/4 |
| | Tìm hiểu về pre-trained model, fine-tune | Không thể build model từ scratch nên việc tận dụng lại các model đã có nền tảng về ngôn ngữ mạnh mẽ, kích thước phù hợp sẽ giảm thiểu thời gian xây dựng và tăng chất lượng của sản phẩm | 1/4 – 3/4 |
| | Tìm nguồn dữ liệu | Nguồn dữ liệu phải phục vụ cho mục đích giải quyết vấn đề, đảm bảo chất lượng lẫn số lượng cũng như có các nghiên cứu trước đó nhằm so sánh hiệu suất. | 2/4 – 4/4 |
| | Lựa chọn các mô hình tiềm năng, độ đo phù hợp để phát triển | Lọc ra 3-4 mô hình tiềm năng được đánh giá dựa trên inferemce, pre-trained data, số lượng tham số và độ đo được sử dụng cho việc đánh giá benchmark. | 4/4 – 7/4 |
| | Vẽ pipeline xây dựng sản phẩm | Pipeline xây dựng giúp quá trình phát triển sản phẩm rõ ràng và dễ nắm bắt hơn khi quản lý, debugging. | 7/4 – 9/4 |
| **Thử nghiệm và đánh giá** | Tìm kiếm tài nguyên huấn luyện | Tìm kiếm các nguồn resource hỗ trợ cung cấp phần cứng mạnh mẽ cho việc huấn luyện mô hình. | 9/4 |
| | Code reading từ Github | Tham khảo quy trình, mã nguồn phát triển các mô hình QA dựa trên BERT từ các mã nguồn mở để hiểu, so sánh và cải thiện chất lượng mô hình. | 9/4 – 11/4 |
| | Inference pre-trained model | Inference cho biết chất lượng ban đầu | 9/4 – 12/4 |

| | | | |
|---|---|---|---|
| | | của mô hình phù hợp với tác vụ này thế nào. | |
| | Tiền xử lý dữ liệu | Xử lý tập dữ liệu với các vấn đề chuyển đổi format, tokenize, xử lý missing value. | 12/4 |
| | Huấn luyện mô hình cơ bản | Lựa chọn ngẫu nhiên các hyper parameter, thiết lập phần cứng cơ bản, các kĩ thuật chung nhất cho mọi quy trình huấn luyện. | 12/4 – 15/4 |
| | Đánh giá chất lượng huấn luyện | Kiểm tra xem mô hình có đang học được không, chỉ số cải thiện với mức độ nào, độ hao tổn tài nguyên và chất lượng đầu ra. | 15/4 |
| | Cải thiện kĩ thuật huấn luyện | Bổ sung các kĩ thuật nhằm tối ưu quá trình huấn luyện tập trung vào việc lựa chọn ra siêu tham số tối ưu nhất, cấu hình phần cứng, quản lí bộ nhớ, early stopping, theo dõi quá trình học theo thời gian thực. | 15/4 – 25/4 |
| | Đánh giá hiệu suất mô hình | Đánh giá độ cải thiện của mô hình sau cùng, nhận xét các chỉ số benchmark và giải thích, so sánh với các mã nguồn giải quyết vấn đề tương tự. | 25/4 – 30/4 |
| | Thiết kế giao diện sản phẩm | Sử dụng figma vẽ bản phát thảo cho giao diện sản phẩm, đảm bảo tối giản và thể hiện đầy đủ performance của mô hình. | 10/4 |
| **Triển khai** | Cài đặt môi trường phát triển frontend (Vue.js) và backend (FastAPI) | Cài đặt Node.js và npm, khởi tạo dự án Vue.<br>Cài đặt FastAPI, Uvicorn, Transformers, Torch, Pydantic và các thư viện liên quan. Tải mô hình từ Hugging Face. | 12/4 – 15/4 |
| | Phát triển giao diện frontend (Vue.js) và API backend (FastAPI) | Xây dựng giao diện để người dùng nhập câu hỏi và context. Sử dụng Axios để kết nối với API backend, gửi câu hỏi và nhận câu trả lời từ mô hình | 15/4 – 19/4 |

| | | | |
|---|---|---|---|
| | | NLP.<br>Xây dựng API cho backend, nạp mô hình NLP từ thư mục local. Cấu hình CORS cho phép frontend ở domain khác có thể gọi API. | |
| | Kiểm thử tích hợp và hoàn thiện | Đảm bảo frontend có thể truy cập API từ các domain khác mà không gặp lỗi CORS.<br>Đảm bảo API có thể xử lý nhiều yêu cầu đồng thời tránh tốc độ chậm hoặc gặp sự cố.<br>Tối ưu giao diện người dùng cho trải nghiệm mượt mà, kiểm tra tính tương thích với các thiết bị và độ phân giải khác nhau. | 19/4 – 30/4 |
| | Hoàn thiện tài liệu và hướng dẫn sử dụng | Viết tài liệu hướng dẫn cho người dùng cuối và hướng dẫn triển khai cho nhóm phát triển, bao gồm cả cách chạy và sử dụng backend và frontend. | 30/4 – 11/5 |

## II. Phân công công việc

- Huỳnh Tấn Lộc:
  - Thiết kế giao diện sản phẩm.
  - Phát triển giao diện frontend (Vue.js) và API backend (FastAPI).
  - Tích hợp mô hình vào giao diện trình bày sản phẩm.
  - Hoàn thiện tài liệu và hướng dẫn sử dụng.
  - Tái tổ chức toàn bộ các thư mục, viết tài liệu và hướng dẫn sử dụng sản phẩm.
- Đinh Viết Lợi:
  - Tham khảo các pre-trained model.
  - Tìm kiếm tập dữ liệu.
  - Inference các pre-trained model, tìm hiểu các độ đo benchmark.
  - Viết báo cáo tổng hợp đồ án.
- Nguyễn Nhật Long:
  - Quản lí chính quá trình xây dựng mô hình.
  - Biểu diễn kỹ thuật cơ bản huấn luyện mô hình.

- Đánh giá chất lượng mô hình, so sánh với các mã nguồn mở tương tự.
- Cải thiện chất lượng mô hình, bổ sung các kĩ thuật nhằm tối ưu quá trình huấn luyện.

# PHẦN 6: TÀI LIỆU THAM KHẢO

## I. Tài liệu nhóm

- Kho lưu trữ tài liệu: NLP - Google Drive

## II. Tài liệu tham khảo

[1] Vinbigdata, 'Large language model là gì? Tất cả những điều bạn cần biết về mô hình ngôn ngữ này'. Đường dẫn: Large language model là gì? Tất cả những điều bạn cần biết về mô hình ngôn ngữ này - VinBigData

[2] Nguyễn Văn Quân, '[Từ Transformer Đến Language Model] Tổng quan về Large Language Model (phần 1)'. Đường dẫn: [Từ Transformer Đến Language Model] Tổng quan về Large Language Model (phần 1)

[3] Khoa học dữ liệu- Khanh's blog, 'Bài 36-BERT model'. Đường dẫn: Khoa học dữ liệu

[4] Jimenez Eladio and Hao Wu, 'emrQA-msquad: A Medical Dataset Structured with the SQuAD V2.0 Framework, Enriched with emrQA Medical Information'. Đường dẫn: [2404.12050] emrQA-msquad: A Medical Dataset Structured with the SQuAD V2.0 Framework, Enriched with emrQA Medical Information

[5] Ducnh279, 'Context-based-Question-Answering-for-Vietnamese'. Đường dẫn: ducnh279/Context-based-Question-Answering-for-Vietnamese: Context-based-Question-Answering-for-Vietnamese

[6] PrimeVue. Đường dẫn: Setup - PrimeVue