TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———— * ————

Báo cáo Project III

Xây dựng ChatBot dựa trên mô hình LLaMA-2

Giảng viên hướng dẫn:

PGS.TS. Phạm Văn Hải

Sinh viên: Đỗ Thành Đức 20200159

9/2023

Xây dựng ChatBot dựa trên mô hình LLaMA-2

TÓM TẮT— *ChatBot là một ứng dụng quan trọng trong lĩnh vực trí tuệ nhân tạo và có nhiều ứng dụng trong thực tế. Bài báo cáo sau đây trình bày hướng tiếp cận cho việc xây dựng 1 ChatBot dựa trên mô hình LLaMA-2(Large Language Model for AI Assistance 2), một mô hình ngôn ngữ trí tuệ nhân tạo mạnh mẽ được phát triển bởi OpenAI. Trong bài báo cáo trình bày về cơ sở lý thuyết về mô hình LLaMA-2 từ kiến trúc mô hình đến các vấn đề xung quanh quá trình huấn luyện cho mô hình cùng với một số đánh giá về độ an toàn và chi phí của mô hình. Cuối cùng sẽ áp dụng mô hình để xây dựng 1 ChatBot có khả năng tương tác với người dùng, trả lời câu hỏi và thực hiện nhiều nhiệm vụ khác dựa trên thông tin mà nó học từ dữ liệu huấn luyện. Đồng thời đưa ra một số đánh giá về tốc độ, hiệu năng, chi phí khi xây dựng để chứng minh tính hiệu quả của mô hình trong việc tạo ra 1 ChatBot.*

I. Giới thiệu

Trong nhiều năm trở lại đây, sự phát triển của trí tuệ nhân tạo trong lĩnh vực khoa học và công nghệ diễn ra nhanh hơn bao giờ hết. Công nghệ Artificial Intelligence - AI (trí tuệ nhân tạo) ngày nay đang được xem là một trong những ngành công nghệ hàng đầu, giúp giải quyết được nhiều vấn đề trong cuộc sống, có thể giao tiếp, học và tự thích nghi với môi trường chung quanh. Đặc biệt trong thời gian gần đây, một lĩnh vực đang thu hút được sự quan tâm của rất nhiều người đó là ChatBot với rất nhiều lợi ích mà chúng đem lại cho doanh nghiệp, khách hàng và người dùng. Chúng được sử dụng để giải quyết nhiều mục đích từ cung cấp thông tin cơ bản đến hỗ trợ khách hàng và thậm chí thực hiện các giao dịch trực tuyến.

Cùng với sự phát triển của Trí tuệ nhân tạo, ngay ngày 19/7/2023 Facebook AI Research và đội ngũ kĩ sư của họ đã cho ra mắt LLaMA-2 **(***Large Language Model Meta AI)*, một mô hình AI thế hệ mới với nhiều năng lực mới hơn so với phiên bản tiền nhiệm

trước đó là LLaMA (xem chi tiết hơn trong [1]). Đây không chỉ là một công cụ giúp xây dựng ChatBot mà còn là một hệ thống trí tuệ nhân tạo đỉnh cao với khả năng hiểu và phản hồi ngôn ngữ tự nhiên của con người với benchmark vượt trội hơn hẳn so với khá nhiều mô hình hiện có.

Trong bài báo cáo này, dựa trên mô hình LLaMA-2 đã được công bố để xây dựng một ChatBot có khả năng tương tác với người dùng, trả lời câu hỏi và thực hiện nhiều nhiệm vụ khác dựa trên thông tin mà nó học từ dữ liệu huấn luyện. Phần I của báo cáo này nêu ra vấn đề và đề xuất cách giải quyết. Phần II đưa ra cơ sở lý thuyết của mô hình LLaMa-2 từ mô hình và các vấn đề xoay quanh quá trình huấn luyện. Phần III đưa ra một số đánh giá về độ an toàn cũng như chi phí của mô hình. Và trong phần IV sẽ trình bày về một số kết quả thực nghiệm đã đạt được trong quá trình xây dựng ChatBot dựa trên mô hình. Và cuối cùng trong phần kết luận, sẽ đưa ra một số đánh giá, thảo luận thêm về kết quả thực nghiệm và đề xuất thêm 1 số hướng phát triển cũng như đưa ra một số hạn chế, khó khăn trong quá trình thực hiện.

II Mô hình LLaMA-2

1. Đầu tiên LLaMA-2 là gì?

Nói một cách ngắn gọn thì LLaMa-2 là phiên bản tiếp theo của LLaMa - một mô hình ngôn ngữ lớn được tạo ra bởi Facebook AI Research và đội ngũ kĩ sư của họ. Mô hình này về mặt kiến trúc thì có vẻ tương tự như LLaMa nhưng được bổ sung thêm dữ liệu, cải thiện chất lượng cũng như đưa thêm các phương pháp tối ưu mới để đạt được hiệu suất cao hơn. Mô hình này cho benchmark vượt trội hơn hẳn so với các open-source model khác và đặc biệt là nó open source cả model, dữ liệu và cho phép sử dụng trong mục đích thương mại.

Với mô hình LLaMA-2 này, MetaAI release 2 phiên bản là pretrained LLM LLaMa-2 và một bản finetuned riêng cho tác vụ chat gọi là LLaMa-2-CHAT. Hai phiên bản này lại gồm nhiều biến thể với số lượng tham số từ 7B đến 70B.

Điểm mới của mô hình này so với LLaMA-1 là:

- **Context length** tăng từ 2048 lên 4096 giúp cho mô hình có thể capture được nhiều thông tin ngữ cảnh hơn.
- **Pretraining corpus** được tăng kích thước lên 40% bằng việc bổ sung thêm nhiều dữ liệu chất lượng của Meta.

- Áp dụng kĩ thuật **Grouped Query Attention** (xem chi tiết ở trong [2]) để làm tăng độ hiệu quả khi inference.

Tiếp theo chúng ta sẽ bàn đến về mô hình kiến trúc của LLaMA-2. Trong paper tác giả không nói rõ về kiến trúc mô hình, họ chỉ tiết lộ rằng kiến trúc mô hình tuân theo kiến trúc Transformer chuẩn và tương tự như kiến trúc của LLaMA-1. Vậy thì để hiểu sâu hơn về kiến trúc của LLaMa-2 chúng ta sẽ tìm hiểu qua về kiến trúc của mô hình LLaMa.

(Cả kiến trúc của LLaMA và LLaMA-2 đều là các **Generative Pretrained Transformer** dựa trên kiến trúc Transformer.)

2. Kiến trúc Tranformer:

Đây là sự kết hợp ưu điểm của mạng sâu như CNN(mạng nơ ron tích chập) và RNN (mạng nơ ron hồi tiếp). Đây là một kiến trúc song song học chuỗi hồi tiếp với cơ chế tập trung đồng thời mã hóa vị trí của từng phần tử trong chuỗi.

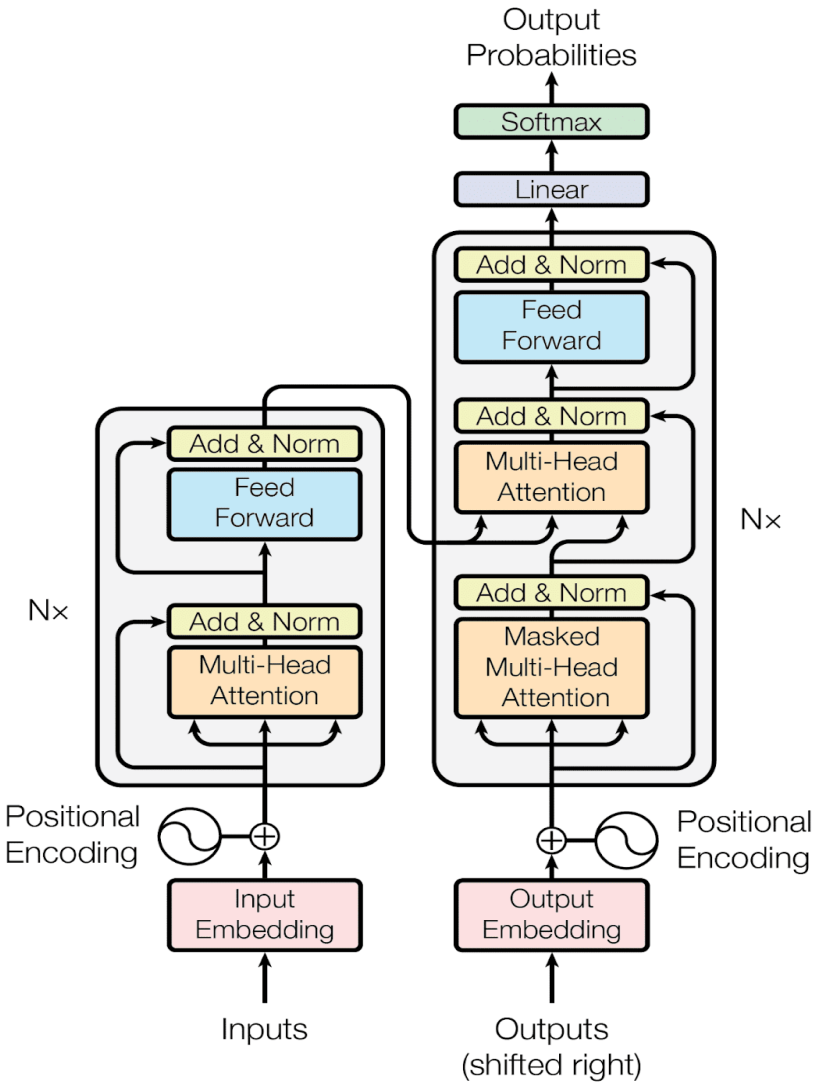
Transformer dựa trên kiến trúc encoder-decoder (mã hóa – giải mã). Tuy nhiên, nó thay thế các tầng hồi tiếp trong seq2seq bằng các tầng tập trung đa đầu (multi-head attention), kết hợp thông tin vị trí thông qua biểu diễn vị trí (positional encoding) và áp dụng chuẩn hóa tầng (layer normalization).

Các embedding của chuỗi nguồn được đưa vào n khối lặp lại. Đầu ra của khối mã hóa cuối cùng sau đó được sử dụng làm bộ nhớ tập trung cho bộ giải mã. Tương tự, các embedding của chuỗi đích được đưa vào n khối lặp lại trong bộ giải mã. Ta thu được đầu ra cuối cùng bằng cách áp dụng một tầng dày đặc có kích thước bằng kích thước bộ từ vựng lên các đầu ra của khối giải mã cuối cùng.

Kiến trúc của transformer gồm 2 phần chính là encoders(là 1 ngăn xếp gồm 6 khối encoder kiến trúc giống nhau) và decoders(là 1 ngăn xếp gồm 6 khối decoder giống nhau):

Mỗi khối encoder có 2 layer chính: self-attention và feed forward.

Mỗi khối decoder có 3 layer chính: sef-attention, encoder-decoder attention và feed forward.



The Encoder-Decoder Structure of the Transformer Architecture Taken from "Attention Is All You Need"

Kiến trúc của mỗi khối Encoder:

Input embedding

Đầu vào của khối encoder đầu tiên là các vector embeddings của các từ trong câu. Đầu vào của các khối encoder còn lại là đầu ra của khối endcoder phía dưới. Các embeddings được tạo thành từ việc kết hợp vector word embedding và positional embedding.

Vector word embedding là vector biểu diễn các từ được tạo ra từ các pre-model như word2vec, glove…

Vector positional embedding là vector biểu diễn thứ tự của các từ trong chuỗi, chúng mang thông tin về vị trí và khoảng cách của các từ. Vector positional embedding(PE) được tính theo công thức:

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

Trong đó dmodel là số chiều của vector, pos là số vị trí(0, 1, 2, 3….), i là chiều thứ i của vector(i ∈ 0, 1, 2...dmodel)

Layer self-attention

Phép tính đầu tiên trong sef-attention là nhân mỗi vector embedding đầu vào với 3 ma trận trọng số Wq, Wk, Wv để tạo ra 3 vector q, k, v. Các ma trận trọng số này sẽ được cập nhật trong quá trình đào tạo. Vector q và k được dùng để tính trọng số khuếch đại thông tin cho các từ trong câu. Vector v là vector biểu diễn của các từ trong câu.

Ví dụ ta có 2 vector embeddings(tương ứng với 2 từ đầu vào “Trời”, “lạnh”) là x1, x2. Nhân 2 vector này với 3 ma trân Wq, Wk, Wv ta được tập các vector: {q1, q2}, {k1, k2}, {v1, v2}. Để tính toán vector biểu diễn cho từ “Trời”. Đầu tiên ta cần tính trọng số khuếch đại thông tin cho mỗi từ(gọi là Attention), Attention cho từ “Trời”(a1) và từ “lạnh”(a2) được tính theo công thức sau:

$$a1 = \text{softmax}(q1*k1/\sqrt{d})$$

$$a2 = \text{softmax}(q1*k2/\sqrt{d})$$

Trong đó d là số chiều của vector k. Cuối cùng vector biểu diễn cho từ “Trời” được tính theo công thức:

$$z1 = a1*v1 + a2*v2$$

Tương tự việc tính toán vector biểu diễn cho các từ còn lại cũng được thực hiện như trên. Việc sử dụng multi-gpu để thực hiện các phép tính song song không được thực hiện ở bước này vì để tính được vector z của 1 từ ta cần có vector k và v của các từ khác.

Multi-Head Attention

Kiến trúc transformer được thiết kế với 8 lớp self-attention kiến trúc giống hệt nhau nhưng trọng số của 3 ma trận Q, K, V khác nhau. Việc tính toán của 8 layer này được thực hiện song song. Các vector biểu diễn qua mỗi lớp self-attention sẽ được nối lại với nhau sau đó được nhân với một ma trận trọng số Wo để nén thông tin từ 8 vector(8 vector này cùng biểu diễn cho 1 từ) thành một vector duy nhất. Vector này sau đó đi qua một bước gọi là Add & Normalize nữa trước khi đưa vào layer Feed Forward.

Ý nghĩa của cơ chế multi-head này là để tăng thêm phần chắc chắn trong việc quyết định thông tin nào cần khuếch đại, thông tin nào cần bỏ qua. Vì rằng 8 cái đầu sẽ cùng vote và đưa ra lựa chọn khách quan, đáng tin cậy hơn 1 cái đầu.

Position-wise Feed-Forward Networks(FFN)

Các vector sau khi đi qua bước Add & Normalize(sẽ được nói ở mục sau) sẽ được gửi tới FFN. Lớp này bao gồm 2 tầng biến đổi thông tin và 1 hàm ReLU(các giá trị < 0 được gán lại = 0) ở giữa. Dropout với tỉ lệ 0.1 cũng được áp dụng ở lần biến đổi thứ nhất sau khi các vector qua hàm ReLU.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Sau khi qua layer FFN các vector cũng phải qua bước Add & Normalize trước khi đi vào khối encoder kế tiếp.

Ý nghĩa của layer FFN này là để học mối quan hệ tiềm ẩn giữa các vector độc lập mà chưa được mô tả rõ ràng. Khác với mối quan hệ giữa các từ được khuếch đại qua lớp self-attention, vẫn còn những mối quan hệ tiềm ẩn khác không thể diễn giải bằng công thức toán học sẽ được học thông qua lớp này.

Add & Normalize

Ở bước này, Các vector đầu ra từ lớp con (multi-head self-attention và feed forward) sau đó qua bước dropout với tỉ lệ 0.1, rồi cộng thêm vector đầu vào(vector trước khi bị biến đổi), cuối cùng được normalized theo một công thức nào đó rồi chuyển vào layer kế tiếp. Ý nghĩa của bước này là để bổ sung thêm thông tin nguyên bản, tránh bị mất mát quá nhiều thông tin sau khi qua các phép biến đổi ở các layer multi-head self-attention và feed forward.

Đầu ra của khối encoder cuối cùng

Các vector sau khi qua lớp FFN của khối encoder cuối cùng sẽ được nhân với 2 ma trận trọng số K và V để tạo thành các cặp vector {(k1, v1), (k2, v2),...(kn, vn)} ứng với

câu có n từ. Các vector này sẽ được dùng để tính vector biểu diễn z trong lớp encoder-decoder attention.

Kiến trúc của mỗi khối decoder
Các layer trong khối decoder được thiết kế tương tự khối encoder tuy nhiên có 1 số điểm khác biệt sau:

Đầu vào của lớp sef-attention ở lần đầu tiên được là vector được tạo thành bởi embedding của 1 ký tự [start] + vector positonal embedding. Vector đầu vào ở các lần kế tiếp được tạo thành bởi vector output của layer FFN của khối decoder cuối cùng + vector positonal embedding.

Lớp self-attention chỉ kết hợp thông tin từ các từ trước nó.

Lớp encoder-decoder attention chỉ tính toán vector q dựa trên đầu ra của self-attention, vector k và v được lấy từ output của khối encoder.

Việc tính toán được thực hiện cho đến khi decoder dự đoán được ký tự kết thúc [end]

Đầu ra của lớp FFN cuối cùng sẽ được đi qua lớp Linear để biến đổi các vector này thành một vector có số chiều bằng số từ trong bộ vocabulary. Mỗi giá trị của 1 phần tử trong vector thể hiện điểm số cho 1 từ trong bộ từ vựng. Vector này sau đó được cho qua 1 hàm softmax để biến chúng thành một phân phối xác suất(tất cả phần tử >0 và tổng =1). Từ mà có xác suất cao nhất sẽ là từ được chọn.

3. Quá trình pretrained và Fine-tuning

Sự khác biệt giữa mô hình Llama-2 với kiến trúc GPT tiêu chuẩn:

- LLaMa sử dụng RMSNorm để chuẩn hoá input dầu vào cho mỗi layer transformer thay vì đầu ra
- Sử dụng **SwiGLU** activation thay vì **ReLu** giúp cho improve performance của quá trình huấn luyện.
- Sử dụng phương pháp tương tự như trong **GPT-Neo-X** LLaMA sử dụng **rotary positional embeddings (RoPE)** trong các layer của mạng

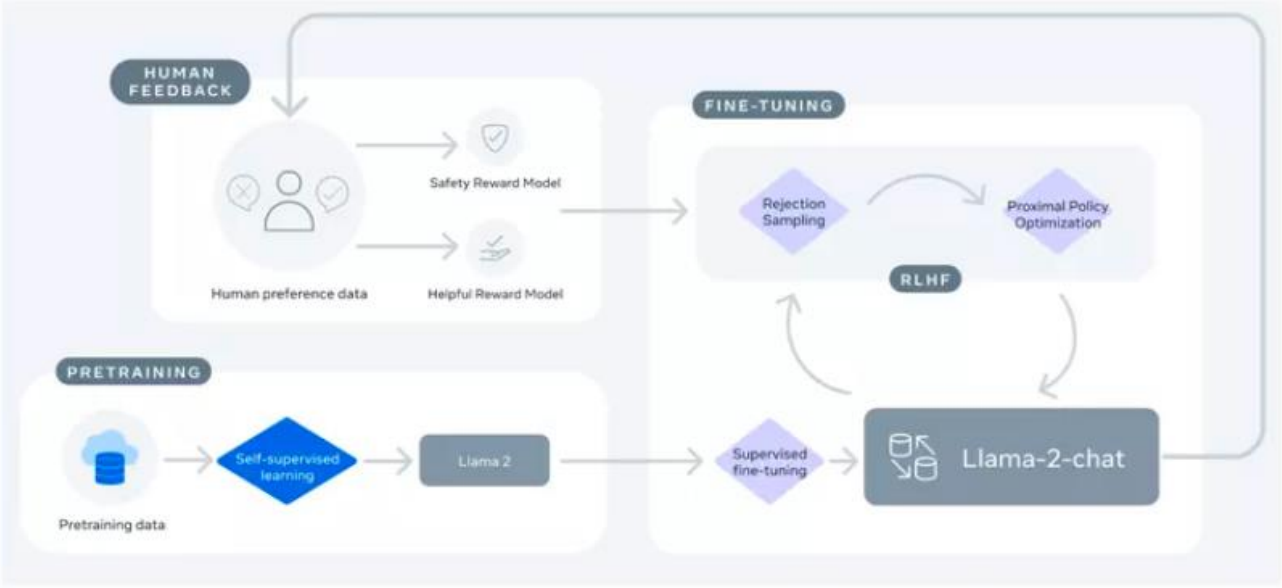
Theo như report trong paper của LLaMa-2 thì họ chỉ thay đổi duy nhất trong kiến trúc nằm ở kích thước của context length và sử dụng grouped-query attention. Việc tăng kích thước của conetxt length giúp cho mô hình có thể tạo ra và xử lý được nhiều thông tin hơn, nó rất thuận tiện cho việc hiểu các long documents. Việc thay thế ***multi-head atttention*** trong kiến trúc Transformer tiêu chuẩn - có nhiều query có thể tương ứng với một key-value projection thành grouped-query attention với 8 key-value projection cho phép tăng tốc độ huấn luyện, nó giúp dễ dàng tăng độ phức tạp của mô hình cũng như tăng

batchsize và context length. Những thay đổi này giúp cho mô hình LLaMa-2 có thể xử lý tốt hơn nhiều mô hình open source LLM trước đó trong nhiều tác vụ khác nhau như Falcon hay MPT

Nhìn chung phần mô hình này cũng không có nhiều điều để nói. Phần lớn thời gian của paper này viết về cách huấn luyện cũng như đảm bảo độ án toàn của mô hình hình hơn là nói về kiến trúc. Thế mới biết là kiến trúc mô hình quan trọng nhưng các kĩ thuật về xử lý dữ liệu và phương pháp huân luyện còn quan trọng hơn gấp nhiều lần.

Quá trình huấn luyện ở đây có thể coi là linh hồn của việc tạo ra các mô hình ngôn ngữ lớn. Tổng quan của quá trình này có thể tóm gọn lại trong 3 bước:

- **Pretraining:** Huấn luyện một foundation model sử dụng các nguồn dữ liệu online có sẵn với kĩ thuật training sử dụng self supervised learning như các mô hình Transformer gốc.
- **Supervised Finetuning**: Tạo ra một phiên bản đầu tiên của LLaMa-2-Chat sử dụng tập dữ liệu được gán nhãn sẵn bởi con người, tập dữ liệu này có dạng instruction bao gồm prompt và câu trả lời tương ứng.
- **RLHF:** Sau đó mô hình được tinh chỉnh liên tục dựa trên kĩ thuật RLHF thông qua hai thuật toán là PPO và Rejection Sampling. Trong quá trình RLHF, mô hình tính toán toán reward được cập nhật liên tục song song với mô hình Chat để đảm bảo rằng hai mô hình này có distribution của dữ liệu giống nhau.



Hình 3. Quá trình huấn luyện mô hình LLaMA-2

Tiếp theo chúng ta sẽ đi vào từng phần chính trong quá trình huấn luyện trên.

Pretraining:

Huấn luyện mô hình transformer trên một tập dữ liệu rất lớn sử dụng các kĩ thuật self-supervised learning. Nói đến kĩ thuật này thì nó là một kĩ thuật được sử dụng phổ biến giúp mô hình có thể học được từ nội tại dataset thông qua các phép biến đổi trực tiếp trong dữ liệu. Điểm mấu chốt ở đây, là họ sử dụng thêm khoảng 40% dữ liệu kết hợp với các nguồn dữ liệu có sẵn và họ dành rất nhiều effort để lọc các thông tin cá nhân có thể gây độc hại cho mô hình. Nên có thể coi đây là một tập dữ liệu khá sạch. Tập dữ liệu này khoảng 2 trillion tức 2000 tỉ tokens.

Quá trình pretraining sử dụng kiến trúc Transformer, chuẩn hóa trước bằng RMSNorm:

$$\text{rmsnorm}(x_i) = \frac{x_i - \text{mean}(X)}{\text{std}(X)}$$

Trong dó, $X=\{x1,x2,...,xn\}$ với n mẫu dữ liệu và

Tính giá trị trung bình (mean):

$$\text{mean}(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Tính độ lệch chuẩn (standard deviation):

$$\text{std}(X) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \text{mean}(X))^2}$$

Hàm kích hoạt SwiGLu là một biến thể của hàm kích hoạt Swish, một hàm kích hoạt thường được sử dụng trong mô hình mạng nơ-ron. SwiGLu kết hợp các phần tử của hàm Swish và ReLU (Rectified Linear Unit) để tạo ra một hàm kích hoạt mới.

$$\text{SwiGLu}(x) = \text{ReLU}(x) \times \sigma(\beta x)$$

Trong đó:

x là đầu vào.
σ là hàm sigmoid, được tính bằng công thức $\sigma(x) = \frac{1}{1+e^{-x}}$.
β là một tham số cân chỉnh.
$\mathrm{ReLU}(x)$ là hàm ReLU, được định nghĩa bởi $\mathrm{ReLU}(x) = \max(0, x)$.

kết hợp với việc sử dụng Positional Embedding.

Họ huấn luyện mô hình Transformer sử dụng thuật toán AdamW (xem chi tiết trong [3]), sử dụng learning rate scheduler với warmup 2000 bước đầu tiên. Điểm thú vị là sau khi train với 2000 tỉ token thì training loss vẫn chưa có dấu hiệu bão hoà, tức là nếu tiếp tục bổ sung thêm dữ liệu và thời gian training thì có thể chúng ta sẽ thu thập được mô hình tốt hơn. Tuy nhiên rằng với phiên bản hiện tại của LLaMa đã đủ out-perform tất cả các open-source model khác trên các chỉ số benchmark khác nhau. Và cũng có kết quả khá cạnh tranh với các closed source LLM như ChatGPT nhưng có vẻ vẫn kém khá xa so với GPT-4.

Đối với việc sử dụng mô hình Tokenizer được áp dụng như trong Llama 1 với việc sử dụng thuật toán bytepair encoding (BPE): chia các ký tự thành các số riêng lẻ và sử dụng byte để phân tách ký tự UTF 8.

Byte Pair Encoding Data Compression Example

aaabdaaabac

aaabdaaabac Replace Z = aa

ZabdZabac Replace Y = ab

ZYdZYac Replace X = ZY

Fine-tuning

Chất lượng dữ liệu là yếu tố quan trọng nhất trong việc fine-tuning mô hình. Tuy nhiên, việc sử dụng hàng triệu dữ liệu third-party SFT từ nhiều nguồn khác nhau có thể gặp phải vấn đề về tính đa dạng và chất lượng. Thay vào đó, tác giả tập trung vào việc sử dụng một tập nhỏ hơn nhưng có chất lượng dữ liệu cao hơn từ đội ngũ làm dữ liệu của mình. Kết quả cho thấy rằng chỉ cần khoảng 10000 mẫu dữ liệu để đạt được hiệu quả tốt. Sau khi có mô hình pretrained, tác giả tiến hành fine-tuning sang dữ liệu dạng chat. Họ sử dụng instruction tuning với các dữ liệu public để làm mồi cho mô hình. Quá trình fine-tuning được thực hiện với batchsize 64 và độ dài chuỗi là 4096. Để đảm bảo độ dài chuỗi của mô

hình được điền đầy đủ, tất cả các câu hỏi và câu trả lời từ tập huấn luyện được ghép lại với nhau bằng một token đặc biệt để phân tách phần câu hỏi và phần câu trả lời. Việc backpropagation chỉ được thực hiện trên các token trong phần câu trả lời. Mô hình được fine-tune trong 2 epochs.

RLHF (Raintforcement Learning with Human Feedback)

Meta đã chỉ ra cho chúng ta cách mà họ dùng kĩ thuật RLHF để cải thiện chất lượng của mô hình như thế nào. Nó không chỉ là những khái niệm hay các lý thuyết mơ hồ như trong các report khác của GPT-4 mà nó rất chi tiết. Mục tiêu của huấn luyện LLM là để có thể alignment hay còn gọi là điều chỉnh mô hình cho phù hợp với sở thích và hành vi của con người. Để thực hiện căn chỉnh này người ta đề xuất ra kĩ thuật RLHF. Trong RLHF chúng ta cần quan tâm đến hai yếu tố:

- **Reinforcement Learning:** Trước hết nói về Học tăng cường RL: thì là kĩ thuật của học máy giúp các huấn luyện ra các mô hình để tối đa hoá một reward nào đó. Trong trường hợp của LLM thì reward ở đây có thể hiểu chính là cái cảm xúc của con người, họ thấy hay, họ thấy wow, họ thấy hữu ích, họ thấy tự nhiên với một câu trả lời của chatbot.

- **Human feedback:** chính là cái đánh giá của con người trên những kết quả trả ra của mô hình, đánh giá này sẽ được lượng hoá bằng một số điểm cụ thể. Các điểm đánh giá này được sử dụng để huấn luyện mô hình tính toán phần thưởng gọi là Reward Model.

Reward Model:

Mục tiêu đào tạo (training objectives): chuyển dữ liệu sở thích con người sang nhãn xếp hạng nhị phân:

$$\mathcal{L}_{\text{ranking}} = -\log(\sigma(r_\theta(x, y_c) - r_\theta(x, y_r)))$$

ở đó $r_\theta(x, y)$ là đầu ra của lời nhắc x và phản hồi y với trọng số mô hình θ. Với y_c là phản hồi ưa thích và y_r là phản hồi bị từ chối

Khi phân tách theo thang điểm sẽ hữu ích trong quá trình đào tạo tức là gán điểm số khác biệt cho các mẫu đặc biệt. Đồng thời ký quỹ vào khoản lỗ:

$$\mathcal{L}_{\text{ranking}} = -\log(\sigma(r_\theta(x, y_c) - r_\theta(x, y_r) - m(r)))$$

Trong đó, m(r) là biên độ trùm rời rạc xếp hạng ưu tiên.

Khi mức xếp hạng lớn sẽ đưa ra phản hồi khác biệt còn ngược lại sẽ đưa ra phản hồi tương tự.

Sau khi có một reward model, nó cũng là một mạng nơ ron thì người ta sử dụng các chiến lược huấn luyện RLHF phổ biến để huấn luyện

- **Rejection Sampling:** lấy k đầu ra và chọn ứng cử viên tốt nhất cho mô hình Reward.
- **Proximal Policy Optimization (PPO):** sử dụng reward model để ước tính cho hàm reward thật sự và mô hình ngôn ngữ được pretrain làm chính sách tối ưu. Ở đây cần tối ưu hàm:

$$\arg\max_{\pi} \mathbb{E}_{p\sim\mathcal{D},g\sim\pi}[R(g \mid p)]$$

Cải thiện chính sách bằng cách lấy mẫu lời nhắc p từ tập D và tạo g từ chính sách π đồng thời sử dụng PPO và hàm mất mát để đạt được mục tiêu.

Hàm sử dụng cuối:

$$R(g \mid p) = \tilde{R}_c(g \mid p) - \beta D_{KL}(\pi_\theta(g \mid p) \parallel \pi_0(g \mid p))$$

Với πo là chính sách ban đầu còn R_c là sự kết hợp của R_s (safety) và R_h (helpfulness)

$$R_c(g \mid p) = \begin{cases} R_s(g \mid p) & \text{if IS\_SAFETY}(p) \text{ or } R_s(g \mid p) < 0.15 \\ R_h(g \mid p) & \text{otherwise} \end{cases}$$
$$\tilde{R}_c(g \mid p) = \text{WHITEN}(\text{LOGIT}(R_c(g \mid p)))$$

Các mô hình đều sử dụng trình tối ưu hóa AdamW.

Trong quá tình tuning thì reward model sẽ cập nhật liên tục với các dữ liệu mới. RLHF giúp cho mô hình ngày càng gần hơn với phân phối thực tế, hình sau có thể cho thấy sự chuyển dịch phân phối từ mô hình được huấn luyện với Superivsed Fine-tuning sang phân phối của RLHF. Có thể thấy phân phối RLHF càng gần với con người hơn.

Tiếp theo là về cách thức huấn luyện của Reward Modeling. Như đã nói ở phía trên, việc training một reward model tốt có thể coi là yếu tố then chốt dẫn đến thành bại của kĩ thuật

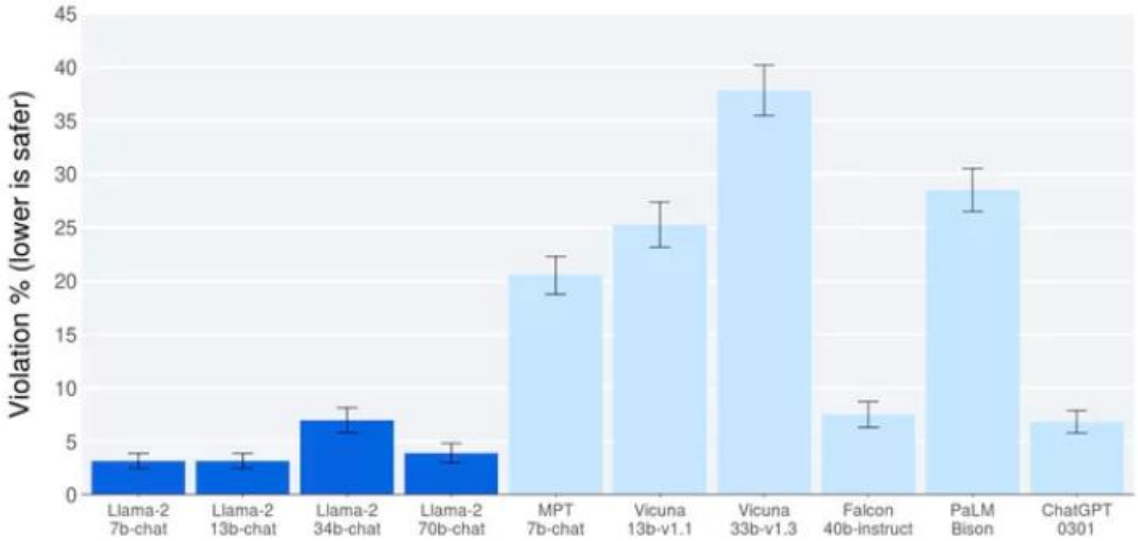
RLHF. Trong paper này tác giả trình bày chi tiết về xây dựng reward model. Họ có hai reward model riêng biệt:

- Sử dụng riêng biệt hai reward model cho hai khía cạnh khác nhau là độ an toàn **safety** và độ hữu dụng **helpfulness**
- Sử dụng scaling law để tính toán số lượng dữ liệu và tài nguyên càn thiết cho huấn luyện reward model.

Để làm rõ hơn về hai mô hình này, trong paper có nói rằng **reward model** được chia làm hai loại, một loại tối ưu cho tính an toàn **safety** và một loại được otois ưu cho tính hữu ích **helpfulness**. Cả hai mô hình này đều được base trên mô hình chat (tức là LLaMa-2-CHAT) chỉ khác mỗi một điều là thay thế các head của mô hình ngôn ngữ (next-token prediction) thành regression head để đầu ra là scalar. Về lý do tại sao lại sử dụng chung một base model với Chat model thì tác giả của bài báo có giải thích răng để cho *In short, the reward model "knows" what the chat model knows* tức là hai mô hình này share cùng một bộ não, tránh việc suy luận nhập nhằng, không tương ứng. Vậy nên **họ sử dụng các checkpoint gần nhất của chatmodel để làm base cho reward model**.

4. Tính an toàn, chi phí của LLaMA-2

Đầu tiên chúng ta sẽ nói về tính **an toàn** của mô hình. Hơn một nửa paper nói về cách đảm bảo tính an toàn cho mô hình. Khía cạnh an toàn của mô hình và báo cáo này là sự tiến bộ lớn nhất so với các mô hình nguồn mở có sẵn. Có thể thấy ngay hình bên dưới thì mô hình cho độ an toàn tốt hơ rất nhiều so với các mô hình Open-source và cả Close-Source khác.



Hình 5. So sánh tính an toàn của LLaMA-2 với các mô hình tương tự

Tiếp theo là về **chi phí** dành cho mô hình LLaMA-2. Chỉ tính nguyên chi phí làm dữ liệu cho LLaMa-2 người ta ước tính nó tiêu tốn của Meta đến **8 triệu đô la tương đương khoảng 200 tỉ VND**. Dưới đây là thống kê của các dataset sử dụng cho human preference data để huấn luyện reward modeling. Nó có khoảng gần 3 triệu sample gồm prompt và các câu trả lời tương ứng.

Dataset	Num. of Comparisons	Avg. # Turns per Dialogue	Avg. # Tokens per Example	Avg. # Tokens in Prompt	Avg. # Tokens in Response
Anthropic Helpful	122,387	3.0	251.5	17.7	88.4
Anthropic Harmless	43,966	3.0	152.5	15.7	46.4
OpenAI Summarize	176,625	1.0	371.1	336.0	35.1
OpenAI WebGPT	13,333	1.0	237.2	48.3	188.9
StackExchange	1,038,480	1.0	440.2	200.1	240.2
Stanford SHP	74,882	1.0	338.3	199.5	138.8
Synthetic GPT-J	33,139	1.0	123.3	13.0	110.3
Meta (Safety & Helpfulness)	1,418,091	3.9	798.5	31.4	234.1
Total	2,919,326	1.6	595.7	108.2	216.9

Hình 6. Thống kê DataSet huấn luyện Reward modeling

Còn nói về chi phí cho GPUs thì paper cũng đã nói rõ ràng con số sử dụng. Họ tiêu tốn khoảng gần 3.4 triệu giờ GPUs. Mỗi GPU là NVIDIA A100s và clusster ước tính cỡ khoảng 6000 GPUs. Chi riêng tính toán chi phí để mua dàn GPU này đã tốn đến con số hàng nghìn tỉ VND.

Cuối cùng là những tổn hại đến môi trường. Đây cũng là paper đầu tiên nói về lượng điện tiêu thụ và khí thải carbon để huấn luyện. Nó tiêu tốn cỡ 3.3 triệu giờ GPU tương đương khoảng 1.3 triệu KW điện tiêu thụ và lượng khí thải carbon là 539 tấn CO2 thải vào môi trường.

		Time (GPU hours)	Power Consumption (W)	Carbon Emitted (tCO_2eq)
Llama 2	7B	184320	400	31.22
	13B	368640	400	62.44
	34B	1038336	350	153.90
	70B	1720320	400	291.42
Total		3311616		539.00

Hình 7. Tổn hại đến môi trường LLaMA-2

III Thiết lập và đánh giá mô hình, các giá trị siêu tham số

Bài toán xây dựng chatbot dựa trên mô hình Llama 2, với việc áp dụng mô hình được lưu trữ, thiết lập trước đó và đánh giá.

Trước hết ta tải xuống bộ dữ liệu nhằm đánh giá mô hình, bộ dữ liệu này nhằm đánh giá cơ bản cho chatbot và tinh chỉnh mô hình và được lấy xuống như sau:

```
dataset_name = "vicgalle/alpaca-gpt4"
```

```
# tải tập dữ liệu
dataset = load_dataset(dataset_name, split="train")
```

Bộ dữ liệu được mô tả:

```
Dataset({
    features: ['instruction', 'input', 'output', 'text'],
    num_rows: 52002
})
```

Tiếp tục tải mô hình và cấu hình Qlora: thiết lập quantizer model sử dụng 4 bit cho việc tính toán tức là mô hình thiết lập với độ chính xác là 4 bit.

```
# Activate 4-bit precision base model loading
use_4bit = True

# Compute dtype for 4-bit base models
bnb_4bit_compute_dtype = "float16"

# Quantization type (fp4 or nf4)
bnb_4bit_quant_type = "nf4"

# Activate nested quantization for 4-bit base models (double quantization)
use_nested_quant = False
```

```
compute_dtype = getattr(torch, bnb_4bit_compute_dtype)

bnb_config = BitsAndBytesConfig(
    load_in_4bit=use_4bit,
    bnb_4bit_quant_type=bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=use_nested_quant,
)
```

Kiểm tra độ tương thích của thiết bị đối với mô hình thiết lập:

```
if compute_dtype == torch.float16 and use_4bit:
    major, _ = torch.cuda.get_device_capability()
    if major >= 8:
        print("=" * 80)
        print("Your GPU supports bfloat16: accelerate training with bf16=True")
        print("=" * 80)
```

Sau khi thiết lập được mô hình quantizer và cấu hình thiết bị phù hợp ta tiến hành tải mô hình xuống để sử dụng:

```
model_name = "NousResearch/Llama-2-7b-chat-hf"
```

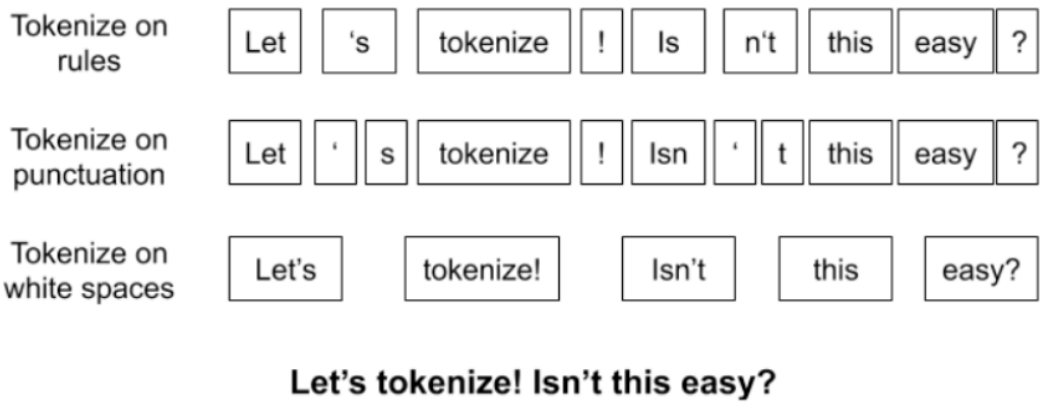
```
# Load the entire model on the GPU 0
device_map = {"": 0}
```

```
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    device_map=device_map
)
model.config.use_cache = False
model.config.pretraining_tp = 1
```

Tiếp tục tải xuống Llama tokenizer, đây là mô hình tokenizer sử dụng cho mô hình

```
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right" # Fix weird overflow issue with fp16 training
```

Có thể minh họa mô hình Tokenizer như sau:



Sau khi có mô hình và các yếu tố cần thiết ta thử nghiệm với một lời nhắc nhằm kiểm tra xem mô hình hoạt động ra sao.

Tiến hành xây dựng hàm nhằm trả về phản hồi khi ta đưa lời nhắc vào sử dụng việc sinh text tự động thông qua mô hình thiết lập với độ dài tối đa là 200

```
def llama_inference(prompt):
    input_ids = tokenizer.encode(prompt, return_tensors="pt").to('cuda')
    # sử dụng Greedy decoding và điều chỉnh độ dài tối đa
    output = model.generate(input_ids, max_length=200)
    return tokenizer.decode(output[0])
```

Ta thử nghiệm với một lời nhắc:

```
prompt = "thành phố Hồ Chí Minh?"

response = llama_inference(prompt)
print(response)
```

Với việc áp dụng hàm thiết lập trước đó ta được phản hồi:

“In Vietnamese, "thành phố" (city) is a general term that can refer to any city in Vietnam. However, when referring to the city of Ho Chi Minh City specifically, the term "thành phố Hồ Chí Minh" (Ho Chi Minh City) is used.

So, in a sentence, you might hear something like:

* "Tôi đang đến thành phố Hồ Chí Minh" (I'm going to Ho Chi Minh City)
* "Thành phố Hồ Chí Minh là thành phố lớn nhất ở Việt Nam"”

Việc áp dụng lời nhắc như trên cho thấy mô hình hiểu được tiếng việt và đồng thời có thể đưa ra một phản hồi khá là tốt và đáp ứng được độ dài yêu cầu.

Trước hết ta đánh giá mô hình với tập dữ liệu đã cho với việc sử dụng mô hình paraphrase-MiniLM-L6-v2 là một mô hình biến đổi câu: Nó ánh xạ các câu và đoạn văn vào một không gian vectơ dày đặc 384 chiều và có thể được sử dụng cho các tác vụ như phân cụm hoặc tìm kiếm ngữ nghĩa. Nhằm đưa các câu trả lời thành các vector embedding sau đó tiến hành thông qua độ đo tương tự cosin nhằm kiểm tra độ tương tự giữa phản hồi được sinh ra và phản hồi có trong dữ liệu

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{\|A\|\|B\|}$$

Trong đó:

$A \cdot B$ đại diện cho tích vô hướng (dot product) giữa hai vectơ A và B.

$\|A\|$ và $\|B\|$ đại diện cho độ dài (norm) của vectơ A và B, tương ứng.

Từ đó có thể đánh giá tương quan giữa mô hình và tập dữ liệu đào tạo

```
def evaluate_model_on_dataset(dataset, num_rows=100):
    correct_count = 0

    # Initialize sentence transformer model
    #free embeddings!
    embedder = SentenceTransformer('paraphrase-MiniLM-L6-v2')

    for idx in range(num_rows):

        question = dataset['instruction'][idx]
        #options = dataset['options'][idx]
        #correct_answer_idx = dataset['answer_idx'][idx]
        answer = dataset['output'][idx]
```

```
        # Run the model's inference on the medical question
        prompt = question
        response = llama_inference(prompt)
        # print('Response:' + response)

        # Generate embeddings for the response and correct answer
        response_embedding = embedder.encode(response, convert_to_tensor=True)
        correct_answer_embedding = embedder.encode(answer, convert_to_tensor=True)

        # Compute cosine similarity
        cosine_similarity = util.pytorch_cos_sim(response_embedding, correct_answer
        # print('the similarity is ' + str(cosine_similarity))
        is_correct = cosine_similarity >= 0.3 # Adjust the threshold as needed, >30

        if is_correct:
            correct_count += 1
        # print(f"Correct Answer: {answer}")
        # print(f"Is Model's Response Correct? {is_correct}\n")
    # Print the accuracy
    accuracy = correct_count / num_rows * 100
    print(f"Accuracy on the first {num_rows} rows: {accuracy}%")

# Evaluate the model on the first 2 rows of the training set
evaluate_model_on_dataset(dataset, num_rows=100)
```

Sau đó ta tiếp tục đánh giá mô hình khi mô hình với tokenizer tương ứng với mô hình đồng thời sử dụng thêm đoạn hội thoại lịch sử nhằm nhớ lại lịch sử tương tác trước đó:

```python
# Define a function to evaluate the model
def evaluate_model(model, tokenizer, dataset, conversation_history, num_rows=100):
    correct = 0
    total = 0
    embedder = SentenceTransformer('paraphrase-MiniLM-L6-v2')

    # Iterate through the dataset
    for idx in range(num_rows):

        question = dataset['instruction'][idx]
        distext = dataset['text'][idx]
        correct_answer_idx = dataset['output'][idx]

        # Combine the question with the text
        input_text = conversation_history + question + " " + " ".join(distext)
```

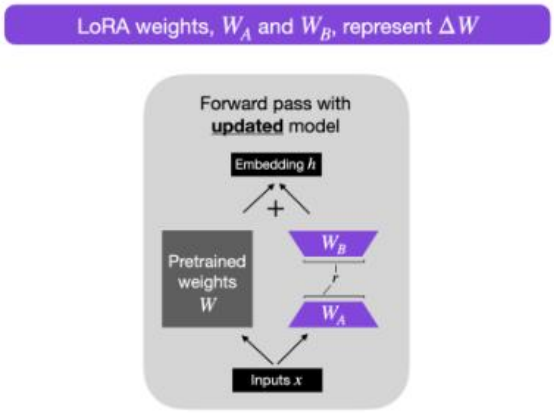
```python
        # Generate model's answer
        input_ids = tokenizer.encode(input_text, return_tensors="pt").to('cuda')
        outputs = model.generate(input_ids, max_length=200)
        generated_text = tokenizer.decode(outputs[0]).strip()

        # print(generated_text.strip())

        # Generate embeddings for the response and correct answer
        response_embedding = embedder.encode(genera
        correct_answer_embedding = embedder.encode(correct_answer_idx, convert_to_tensor=True)

        # Compute cosine similarity
        cosine_similarity = util.pytorch_cos_sim(response_embedding, correct_answer_embedding).item()
        # print('the similarity is ' + str(cosine_similarity))
        is_correct = cosine_similarity >= 0.3 # Adjust the threshold as needed, >30% threshold

        if is_correct:
            correct += 1

        # # Extract the selected option from the generated text
        # predicted_answer_idx = generated_text[0]  # Assuming the generated text starts with the selected option letter

        # # Compare with the correct answer
        # if correct_answer_idx == predicted_answer_idx:
        #     correct += 1

        total += 1

    return correct / total
```

(variable) correct_answer_idx: Any

```python
# Evaluate the model
accuracy = evaluate_model(model, tokenizer, dataset, "", 100)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Sau khi đánh giá được mô hình, ta tiến hành tinh chỉnh mô hình với bộ dữ liệu được tải xuống:

Đầu tiên cần thiết lập hàm Low-rank :




```python
input_dim = 768  # e.g., the hidden size of the pre-trained model
output_dim = 768  # e.g., the output size of the layer
rank = 8  # The rank 'r' for the low-rank adaptation

W = ... # from pretrained network with shape input_dim x output_dim

W_A = nn.Parameter(torch.empty(input_dim, rank)) # LoRA weight A
W_B = nn.Parameter(torch.empty(rank, output_dim)) # LoRA weight B

# Initialization of LoRA weights
nn.init.kaiming_uniform_(W_A, a=math.sqrt(5))
nn.init.zeros_(W_B)


def regular_forward_matmul(x, W):
    h = x @ W
return h


def lora_forward_matmul(x, W, W_A, W_B):
    h = x @ W  # regular matrix multiplication
    h += x @ W_A @ W_B # use LoRA weights
return alpha * h
```

```python
# LoRA attention dimension
lora_r = 64

# Alpha parameter for LoRA scaling
lora_alpha = 16

# Dropout probability for LoRA layers
lora_dropout = 0.1
```

```python
# Load LoRA configuration
peft_config = LoraConfig(
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    r=lora_r,
    bias="none",
    task_type="CAUSAL_LM",
)
```

Tiếp tục tiến hành tham số cho quá trình huấn luyện:

```
# Set training parameters
training_arguments = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=num_train_epochs,
    per_device_train_batch_size=per_device_train_batch_size,
    gradient_accumulation_steps=gradient_accumulation_steps,
    optim=optim,
    save_steps=1000,
    logging_steps=100,
    learning_rate=learning_rate,
    weight_decay=weight_decay,
    fp16=fp16,
    bf16=bf16,
    max_grad_norm=max_grad_norm,
    max_steps=1000,
    warmup_ratio=warmup_ratio,
    group_by_length=group_by_length,
    lr_scheduler_type=lr_scheduler_type,
    report_to="tensorboard"
)
```

Đoạn mã này đang thiết lập các tham số cho quá trình huấn luyện mô hình. Để hiểu chi tiết hơn, chúng ta sẽ xem xét từng tham số một cách cụ thể:

output_dir: Đây là đường dẫn tới thư mục nơi các kết quả của quá trình huấn luyện (như mô hình đã huấn luyện, các thông tin liên quan) sẽ được lưu.

num_train_epochs: Số lượng epoch (vòng lặp qua toàn bộ dữ liệu huấn luyện) mà huấn luyện sẽ diễn ra.

per_device_train_batch_size: Kích thước mỗi batch huấn luyện trên mỗi thiết bị.

gradient_accumulation_steps: Số lượng steps mà gradient được tính toán trước khi cập nhật mô hình.

optim: Loại tối ưu hóa được sử dụng (như Adam, SGD, ...).

save_steps: Khoảng cách giữa các lần lưu mô hình trong quá trình huấn luyện.

logging_steps: Khoảng cách giữa các lần ghi log trong quá trình huấn luyện.

learning_rate: Tốc độ học (learning rate) cho quá trình huấn luyện.

weight_decay: Hệ số giảm trọng lượng (weight decay) để tránh quá khớp.

fp16 và bf16: Sử dụng half-precision (FP16) hoặc bfloat16 để tối ưu hóa việc tính toán và bộ nhớ.

max_grad_norm: Ngưỡng giới hạn độ lớn của gradient để tránh hiện tượng gradient explosion.

max_steps: Số lượng steps tối đa trong quá trình huấn luyện.

warmup_ratio: Tỷ lệ warmup cho hàm tối ưu hóa (thường được sử dụng trong các phương pháp tối ưu hóa như AdamW).

group_by_length: Gom nhóm các mẫu huấn luyện theo độ dài để tối ưu hóa hiệu suất.

lr_scheduler_type: Loại lịch trình tốc độ học (learning rate scheduler) sử dụng trong quá trình huấn luyện.

report_to: Báo cáo kết quả huấn luyện đến TensorBoard hoặc các nơi tương tự.

Những tham số này định rõ cách quá trình huấn luyện mô hình sẽ diễn ra và cách mô hình sẽ được tối ưu hóa. Các giá trị của chúng có thể được điều chỉnh để cân nhắc giữa hiệu suất và tốc độ huấn luyện của mô hình.

```
# Set supervised fine-tuning parameters
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config,
    dataset_text_field="text",
    max_seq_length=max_seq_length,
    tokenizer=tokenizer,
    args=training_arguments,
    packing=packing,
)

# Train model
trainer.train()

# Save trained model
trainer.model.save_pretrained(new_model)
```

Tiến hành kiểm tra quá trình phát triển, tinh chỉnh và làm việc của mô hình

Cuối cùng là chạy thử nghiệm mô hình đã xây dựng:

```python
def generate_response(user_input):
    # Create a pipeline for text generation
    pipe = pipeline(task="text-generation", model=model, tokenizer=tokenizer, max_length=200)
    # Generate a response based on the user's input
    result = pipe(f"<s>[INST] {user_input} [/INST]")
    return result[0]['generated_text']
```

```python
# Ignore warnings
logging.set_verbosity(logging.CRITICAL)
while True:
    user_input = input("You: ")
    if user_input.lower() in ['exit', 'quit', 'bye']:
        print("Goodbye!")
        break
    response = generate_response(user_input)
    print("Bot:", response)
```

Thiết lập một pipeline với nhiệm vụ sinh phản hồi với việc sử dụng mô hình đã thiết lập và tokenizer đồng thời do giới hạn tài nguyên nên độ dài tối đa của đoạn văn bản sinh ra được giới hạn ở 200 từ nhằm tiết kiệm thời gian chi phí tránh tiêu hao.

Mô hình đơn giản được thiết lập bởi một vòng lặp nhận đầu vào làm nhiệm vụ và sinh ra phản hồi tương ứng. Quá trình này chỉ kết thúc khi người dùng đưa ra các từ ‘exit’, ‘quit’, ‘bye’.

IV Kết quả thực nghiệm

Đối với mô hình Llama 2 áp dụng ban đầu sử dụng trên mô hình cơ bản sinh phản hồi khi nhận được lời nhắc khá tốt và không mất quá nhiều thời gian đồng thời có thể giới hạn được độ dài văn bản sinh. Khi triển khai với một ví dụ mô hình cơ bản ban đầu cho ra kết quả như sau:

```
<s> thành phố Hồ Chí Minh?
 Unterscheidung between "thành phố" and "thành phố Hồ Chí Minh"

In Vietnamese, "thành phố" (city) is a general term that can refer to
any city in Vietnam. However, when referring to the city of Ho Chi
Minh City specifically, the term "thành phố Hồ Chí Minh" (Ho Chi Minh
City) is used.

So, in a sentence, you might hear something like:

* "Tôi đang đến thành phố Hồ Chí Minh" (I'm going to Ho Chi Minh City)
* "Thành phố Hồ Chí Minh là thành phố lớn nhất ở Việt Nam"
```

Với bộ dữ liệu đánh giá là tập dataset alpaca-Gpt4 được lưu trữ trên Huggingface với 52000 hàng và 4 cột. Dữ liệu được lấy từ các phản hồi từ mô hình GPT4 đồng thời đánh giá cho cả 2 mô hình khi trước đó cho thấy mô hình GPT4 được đánh giá cao hơn.

instruction (string)	input (string)	output (string)	text (string)
"Give three tips for staying healthy."	""	"1. Eat a balanced and nutritious diet: Make sure your meals are inclusive of a variety of fruits and vegetables, lean protein, whole grains, and healthy fats. This helps to provide your body with the essential nutrients to function at its best and can help prevent chronic diseases. 2. Engage in regular physical activity: Exercise is crucial for maintaining strong bones, muscles, and cardiovascular health. Aim for at least 150 minutes of moderate aerobic exercise or 75 minutes of vigorous exercise each week. 3. Get enough sleep: Getting enough quality sleep is crucial for physical and mental well-being. It helps to regulate mood, improve cognitive function, and supports healthy growth and immune function. Aim for 7-9 hours of sleep each night."	"Below is an instruction that describes a task. Write a response that appropriately completes the request. ### Instruction: Give three tips for staying healthy. ### Response: 1. Eat a balanced and nutritious diet: Make sure your meals are inclusive of a variety of fruits and vegetables, lean protein, whole grains, and healthy fats. This helps to provide your body with the essential nutrients to function at its best and can help prevent chronic diseases. 2. Engage in regular physical activity: Exercise is crucial for maintaining strong bones, muscles, and cardiovascular health. Aim for at least 150 minutes of moderate aerobic exercise or 75 minutes of vigorous exercise each week. 3. Get enough sleep: Getting enough quality sleep is crucial for physical and mental well-being. It helps to regulate mood, improve cognitive function, and supports healthy growth and immune function. Aim for 7-9 hours of sleep each night."
"What are the three primary colors?"	""	"The three primary colors are red, blue, and yellow. These colors are called…	"Below is an instruction that describes a task. Write a response that appropriatel…
"Describe the structure of an atom."	""	"An atom is the basic building block of all matter and is made up of three type…	"Below is an instruction that describes a task. Write a response that appropriatel…
"How can we reduce air pollution?"	""	"There are several ways to reduce air pollution, including: 1. Reduce energy…	"Below is an instruction that describes a task. Write a response that appropriatel…

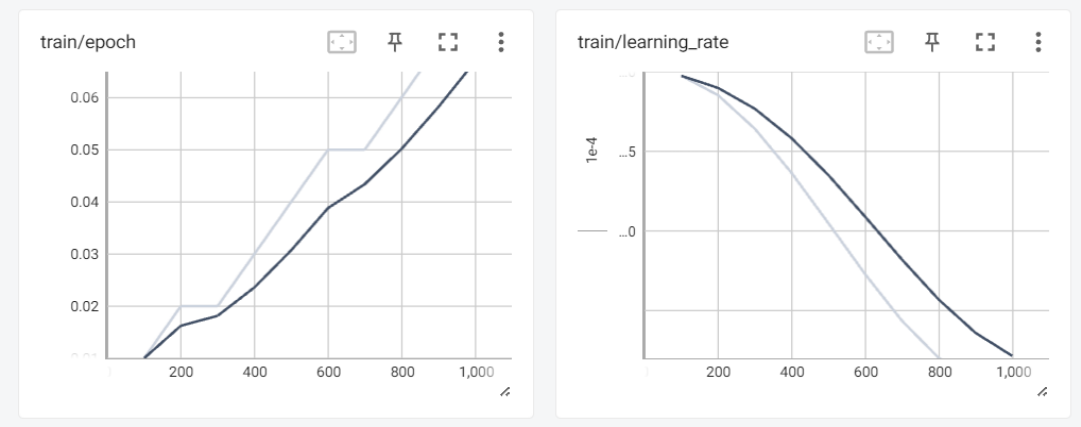
Kết quả đánh giá mô hình với một phần bộ dữ liệu trên ta được độ chính xác (accuracy) là 74%.

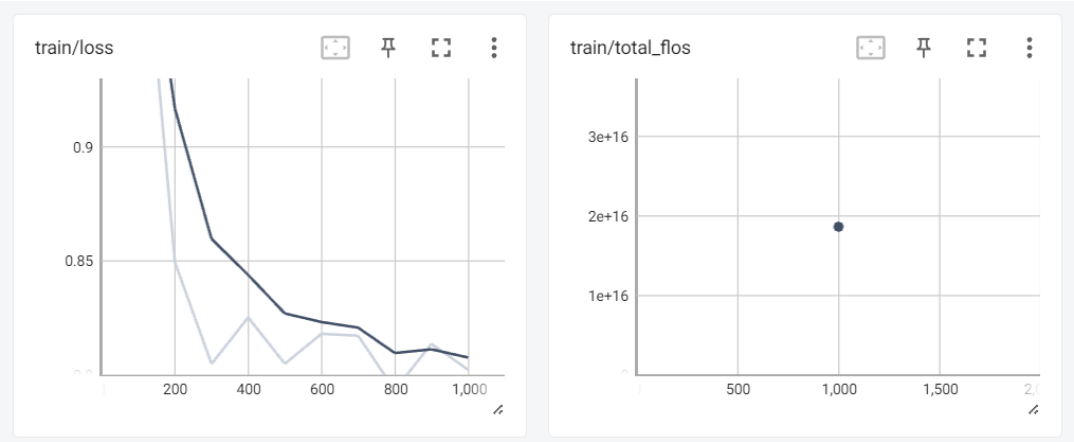
Tương tự ta đánh giá so với mô hình khi sử dụng thêm tokenizer cùng với lịch sử đoạn hội thoại ta được độ chính xác là 67%

Sau đó tiến hành tinh trỉnh mô hình và mất mát dữ liệu trong quá trình đào tạo được kết quả (áp dụng với một phần của tập dữ liệu):

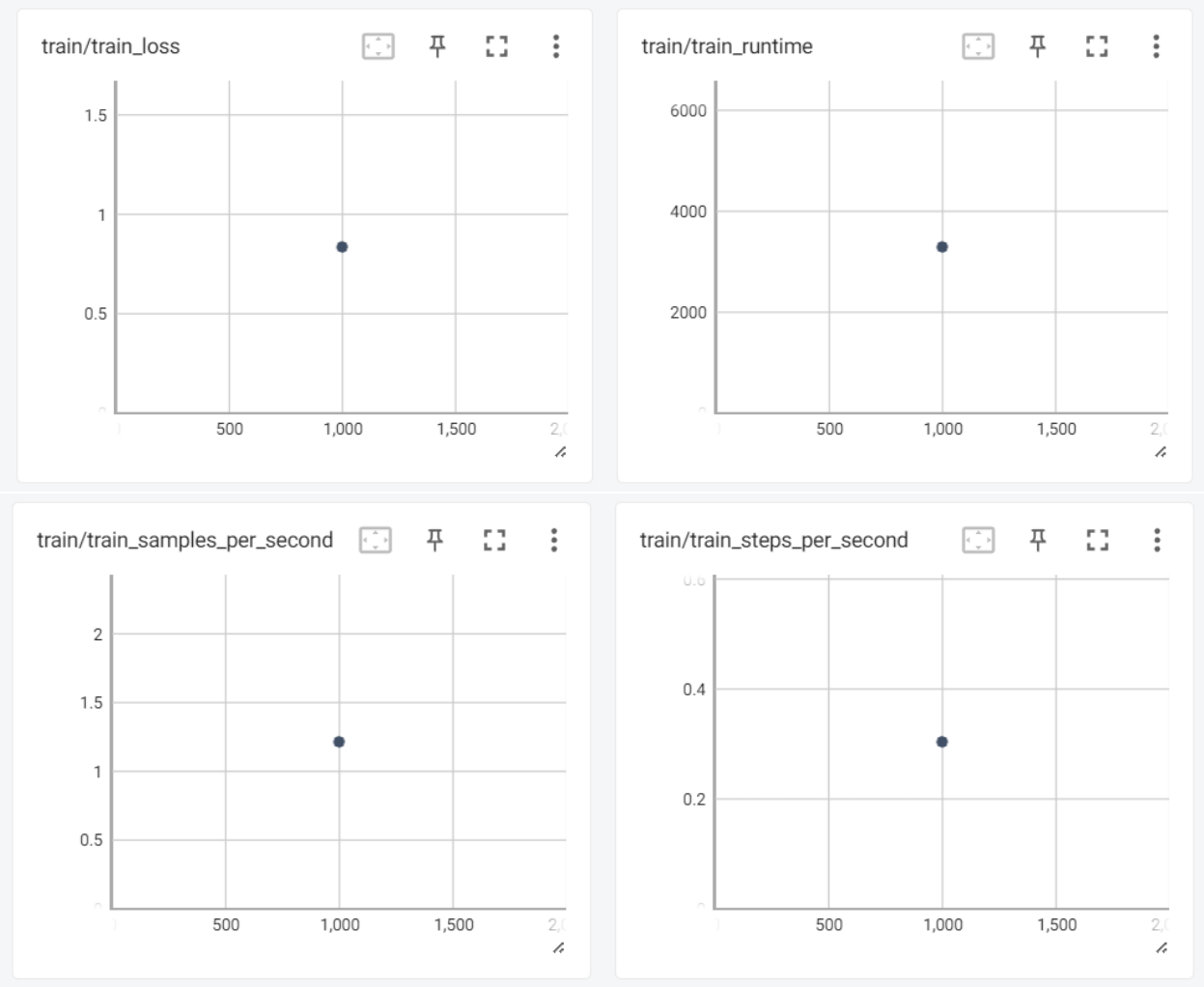
Step	Training Loss
100	1.028900
200	0.849500
300	0.804900
400	0.825300
500	0.805000
600	0.818100
700	0.817300
800	0.793500
900	0.813700
1000	0.802400

Trong quá trình đào tạo ta thu được kết quả được biểu diễn trực quan:





Tương tự một số kết quả như thời gian đào tạo từng mẫu hay từng batch có kết quả không thay đổi nhiều:



Cuối cùng với mô hình đã xây dựng và trải qua các quá trình đào tạo hay tinh chỉnh, ta tiến hành thử nghiệm và đạt được kết quả khá tích cực khi phản hồi sinh ra đã đáp ứng tốt những lời nhắc đưa vào cho mô hình:

You: How dangerous is cancer?
Bot: <s>[INST] How dangerous is cancer? [/INST] Cancer is a serious and potentially life-threatening disease. Everybody has some risk of developing cancer, and the risk increases with age. According to the American Cancer Society, in 2020, an estimated 18.1 million new cases of cancer were diagnosed worldwide, and 9.6 million cancer deaths occurred.

The danger of cancer depends on several factors, including the type of cancer, the stage at which it is diagnosed, and the effectiveness of the treatment. Some types of cancer, such as breast cancer, have a high survival rate when detected early, while others, such as pancreatic cancer, have a lower survival rate.

Some of the dangers of cancer include:

1. Cancer can spread to other parts of the body: Cancer cells can break away from the primary tumor and travel through the

You: Where is Vietnam?
Bot: <s>[INST] Where is Vietnam? [/INST] Vietnam is a country located in Southeast Asia. everybody. It is bordered by China to the north, Laos and Cambodia to the west, and Malaysia to the southwest. Its coastline faces the South China Sea to the east and the Gulf of Thailand to the south.

Vietnam is a long, thin country, stretching for over 1,000 miles (1,600 km) along the coast of the South China Sea. It is divided into three main regions: the Red River Delta in the north, the central highlands, and the Mekong Delta in the south. The capital city of Vietnam is Hanoi, and the largest city is Ho Chi Minh City.

Vietnam has a diverse landscape, with mountains, forests, and beaches. The country is home to many different ethnic groups, and its

You: Can you describe Albert Einstein's theory of relativity?
Bot: <s>[INST] Can you describe Albert Einstein's theory of relativity? [/INST] Albert Einstein's theory of relativity is a fundamental concept in modern physics that describes how space and time are intertwined. Nobody can move faster than the speed of light, and time can appear to slow down or speed up depending on how fast you are moving.

Einstein's theory of relativity is based on two main postulates:

1. The laws of physics are the same for all observers in uniform motion relative to one another.
2. The speed of light is always constant, regardless of the motion of the observer or the source of light.

From these two postulates, Einstein derived the theory of special relativity, which challenged many long-held assumptions about space and time. Some of the key implications of special relativity include:

1. Time dilation: Time appears to

You: bye
Bot: Goodbye!

Mô hình cho thấy có thể hiểu được các nhiệm vụ đưa vào và đồng thời sinh ra các đoạn mã tương đối tốt mặc dù có hạn chế về độ dài tốt đa cho phản hồi sinh ra nhưng nhìn chung kết quả khá là tích cực.

V Kết luận

Bài toán được xây dựng dựa trên mô hình Llama-2 phần nào đã được giải quyết khi có thể đưa ra các kết quả khá khả quan. Tuy nhiên mô hình vẫn còn sinh khá chậm và bị giới hạn bởi tài nguyên. Do đó để mô hình sinh nhanh hơn và đáp ứng được nhu cầu thiết yếu tốt hơn ta cần sử dụng thiết bị tốt hơn và cao cấp hơn đồng thời đáp ứng được vấn đề mở rộng mô hình.

Quá trình tinh chỉnh với tập dữ liệu không giúp được nhiều cho mô hình do thiết bị hạn chế nên việc đào tạo hay tinh chỉnh mô hình với tập dữ liệu khổng lồ là không khả thi. Tuy nhiên, việc tinh chỉnh mô hình sẽ làm mô hình sạch hơn và tối ưu hơn.

Mô hình có thể áp dụng cho nhiều bài toán khác nhau tuy nhiên đối với những người dùng cá nhân thì nên thiết lập mô hình với một bộ dữ liệu và một lĩnh vực cụ thể nào đó thì có thể tối ưu hóa dữ liệu tốt nhất nhằm đáp ứng cao nhu cầu cho từng ngành.

Hướng tiếp cận bài toán dựa trên việc tinh chỉnh như trên có thể phát triển thêm việc sử dụng con người cho các phản hồi đồng thời có thể dùng mô hình lớn hơn và tài nguyên đáp ứng được cho việc cài đặt. Cần phát triển mô hình thành ứng dụng sử dụng cho các nền tảng và tiếp cận tới người dùng và đáp ứng được nhu cầu của con người cũng như các bài toán đặt ra.

VI Tài liệu tham khảo

[1] https://arxiv.org/pdf/2307.09288.pdf

[2] https://arxiv.org/abs/2305.13245

[3] https://arxiv.org/pdf/1706.08500.pdf

[4] https://d2l.aivivn.com/chapter_attention-mechanisms/transformer_vn.html

[5] https://github.com/llSourcell/Doctor-Dignity