AI VIET NAM – COURSE 2024

Hiểu về LLMs: Tổng quan toàn diện từ việc Huấn luyện đến việc Suy luận

Ngày 30 tháng 3 năm 2024

Ngày công bố:	6/1/2024
Tác giả:	Yiheng Liu, Hao He, Tianle Han, Xu Zhang, Mengyuan Liu, Jiaming Tian, Yutong Zhang, Jiaqi Wang, Xiaohui Gao, Tianyang Zhong, Yi Pan, Shaochen Xu, Zihao Wu, Zhengliang Liu, Xin Zhang, Shu Zhang, Xintao Hu, Tuo Zhang, Ning Qiang, Tianming Liu and Bao Ge
Nguồn:	arXiv
Nguồn dữ liệu (nếu có):	
Từ khóa:	Large Language Models, Training, Inference, Survey
Người tóm tắt:	Hoàng Văn Nhân

I. Tổng quan bài báo:

Dựa trên xu hướng mới nổi là huấn luyện và triển khai LLM với chi phí thấp, bài báo đã xem xét sự phát triển của các kỹ thuật đào tạo mô hình ngôn ngữ lớn và công nghệ triển khai suy luận phù hợp xu hướng đó. Các đối tượng trong từng phần thảo luận của bài báo:

- Phần 1: Lí do viết bài báo.
- Phần 2: Kiến thức nền tảng về LLM.
- Phần 3: Các khía cạnh kĩ thuật của việc huấn luyện LLM.
- Phần 4: Các công nghệ liên quan đến việc suy luận và triển khai LLM.
- Phần 5: Sử dụng LLM.
- Phần 6: Các hướng phát triển trong tương lai và ý nghĩa của chúng.

Bài tóm tắt dưới đây sẽ đi nhanh phần tổng quan 1 và 2 để đến với nội dung chuyên sâu thuộc phần 3 của bài báo.

II. Tóm tắt chi tiết bài báo

1. Tầm quan trọng của việc nghiên cứu cách thức huấn luyện, suy luận hiệu quả cho LLM. Một số kiến thức, khái niệm cần có liên quan đến LLM.

1.1. Tầm quan trọng:

Pre-trained language models (PLM) hay mô hình đào tạo sẵn là một trong những phương pháp chính, phổ biến hiện nay trong lĩnh vực xử lí ngôn ngữ tự nhiên. Những mô hình này thường được huấn luyện trên một tập dữ liệu tổng quát lớn và sau đó sẽ được tinh chỉnh cho một nhiệm vụ cụ thể.

PLM với kích thước tham số lớn hơn đáng kể (6-10 tỷ tham số) và dữ liệu huấn luyện rộng lớn được gọi là Mô hình ngôn ngữ lớn (LLM). Việc có hiểu biết để tự phát triển LLM có thể đóng vai trò thay

thế cho ChatGPT khi mà nó đã không còn là mã nguồn mở, và việc phát triển LLM dành riêng cho từng lĩnh vực đã trở nên rất cần thiết. Và một phần quan trọng trong việc nghiên cứu đó là nghiên cứu cách thức huấn luyện, suy luận hiệu quả cho LLM.

1.2. Kiến thức nền tảng về LLM:

a) Transformer (Bộ chuyển đổi):

Transformer là một mô hình học sâu dựa trên cơ chế chú ý để xử lý dữ liệu chuỗi có thể giải quyết hiệu quả các vấn đề xử lý ngôn ngữ tự nhiên phức tạp. Do tính phù hợp với tính toán song song và độ phức tạp của mô hình, Transformer vượt trội hơn các mạng thần kinh hồi quy (RNN) phổ biến trước đây về độ chính xác và hiệu suất.

Transformer bao gồm Bộ mã hóa (Encoder) và Bộ giải mã (Decoder) cùng cơ chế Chú ý tự thân.

****Cơ chế tự chú ý (Self-Attention):***

Self-Attention là cơ chế giúp Transformers "hiểu" được sự liên quan giữa các từ trong một câu. Nó là một biến thể của cơ chế chú ý, làm giảm sự phụ thuộc vào thông tin bên ngoài và vượt trội trong việc nắm bắt các mối tương quan bên trong trong dữ liệu và giữa các đặc trưng.

Ở trong kiến trúc tổng thể module Multi-head Attention (bản chất là Self-Attention) có 3 mũi tên, đó là 3 vectors Querys (Q) – từ đang được quan tâm, Keys (K) – tất cả các từ trong câu, và Values (V) – lưu trữ thông tin liên quan đến mỗi từ. Từ 3 vectors này ta sẽ tính vector attention cho một từ theo công thức:

$$Attention(Q,\ K,\ V) = softmax\left(\frac{QK^T}{\sqrt{Dimension\ of\ vector\ K}}\right)V$$

Multi-head Attention là cơ chế được mở rộng thêm so với cơ chế Self-Attention bằng cách thực hiện nó nhiều lần một cách song song. Bằng cách này, mô hình có thể nắm bắt cả phần phụ thuộc cục bộ và toàn cục, cho phép hiểu biết toàn diện hơn về chuỗi đầu vào. Sự song song hóa này cũng nâng cao khả năng của mô hình trong việc nắm bắt các mối quan hệ phức tạp giữa các từ.

***Encoder (Bộ mã hóa):**

Module encoder của Transformer bao gồm nhiều lớp con giống hệt nhau.

Cơ chế Multi-head Attention tính toán mức độ quan trọng giữa các vị trí khác nhau trong chuỗi đầu vào để nắm bắt mối liên hệ giữa chúng. Sau đó, mạng thần kinh truyền thẳng (chuyển tiếp) được sử dụng để xử lý và trích xuất thêm các đặc trưng từ đầu ra của cơ chế Multi-head Attention.

Mô-đun encoder dần dần trích xuất các đặc trưng của chuỗi đầu vào thông qua việc xếp chồng nhiều lớp như vậy và chuyển kết quả mã hóa cuối cùng đến mô-đun giải mã để giải mã.

***Decoder (Bộ giải mã):**

Decoder còn có thêm một cơ chế bổ sung là Masked Multi-head Attention (Encoder-Decoder Attention) so với encoder. Quá trình giải mã về cơ bản là giống với quá trình mã hóa, chỉ khác là Decoder giải mã từng từ một và input của Decoder bị masked – nó chỉ được giải mã dựa trên những từ trước đó, không được phép truy cập đến phần thông tin mà nó cần giải mã lúc sau.

***Positional Embedding:**

Khi mỗi thành phần (từ, token) trong một câu đi qua ngăn xếp Bộ mã hóa/Bộ giải mã của Transformer, bản thân mô hình sẽ không có bất kỳ ý nghĩa nào về vị trí/thứ tự cho mỗi token. Kĩ thuật cho phép mô hình có thêm thông tin về vị trí của từng token trong câu nhằm kết hợp tuần tự của các token vào biểu diễn đầu vào được gọi là Positional Embedding.

Trong mô hình Transformer, công thức cốt lõi của Positional Embedding có thể được biểu diễn dưới dạng:

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

Trong đó, PE đại diện cho ma trận positional embedding, pos là vị trí của token trong câu, i là chỉ phần tử thứ i trong embedding và d_{model} biểu diễn cho số chiều của lớp ẩn trong mô hình Transformer.

Hai phương thức mã hóa vị trí phổ biến thường được sử dụng trong Transformer là: Mã hóa vị trí tuyệt đối và Mã hóa vị trí tương đối, mỗi phương pháp cung cấp thông tin vị trí theo cách riêng biệt để mô hình có thể xử lý dữ liệu tuần tự một cách hiệu quả.

b) Prompt learning (Học dựa trên lời nhắc):

Prompt learning là một phương pháp học máy, được xây dựng dựa trên các mô hình ngôn ngữ được đào tạo trước và hướng dẫn mô hình thực hiện các nhiệm vụ khác nhau thông qua thiết kế các câu lệnh nhắc, mang lại sự linh hoạt cao hơn cho việc tùy chỉnh các ứng dụng của mô hình.

***Các thành phần và quy trình cơ bản của Prompt learning:**

Prompt learning bao gồm các thành phần cơ bản sau:

- Mẫu Prompt: Là phần chính của prompt, có hai loại mẫu thông dụng:
  - Mẫu "điền vào chỗ trống": loại mẫu chọn một hoặc nhiều vị trí trong văn bản và thể hiện chúng bằng các thẻ [MASK], dùng để nhắc mô hình điền các từ tương ứng.
  - Mẫu "tạo ra dựa trên tiền tố": Việc tạo mẫu dựa trên tiền tố bao gồm việc thêm một tiền tố cụ thể trước một câu để hướng dẫn mô hình tạo văn bản phù hợp.

  Việc lựa chọn mẫu đóng vai trò rất quan trọng trong Prompt learning. Các cách chọn mẫu:
  - Tạo mẫu nhân tạo: là phương pháp trực quan nhất và có hiệu suất tốt trong các ứng dụng thực tế. Tuy nhiên, các mẫu được xây dựng nhân tạo cũng có một số nhược điểm: cần có kiến thức trước khi thiết kế và có thể mắc lỗi.
  - Có hai loại mẫu tạo theo kiểu tự động: mẫu lời nhắc rời rạc và lời nhắc liên tục.

  Có thể sử dụng nhiều mẫu để cải thiện hiệu suất mô hình.
- Ánh xạ câu trả lời: là quá trình chuyển đổi các nhãn (label) của tác vụ thành các từ vựng phù hợp với ngữ cảnh, sau đó chọn đáp án có khả năng cao nhất làm kết quả dự đoán.
- Mô hình ngôn ngữ đã qua huấn luyện (PLMs): Chọn một mô hình ngôn ngữ đã được huấn luyện trước phù hợp để sử dụng làm bộ mã hóa cơ sở.

Quy trình làm việc của Prompt learning chủ yếu gồm 4 phần:

- Sử dụng PLMs làm bộ mã hóa cơ sở.
- Thêm ngữ cảnh (mẫu lời nhắc) với một vị trí [MASK].
- Ánh xạ câu trả lời: chuyển đổi các nhãn (labels) thành các từ nhãn (label words) – verbalizer.
- Phần cầu nối giữa quá trình tiền huấn luyện và tinh chỉnh mô hình (fine-tuning).

***Chiến lược học của Prompt learning:**

Dựa trên nhiệm vụ cụ thể và nhu cầu để lựa chọn chiến lược học phù hợp:

- Tiền huấn luyện rồi tinh chỉnh: là chiến lược phổ biến nhất, phù hợp với hầu hết các nhiệm vụ.
- Không cần tinh chỉnh: phù hợp với các tác vụ đơn giản, chiến lược này có thể giảm đáng kể thời gian đào tạo và mức tiêu thụ tài nguyên tính toán.
- Tinh chỉnh prompt (lời nhắc) với LM (mô hình ngôn ngữ) cố định.

- Tinh chinh LM với prompt cố định.

  Hai chiến lược học phía trên phù hợp với các tác vụ yêu cầu điều khiển chính xác hơn và có thể tối ưu hóa hiệu suất mô hình bằng cách điều chỉnh các tham số prompt hoặc tham số LM.

- Tinh chỉnh prompt kết hợp LM: kết hợp các ưu điểm của cả hai chiến lược trước, có thể cải thiện nhiều hơn nữa hiệu suất mô hình.

Tóm lại, việc sử dụng prompt thiết kế phù hợp và chiến lược học tập hiệu quả sẽ giúp cải thiện hiệu suất mô hình trên các nhiệm vụ khác nhau.

2. Các khía cạnh kĩ thuật của việc huấn luyện LLM:

Việc đào tạo LLM có thể được chia thành ba bước:

- Thu thập rồi xử lý dữ liệu.
- Quá trình tiền huấn luyện: bao gồm việc xác định kiến trúc của mô hình và các nhiệm vụ tiền huấn luyện, đồng thời sử dụng các thuật toán đào tạo song song phù hợp để hoàn thành quá trình huấn luyện.
- Bước thứ ba liên quan đến fine-tuning và căn chỉnh cho phù hợp.

2.1. Chuẩn bị dữ liệu và tiền xử lí dữ liệu:

a) Chuẩn bị tập dữ liệu:

Huấn luyện LLM yêu cầu lượng lớn dữ liệu văn bản và chất lượng của dữ liệu này ảnh hưởng đáng kể đến hiệu suất LLM. Tiền huấn luyện về ngữ liệu quy mô lớn cung cấp cho LLM sự hiểu biết cơ bản về ngôn ngữ và một số khả năng sáng tạo. Các nguồn dữ liệu cho việc tiền huấn luyện thường bao gồm văn bản web, dữ liệu hội thoại, sách và tài liệu chuyên ngành để tăng cường khả năng của LLMs trong các lĩnh vực đó. Có 5 nhóm bộ dữ liệu thường được sử dụng để huấn luyện LLM là:

- Books: Các tập dữ liệu sách như BookCorpus và Gutenberg bao gồm nhiều thể loại văn học khác nhau, từ tiểu thuyết đến lịch sử, khoa học, triết học và nhiều hơn nữa. Chúng được sử dụng rộng rãi trong việc đào tạo các mô hình ngôn ngữ lớn (LLMs) để cung cấp hiểu biết sâu rộng về ngôn ngữ qua các lĩnh vực khác nhau.
- CommonCrawl: Là một kho lưu trữ dữ liệu web crawl lớn, bao gồm hơn 250 tỷ trang web được thu thập trong 16 năm. Dữ liệu từ CommonCrawl chiếm 82% nguồn dữ liệu đào tạo cho GPT-3. Tuy nhiên, do chứa nhiều dữ liệu chất lượng thấp, việc tiền xử lý dữ liệu là cần thiết khi làm việc với CommonCrawl.
- Reddit Links: Reddit là một nền tảng truyền thông xã hội nơi người dùng có thể gửi liên kết và bài đăng, và những người khác có thể bình chọn chúng. Điều này làm cho nó trở thành một nguồn tài nguyên quý giá để tạo ra các tập dữ liệu chất lượng cao.
- Wikipedia: Một dự án bách khoa toàn thư trực tuyến mở rộng với nội dung chất lượng cao về nhiều chủ đề khác nhau. Wikipedia được sử dụng rộng rãi trong việc đào tạo nhiều LLM, đóng vai trò là nguồn tài nguyên quý giá cho việc hiểu và tạo ngôn ngữ.
- Code: Hiện tại có một số lượng hạn chế các tập dữ liệu mã nguồn mở có sẵn công khai. Các nỗ lực hiện tại chủ yếu liên quan đến việc trích xuất mã nguồn có bản quyền mở từ internet, chủ yếu từ Github và Stack Overflow.

b) Tiền xử lí dữ liệu:

Chất lượng xử lý trước dữ liệu ảnh hưởng trực tiếp đến hiệu suất và tính bảo mật của mô hình. Các bước tiền xử lý quan trọng bao gồm:

- Lọc chất lượng: Việc lọc dữ liệu chất lượng thấp thường được thực hiện bằng phương pháp dựa trên quy tắc hoặc phân loại. Các quy tắc có thể bao gồm chỉ giữ lại văn bản chứa số, loại bỏ câu chỉ gồm chữ hoa, và bỏ các tệp có tỷ lệ ký tự và từ vượt quá 0.1.

- Loại bỏ trùng lặp: Mô hình ngôn ngữ đôi khi có thể tạo ra nội dung lặp đi lặp lại do sự lặp lại cao trong dữ liệu huấn luyện. Việc loại bỏ dữ liệu trùng lặp giúp duy trì sự ổn định trong quá trình huấn luyện và cải thiện hiệu suất của LLMs.

- Làm sạch dữ liệu để bảo vệ quyền riêng tư (Privacy scrubbing): Để đảm bảo an ninh mô hình, quá trình tiền xử lý dữ liệu ngôn ngữ cần phải loại bỏ thông tin nhạy cảm, bao gồm các kĩ thuật như ẩn danh, che giấu hoặc mã hóa thông tin cá nhân, địa điểm địa lý và dữ liệu bảo mật khác.

- Lọc văn bản độc hại và thiên kiến: Loại bỏ nội dung độc hại và thiên kiến là một bước quan trọng để phát triển mô hình ngôn ngữ công bằng và không thiên kiến. Điều này đòi hỏi việc áp dụng các kỹ thuật kiểm duyệt nội dung mạnh mẽ để xác định và loại bỏ văn bản có thể duy trì định kiến có hại, ngôn ngữ xúc phạm hoặc quan điểm thiên vị.

2.2. Kiến trúc của mô hình:

Hiện tại, tất cả các LLM đều được xây dựng dựa trên kiến trúc Transformer. Thông thường, kiến trúc PLM thuộc ba loại: Chỉ bộ mã hóa, Bộ giải mã – mã hóa và Chỉ bộ giải mã.

a) Kiến trúc bộ mã hóa – giải mã:

Kiến trúc bộ mã hóa-giải mã bao gồm hai thành phần chính: Bộ mã hóa và bộ giải mã. Encoder chứa nhiều lớp Multi-Head Self-Attention, mã hóa chuỗi đầu vào. Decoder tạo ra chuỗi đích bằng cách sử dụng cross-attention qua biểu diễn đầu ra của Encoder và tạo ra chuỗi đích một cách tự động theo thứ tự.

b) Kiến trúc chỉ bộ giải mã:

LLMs với kiến trúc chỉ bộ giải mã chỉ sử dụng phần decoder của kiến trúc Transformer truyền thống. Mô hình này tạo ra token một cách tuần tự, chú ý đến các token trước đó trong chuỗi. Kiến trúc này thể hiện được tính hiệu quả trong các tác vụ khác nhau như tạo văn bản mà không cần giai đoạn mã hóa rõ ràng. Kiến trúc này có thể được phân loại thành hai loại: kiến trúc Bộ giải mã nguyên nhân (Causal Decoder Architecture) và kiến trúc Bộ giải mã tiền tố (Prefix Decoder Architecture).

- Causal Decoder Architecture: Mỗi token chỉ chú ý đến các token trước đó trong chuỗi đầu vào, sử dụng một cấu hình mặt nạ đặc biệt để thực hiện quá trình giải mã một chiều. Kiến trúc này được áp dụng trong các mô hình như GPT series.

- Prefix Decoder Architecture: Kết hợp ưu điểm của cả hai kiến trúc Encoder-decoder và Causal Decoder, cho phép sự chú ý hai chiều đối với các token tiền tố trong khi vẫn duy trì sự chú ý một chiều khi tạo ra các token tiếp theo. Kiến trúc này được sử dụng trong các mô hình như PaLM và GLM.

Cả hai kiến trúc này đều tập trung vào việc tạo ra chuỗi văn bản một cách tự động, từng token một, dựa trên ngữ cảnh được cung cấp bởi các token trước đó.

2.3. Nhiệm vụ tiền huấn luyện:

Mô hình ngôn ngữ lớn (LLM) thường học cách biểu diễn ngôn ngữ phong phú thông qua quá trình tiền huấn luyện. Trong quá trình đào tạo trước, các mô hình này tận dụng kho dữ liệu mở rộng và trải qua quá trình đào tạo thông qua các phương pháp học tập tự giám sát. Một trong số đó là Mô hình hóa ngôn ngữ (LM), LM tiền huấn luyện thông qua việc dự đoán từ tiếp theo trong một ngữ cảnh cho trước, giúp mô hình học được cách sử dụng từ vựng, ngữ pháp, và ngữ nghĩa. Quá trình học dần dần này cho phép mô hình nắm bắt các mẫu và thông tin vốn có trong ngôn ngữ, mã hóa một lượng lớn kiến thức ngôn ngữ thành các tham số của nó. Sau khi quá trình đào tạo trước hoàn tất, các tham số

mô hình này có thể được tinh chỉnh cho các tác vụ xử lý ngôn ngữ tự nhiên khác nhau để thích ứng với các yêu cầu tác vụ cụ thể.

Ngoài mô hình hóa ngôn ngữ, có các nhiệm vụ tiền huấn luyện khác như sử dụng văn bản với một số phần được thay thế ngẫu nhiên, sau đó sử dụng phương pháp tự động hồi quy để phục hồi các phần đã thay thế.

2.4. Huấn luyện mô hình:

a) Huấn luyện song song:

Huấn luyện song song là một phương pháp huấn luyện mô hình học máy, đặc biệt là các LLM bằng cách sử dụng nhiều bộ xử lý hoặc GPU cùng một lúc để xử lý dữ liệu và tham số mô hình. Mục tiêu của parallel training là giảm thời gian cần thiết để huấn luyện mô hình và tăng khả năng mở rộng của mô hình khi xử lý dữ liệu lớn và phức tạp.

Giao tiếp tập thể (collective communication) là một khái niệm quan trọng trong huấn luyện song song. giao tiếp tập thể liên quan đến việc trao đổi dữ liệu giữa các bộ xử lý hoặc GPU trong quá trình huấn luyện mô hình. Các hoạt động giao tiếp tập thể bao gồm:

- Broadcast: Gửi dữ liệu từ một GPU đến tất cả các GPU khác.
- Reduce: Tổng hợp dữ liệu từ tất cả các GPU và gửi kết quả đến một GPU.
- All Reduce: Tổng hợp dữ liệu từ tất cả các GPU và phân phối kết quả đến tất cả các GPU.
- Reduce Scatter: Tổng hợp dữ liệu từ tất cả các GPU và phân phối các phần dữ liệu đến từng GPU.
- All Gather: Thu thập dữ liệu từ tất cả các GPU và phân phối đến tất cả các GPU.

Trong huấn luyện song song có 4 phương pháp chính:

- Data Parallel: Sử dụng một máy chủ tham số để lưu trữ các tham số của mô hình và toàn bộ batch dữ liệu. Các GPU đồng bộ hóa tham số mô hình và chia dữ liệu thành từng phần, mỗi GPU xử lý một phần dữ liệu. Sau đó, các gradient được tổng hợp và gửi trở lại máy chủ tham số.
- Model Parallel: Phân chia mô hình thành các phần nhỏ hơn và mỗi GPU sẽ xử lý một phần của mô hình. Điều này giúp xử lý các mô hình lớn không vừa với bộ nhớ của một GPU đơn lẻ.
- ZeRO: Tối ưu hóa việc sử dụng bộ nhớ và băng thông mạng bằng cách loại bỏ sự trùng lặp của dữ liệu trên các GPU, giúp mở rộng quy mô huấn luyện mà không tăng cường độ sử dụng bộ nhớ.
- Pipeline Parallel: Chia nhỏ quá trình huấn luyện thành các giai đoạn và mỗi GPU sẽ xử lý một giai đoạn cụ thể. Điều này giúp tăng hiệu suất bằng cách giảm thời gian chờ đợi giữa các GPU.

b) Huấn luyện chính xác hỗn hợp:

Huấn luyện độ chính xác hỗn hợp (Mixed Precision Training) là một kỹ thuật trong huấn nhằm tăng tốc độ tính toán và giảm bớt việc sử dụng bộ nhớ. Kỹ thuật này sử dụng cả số thực dấu phẩy động 16-bit (FP16) và 32-bit (FP32) trong quá trình huấn luyện.

FP16 có phạm vi số học nhỏ hơn và độ chính xác thấp hơn so với FP32, nhưng lại cho phép tính toán nhanh hơn. Để cải thiện tốc độ tính toán, chúng ta có thể chuyển từ FP32 sang FP16, vì số lượng tham số trong một mô hình thường nằm trong phạm vi số của FP16. Khi cập nhật tham số, việc nhân gradient với tốc độ học có thể dẫn đến mất mát do tràn dấu phẩy động nếu sử dụng FP16, vì vậy tham số cập nhật được biểu diễn dưới dạng FP32. Trong trình tối ưu hóa, lượng cập nhật được lưu dưới dạng FP32 và tích lũy nó một cách hiệu quả thông qua một tham số FP32 tạm thời, sau đó được chuyển đổi trở lại thành các tham số FP16.

c) Offloading:

Offloading là quá trình chuyển các tham số của trình tối ưu hóa từ GPU sang CPU để giảm tải tính toán, sử dụng ZeRO3 để giảm kích thước của tham số, gradient và trình tối ưu hóa xuống 1/n số lượng GPU.

d) Overlapping:

Overlapping là một kỹ thuật tối ưu hóa hiệu suất bằng cách thực hiện các phép tính một cách không đồng bộ. Điều này cho phép mô hình tiếp tục thực hiện các phép tính khác trong khi đang chờ xử lý một yêu cầu bộ nhớ, giúp tận dụng tối đa khả năng tính toán của hệ thống và tối ưu hóa quá trình lan truyền chuyển tiếp trong huấn luyện mô hình.

e) Checkpoint:

Sử dụng cơ chế checkpoint không lưu trữ tất cả kết quả trung gian trong bộ nhớ GPU mà chỉ giữ lại các điểm kiểm tra nhất định, giảm lượng bộ nhớ cần thiết và cho phép tái tính toán trong quá trình lan truyền ngược.

2.5. Fine-tuning (tinh chỉnh):

Có 3 loại kĩ thuật tinh chỉnh: tinh chỉnh có giám sát (SFT), điều chỉnh phù hợp (alignment tuning), tinh chỉnh tham số hiệu quả.

a) Tinh chỉnh có giám sát:

SFT là điều chỉnh tiếp mô hình đã được đào tạo trước, dựa trên tập dữ liệu được gắn nhãn cho một nhiệm vụ cụ thể (ví dụ: dịch thuật, trả lời câu hỏi).

Điều chỉnh hướng dẫn (Instruction tuning): Một kỹ thuật SFT phổ biến trong đó LLM được đào tạo thêm trên một bộ dữ liệu gồm các cặp "(hướng dẫn, đầu ra)" để cải thiện khả năng kiểm soát bằng cách hiểu và tuân theo hướng dẫn của con người.

b) Điều chỉnh phù hợp:

Điều chỉnh phù hợp là việc điều chỉnh để đảm bảo rằng mô hình tạo ra kết quả:

- Hữu ích: Mô hình cần cung cấp thông tin có giá trị, phục vụ tốt cho nhu cầu hoặc mục tiêu của người dùng.
- Trung thực: Mô hình phải tạo ra thông tin chính xác, không làm sai lệch sự thật, giữ được niềm tin của người dùng.
- Không gây hại: Mô hình không được tạo ra nội dung có hại, xúc phạm hoặc nguy hiểm, đảm bảo an toàn cho người dùng và xã hội.

Phương pháp: Sử dụng phương pháp Học Tăng Cường với Phản Hồi Từ Con Người (RLHF), thu thập dữ liệu phản hồi từ con người để huấn luyện một mô hình phần thưởng (RM), sử dụng trong quá trình học tăng cường để tinh chỉnh LLM.

c) Tinh chỉnh tham số hiệu quả:

Kỹ thuật tinh chỉnh hiệu quả tham số giúp giảm đáng kể chi phí tính toán và bộ nhớ, bằng cách chỉ tinh chỉnh một phần nhỏ hoặc bổ sung của các tham số mô hình. Các phương pháp phổ biến như Low-Rank Adaptation (LoRA), Prefix Tuning và P-Tuning cho phép tinh chỉnh mô hình một cách hiệu quả ngay cả trong môi trường có hạn chế về tài nguyên.

d) Tinh chỉnh an toàn:

Tăng cường an toàn và trách nhiệm của các mô hình ngôn ngữ lớn (LLMs) bằng cách tích hợp thêm kỹ thuật an toàn trong quá trình tinh chỉnh. Điều này bao gồm cả 3 kĩ thuật chính.

- Tinh chỉnh an toàn có giám sát: Dùng dữ liệu do chuyên gia tạo ra để huấn luyện LLM nhận diện và tránh nội dung nguy hiểm.
- RLHF an toàn: Khen thưởng LLM khi đưa ra phản hồi an toàn nhất, ví dụ như từ chối tạo nội dung độc hại.

- Lọc ngữ cảnh an toàn: Huấn luyện LLM dựa trên dữ liệu an toàn được tạo ra bằng cách thêm lời nhắc an toàn vào đầu lời nhắc nguy hiểm.

2.6. Đánh giá:

a) Bộ dữ liệu kiểm tra:

Quá trình đánh giá phụ thuộc rất nhiều vào các bộ dữ liệu thích hợp để đánh giá khả năng LLM. Một số bộ dữ liệu thường được sử dụng là:

- Dành cho các mô hình hình ảnh**:** ImageNet (thị giác máy tính), Open Images.
- Dành cho các mô hình dựa trên văn bản**:** GLUE, SuperGLUE, MMLU (khả năng chung), CMMLU (tiếng Trung), XTREME/XTREME-R (đa ngôn ngữ)
- Dành cho các nhiệm vụ cụ thể**:** MATH/GSM8K (toán học), HumanEval/MBPP (tạo mã), HelloSwag/PIQA/ BoolQ/v.v. (lý giải thông thường), MedQA-USMLE/MedMCQA (tri thức về y khoa).

b) Đánh giá khả năng trả lời câu hỏi mở (ODQA):

Đánh giá ODQA là quan trọng vì nó ảnh hưởng đến trải nghiệm người dùng. Các bộ dữ liệu phổ biến bao gồm SquAD và Natural Questions, với F1 và EM là chỉ số đánh giá. Tuy nhiên, đánh giá của con người vẫn cần thiết do những hạn chế trong phương pháp ghép từ.

c) Đánh giá an ninh:

LLM yêu cầu đánh giá an ninh cẩn thận để giải quyết các mối đe dọa và lỗ hổng tiềm ẩn:

- Thiên kiến tiềm ẩn: Đánh giá an ninh cần xem xét liệu mô hình có tạo ra hoặc tăng cường những định kiến như giới tính hay chủng tộc không và cách giảm bớt hoặc sửa chữa chúng.
- Bảo vệ quyền riêng tư: Cần đảm bảo bảo vệ quyền riêng tư dữ liệu người dùng hiệu quả để ngăn chặn rò rỉ và bị lạm dụng.
- Tấn công thù địch: LLM có thể dễ bị thao túng thông qua các cuộc tấn công thù địch. Giải quyết các lỗ hổng này là rất quan trọng.

d) Phương pháp đánh giá:

Nên kết hợp đánh giá tự động và thủ công để đánh giá toàn diện hiệu suất mô hình ngôn ngữ:

- Đánh giá tự động: Các chỉ số như BLEU, ROUGE và BERTScore cung cấp các phương pháp định lượng đo lường hiệu suất. Chúng hiệu quả khi phân tích dữ liệu quy mô lớn nhưng không thể hiểu hết được sự phức tạp của ngôn ngữ.
- Đánh giá thủ công: Các chuyên gia đánh giá chủ quan chất lượng đầu ra LLM. Phương pháp này có giá trị cho các nhiệm vụ cụ thể và xác định các vấn đề, lỗi tinh vi mà đánh giá tự động có thể bỏ qua, nhưng nó tốn thời gian và chủ quan.

2.7. LLM Framework:

LLM Framework cung cấp các công nghệ và phương pháp tiên tiến để xử lý số lượng lớn tham số của LLMs, giúp tối ưu hóa quá trình huấn luyện và cải thiện hiệu suất của mô hình trên các tài nguyên tính toán hạn chế.

Một số LLM Framework sử dụng công nghệ đào tạo phân tán tận dụng GPU, CPU và bộ nhớ NVMe là:

- Transformers: Thư viện mã nguồn mở của Hugging Face, cung cấp API thân thiện để tùy chỉnh các mô hình đã được huấn luyện trước sử dụng kiến trúc Transformer.

- DeepSpeed: Phát triển bởi Microsoft, thư viện này hỗ trợ công nghệ ZeRO cho việc huấn luyện LLM hiệu quả với các tính năng như phân chia trạng thái tối ưu hóa và huấn luyện chính xác hỗn hợp tùy chỉnh.
- BMTrain: Bộ công cụ của Đại học Thanh Hoa cho phép huấn luyện phân tán các mô hình lớn một cách đơn giản mà không cần tái cấu trúc mã, hỗ trợ các kỹ thuật tối ưu hóa khác nhau.
- Megatron-LM: Thư viện của NVIDIA cho việc huấn luyện các mô hình ngôn ngữ quy mô lớn, sử dụng song song hóa mô hình/dữ liệu và huấn luyện chính xác hỗn hợp để sử dụng GPU một cách hiệu quả.

3. Suy luận (Inference) với các mô hình ngôn ngữ lớn:

LLM đang ngày càng trở nên mạnh mẽ, nhưng quy mô lớn của chúng dẫn đến nhu cầu đáng kể về tài nguyên. Dưới đây là một số kỹ thuật để tối ưu hóa việc suy luận của LLM nhằm giảm chi phí tính toán và sử dụng bộ nhớ.

3.1. Nén mô hình:

- Chắt lọc kiến thức: Chuyển tải kiến thức từ mô hình cồng kềnh sang mô hình nhỏ hơn, phù hợp hơn cho việc triển khai.
- Cắt tỉa mô hình: Loại bỏ các phần dư thừa khỏi ma trận tham số của các mô hình lớn. Có 2 cách thực hiện:
  - Cắt tỉa không có cấu trúc: loại bỏ các kết nối hoặc trọng số riêng lẻ trong mạng lưới thần kinh mà không tuân theo bất kỳ mẫu cấu trúc cụ thể nào.
  - Cắt tỉa có cấu trúc: chỉ loại bỏ các mẫu cấu trúc cụ thể.
- Lượng tử hóa mô hình (gọi như vậy vì ta phải biến hóa các đối tượng vốn có thành đối tượng cơ bản nhất như trong vật lí là lượng tử): Giảm số lượng bit được sử dụng để tính toán số, giảm yêu cầu lưu trữ và tính toán. Tuy nhiên, có một sự đánh đổi giữa độ chính xác và hiệu suất. Ví dụ: từ dạng số thực sang số nguyên.
- Chia sẻ trọng số: Sử dụng cùng một bộ tham số cho nhiều phần của LLM. Điều này giúp giảm số lượng tham số cần học, tăng hiệu quả tính toán và giảm nguy cơ quá khớp.
- Phép xấp xỉ hạng thấp: Tạo ra các mô hình nhỏ gọn hơn với ít tham số hơn bằng cách phân tách các ma trận lớn thành các ma trận nhỏ hơn, cho phép suy luận hiệu quả trên các thiết bị bị hạn chế tài nguyên.

3.2. Lập lịch bộ nhớ:

Tối ưu hóa việc sử dụng bộ nhớ trong quá trình suy luận LLM bằng cách quản lý các mẫu truy cập bộ nhớ một cách hiệu quả. Điều này rất quan trọng vì các mô hình lớn thường có kiến trúc phức tạp với yêu cầu bộ nhớ đáng kể. Một ví dụ là BMInf, tận dụng các nguyên tắc bộ nhớ ảo để lên lịch hiệu quả các tham số của mỗi lớp giữa GPU và CPU.

3.3. Song song hóa:

Sử dụng nhiều đơn vị xử lý để phân phối khối lượng công việc suy luận LLM, cải thiện thông lượng tổng thể và giảm độ trễ.

- Song song hóa dữ liệu: Phân phối dữ liệu trên nhiều GPU để xử lý nhanh hơn.
- Song song hóa tensor: Tham số của mô hình được chia thành nhiều tensor và mỗi tensor được tính toán trên các đơn vị xử lý khác nhau.

- Song song hóa pipeline: Chia nhỏ tính toán thành các giai đoạn được xử lý bởi các GPU khác nhau, cho phép hỗ trợ cho các mô hình lớn hơn và nâng cao khả năng sử dụng thiết bị.

3.4. Tối ưu hóa cấu trúc:

Tập trung vào việc giảm thiểu truy cập bộ nhớ trong quá trình suy luận LLM để cải thiện tốc độ. Các kỹ thuật như FlashAttention và PagedAttention nâng cao tốc độ tính toán bằng cách sử dụng phương pháp tính toán theo khối, giảm thiểu chi phí lưu trữ liên quan đến ma trận.

3.5. Inference Framework:

Cung cấp nền tảng để triển khai và chạy LLM. Các kỹ thuật như tính toán song song, nén mô hình, lập lịch bộ nhớ và tối ưu hóa cấu trúc transformer đã được tích hợp hiệu quả vào các inference framework chính. Các framework này cung cấp các công cụ và giao diện giúp hợp lý hóa các quy trình triển khai và suy luận cho các trường hợp sử dụng khác nhau.

4. Việc sử dụng các mô hình ngôn ngữ lớn:

LLMs có thể được áp dụng trong hầu hết các lĩnh vực chuyên môn. Sau khi được huấn luyện sơ bộ và tinh chỉnh, LLMs chủ yếu được sử dụng bằng cách thiết kế các lời nhắc phù hợp cho nhiều nhiệm vụ khác nhau qua một số khả năng sau: Zero-shot, few-shot, kết hợp các lời nhắc thành một chuỗi suy nghĩ giúp cải thiện học tập về ngữ cảnh,...

Chiến lược triển khai LLM:

- Truy cập API: Sử dụng khả năng của các mô hình độc quyền, mạnh mẽ được cung cấp thông qua các API mở (ví dụ: ChatGPT).
- Triển khai LLM mã nguồn mở để sử dụng cục bộ.
- Tinh chỉnh LLM mã nguồn mở để đáp ứng các tiêu chuẩn cụ thể của lĩnh vực nhất định và sau đó triển khai chúng cục bộ cho các ứng dụng chuyên biệt.

5. Các hướng phát triển trong tương lai và ý nghĩa của chúng:

Xu hướng phát triển LLM trong tương lai:

- Mở rộng mô hình: Dự kiến LLMs sẽ tiếp tục mở rộng quy mô, nâng cao hiệu suất và khả năng học hỏi.
- Phát triển đa phương tiện: LLMs tương lai có thể xử lý không chỉ văn bản mà còn cả hình ảnh, video và âm thanh, mở rộng ứng dụng của chúng.
- Hiệu quả: Tập trung vào giảm chi phí đào tạo và suy luận thông qua các kỹ thuật như chắt lọc kiến thức, nén mô hình.
- Đào tạo cho ngành cụ thể: Tùy chỉnh LLMs cho các ngành cụ thể để hiểu rõ hơn về thuật ngữ và bối cảnh đặc thù của ngành.
- Kiến trúc mới: Khám phá các kiến trúc thay thế như RNN cho LLM là một hướng nghiên cứu hấp dẫn, có khả năng giải quyết các hạn chế như yêu cầu đầu vào kích thước cố định trong transformer.

Đối với các nhà nghiên cứu AI, sự hợp tác giữa các ngành và lĩnh vực khác nhau đang trở nên thiết yếu. Họ phải triển kỹ năng kỹ thuật để giải quyết những thách thức trong phát triển LLMs. Các vấn đề đạo đức và ảnh hưởng xã hội của LLMs cũng là những lĩnh vực quan trọng cần nghiên cứu và cải thiện liên tục.

AI VIET NAM – RESEARCH TEAM

Understanding LLMs: A Comprehensive Overview from Training to Inference

March 30, 2024

Date of publication:	6/1/2024
Authors:	Yiheng Liu, Hao He, Tianle Han, Xu Zhang, Mengyuan Liu, Jiaming Tian, Yutong Zhang, Jiaqi Wang, Xiaohui Gao, Tianyang Zhong, Yi Pan, Shaochen Xu, Zihao Wu, Zhengliang Liu, Xin Zhang, Shu Zhang, Xintao Hu, Tuo Zhang, Ning Qiang, Tianming Liu and Bao Ge
Sources:	arXiv
Data sources (if any):	
Keywords:	Large Language Models, Training, Inference, Survey
Summary by:	Hoàng Văn Nhân

I. Article Overview:

Based on the emerging trend of training and deploying Large Language Models (LLMs) at low cost, this article examines the development of techniques for training large language models and the relevant inference technologies. The discussion is organized into the following sections:

1. The motivation behind writing the article.
2. Background Knowledge about LLMs.
3. Technical Aspects of LLM Training.
4. Technologies Related to Inference and Deployment of LLMs.
5. Utilization of LLMs.
6. Future Directions and Implications.

The summary below will briefly cover Sections 1 and 2, leading to the in-depth content of Section 3 in the article.

II. Detailed Article Summary

1. Importance of Researching Effective Training and Inference Methods for LLMs. Background Knowledge about LLMs.

1.1. Importance of Researching Effective Training and Inference Methods for LLMs:

Pre-trained Language Models (PLMs) are widely used in natural language processing. These models are trained on large general datasets and then fine-tuned for specific tasks.

PLM with significantly larger parameter sizes (6-10 billion parameters) and extensive training data is called Large Language Models (LLM). Having the knowledge to develop LLMs independently can play a role in replacing ChatGPT when it is no longer open source, and developing LLMs for specific

domains has become very necessary. And an important part of that research is researching effective training and reasoning methods for LLMs.

1.2. Background Knowledge about LLMs:

a) Transformer:

Transformers are deep learning models based on attention mechanisms. They effectively handle complex natural language processing tasks. Their parallel computation capability and model complexity make Transformers superior to previously popular recurrent neural networks (RNNs) in terms of accuracy and performance.

Transformer consists of an Encoder and a Decoder along with the Self-Attention mechanism.

****Self-Attention Mechanism:***

Self-Attention helps Transformers understand relationships between words in a sentence. It is a variant of the attention mechanism that reduces the dependence on external information and excels in capturing internal correlations within data and across features.

In the overall architecture of the Multi-head Attention module (which is essentially Self-Attention), there are 3 arrows, which are 3 vectors: Queries (Q) - the word being focused on, Keys (K) - all the words in the sentence, and Values (V) - storing information related to each word. From these 3 vectors, we will calculate the attention vector for a word according to the formula:

$$Attention(Q,\ K,\ V) = softmax\left(\frac{QK^T}{\sqrt{Dimension\ of\ vector\ K}}\right)V$$

Multi-head Attention is an extension of the Self-Attention mechanism by performing it multiple times in parallel. In this way, the model can capture both local and global dependencies, allowing for a more comprehensive understanding of the input sequence. This parallelization also enhances the model's ability to capture complex relationships between words.

***Encoder:**

The encoder module of Transformer consists of many identical sub-layers.

The Multi-head Attention mechanism calculates the importance between different positions in the input sequence to capture the relationship between them. Then, the feed-forward neural network is used to process and extract additional features from the output of the Multi-head Attention mechanism.

The encoder module gradually extracts the features of the input sequence by stacking multiple layers in this way and passes the final encoded result to the decoder module for decoding.

***Decoder:**

The decoder also has an additional mechanism called Masked Multi-head Attention (Encoder-Decoder Attention) compared to the encoder. The decoding process is basically the same as the encoding process, except that the Decoder decodes one word at a time and the input of the Decoder is masked - it can only be decoded based on the previous words, and is not allowed to access the information it needs to decode later.

***Positional Embedding:**

When each element (word, token) in a sentence passes through the Encoder/Decoder stack of Transformer, the model itself will not have any meaning about the position/order for each token. The technique that allows the model to have additional information about the position of each token in the sentence in order to incorporate the sequence of tokens into the input representation is called Positional Embedding.

In the Transformer model, the core formula of Positional Embedding can be expressed as:

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

where PE represents the positional embedding matrix, pos is the position of the token in the sentence, i is the index of the ith element in the embedding, and d_{model} represents the number of dimensions of the hidden layer in the Transformer model.

Two common position encoding methods are commonly used in Transformer: Absolute Position Encoding and Relative Position Encoding, each providing position information in a separate way so that the model can process sequential data effectively.

b) Prompt learning:

Prompt learning is a machine learning method that builds on pre-trained language models and guides the model to perform different tasks by designing prompts, which provides greater flexibility for customizing the model's applications.

***Basic Components and Process of Prompt Learning:**

Prompt learning includes the following basic components:

- Prompt Template: The main part of the prompt, there are two common types of templates:
  - Fill-in-the-blank template: This type of template selects one or more positions in the text and represents them with [MASK] tags, which are used to prompt the model to fill in the corresponding words.
  - Prefix-based generation template: Prefix-based templating involves adding a specific prefix before a sentence to guide the model to generate appropriate text.

  The choice of prompt template plays a crucial role in prompt learning.
  - Artificially Designed Templates: These visually intuitive templates perform well in practical applications. However, manually designed templates may have limitations, requiring prior knowledge and being prone to errors.
  - There are two types of automatically generated templates: discrete templates and continuous templates.

  Combining multiple type of templates can improve model performance.
- Answer Mapping: The process of converting task labels into vocabulary that is appropriate for the context, and then selecting the answer with the highest probability as the predicted result.
- Pre-trained Language Model (PLM): Select an appropriate PLM to use as the base encoder.

The workflow of Prompt learning mainly includes the following four parts:

- Use PLMs as base encoders
- Add additional context (template) with a [MASK] position.
- Answer mapping: convert labels to label words - verbalizer.
- Be the GAP between pre-training and fine-tuning

***Learning Strategies of Prompt Learning:**

Choose the appropriate learning strategy based on the specific task and needs:

- Pre-training then fine-tuning: The most common strategy, suitable for most tasks.
- Tuning free promotion: Suitable for simple tasks, this strategy can significantly reduce training time and computing resource consumption.
- Fixed LM prompt tuning.

- Fix prompt LM tuning.

  The above two learning strategies are suitable for tasks that require more precise control and can optimize the model's performance by adjusting the prompt parameters or LM parameters.

- Prompt+LM tuning: Combines the advantages of both previous strategies, which can further improve the performance of the model.

In summary, using a well-designed prompt and an effective learning strategy will help improve the performance of the model on different tasks.

2. Technical Aspects of LLM Training:

The LLM training can be divided into three steps:

- Data Collection and Preprocessing.
- Pre-training process: includes defining the model architecture and pre-training tasks, and using appropriate parallel training algorithms to complete the training process.
- The third step involves fine-tuning and alignment.

2.1. Data Collection and Preprocessing:

a) Dataset Preparation:

Training LLMs require vast amounts of text data, and the quality of this data significantly impacts LLM performance. Pre-training on large-scale corpora provides LLMs with a fundamental understanding of language and some generative capability. Data sources for pre-training typically include web text, conversational data, books, and domain-specific documents to enhance LLMs' capabilities in those areas. There are 5 groups of datasets commonly used to train LLM:

- Books: Book datasets like BookCorpus and Gutenberg cover a wide variety of literary genres, from fiction to history, science, philosophy and more. They are widely used in training LLMs to provide deep understanding of languages across different domains.
- CommonCrawl: Is a large web crawl data repository, including more than 250 billion web pages collected over 16 years. Data from CommonCrawl accounts for 82% of the training data source for GPT-3. However, because it contains a lot of low-quality data, data preprocessing is necessary when working with CommonCrawl.
- Reddit Links: Reddit is a social media platform where users can submit links and posts, and others can vote on them. This makes it a valuable resource for creating high-quality datasets.
- Wikipedia: An extensive online encyclopedia project with high-quality content across various topics. Wikipedia is widely used in the training of many LLMs, serving as a valuable resource for language comprehension and production.
- Code: There are currently a limited number of open source datasets publicly available. Current efforts primarily involve extracting open copyrighted source code from the internet, primarily from Github and Stack Overflow.

b) Data Preprocessing:

The quality of data preprocessing directly affects the performance and security of the model. Important preprocessing steps include:

- Quality filtering: Low-quality data filtering is often performed using rule-based or classification methods. Rules may include only keeping text containing numbers, removing sentences that are all caps, and discarding files with a character-to-word ratio greater than 0.1.
- Deduplicate: Language models can sometimes generate repetitive content due to high repetition in the training data. Removing duplicate data helps maintain stability during training and improves the performance of LLMs.
- Privacy scrubbing: To ensure model security, language data preprocessing needs to remove sensitive information, including techniques such as anonymizing, redaction, or tokenization to eliminate personally identifiable details, geolocation, and other confidential data.
- Filtering out toxic and biased text: Removing toxic and biased content is an important step in developing fair and unbiased language models. This requires applying strong content moderation techniques to identify and remove text that may perpetuate harmful stereotypes, offensive language, or biased viewpoints.

2.2. Model Architecture:

Currently, all LLMs are built on the Transformer architecture. Typically, PLM architectures fall into three categories: Encoder-only, Decoder-only, and Encoder-Decoder.

a) Encoder-Decoder Architecture:

The encoder-decoder architecture consists of two main components: the encoder and the decoder. The encoder contains multiple layers of Multi-Head Self-Attention, which encodes the input sequence. The decoder generates the target sequence using cross-attention over the encoder's output representation and generates the target sequence autoregressively.

b) Decoder-only Architecture:

LLMs with decoder-only architecture only use the decoder part of the traditional Transformer architecture. This model generates tokens sequentially, attending to previous tokens in the sequence. This architecture has shown its effectiveness in various tasks such as text generation without an explicit encoding phase. This architecture can be classified into two types: Causal Decoder Architecture and Prefix Decoder Architecture.

- Causal Decoder Architecture: Each token attends only to the preceding tokens in the input sequence. It uses a special mask configuration to perform one-directional decoding. This architecture is applied in models such as the GPT series.
- Prefix Decoder Architecture: Combines the advantages of both encoder-decoder and causal decoder architectures. Allows bidirectional attention for prefix tokens while maintaining one-directional attention when generating subsequent tokens. This architecture is used in models such as PaLM and GLM.

Both architectures focus on automatically generating text, token by token, based on the context provided by the preceding tokens.

2.3. Pre-training Tasks:

Large Language Models (LLMs) typically learn rich language representations through a pre-training process. During pre-training, these models leverage extensive corpora, such as text data from the internet, and undergo training through self-supervised learning methods. One such method is Language Modeling (LM), where the LM is pre-trained by predicting the next word in a given context. This helps the model learn how to use vocabulary, grammar, semantics, and text structure. This gradual learning process allows the model to capture patterns and information inherent in the language, encoding a large amount of language knowledge into its parameters. Once the pre-training process is complete, these

model parameters can be fine-tuned for various natural language processing tasks to adapt to specific task requirements.

In addition to language modeling, there are other pre-training tasks such as using text with some parts randomly replaced, and then using the autoregression method to recover the replaced parts.

2.4. Model training:

a) Parallel Training:

Parallel training is a method for training machine learning models, especially LLMs, by using multiple processors or GPUs simultaneously to process data and model parameters. The goal of parallel training is to reduce the time required to train the model and increase the scalability of the model when handling large and complex data.

Collective communication is an important concept in parallel training. Collective communication involves the exchange of data between processors or GPUs during model training. Collective communication operations include:

- Broadcast: Send data from one GPU to all other GPUs.
- Reduce: Reduce(sum/average) data of all GPUs, send to one GPU.
- All Reduce: Reduce all data of GPUs, send to all GPU
- Reduce Scatter: Reduce all data of GPUs, send portions to all GPUs
- All Gather: Gather data of all GPUs, send all GPUs.

There are 4 main methods in parallel training:

- Data Parallel: Uses a parameter server to store the model parameters and the entire batch of data. The GPUs synchronize the model parameters and divide the data into chunks, with each GPU processing a chunk of data. The gradients are then aggregated and sent back to the parameter server.
- Model Parallel: Divides the model into smaller parts and each GPU will process a part of the model. This helps to process large models that do not fit into the memory of a single GPU.
- ZeRO: Optimizes memory and network bandwidth usage by eliminating data duplication on GPUs, enabling scaling of training without increasing memory usage.
- Pipeline Parallel: Divides the training process into stages and each GPU will process a specific stage. This helps to improve performance by reducing the waiting time between GPUs.

b) Mixed Precision Training:

Mixed precision training is a technique used in training to increase computation speed and reduce memory usage. This technique uses both 16-bit (FP16) and 32-bit (FP32) floating-point numbers during training.

FP16 has a smaller number range and lower precision than FP32, but it allows for faster computation. To improve computation speed, we can switch from FP32 to FP16, since the number of parameters in a model typically falls within the number range of FP16. When updating parameters, multiplying the gradient by the learning rate can lead to loss of precision due to floating point overflow if FP16 is used, so the updated parameter is represented as FP32. In the optimizer, the amount of update is stored as FP32 and accumulated efficiently through a temporary FP32 parameter, which is then converted back to the FP16 parameters.

c) Offloading:

Offloading is the process of transferring optimizer parameters from GPU to CPU to reduce computation load, using ZeRO3 to reduce the size of parameters, gradients and optimizer to 1/n number of GPUs.

d) Overlapping:

Overlapping is a performance optimization technique that performs computations asynchronously. This allows the model to continue performing other computations while waiting for a memory request to be processed, which helps to maximize the computation capability of the system and optimize the forward propagation process in model training.

e) Checkpoint:

The checkpoint mechanism does not store all intermediate results in GPU memory, but only keeps certain checkpoints, which reduces the amount of memory required and allows for recomputation during backpropagation.

2.5. Fine-tuning:

There are three types of fine-tuning techniques: supervised fine-tuning (SFT), alignment tuning, and parameter-efficient fine-tuning.

a) Supervised Fine-tuning:

SFT is the process of further adjusting a pre-trained model based on a labeled dataset for a specific task (e.g., translation, question answering).

Instruction tuning: A popular SFT technique where the LLM is further trained on a dataset of "(instruction, output)" pairs to improve controllability by understanding and following human instructions.

b) Alignment Tuning:

Alignment tuning is the process of adjusting the model to ensure that it generates outputs that are:

- Useful: The model should provide valuable information that serves the user's needs or goals well.
- Truthful: The model must generate accurate information, without distorting the truth, and maintain user trust.
- Harmless: The model should not generate harmful, offensive, or dangerous content, ensuring the safety of users and society.

Method: Use the Reinforcement Learning with Human Feedback (RLHF) method to collect human feedback data to train a reward model (RM), which is used in reinforcement learning to fine-tune the LLM.

c) Parameter-efficient Tuning:

Efficient parameter fine-tuning techniques significantly reduce computational and memory costs by fine-tuning only a small subset or addition of model parameters. Popular methods like Low-Rank Adaptation (LoRA), Prefix Tuning, and P-Tuning allow efficient model fine-tuning even in resource-constrained environments.

d) Safe Fine-tuning:

Enhancing the safety and responsibility of large language models (LLMs) by integrating additional safety techniques during fine-tuning. This includes all three main techniques.

- Supervised safety fine-tuning: Use expert-created data to train the LLM to identify and avoid dangerous content.
- Safe RLHF: Reward the LLM for providing the safest response, such as refusing to generate harmful content.
- Safe context distillation: Train the LLM on safe data created by adding safe prompts to the beginning of dangerous prompts.

2.6. Evaluation:

a) Static testing dataset:

The evaluation process relies heavily on appropriate datasets to assess LLM capabilities. Some commonly used datasets are:

- For image models**:** ImageNet (computer vision), Open Images.
- For text-based models**:** GLUE, SuperGLUE, MMLU (general ability), CMMLU (Chinese), XTREME/XTREME-R (multilingual)
- For specific tasks: MATH/GSM8K (mathematics), HumanEval/MBPP (code generation), HelloSwag/PIQA/BoolQ/etc. (common sense reasoning), MedQA-USMLE/MedMCQA (medical knowledge).

b) Open domain Q&A evaluation:

Evaluating ODQA is important because it affects the user experience. Popular datasets include SquAD and Natural Questions, with F1 and EM as evaluation metrics. However, human evaluation is still necessary due to the limitations of the word matching method.

c) Security evaluation:

LLMs require careful security evaluation to address potential threats and vulnerabilities:

- Potential bias: Security evaluation needs to consider whether the model creates or reinforces stereotypes such as gender or race, and how to mitigate or correct them.
- Privacy protection: It is necessary to ensure effective protection of user data privacy to prevent leaks and abuse.
- Adversarial attacks: LLMs can be easily manipulated through adversarial attacks. Addressing these vulnerabilities is critical.

d) Evaluation method:

Automated and manual evaluation should be combined for a comprehensive evaluation of language model performance:

- Automated evaluation: Metrics such as BLEU, ROUGE, and BERTScore provide quantitative methods for measuring performance. They are effective when analyzing large-scale data but cannot fully understand the complexity of language.
- Manual evaluation: Experts subjectively evaluate the quality of the LLM output. This method is valuable for specific tasks and identifying problems and subtle errors that automatic evaluation may miss, but it is time-consuming and subjective.

2.7. LLM Framework:

LLM Framework provides advanced technologies and methods to handle the large number of parameters of LLMs, helping to optimize the training process and improve the performance of the model on limited computing resources.

Some LLM Frameworks that use distributed training technology to take advantage of GPUs, CPUs, and NVMe memory are:

- Transformers: Hugging Face's open source library provides a user-friendly API for customizing pre-trained models using the Transformer architecture.

- DeepSpeed: Developed by Microsoft, this library supports ZeRO technology for efficient LLM training with features such as optimized state partitioning and custom mixed precision training.
- BMTrain: Tsinghua University's toolkit enables distributed training of large models simply without code restructuring, supporting various optimization techniques.
- Megatron-LM: NVIDIA's library for training large-scale language models, using model/data parallelism and mixed precision training to efficiently utilize GPUs.

3. Inference with Large Language Models:

LLMs are becoming increasingly powerful, but their large size leads to a significant need for resources. Here are some techniques for optimizing LLM inference to reduce computational cost and memory usage:

3.1. Model Compression:

- Knowledge Distillation: Transfer knowledge from a cumbersome model to a smaller model that is more suitable for deployment.
- Model Pruning: Removes redundant parts from the parameter matrix of large models. There are two ways to do this:
  - Unstructured pruning: Removes individual connections or weights in the neural network without following any specific structural pattern.
  - Structured pruning: Only removes specific structural patterns.
- Model Quantization: Reduces the number of bits used to compute numbers, reducing storage and computation requirements. However, there is a trade-off between accuracy and performance. For example, from real numbers to integers.
- Weight Sharing: Uses the same set of parameters for multiple parts of the LLM. This helps reduce the number of parameters to be learned, increases computational efficiency, and reduces the risk of overfitting.
- Low-Rank Approximation: Creates smaller models with fewer parameters by decomposing large matrices into smaller matrices, enabling efficient inference on resource-constrained devices.

3.2. Memory Scheduling:

Optimizes memory usage during LLM inference by efficiently managing memory access patterns. This is important because large models often have complex architectures with significant memory requirements. An example is BMInf, which leverages virtual memory principles to efficiently schedule the parameters of each layer between the GPU and CPU.

3.3. Parallelism:

Uses multiple processing units to distribute the LLM inference workload, improving overall throughput and reducing latency.

- Data Parallelism: Distributes data across multiple GPUs for faster processing.
- Tensor Parallelism: Model parameters are split into multiple tensors and each tensor is computed on different processing units.
- Pipeline Parallelism: Divides the computation into stages processed by different GPUs, allowing support for larger models and improving device utilization.

3.4. Structural Optimization:

Focuses on minimizing memory access during LLM inference to improve speed. Techniques such as FlashAttention and PagedAttention improve computation speed by using a chunked computation method, minimizing the storage costs associated with the matrix.

3.5. Inference Framework:

Provides a platform for deploying and running LLMs. Techniques such as parallel computing, model compression, memory scheduling, and transformer architecture optimization have been effectively integrated into major inference frameworks. These frameworks provide tools and interfaces that streamline deployment and inference processes for different use cases.

4. Utilization of LLMs:

LLMs can be applied in most specialized domain. After being pre-trained and fine-tuned, LLMs are mainly used by designing appropriate prompts for various tasks through a number of capabilities, including: Zero-shot, Few-shot, Chain-of-thought prompts,...

LLM Deployment Strategies:

- API Access: Use the capabilities of proprietary, powerful models provided through open APIs (e.g., ChatGPT).
- Deploy open-source LLMs for local use.
- Fine-tune open-source LLMs to meet the specific standards of a particular field and then deploy them locally for specialized applications.

5. Future Development Directions and Their Implications:

Future Development Trends of LLMs:

- Model Scaling: LLMs are expected to continue to scale in size, improving performance and learning ability.
- Multimodality: Future LLMs will be able to process not only text, but also images, videos, and audio, expanding their applications.
- Efficiency: Focus on reducing training and inference costs through techniques such as knowledge distillation and model compression.
- Industry-Specific Training: Customize LLMs for specific industries to better understand industry-specific terminology and context.
- New Architectures: Exploring alternative architectures such as RNNs for LLMs is an exciting research direction with the potential to address limitations such as the fixed-size input requirement in transformers.

For AI researchers, collaboration between different industries and fields is becoming essential. They must develop technical skills to address the challenges in LLM development. The ethical issues and social impact of LLMs are also important areas that need continuous research and improvement.