Day 1

Today's Assignments

1. Complete the Intro Unit – "Foundational Large Language Models & Text Generation", which is:

- [Optional] Listen to the summary podcast episode for this unit (created by NotebookLM).
- Read the "Foundational Large Language Models & Text Generation" whitepaper.

2. Complete Unit 1 – "Prompt Engineering", which is:

- [Optional] Listen to the summary podcast episode for this unit (created by NotebookLM).
- Read the "Prompt Engineering" whitepaper.
- Complete this code lab on Kaggle where you'll learn prompting fundamentals. Make sure you phone verify your account before starting, it's necessary for the code labs.

What You'll Learn

Today you'll explore the evolution of LLMs, from transformers to techniques like fine-tuning and inference acceleration. You'll also get trained in the art of prompt engineering for optimal LLM interaction.

The code lab will walk you through getting started with the Gemini API and cover several prompt techniques and how different parameters impact the prompts.

Reminders

- Tomorrow at 11:00 am ET, Paige Bailey is hosting the first livestream on our YouTube channel to discuss the assignments with the course authors and other special guests from Google. Tomorrow's guests are Mohammadamin Barekatain, Lee Boonstra, Logan Kilpatrick, Daniel Mankowitz, Majd Merey Al, Anant Nawalgaria, Aliaksei Severyn and Chuck Sugnet. It'll be recorded in case you're unable to attend.
- Discord is the best place to ask questions – specifically in the #5dgai-q-and-a channel. In addition to other participants, several Googlers are there to help. During the livestream, we'll also pick several questions from Discord to discuss. You'll get Kaggle swag if your question is chosen!
- We created a new channel in Discord called #5dgai-announcements that will be used exclusively for course announcements from us.
- We want this course community to be positive and supportive. Please follow Kaggle's community guidelines found here.

Happy learning and see you tomorrow!

The Kaggle Team

Table of contents

- **Layer normalization và residual connections**
- **Feedforward layer**
- **Encoder và decoder**
- **Huấn luyện transformer**

The evolution of transformers (Tiến hóa của các mô hình transformer)

- GPT-1
- BERT
- GPT-2
- GPT-3/3.5/4
- LaMDA
- Gopher
- **GLaM**
- **Chinchilla**
- **PaLM**
  - **PaLM 2**
- **Gemini**
- **Các mô hình mở khác**
- **Comparison - So sánh**

Fine-tuning LLM

- Supervised fine-tuning
- Reinforcement learning from human feedback
- Parameter Efficient Fine-Tuning

Using large language models (Sử dụng các mô hình ngôn ngữ lớn (LLMs))

- **Prompt engineering**
- **Sampling Techniques and Parameters (Kỹ thuật Sampling và Các Tham số)**

Accelerating inference (Tăng tốc suy luận)

- **Trade offs (Sự đánh đổi)**
  - **The Quality vs Latency/Cost Tradeoff (Đánh đổi giữa chất lượng và độ trễ/chi phí)**
  - **The Latency vs Cost Tradeoff (Đánh đổi giữa độ trễ và chi phí)**
- **Output-approximating methods (Các phương pháp gần đúng đầu ra)**
  - **Quantization (Lượng tử hóa)**
  - **Distillation (Chưng cất kiến thức)**
- **Output-preserving methods (Các phương pháp bảo toàn đầu ra)**
  - **Flash Attention (Tăng tốc Attention)**
  - **Prefix Caching (Lưu trữ trước các giá trị tiền tố)**
  - **Speculative Decoding (Giải mã suy đoán)**
- **Batching and Parallelization**

Applications (Ứng dụng)

Introduction (Giới thiệu)

Sự xuất hiện của các mô hình ngôn ngữ lớn (Large Language Models - LLMs) đại diện cho một bước ngoặt lớn trong thế giới trí tuệ nhân tạo. Khả năng xử lý, tạo ra, và hiểu ý định của người dùng của các mô hình này đang thay đổi cơ bản cách chúng ta tương tác với thông tin và công nghệ.

LLM là một hệ thống trí tuệ nhân tạo tiên tiến, chuyên về xử lý, hiểu và tạo ra văn bản giống con người. Các hệ thống này thường được triển khai dưới dạng mạng nơ-ron sâu và được đào tạo trên một lượng lớn dữ liệu văn bản. Điều này cho phép chúng học được các mẫu phức tạp của ngôn ngữ, từ đó có khả năng thực hiện nhiều nhiệm vụ khác nhau như dịch máy, tạo văn bản sáng tạo, trả lời câu hỏi, tóm tắt văn bản và nhiều nhiệm vụ suy luận và ngôn ngữ khác.

Bài viết này đi sâu vào quá trình phát triển của các kiến trúc và phương pháp đã dẫn đến sự ra đời của các mô hình ngôn ngữ lớn, cũng như các kiến trúc được sử dụng tại thời điểm xuất bản. Bài viết cũng thảo luận về các kỹ thuật tinh chỉnh để tùy chỉnh LLM cho một lĩnh vực hoặc nhiệm vụ cụ thể, các phương pháp để làm cho quá trình đào tạo hiệu quả hơn, cũng như các phương pháp để tăng tốc suy luận. Phần cuối bao gồm nhiều ứng dụng và ví dụ về mã.

Why language models are important (Tại sao các mô hình ngôn ngữ lại quan trọng)

Các mô hình ngôn ngữ lớn (LLMs) mang lại sự cải tiến vượt bậc về hiệu suất so với các phương pháp trước đó trong nhiều nhiệm vụ khác nhau và phức tạp, yêu cầu trả lời câu hỏi hoặc suy luận phức tạp, giúp hiện thực hóa nhiều ứng dụng mới. Những ứng dụng này bao gồm dịch ngôn ngữ, tạo mã và hoàn thiện mã, tạo văn bản, phân loại văn bản và trả lời câu hỏi, chỉ kể một vài ví dụ. Mặc dù các LLM cơ bản được huấn luyện trên nhiều nhiệm vụ với lượng dữ liệu lớn có thể hoạt động rất tốt mà không cần điều chỉnh, và thậm chí thể hiện các hành vi nổi bật (ví dụ như khả năng thực hiện các nhiệm vụ mà chúng chưa được huấn luyện trực tiếp), chúng cũng có thể được điều chỉnh để giải quyết các nhiệm vụ cụ thể khi hiệu suất mặc định chưa đạt yêu cầu thông qua quá trình gọi là tinh chỉnh. Quá trình này yêu cầu ít dữ liệu và tài nguyên tính toán hơn nhiều so với việc huấn luyện LLM từ đầu. Các LLM còn có thể được điều chỉnh theo hướng mong muốn nhờ vào kỹ thuật lập trình gợi ý (prompt engineering): nghệ thuật và khoa học xây dựng các gợi ý và tham số của LLM để có được phản hồi như ý.

Câu hỏi lớn đặt ra là: các mô hình ngôn ngữ lớn này hoạt động như thế nào? Phần tiếp theo sẽ khám phá các thành phần cốt lõi của LLM, tập trung vào các kiến trúc transformer và sự phát triển của chúng từ bài báo gốc “Attention is all you need” đến các mô hình mới nhất như Gemini, LLM mạnh nhất của Google. Chúng tôi cũng đề cập đến các kỹ thuật huấn luyện và tinh chỉnh, cũng như các phương pháp để cải thiện tốc độ tạo phản hồi. Bài viết sẽ kết thúc với một vài ví dụ về cách các mô hình ngôn ngữ được sử dụng trong thực tế.

Các mô hình ngôn ngữ lớn

Một mô hình ngôn ngữ dự đoán xác suất của một chuỗi từ. Thông thường, khi được cung cấp một đoạn văn bản ban đầu, mô hình ngôn ngữ sẽ gán các xác suất cho các từ tiếp theo. Ví dụ, với đoạn văn bản “Thành phố nổi tiếng nhất ở Mỹ là...”, mô hình ngôn ngữ có thể dự đoán xác suất cao cho các từ “New York” và “Los Angeles” và xác suất thấp cho các từ như “laptop” hoặc “apple”. Bạn có thể tạo một mô hình ngôn ngữ cơ bản bằng cách lưu trữ một bảng n-gram, trong khi các mô hình ngôn ngữ hiện đại thường dựa trên các mô hình nơ-ron, như transformers.

Trước khi các transformer ra đời, mạng nơ-ron hồi quy (Recurrent Neural Networks - RNNs) là phương pháp phổ biến để mô hình hóa các chuỗi. Đặc biệt, các kiến trúc “bộ nhớ ngắn hạn dài” (LSTM) và “đơn vị hồi quy có cổng” (GRU) rất thông dụng. Lĩnh vực này bao gồm các vấn đề ngôn ngữ như dịch máy, phân loại

văn bản, tóm tắt văn bản, và trả lời câu hỏi, trong số các nhiệm vụ khác. RNN xử lý các chuỗi đầu vào và đầu ra theo tuần tự, tạo ra một chuỗi các trạng thái ẩn dựa trên trạng thái ẩn trước đó và đầu vào hiện tại. Tính **tuần tự của RNN khiến chúng tiêu tốn nhiều tài nguyên tính toán và khó song song hóa trong quá trình huấn luyện** (mặc dù nghiên cứu gần đây về mô hình không gian trạng thái đang cố gắng khắc phục những thách thức này).

Trong khi đó, transformers là một loại mạng nơ-ron có thể xử lý các chuỗi token song song nhờ vào cơ chế tự chú ý. Điều này có nghĩa là transformers có thể mô hình hóa tốt hơn các ngữ cảnh dài hạn và dễ dàng song song hóa hơn RNN, giúp chúng huấn luyện nhanh hơn đáng kể và mạnh mẽ hơn trong việc xử lý các phụ thuộc dài hạn trong các nhiệm vụ có chuỗi dài. Tuy nhiên, **chi phí của cơ chế self-attention trong các transformers ban đầu là lũy thừa hai theo độ dài ngữ cảnh**, giới hạn kích thước của ngữ cảnh, trong khi RNN có lý thuyết về độ dài ngữ cảnh vô hạn. Transformers đã trở thành phương pháp phổ biến nhất cho các vấn đề mô hình hóa và chuyển đổi chuỗi trong những năm gần đây.

Chúng tôi sẽ thảo luận về phiên bản đầu tiên của mô hình transformer và sau đó sẽ trình bày các mô hình và thuật toán tiên tiến hơn gần đây.

Transformer

Kiến trúc transformer được phát triển tại Google vào năm 2017 để sử dụng trong mô hình dịch thuật. Đây là một mô hình chuỗi sang chuỗi (seq2seq) có khả năng *chuyển đổi các chuỗi từ một lĩnh vực sang các chuỗi trong lĩnh vực khác*. Ví dụ, dịch các câu tiếng Pháp sang tiếng Anh. **Kiến trúc transformer ban đầu bao gồm hai phần: encoder (bộ mã hóa) và decoder (bộ giải mã)**. Encoder chuyển đổi văn bản đầu vào (ví dụ, câu tiếng Pháp) thành một biểu diễn, sau đó được truyền đến decoder. Decoder sử dụng biểu diễn này để tạo ra văn bản đầu ra (ví dụ, bản dịch tiếng Anh) theo cách tự hồi quy (autoregressively).

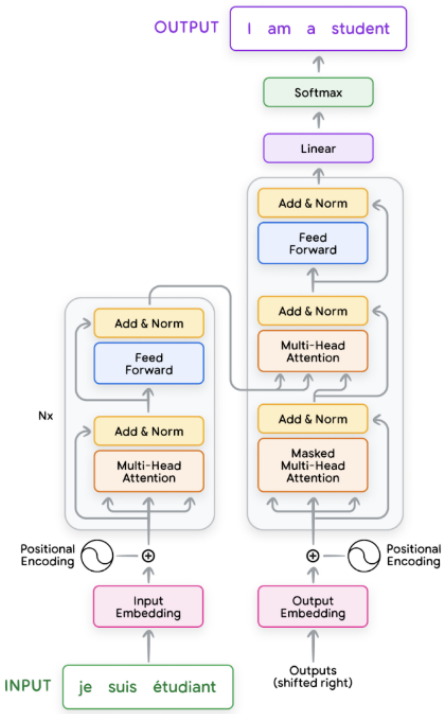
Đáng chú ý, kích thước đầu ra của encoder trong transformer tỷ lệ tuyến tính với kích thước đầu vào của nó. Hình 1 cho thấy thiết kế của kiến trúc transformer ban đầu.

Transformer bao gồm nhiều lớp. Một lớp trong mạng nơ-ron bao gồm một tập hợp các tham số thực hiện một phép biến đổi cụ thể trên dữ liệu. Trong sơ đồ, bạn có thể thấy ví dụ về một số lớp như Multi-Head Attention, Thêm & Chuẩn hóa (Add &

Norm), Feed-Forward, Linear, Softmax, v.v. Các lớp có thể được chia nhỏ thành các lớp đầu vào, lớp ẩn và lớp đầu ra.

Lớp đầu vào (Input layer) (ví dụ, Input/output embedding) là lớp mà raw data đi vào network. Các input embeddings được sử dụng để đại diện cho các input token của mô hình. Các output embeddings được sử dụng để đại diện cho các output token mà mô hình dự đoán. Ví dụ, trong mô hình dịch máy, các input embeddings sẽ đại diện cho các từ trong ngôn ngữ nguồn, trong khi các output embeddings sẽ đại diện cho các từ trong ngôn ngữ đích.

Lớp đầu ra (Output layer) (ví dụ, Softmax) là lớp cuối cùng tạo ra đầu ra của network. Các lớp ẩn (ví dụ, Multi-Head Attention) nằm giữa lớp đầu vào và đầu ra và là nơi diễn ra các phép tính quan trọng!



Hình 1 : Transformer gốc.

Để hiểu rõ hơn về các lớp khác nhau trong transformer, hãy sử dụng ví dụ về nhiệm vụ dịch tiếng Pháp sang tiếng Anh. Chúng ta sẽ giải thích cách một câu tiếng Pháp được đưa vào transformer và bản dịch tiếng Anh tương ứng được tạo ra. Chúng tôi cũng sẽ mô tả từng thành phần bên trong transformer như trong Hình 1.

Why language models are important (Chuẩn bị đầu vào và embedding)

Để chuẩn bị dữ liệu ngôn ngữ đầu vào cho transformers, chúng ta chuyển một chuỗi đầu vào thành các token và sau đó thành các embeddings đầu vào. Ở cấp độ cao, một embedding đầu vào là một vector không gian cao đại diện cho ý nghĩa của từng token trong câu. Embedding này sau đó được đưa vào transformer để xử lý. Quá trình tạo embedding đầu vào bao gồm các bước sau:

1. **Chuẩn hóa (Tùy chọn)**: Chuẩn hóa văn bản bằng cách loại bỏ khoảng trắng dư thừa, dấu, v.v.
2. **Tokenization**: Phân tách câu thành các từ hoặc từ con và ánh xạ chúng thành các mã token nguyên từ một từ vựng.
3. **Embedding**: Chuyển đổi mỗi mã token thành vector không gian cao tương ứng, thường sử dụng một bảng tra cứu. Các embedding này có thể được học trong quá trình huấn luyện.
4. **Mã hóa vị trí (Positional Encoding)**: Thêm thông tin về vị trí của từng token trong chuỗi để giúp transformer hiểu thứ tự của các từ.

Những bước này giúp chuẩn bị đầu vào cho transformers, giúp chúng hiểu rõ hơn về ý nghĩa của văn bản.

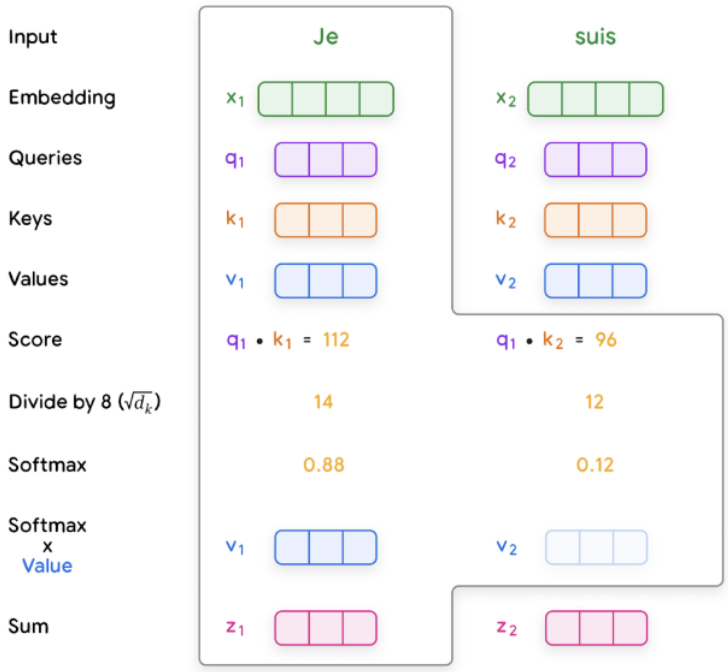
Multi-head attention

Sau khi chuyển đổi các token đầu vào thành các vector embedding, bạn đưa các embedding này vào module multi-head attention (xem Hình 1). Self-attention là một cơ chế quan trọng trong transformers; nó cho phép mô hình tập trung vào các phần cụ thể của chuỗi đầu vào có liên quan đến nhiệm vụ hiện tại và giúp nắm bắt các phụ thuộc dài hạn trong chuỗi hiệu quả hơn so với RNN truyền thống.

Hiểu về self-attention

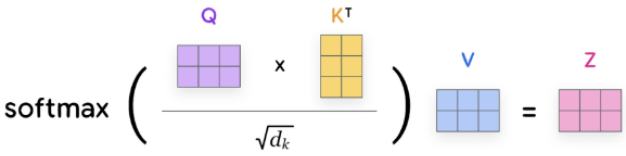
Hãy xem xét câu sau: "The tiger jumped out of a tree to get a drink because it was thirsty." Self-attention giúp xác định mối quan hệ giữa các từ và cụm từ khác nhau trong câu. Ví dụ, trong câu này, "the tiger" và "it" là cùng một đối tượng, vì vậy ta kỳ vọng hai từ này sẽ có liên kết mạnh với nhau. Self-attention đạt được điều này thông qua các bước sau (Hình 2):

1. **Tạo queries, keys, và values**: Mỗi embedding đầu vào được nhân với ba ma trận trọng số đã học (W_q, W_k, W_v) để tạo ra các vector query (Q), key (K), và value (V). Đây giống như các biểu diễn chuyên biệt của từng từ.
    - **Query**: Vector query giúp mô hình đặt câu hỏi, “Những từ nào khác trong chuỗi có liên quan đến tôi?”
    - **Key**: Vector key giống như một nhãn giúp mô hình xác định một từ có thể liên quan như thế nào đến các từ khác trong chuỗi.
    - **Value**: Vector value chứa thông tin nội dung thực tế của từ.
2. **Tính toán scores**: Các điểm số được tính để xác định mức độ mà mỗi từ nên "chú ý" đến các từ khác. Điều này được thực hiện bằng cách lấy tích vô hướng (dot product) của vector query của một từ với các vector key của tất cả các từ trong chuỗi.
3. **Normalization**: Các điểm số được chia cho căn bậc hai của kích thước vector key ($\sqrt{d_k}$) để đảm bảo tính ổn định, sau đó được đưa qua hàm softmax để tạo ra các trọng số attention. Các trọng số này cho biết mức độ mà mỗi từ được liên kết mạnh mẽ với các từ khác.
4. **Weighted values**: Mỗi vector value được nhân với trọng số attention tương ứng. Các kết quả này được cộng lại, tạo ra một biểu diễn có ngữ cảnh cho từng từ.



Hình 2: The process of computing self-attention in the multi-head attention module1

Trên thực tế, các phép tính này được thực hiện đồng thời bằng cách xếp chồng các vector query, key và value của tất cả các token vào các ma trận Q, K và V và nhân chúng lại với nhau như minh họa trong Hình 3.



Hình 3. Hoạt động cơ bản của attention, với Q=query, K=keys và V=value, Z=attention, d_k=kích thước của queries và keys

Multi-head attention: sức mạnh trong tính đa dạng

Multi-head attention sử dụng nhiều bộ ma trận trọng số Q, K, V. Các bộ này chạy song song, mỗi "head" có thể tập trung vào các khía cạnh khác nhau của mối quan hệ giữa các đầu vào. Đầu ra từ mỗi head được concatenated (ghép nối) và biến đổi tuyến tính, cung cấp cho mô hình một biểu diễn phong phú hơn về chuỗi đầu vào.

Việc sử dụng multi-head attention cải thiện khả năng của mô hình trong việc xử lý các mẫu ngôn ngữ phức tạp và các phụ thuộc dài hạn. Điều này rất quan trọng đối với các nhiệm vụ đòi hỏi sự hiểu biết tinh tế về cấu trúc và nội dung ngôn ngữ, chẳng hạn như dịch máy, tóm tắt văn bản và trả lời câu hỏi. Cơ chế này cho phép transformer xem xét nhiều cách diễn giải và biểu diễn của đầu vào, từ đó nâng cao hiệu suất của nó trong các nhiệm vụ này.

Layer normalization và residual connections

Mỗi lớp trong transformer, bao gồm một module multi-head attention và một lớp feed-forward, đều sử dụng layer normalization và residual connections. Đây là lớp *Add & Norm* trong Hình 1, trong đó "Add" tương ứng với residual connection và "Norm" tương ứng với layer normalization. Layer normalization tính toán giá trị trung bình (mean) và phương sai (variance) của các activations (kích hoạt) để chuẩn hóa activations (kích hoạt) trong một lớp nhất định. Điều này thường được thực hiện để giảm sự thay đổi đồng biến (covariate shift) cũng như cải thiện gradient flow (dòng chảy), giúp quá trình hội tụ nhanh hơn trong quá trình huấn luyện và cải thiện hiệu suất tổng thể.

Residual connections truyền các đầu vào đến đầu ra của một hoặc nhiều lớp. Điều này giúp quy trình tối ưu hóa trở nên dễ học hơn và giúp xử lý các vấn đề gradient bị mất (vanishing) và bùng nổ (exploding).

Lớp *Add & Norm* được áp dụng cho cả module multi-head attention và lớp feed-forward sẽ được mô tả trong phần tiếp theo.

Feedforward layer

Đầu ra của module multi-head attention và lớp "Add & Norm" sau đó được đưa vào lớp feedforward của mỗi khối transformer. Lớp này áp dụng một phép biến đổi theo từng vị trí với dữ liệu, độc lập cho từng vị trí trong chuỗi, cho phép tích hợp thêm tính phi tuyến và độ phức tạp vào các biểu diễn của mô hình. Lớp feedforward thường bao gồm hai phép biến đổi tuyến tính với một hàm kích hoạt phi tuyến như ReLU hoặc GELU ở giữa. Cấu trúc này tăng cường sức mạnh biểu diễn cho mô hình. Sau khi được xử lý qua lớp feedforward, dữ liệu sẽ trải qua một bước "Add and Norm" khác, góp phần vào sự ổn định và hiệu quả của các mô hình transformer sâu.

Encoder và decoder

Kiến trúc transformer ban đầu dựa vào sự kết hợp giữa các module encoder và decoder. Mỗi encoder và decoder bao gồm một loạt các lớp, với mỗi lớp bao gồm các thành phần chính: cơ chế self-attention multi-head, a position-wise feed-forward network (mạng feed-forward theo từng vị trí), các lớp normalization, và các residual connections.

Chức năng chính của encoder là xử lý chuỗi đầu vào thành một biểu diễn liên tục chứa thông tin ngữ cảnh cho từng token. Chuỗi đầu vào trước tiên được chuẩn hóa, chuyển đổi thành các token và chuyển thành các embeddings. Positional encodings được thêm vào các embeddings này để giữ lại thông tin về thứ tự trong chuỗi. Thông qua các cơ chế self-attention, mỗi token trong chuỗi có thể tự động tập trung vào bất kỳ token nào khác, từ đó hiểu được các mối quan hệ ngữ cảnh trong chuỗi. Đầu ra từ encoder là một chuỗi các vector embedding Z đại diện cho toàn bộ chuỗi đầu vào.

Decoder có nhiệm vụ tạo ra một chuỗi đầu ra dựa trên ngữ cảnh được cung cấp bởi đầu ra Z của encoder. Nó hoạt động theo cách từng token một, bắt đầu bằng một token đầu chuỗi (start-of-sequence). Các lớp của decoder sử dụng hai loại cơ chế attention: masked self-attention và encoder-decoder cross-attention. Masked self-attention đảm bảo rằng mỗi vị trí chỉ có thể tập trung vào các vị trí trước đó trong chuỗi đầu ra, duy trì tính tự hồi quy. Điều này rất quan trọng để ngăn decoder truy cập vào các token tương lai trong chuỗi đầu ra. Cơ chế encoder-decoder cross-attention cho phép decoder tập trung vào các phần liên quan của chuỗi đầu vào, tận dụng các embeddings ngữ cảnh do encoder tạo ra. Quá trình lặp lại này tiếp tục cho đến khi decoder dự đoán một token kết thúc chuỗi (end-of-sequence), hoàn thành quá trình tạo chuỗi đầu ra.

Phần lớn các mô hình ngôn ngữ lớn (LLMs) hiện nay áp dụng biến thể chỉ sử dụng decoder trong kiến trúc transformer. Phương pháp này loại bỏ sự phân tách encoder-decoder truyền thống, thay vào đó tập trung trực tiếp vào việc tạo chuỗi đầu ra từ đầu vào. Chuỗi đầu vào trải qua quá trình embedding và mã hóa vị trí tương tự trước khi được đưa vào decoder. Decoder sau đó sử dụng masked self-attention để dự đoán từng token tiếp theo dựa trên các token đã được tạo ra trước đó. Cách tiếp cận đơn giản hóa này giúp kiến trúc trở nên tinh gọn cho các nhiệm vụ cụ thể mà encoding và decoding có thể được hợp nhất hiệu quả.

Huấn luyện transformer

Khi nói về các mô hình machine learning, điều quan trọng là phân biệt giữa huấn luyện (training) và suy luận (inference). Huấn luyện thường ám chỉ việc điều chỉnh các tham số của mô hình, bao gồm các hàm mất mát và lan truyền ngược. Suy luận là khi mô hình chỉ được sử dụng để dự đoán đầu ra mà không cập nhật trọng số của mô hình; các tham số của mô hình được cố định trong quá trình suy luận. Cho đến nay, chúng ta đã tìm hiểu cách transformers tạo đầu ra trong quá trình suy luận. Tiếp theo, chúng ta sẽ tập trung vào cách huấn luyện transformers để thực hiện một hoặc nhiều nhiệm vụ nhất định.

Chuẩn bị dữ liệu

Bước đầu tiên là chuẩn bị dữ liệu, bao gồm một số bước quan trọng. Đầu tiên, làm sạch dữ liệu bằng cách áp dụng các kỹ thuật như lọc, loại bỏ dữ liệu trùng lặp và chuẩn hóa. Bước tiếp theo là tokenization, trong đó tập dữ liệu được chuyển đổi thành các token bằng các kỹ thuật như Byte-Pair Encoding và Unigram tokenization. Tokenization tạo ra một từ vựng, là tập hợp các token duy nhất mà LLM sử dụng. Từ vựng này đóng vai trò là "ngôn ngữ" của mô hình để xử lý và hiểu văn bản. Cuối cùng, dữ liệu thường được chia thành tập huấn luyện để huấn luyện mô hình và tập kiểm tra để đánh giá hiệu suất của mô hình.

Huấn luyện và hàm mất mát

Một vòng lặp huấn luyện transformer điển hình bao gồm một số phần: Đầu tiên, các batch chuỗi đầu vào được lấy mẫu từ một tập dữ liệu huấn luyện. Với mỗi chuỗi đầu vào, có một chuỗi mục tiêu tương ứng. Trong huấn luyện không giám sát, chuỗi mục tiêu được lấy từ chính chuỗi đầu vào. Batch các chuỗi đầu vào sau đó được đưa vào transformer. Transformer tạo ra các chuỗi đầu ra dự đoán. Sự khác biệt giữa chuỗi dự đoán và chuỗi mục tiêu được đo lường bằng một hàm mất mát (thường là cross-entropy loss). Các gradient của hàm mất mát này được tính toán, và một optimizer sử dụng chúng để cập nhật các tham số của transformer. Quá trình này lặp lại cho đến khi transformer đạt đến một mức hiệu suất nhất định hoặc đã được huấn luyện trên số lượng token đã định trước.

Có các phương pháp khác nhau để xây dựng nhiệm vụ huấn luyện cho transformers tùy thuộc vào kiến trúc sử dụng:

- **Các mô hình chỉ có decoder** thường được huấn luyện trước trên nhiệm vụ language modeling. Chuỗi mục tiêu cho decoder chỉ đơn giản là phiên bản

dịch chuyển của chuỗi đầu vào. Với chuỗi huấn luyện như "the cat sat on the mat," nhiều cặp đầu vào/mục tiêu có thể được tạo cho mô hình. Ví dụ, đầu vào "the cat sat on" sẽ dự đoán "the" và tiếp theo, đầu vào "the cat sat on the" sẽ dự đoán chuỗi mục tiêu "mat".

- **Các mô hình chỉ có encoder** (như BERT) thường được huấn luyện trước bằng cách làm hỏng chuỗi đầu vào theo một cách nào đó và để mô hình cố gắng tái tạo nó. Một phương pháp như vậy là masked language modeling (MLM). Trong ví dụ của chúng ta, chuỗi đầu vào có thể là "The [MASK] sat on the mat" và chuỗi mục tiêu sẽ là câu gốc.
- **Các mô hình encoder-decoder** (như transformer gốc) được huấn luyện trên các nhiệm vụ giám sát theo kiểu chuỗi-sang-chuỗi như dịch thuật (chuỗi đầu vào "Le chat est assis sur le tapis" và chuỗi mục tiêu "The cat sat on the mat"), trả lời câu hỏi (chuỗi đầu vào là câu hỏi và chuỗi mục tiêu là câu trả lời tương ứng), và tóm tắt (chuỗi đầu vào là toàn bộ bài viết và chuỗi mục tiêu là phần tóm tắt tương ứng của nó). Các mô hình này cũng có thể được huấn luyện theo cách không giám sát bằng cách chuyển đổi các nhiệm vụ khác thành định dạng chuỗi-sang-chuỗi. Ví dụ, khi huấn luyện trên dữ liệu Wikipedia, chuỗi đầu vào có thể là phần đầu của một bài viết, và chuỗi mục tiêu là phần còn lại của bài viết.

Một yếu tố bổ sung cần xem xét trong quá trình huấn luyện là "context length" (độ dài ngữ cảnh). Điều này đề cập đến số lượng token trước đó mà mô hình có thể "nhớ" và sử dụng để dự đoán token tiếp theo trong chuỗi. Các ngữ cảnh dài hơn cho phép mô hình nắm bắt các mối quan hệ và phụ thuộc phức tạp hơn trong văn bản, có thể dẫn đến hiệu suất tốt hơn. Tuy nhiên, ngữ cảnh dài hơn cũng đòi hỏi nhiều tài nguyên tính toán và bộ nhớ hơn, có thể làm chậm quá trình huấn luyện và suy luận. Việc chọn độ dài ngữ cảnh phù hợp liên quan đến việc cân bằng giữa các yếu tố này dựa trên nhiệm vụ cụ thể và các tài nguyên sẵn có.

The evolution of transformers (Tiến hóa của các mô hình transformer)

Các phần tiếp theo sẽ cung cấp tổng quan về các kiến trúc transformer khác nhau. Bao gồm các mô hình chỉ mã hóa (encoder-only), mã hóa-giải mã (encoder-

decoder), cũng như các mô hình chỉ giải mã (decoder-only). Chúng ta sẽ bắt đầu với GPT-1 và BERT và kết thúc với họ LLM mới nhất của Google mang tên Gemini.

GPT-1

GPT-1 (Generative pre-trained transformer phiên bản 1) là một mô hình decoder-only được phát triển bởi OpenAI vào năm 2018. Mô hình này được huấn luyện trên tập dữ liệu BooksCorpus (chứa khoảng vài tỷ từ) và có khả năng tạo văn bản, dịch ngôn ngữ, viết các nội dung sáng tạo khác nhau, và trả lời các câu hỏi một cách thông tin. Những đổi mới chính trong GPT-1 bao gồm:

- Kết hợp transformers và unsupervised pre-training: Huấn luyện trước không giám sát là quá trình huấn luyện một mô hình ngôn ngữ trên một tập dữ liệu lớn không gán nhãn. Sau đó, dữ liệu có gán nhãn sẽ được sử dụng để fine-tune (tinh chỉnh) mô hình cho một nhiệm vụ cụ thể, như dịch thuật hoặc phân loại cảm xúc. Trong các công trình trước đây, hầu hết các mô hình ngôn ngữ đều được huấn luyện sử dụng mục tiêu học có giám sát, có nghĩa là mô hình được huấn luyện trên một tập dữ liệu có gán nhãn, trong đó mỗi ví dụ có một nhãn tương ứng. Cách tiếp cận này có *hai hạn chế chính*. ***Thứ nhất***, nó đòi hỏi một lượng lớn dữ liệu có gán nhãn, vốn có thể tốn kém và mất thời gian để thu thập. ***Thứ hai*****,** mô hình chỉ có thể khái quát hóa cho các nhiệm vụ tương tự với các nhiệm vụ mà nó đã được huấn luyện. Semi-supervised sequence learning (Học chuỗi bán giám sát) là một trong những công trình đầu tiên cho thấy rằng huấn luyện trước không giám sát kết hợp với huấn luyện có giám sát có hiệu quả vượt trội so với chỉ huấn luyện có giám sát.

  Unsupervised pre-training giải quyết những hạn chế này bằng cách huấn luyện mô hình trên một tập dữ liệu lớn không gán nhãn. Dữ liệu này có thể được thu thập dễ dàng và rẻ hơn so với dữ liệu có gán nhãn. Ngoài ra, mô hình có thể khái quát hóa cho các nhiệm vụ khác với các nhiệm vụ mà nó đã được huấn luyện. Tập dữ liệu BooksCorpus là một tập hợp lớn (5GB) các văn bản không gán nhãn được sử dụng để huấn luyện mô hình ngôn ngữ GPT-1. Tập dữ liệu này chứa hơn 7.000 cuốn sách chưa xuất bản, cung cấp cho mô hình một lượng lớn dữ liệu để học hỏi. Thêm vào đó, tập hợp này chứa những đoạn văn dài liên tục, giúp mô hình học được các mối quan hệ phụ thuộc dài. Tổng quát, huấn luyện trước không giám sát là một kỹ thuật mạnh mẽ có thể được sử dụng để huấn luyện các mô hình ngôn ngữ chính xác hơn và khả năng khái

quát hóa tốt hơn so với các mô hình chỉ được huấn luyện bằng học có giám sát.

- Task-aware input transformations (Biến đổi đầu vào nhận biết nhiệm vụ): Có các loại nhiệm vụ khác nhau như suy diễn văn bản và hỏi-đáp yêu cầu một cấu trúc cụ thể. Ví dụ, suy diễn văn bản yêu cầu một tiền đề và một giả thuyết; hỏi-đáp yêu cầu một tài liệu ngữ cảnh, một câu hỏi và các câu trả lời có thể. Một trong những đóng góp của GPT-1 là chuyển đổi những loại nhiệm vụ này (yêu cầu đầu vào có cấu trúc) thành một đầu vào mà mô hình ngôn ngữ có thể phân tích, mà không cần các kiến trúc chuyên biệt cho từng nhiệm vụ trên nền kiến trúc đã được huấn luyện trước.
  Đối với suy diễn văn bản, tiền đề (premise)
  p và giả thuyết (hypothesis) *h* được nối với nhau bằng một ký tự phân tách ($) ở giữa - [p, $, h]. Đối với hỏi-đáp, tài liệu ngữ cảnh (context) *c* được nối với câu hỏi (question) *q* và một câu trả lời (answer) có thể có *a*, với một ký tự phân tách ở giữa câu hỏi và câu trả lời - [c, q, $, a].

GPT-1 đã vượt qua các mô hình trước đó trên nhiều tiêu chuẩn đánh giá, đạt được kết quả xuất sắc. Mặc dù GPT-1 là một bước đột phá quan trọng trong xử lý ngôn ngữ tự nhiên (NLP), nhưng nó vẫn có một số hạn chế. Ví dụ, mô hình có xu hướng tạo ra văn bản lặp lại, đặc biệt khi gặp các gợi ý nằm ngoài phạm vi dữ liệu mà nó đã được huấn luyện. Nó cũng không thể suy luận qua nhiều lượt hội thoại và không theo dõi được các phụ thuộc dài hạn trong văn bản. Ngoài ra, tính mạch lạc và trôi chảy của GPT-1 chỉ giới hạn ở các chuỗi văn bản ngắn, và các đoạn văn dài thường thiếu mạch lạc. Dù có những hạn chế này, GPT-1 đã chứng minh sức mạnh của việc huấn luyện trước không giám sát, đặt nền móng cho những mô hình lớn hơn và mạnh mẽ hơn dựa trên kiến trúc transformer.

BERT

BERT, viết tắt của **Bidirectional Encoder Representations from Transformers**, khác biệt với các mô hình transformer truyền thống encoder-decoder ở chỗ nó là một kiến trúc chỉ mã hóa (encoder-only). Thay vì dịch hoặc tạo chuỗi, BERT tập trung vào việc hiểu ngữ cảnh sâu bằng cách huấn luyện trên mục tiêu mô hình ngôn ngữ bị che (masked language model). Trong cấu hình này, các từ ngẫu nhiên trong câu được thay thế bằng một token [MASK], và BERT cố gắng dự đoán từ gốc dựa trên ngữ cảnh xung quanh. Một khía cạnh đổi mới khác trong phương pháp

huấn luyện của BERT là mất mát dự đoán câu tiếp theo (next sentence prediction loss), nơi mà mô hình học cách xác định xem một câu có hợp lý khi theo sau một câu trước đó hay không.

Bằng cách huấn luyện trên các mục tiêu này, BERT nắm bắt các phụ thuộc ngữ cảnh phức tạp từ cả bên trái và bên phải của một từ, và có thể phân biệt mối quan hệ giữa các cặp câu. Những khả năng này khiến BERT đặc biệt phù hợp với các nhiệm vụ yêu cầu hiểu ngôn ngữ tự nhiên, như hỏi-đáp, phân tích cảm xúc và suy diễn ngôn ngữ tự nhiên, trong số những nhiệm vụ khác. Vì đây là một mô hình chỉ mã hóa, BERT không thể tạo văn bản.

GPT-2

GPT-2, người kế nhiệm của GPT-1, được OpenAI ra mắt vào năm 2019. Đổi mới chính của GPT-2 là việc tăng quy mô trực tiếp (direct scale-up), với số lượng tham số và kích thước tập dữ liệu huấn luyện tăng gấp mười lần:

- **Dữ liệu**: GPT-2 được huấn luyện trên một tập dữ liệu lớn (40GB) và đa dạng có tên là WebText, bao gồm 45 triệu trang web từ Reddit với điểm Karma ít nhất là 3. Karma là một thước đo đánh giá trên Reddit, và giá trị 3 cho thấy tất cả các bài đăng đều có chất lượng ở mức hợp lý.
- **Tham số**: GPT-2 có 1,5 tỷ tham số, lớn hơn một bậc so với mô hình trước đó. Số tham số nhiều hơn làm tăng khả năng học của mô hình. Các tác giả đã huấn luyện bốn mô hình ngôn ngữ với 117M (bằng với GPT-1), 345M, 762M, và 1.5B (GPT-2) tham số, và nhận thấy rằng mô hình với nhiều tham số nhất hoạt động tốt hơn ở mọi nhiệm vụ sau đó.

Việc mở rộng này tạo ra một mô hình có khả năng tạo văn bản mạch lạc và chân thực hơn so với GPT-1. Khả năng tạo ra các phản hồi giống con người đã khiến GPT-2 trở thành một công cụ có giá trị cho nhiều nhiệm vụ xử lý ngôn ngữ tự nhiên, chẳng hạn như sáng tạo nội dung và dịch thuật. Cụ thể, GPT-2 đã cho thấy cải thiện rõ rệt trong việc nắm bắt các phụ thuộc dài hạn và lý luận thông thường.

Dù hoạt động tốt ở một số nhiệm vụ, GPT-2 không vượt qua được các mô hình tiên tiến nhất trong đọc hiểu, tóm tắt và dịch thuật. Thành tựu đáng kể nhất của GPT-2 là khả năng thực hiện học không cần ví dụ (zero-shot learning) trên nhiều nhiệm vụ. Zero-shot task transfer là khả năng của một mô hình tổng quát hóa sang một nhiệm vụ mới mà không được huấn luyện trước đó, yêu cầu mô hình phải hiểu

nhiệm vụ dựa trên hướng dẫn cho sẵn. Ví dụ, trong nhiệm vụ dịch từ tiếng Anh sang tiếng Đức, mô hình có thể được cung cấp một câu tiếng Anh, tiếp theo là từ “German” và một promt (":"). Mô hình sau đó sẽ phải hiểu rằng đây là nhiệm vụ dịch và tạo ra bản dịch tiếng Đức của câu tiếng Anh.

GPT-2 có thể thực hiện các nhiệm vụ như dịch máy, tóm tắt văn bản, và đọc hiểu mà không cần bất kỳ sự giám sát nào. Nghiên cứu phát hiện rằng hiệu suất trong các nhiệm vụ zero-shot tăng lên theo dạng log-tuyến tính khi khả năng của mô hình tăng lên. GPT-2 đã cho thấy rằng huấn luyện trên tập dữ liệu lớn hơn và có nhiều tham số hơn đã cải thiện khả năng của mô hình trong việc hiểu các nhiệm vụ và vượt qua các mô hình tiên tiến nhất ở nhiều nhiệm vụ trong bối cảnh zero-shot.

GPT-3/3.5/4

GPT-3, phiên bản thứ ba của mô hình Generative Pre-trained Transformer, đại diện cho một bước tiến hóa đáng kể so với người tiền nhiệm GPT-2, chủ yếu về quy mô, khả năng và tính linh hoạt. Sự khác biệt đáng chú ý nhất là quy mô khổng lồ của GPT-3, với 175 tỷ tham số, so với mô hình lớn nhất của GPT-2 chỉ có 1,5 tỷ tham số. Sự gia tăng kích thước mô hình này cho phép GPT-3 lưu trữ và nhớ lại một lượng thông tin khổng lồ hơn, hiểu các hướng dẫn tinh tế và tạo ra văn bản mạch lạc và phù hợp với ngữ cảnh hơn trên các đoạn văn dài hơn.

Trong khi GPT-2 có thể được fine-tuned trên các nhiệm vụ cụ thể với dữ liệu đào tạo bổ sung, GPT-3 có thể hiểu và thực hiện các nhiệm vụ chỉ với một vài ví dụ, hoặc đôi khi thậm chí không cần bất kỳ ví dụ rõ ràng nào - chỉ dựa trên hướng dẫn được cung cấp. Điều này nhấn mạnh khả năng hiểu và thích ứng linh hoạt hơn của GPT-3, giảm nhu cầu tinh chỉnh cụ thể cho từng nhiệm vụ, vốn phổ biến hơn trong GPT-2.

Cuối cùng, quy mô mô hình lớn và tập dữ liệu đào tạo đa dạng của GPT-3 đã dẫn đến khả năng tổng quát hóa tốt hơn trên một loạt các nhiệm vụ rộng hơn. Điều này có nghĩa là ngay từ đầu, không cần đào tạo thêm, GPT-3 thể hiện hiệu suất cải thiện trên các thách thức NLP đa dạng, từ dịch thuật đến hỏi đáp, so với GPT-2. Cũng cần lưu ý rằng cách tiếp cận phát hành khác nhau: trong khi OpenAI ban đầu đã giữ lại GPT-2 do lo ngại về lạm dụng, họ đã chọn cung cấp GPT-3 dưới dạng API thương mại, phản ánh cả tính hữu dụng của nó và quan điểm đang phát triển của tổ chức về triển khai.

Instruction tuning (Điều chỉnh theo hướng dẫn) đã được giới thiệu với InstructGPT, một phiên bản của GPT-3, sử dụng Supervised Fine-Tuning, trên một tập dữ liệu gồm các ví dụ về hành vi mong muốn của mô hình. Sau đó, đầu ra của mô hình này được xếp hạng và tiếp tục được điều chỉnh tinh bằng cách sử dụng Reinforcement Learning (Học tăng cường) từ Phản hồi của Con người. Điều này dẫn đến việc cải thiện khả năng tuân theo hướng dẫn của mô hình. Một mô hình InstructGPT 1,3 tỷ tham số có đánh giá của con người tốt hơn mô hình GPT-3 175 tỷ tham số. Nó cũng cho thấy sự cải thiện về tính trung thực (truthfulness) và giảm độc hại (toxicity).

Các mô hình GPT-3.5, bao gồm GPT-3.5 turbo, cải thiện so với GPT-3 vì nó có khả năng hiểu và tạo ra code. Nó được tối ưu hóa cho các cuộc đối thoại. Và nó có khả năng nhận các cửa sổ ngữ cảnh (context windows) lên đến 16.385 token và có thể tạo ra đầu ra lên đến 4.096 token.

GPT-4 mở rộng GPT-3.5 thành một mô hình đa phương thức lớn có khả năng xử lý đầu vào hình ảnh và văn bản và tạo ra đầu ra văn bản. Mô hình này có kiến thức chung rộng hơn và khả năng lập luận nâng cao. Nó có thể nhận các cửa sổ ngữ cảnh lên đến 128.000 token và có đầu ra tối đa là 4.096 token. GPT-4 thể hiện sự linh hoạt đáng kể bằng cách giải quyết các nhiệm vụ phức tạp trên nhiều lĩnh vực khác nhau như toán học, lập trình, thị giác máy tính, y học, luật và tâm lý học - tất cả đều không cần hướng dẫn chuyên biệt. Hiệu suất của nó thường ngang bằng hoặc thậm chí vượt quá khả năng của con người và vượt trội đáng kể so với các mô hình trước đó như GPT-3.5.

LaMDA

LaMDA của Google, viết tắt của 'Language Model for Dialogue Applications', là một đóng góp khác vào lĩnh vực các mô hình ngôn ngữ quy mô lớn, được thiết kế chủ yếu để tham gia vào các cuộc trò chuyện mở. Khác với các chatbot truyền thống hoạt động trong các lĩnh vực hạn chế và đã được định sẵn, LaMDA được thiết kế để xử lý một loạt các chủ đề, mang đến các cuộc trò chuyện tự nhiên và mạch lạc hơn. LaMDA được huấn luyện trên dữ liệu tập trung vào đối thoại để khuyến khích sự liên tục trong cuộc trò chuyện, không chỉ là những phản hồi rời rạc, giúp người dùng có thể có những cuộc đối thoại sâu rộng và khám phá.

Trong khi các mô hình GPT, đặc biệt là các phiên bản sau này như GPT-3, đã nỗ lực để giải quyết nhiều nhiệm vụ cùng lúc, từ tạo văn bản đến viết code, trọng tâm

chính của LaMDA là duy trì và nâng cao độ sâu và độ rộng của cuộc trò chuyện. Các mô hình GPT nổi bật với khả năng tạo ra nội dung dài mạch lạc và thực hiện các nhiệm vụ khác nhau với ít sự hướng dẫn, trong khi LaMDA nhấn mạnh vào sự lưu loát và sự phát triển của cuộc đối thoại, cố gắng mô phỏng sự khó đoán và sự phong phú của các cuộc trò chuyện giữa con người.

Gopher

Gopher là một mô hình ngôn ngữ với 280 tỷ tham số, dựa trên kiến trúc transformer chỉ sử dụng bộ giải mã (decoder-only), được phát triển bởi DeepMind vào năm 2021. Gopher có khả năng tạo văn bản, dịch ngôn ngữ, viết các loại nội dung sáng tạo khác nhau và trả lời câu hỏi của bạn một cách thông tin. Tương tự như GPT-3, Gopher tập trung vào việc cải thiện chất lượng bộ dữ liệu và các kỹ thuật tối ưu hóa:

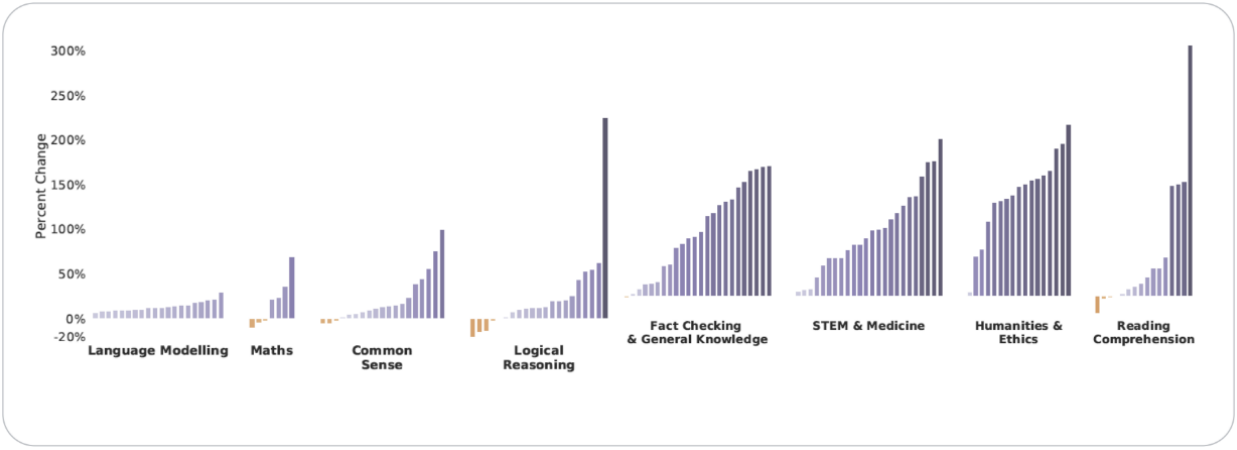
• **Bộ dữ liệu**: Các nhà nghiên cứu đã xây dựng một bộ dữ liệu văn bản chất lượng cao có tên là MassiveText, chứa hơn 10 terabyte dữ liệu và 2.45 tỷ tài liệu từ các trang web, sách, bài báo tin tức và mã nguồn (GitHub). Họ chỉ huấn luyện trên 300 tỷ token, chiếm 12% bộ dữ liệu. Quan trọng là, họ đã cải thiện chất lượng dữ liệu bằng cách lọc, ví dụ như loại bỏ văn bản trùng lặp và loại bỏ các tài liệu tương tự. Điều này đã cải thiện đáng kể hiệu suất của mô hình trên các tác vụ tiếp theo.

• **Tối ưu hóa**: Các nhà nghiên cứu đã sử dụng warmup learning rate trong 1,500 steps và sau đó làm giảm nó bằng cosine schedule. Họ cũng có một quy tắc thú vị là **khi tăng kích thước mô hình, họ giảm learning rate và tăng số lượng token trong mỗi batch**. Ngoài ra, họ nhận thấy rằng việc clipping gradient tối đa là 1, dựa trên global gradient norm, giúp ổn định quá trình huấn luyện.

Gopher đã được đánh giá trên nhiều nhiệm vụ khác nhau, bao gồm toán học, lẽ thường, lý luận logic, kiến thức chung, hiểu biết khoa học, đạo đức và hiểu bài đọc. Gopher đã vượt trội hơn các mô hình tiên tiến trước đó trên 81% các nhiệm vụ. Cụ thể, Gopher hoạt động tốt trên các nhiệm vụ yêu cầu kiến thức chuyên sâu nhưng gặp khó khăn với những nhiệm vụ đòi hỏi lý luận phức tạp, chẳng hạn như đại số trừu tượng (abstract algebra).

Các tác giả cũng đã thực hiện một nghiên cứu về tác động của kích thước mô hình đối với các loại nhiệm vụ khác nhau. Hình 4 cho thấy kết quả của nghiên cứu cắt bỏ này. Cụ thể, các tác giả nhận thấy rằng việc tăng số lượng tham số có ảnh hưởng lớn đến khả năng lý luận logic và hiểu bài đọc, nhưng không cải thiện hiệu

suất nhiều đối với các nhiệm vụ như kiến thức chung, nơi mà hiệu suất cuối cùng gần như không thay đổi.



Hình 4. Nghiên cứu cắt bỏ về tác động của kích thước mô hình đến hiệu suất của Gopher trên các loại nhiệm vụ khác nhau

GLaM

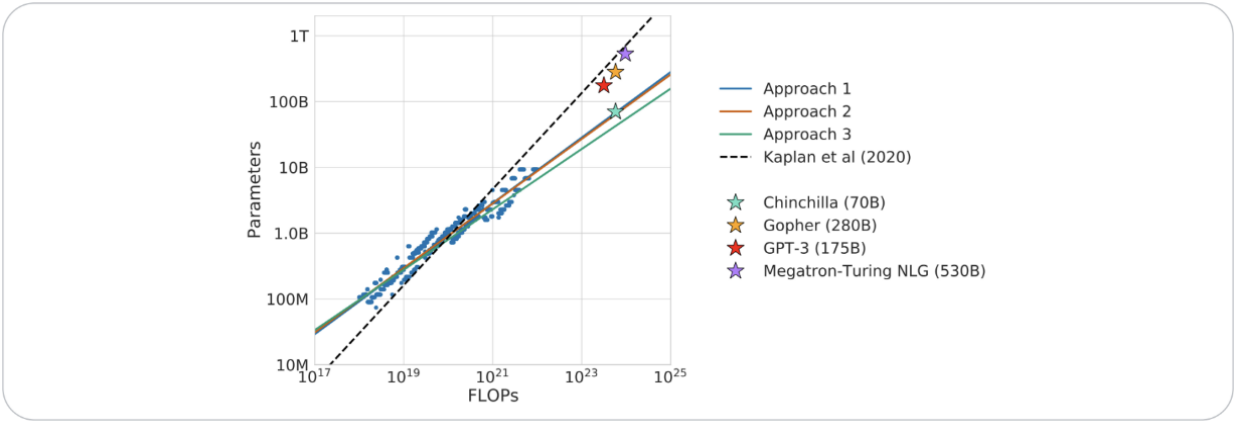
GLaM (Generalist Language Model) là mô hình ngôn ngữ đầu tiên sử dụng phương pháp **mixture-of-experts** với sparsely-activated (kích hoạt thưa). Các mô hình **mixture-of-experts** có hiệu quả tính toán cao hơn rất nhiều so với các mô hình truyền thống về số lượng tham số. Điều này đạt được nhờ vào việc chỉ kích hoạt một tập con của các tham số (tức là các "experts - chuyên gia") cho mỗi token đầu vào. GLaM bao gồm 1,2 nghìn tỷ tham số nhưng chỉ sử dụng 1/3 năng lượng so với GPT-3 để huấn luyện và chỉ tốn một nửa số FLOPs trong quá trình inference (suy luận), trong khi vẫn đạt được hiệu suất tổng thể tốt hơn so với GPT-3.

Chinchilla

Cho đến năm 2022, các mô hình ngôn ngữ lớn (LLMs) chủ yếu được mở rộng bằng cách **tăng kích thước mô hình và sử dụng các bộ dữ liệu tương đối nhỏ** so với các tiêu chuẩn hiện nay (lên đến 300 tỷ token đối với các mô hình lớn nhất). Cách tiếp cận này dựa trên nghiên cứu của Kaplan et al., nghiên cứu về hiệu suất của mô hình ngôn ngữ, đo bằng tổn thất cross-entropy, thay đổi khi ngân sách tính toán, kích thước mô hình và kích thước bộ dữ liệu thay đổi. Cụ thể, với việc tăng gấp 100 lần tài nguyên tính toán (C), Kaplan et al. khuyến nghị tăng kích thước mô hình lên

khoảng 28,8 lần ($N_{opt} \propto C^{0.73}$), trong khi chỉ tăng kích thước bộ dữ liệu 3,5 lần ($D_{opt} \propto C^{0.27}$).

Bài báo Chinchilla đã xem xét lại các quy luật mở rộng tối ưu về tính toán và sử dụng ba phương pháp khác nhau để phát hiện rằng việc mở rộng gần như đồng đều giữa tham số và dữ liệu là tối ưu khi tăng cường tính toán. Vì vậy, việc tăng cường tính toán gấp 100 lần nên dẫn đến việc tăng cả kích thước dữ liệu và kích thước mô hình gấp 10 lần.



Hình 5. Dự đoán chồng lên nhau (Overlaid) từ ba phương pháp khác nhau trong bài báo Chinchilla, cùng với các dự đoán từ nghiên cứu của Kaplan et al.

Để xác minh quy luật mở rộng cập nhật, DeepMind đã huấn luyện một mô hình 70 tỷ tham số (gọi là Chinchilla) sử dụng cùng một ngân sách tính toán như mô hình Gopher đã được huấn luyện trước đó. Chinchilla đã vượt trội và đáng kể so với Gopher (280B), GPT-3 (175B), và Megatron-Turing NLG (530B) trên một loạt các nhiệm vụ đánh giá tiếp theo. Vì Chinchilla nhỏ hơn Gopher 4 lần, cả dung lượng bộ nhớ và chi phí suy luận của Chinchilla cũng thấp hơn.

Những phát hiện của Chinchilla đã có những tác động quan trọng đối với sự phát triển của các LLM trong tương lai. Trọng tâm chuyển sang việc tìm ra cách mở rộng kích thước bộ dữ liệu (đồng thời duy trì chất lượng) cùng với việc tăng số lượng tham số. Việc dự đoán theo xu hướng này gợi ý rằng kích thước bộ dữ liệu huấn luyện có thể sớm bị giới hạn bởi lượng dữ liệu văn bản có sẵn. Điều này đã dẫn đến các nghiên cứu mới của Muennighoff et al. về các quy luật mở rộng trong các môi trường hạn chế dữ liệu.

PaLM

Pathways language model (PaLM) là một mô hình ngôn ngữ lớn dựa trên transformer với 540 tỷ tham số, được phát triển bởi Google AI. PaLM được huấn luyện trên một bộ dữ liệu khổng lồ gồm văn bản và code, và có khả năng thực hiện một loạt các nhiệm vụ, bao gồm lý luận lẽ thường, lý luận toán học, giải thích câu đùa, tạo code, và dịch ngôn ngữ.

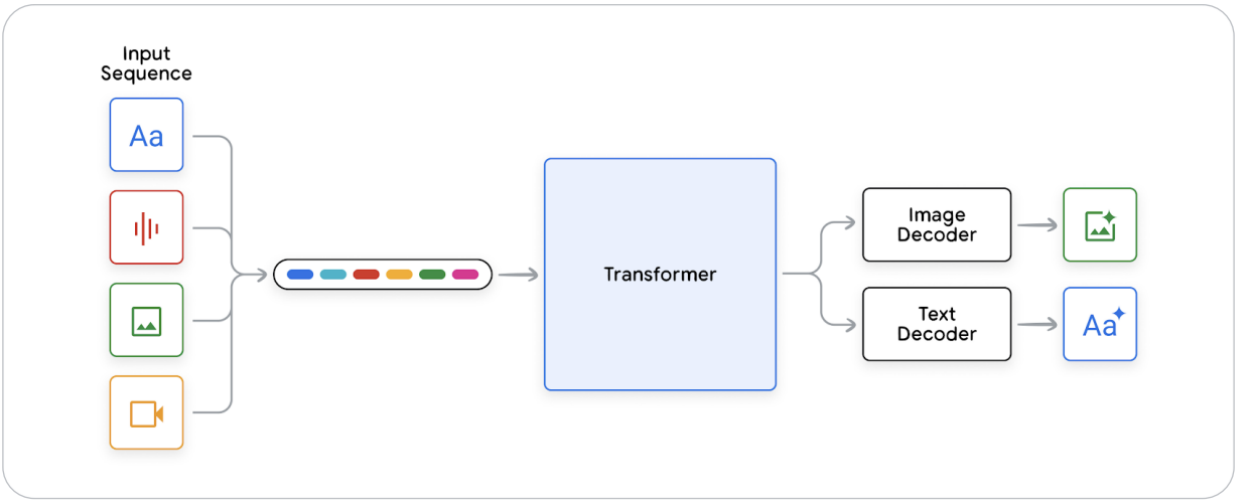
Khi ra mắt, PaLM đã đạt được hiệu suất tiên tiến nhất trong nhiều chuẩn mực ngôn ngữ, ví dụ như GLUE và SuperGLUE. Một trong những tính năng quan trọng của PaLM là khả năng mở rộng hiệu quả, điều này đạt được nhờ vào hệ thống **Pathways**, mà Google phát triển để phân phối quá trình huấn luyện các mô hình ngôn ngữ lớn qua hai **TPU v4 Pods**.

PaLM 2

PaLM 2 là thế hệ kế tiếp của PaLM, được công bố vào tháng 5 năm 2023. Nhờ vào một số cải tiến về kiến trúc và huấn luyện, PaLM 2 mạnh mẽ hơn PaLM, với ít tham số hơn. PaLM 2 nổi bật trong các nhiệm vụ lý luận nâng cao, bao gồm tạo code, toán học, phân loại, trả lời câu hỏi và dịch ngôn ngữ.

PaLM 2 cũng được chứng minh là hiệu quả hơn PaLM và trở thành nền tảng cho một số mô hình thương mại mà Google phát hành như một phần của Google Cloud Generative AI.

Gemini



Hình 6. Gemini có thể nhận các đầu vào đa phương thức bao gồm văn bản, âm thanh, hình ảnh và dữ liệu video. Tất cả các dữ liệu này được mã hóa thành các token và đưa vào mô hình **transformer** của nó. Mô hình **transformer** sẽ tạo ra đầu ra có thể chứa cả hình ảnh và văn bản.

Gemini là một dòng mô hình ngôn ngữ đa phương thức tiên tiến, có thể nhận các chuỗi dữ liệu đầu vào kết hợp từ văn bản, hình ảnh, âm thanh và video. Được xây dựng trên nền tảng các bộ giải mã transformer, Gemini có các cải tiến về kiến trúc để mở rộng quy mô cũng như tối ưu hóa quá trình suy luận trên các **TPU** của Google. Trong phiên bản hiện tại **Gemini 1.5**, các mô hình này được huấn luyện để hỗ trợ các ngữ cảnh có kích thước khác nhau, lên đến 2 triệu token trong phiên bản **Gemini 1.5 Pro** trên **Vertex AI**, và sử dụng các cơ chế như **multi-query attention** để nâng cao hiệu quả.

Gemini cũng sử dụng kiến trúc **Mixture of Experts** (MoE) để tối ưu hóa hiệu suất và khả năng của các mô hình. Khả năng xử lý đa phương thức cho phép các mô hình này làm việc với văn bản, hình ảnh và video đầu vào, và dự kiến sẽ mở rộng sang các phương thức đầu ra trong tương lai.

Các mô hình Gemini được huấn luyện trên các bộ xử lý **TPUv5e** và **TPUv4** của Google, tùy thuộc vào kích thước và cấu hình của mô hình. Dữ liệu huấn luyện bao gồm tài liệu web, sách, mã nguồn, cũng như dữ liệu hình ảnh, âm thanh và video.

Các **mô hình lớn hơn được huấn luyện với số lượng token tối ưu về tính toán** theo phương pháp đã được mô tả trong bài báo **Chinchilla**, trong khi các **mô hình nhỏ hơn được huấn luyện với nhiều token hơn mức tối ưu** để cải thiện hiệu suất cho ngân sách suy luận nhất định.

Dòng mô hình Gemini được tối ưu hóa cho các kích thước khác nhau: **Gemini Ultra**, **Gemini Pro**, **Gemini Nano** và **Gemini Flash**. **Gemini Ultra** được sử dụng cho các tác vụ phức tạp và đạt được kết quả tiên tiến nhất trong 30 trên 32 tác vụ chuẩn. **Gemini Pro** hỗ trợ triển khai quy mô lớn, còn **Gemini Nano** được thiết kế cho các ứng dụng trên thiết bị. Các mô hình **Gemini Nano** sử dụng các kỹ thuật như **distillation** để đạt hiệu suất tiên tiến cho các mô hình ngôn ngữ nhỏ trong các tác vụ như tóm tắt và hiểu bài đọc. Nhờ vào khả năng đa phương thức của mình, việc huấn luyện trên nhiều phương thức đầu vào giúp Gemini đạt được khả năng mạnh mẽ trong từng lĩnh vực.

Gemini 1.5 Pro là mô hình mới nhất trong gia đình Gemini, được giới thiệu vào đầu năm 2024, là một mô hình **mixture-of-experts** đa phương thức hiệu quả về tính toán. Mô hình này cũng đã tăng đáng kể kích thước cửa sổ ngữ cảnh (context window) lên đến hàng triệu token và có khả năng nhớ và lý luận trên các token này, bao gồm nhiều tài liệu dài và hàng giờ video và âm thanh. **Gemini 1.5 Pro** thể hiện khả năng đáng kinh ngạc trong nhiều lĩnh vực:

- **Hiểu mã nguồn (code)**: Có thể xử lý các codebases và trả lời các câu hỏi liên quan đến code rất cụ thể.
- **Học ngôn ngữ**: Mô hình có thể học các ngôn ngữ mới chưa từng thấy trong quá trình huấn luyện chỉ dựa trên tài liệu tham khảo cung cấp trong đầu vào.
- **Lý luận đa phương thức**: Hiểu được hình ảnh và văn bản, cho phép mô hình xác định một cảnh nổi tiếng từ tiểu thuyết "Les Misérables" chỉ từ một bản phác thảo đơn giản.
- **Hiểu video**: Có thể phân tích toàn bộ bộ phim, trả lời các câu hỏi chi tiết và xác định chính xác các dấu mốc thời gian.

Gemini 1.5 Pro thể hiện khả năng vượt trội trong việc tìm kiếm thông tin từ các tài liệu dài. Trong nghiên cứu của mình, nó đã đạt được **100%** tỷ lệ hồi tưởng trên các tài liệu dài đến 530,000 token, và trên **99,7%** trên các tài liệu lên đến 1 triệu token. Đặc biệt, nó duy trì **99,2%** độ chính xác khi tìm kiếm thông tin trong các tài liệu dài tới 10 triệu token.

Ngoài ra, **Gemini 1.5 Pro** còn thể hiện một bước tiến lớn trong khả năng tuân theo các hướng dẫn phức tạp. Trong một bài kiểm tra nghiêm ngặt với 406 yêu cầu nhiều bước, mô hình này đã vượt trội so với các mô hình Gemini trước đó. Mô hình

tuân theo chính xác gần **90%** các yêu cầu và hoàn thành đầy đủ **66%** các nhiệm vụ phức tạp.

Gemini Flash là một bổ sung mới trong gia đình mô hình Gemini, và là mô hình nhanh nhất trong API Gemini. Nó được tối ưu hóa cho các tác vụ có khối lượng lớn và tần suất cao, tiết kiệm chi phí hơn khi triển khai, và có context window dài đột phá lên đến **1 triệu token**. Mặc dù là mô hình nhẹ hơn so với **Gemini 1.5 Pro**, nhưng **Gemini Flash** vẫn cực kỳ mạnh mẽ trong lý luận đa phương thức trên lượng thông tin lớn và mang lại chất lượng ấn tượng cho kích thước của nó.

Cuối cùng, **Gemma** là một dòng mô hình nhẹ mới, được phát triển từ cùng một nghiên cứu và công nghệ tạo ra các mô hình Gemini. Mô hình **Gemma 1** có vốn từ vựng lên đến **256,000 từ** và đã được huấn luyện trên một bộ dữ liệu khổng lồ gồm **6 nghìn tỷ token**, là một sự bổ sung giá trị cho bộ sưu tập các mô hình ngôn ngữ mở. Phiên bản **2B tham số** của Gemma cũng đáng chú ý vì có thể chạy hiệu quả trên một GPU duy nhất.

Gemma 2, với **27 tỷ tham số**, được phát triển bởi Google AI, là một bước tiến lớn trong lĩnh vực mô hình ngôn ngữ mở. Với hiệu suất tương đương với các mô hình lớn hơn như **Llama 3 70B** trên các chuẩn benchmark, **Gemma 2** là công cụ mạnh mẽ và dễ tiếp cận cho nhiều nhà phát triển AI. Sự tương thích của nó với các công cụ tinh chỉnh đa dạng, từ các giải pháp trên đám mây đến các công cụ cộng đồng phổ biến, làm tăng thêm tính linh hoạt của nó.

Các mô hình mở khác

Lĩnh vực các mô hình ngôn ngữ lớn mở (LLMs) đang phát triển nhanh chóng, với ngày càng nhiều mô hình mà cả code và pre-trained weights (các trọng số đã được huấn luyện) đều có sẵn công khai. Dưới đây là một số ví dụ nổi bật:

- **LLaMA 2**: Được phát hành bởi Meta AI, **LLaMA 2** là một gia đình các mô hình LLM đã được pretrained and fine-tuned, với kích thước từ 7 tỷ đến 70 tỷ tham số. Nó cho thấy sự cải thiện đáng kể so với người tiền nhiệm **LLaMA 1**, bao gồm một bộ dữ liệu huấn luyện lớn hơn 40% (2 nghìn tỷ token), chiều dài ngữ cảnh gấp đôi (4096 token), và việc sử dụng **grouped-query attention**. Phiên bản tinh chỉnh **LLaMA 2-Chat** được tối ưu hóa cho đối thoại và có hiệu suất cạnh tranh với các mô hình đóng nguồn cùng kích thước.

- **LLaMA 3.2**: Cũng do Meta AI phát hành, **LLaMA 3.2** là thế hệ tiếp theo của các mô hình LLaMA mở. **LLaMA 3.2** bao gồm các mô hình ngôn ngữ đa ngôn ngữ chỉ có văn bản (1B, 3B) và mô hình ngôn ngữ thị giác (11B, 90B), với các phiên bản quantized của 1B và 3B có kích thước nhỏ hơn trung bình tới 56% và tốc độ nhanh gấp 2-3 lần, lý tưởng cho các ứng dụng trên thiết bị và edge deployments. **LLaMA 3.2** sử dụng **grouped-query attention** và bộ từ vựng 128K token để nâng cao hiệu suất và hiệu quả.
- **Mixtral**: Phát triển bởi Mistral AI, **Mixtral 8×7B** là một mô hình **Sparse Mixture of Experts (SMoE)**. Mặc dù tổng số tham số là 47B, nhưng mô hình này chỉ sử dụng 13B tham số hoạt động mỗi token trong quá trình suy luận, giúp tăng tốc suy luận và cải thiện hiệu suất. Mô hình này nổi bật trong các tác vụ toán học, tạo code và đa ngôn ngữ, thường vượt trội so với **LLaMA 2 70B** trong các lĩnh vực này. **Mixtral** cũng hỗ trợ chiều dài ngữ cảnh 32k token, cho phép xử lý các chuỗi dài hơn. Phiên bản tinh chỉnh **Mixtral 8×7B-Instruct** vượt trội so với một số mô hình closed-source trong các bài đánh giá của con người.
- **Qwen 1.5**: Dòng mô hình LLM của Alibaba có sáu kích thước: 0.5B, 1.8B, 4B, 7B, 14B và 72B. Các mô hình **Qwen 1.5** hỗ trợ chiều dài ngữ cảnh lên đến 32k token và có hiệu suất mạnh mẽ trên nhiều chuẩn mực. Đặc biệt, **Qwen 1.5-72B** vượt trội hơn **LLaMA 2-70B** trên tất cả các bài kiểm tra đã đánh giá, thể hiện khả năng xuất sắc trong việc hiểu ngôn ngữ, lý luận và toán học.
- **Yi**: Được tạo ra bởi 01.AI, dòng mô hình **Yi** bao gồm các mô hình cơ bản 6B và 34B đã được huấn luyện trên một bộ dữ liệu khổng lồ gồm 3.1 nghìn tỷ token tiếng Anh và tiếng Trung. Yi chú trọng vào chất lượng dữ liệu thông qua các quy trình làm sạch và lọc nghiêm ngặt. Mô hình **34B** đạt hiệu suất tương đương với **GPT-3.5** trên nhiều chuẩn mực và có thể được triển khai hiệu quả trên các GPU của người tiêu dùng với việc **quantization 4-bit**. Yi cũng cung cấp các mở rộng như mô hình ngữ cảnh 200k token, mô hình ngôn ngữ- thị giác (**Yi-VL**) và mô hình nâng cấp chiều sâu 9B.
- **Grok-1**: Phát triển bởi xAI, **Grok-1** là một mô hình **Mixture-of-Experts** với 314 tỷ tham số và 25% trọng số hoạt động trên một token nhất định. Đây là mô hình cơ sở chưa được tinh chỉnh cho các tác vụ cụ thể như đối thoại. **Grok-1** vận hành với chiều dài ngữ cảnh 8k token.

Nhịp độ đổi mới trong lĩnh vực LLMs đang diễn ra rất nhanh và không có dấu hiệu chậm lại. Có rất nhiều đóng góp cho lĩnh vực này từ cả môi trường học thuật và

thương mại. Với hơn 20,000 bài báo đã được xuất bản về LLMs trên **arxiv.org**, thật khó để liệt kê hết tất cả các mô hình và nhóm nghiên cứu đã đóng góp vào sự phát triển của LLMs. Tuy nhiên, một danh sách rút gọn các mô hình mở đáng chú ý có thể bao gồm **GPT-NeoX** và **GPT-J** của EleutherAI, **Alpaca** của Stanford, **Vicuna** của LMSYS, **Grok** của xAI, **Falcon** của TII, **PHI** của Microsoft, **NVLM** của Nvidia, **DBRX** của Databricks, **Qwen** của Alibaba, **Yi** của 01.ai, **LLaMA** của Meta đã đề cập ở trên và nhiều mô hình khác. Một số công ty nổi bật phát triển các mô hình nền tảng LLM thương mại bao gồm **Anthropic**, **Cohere**, **Character.ai**, **Reka**, **AI21**, **Perplexity**, **xAI** và nhiều công ty khác ngoài Google và OpenAI đã đề cập trong các phần trước.

Khi sử dụng một mô hình, điều quan trọng là phải xác nhận rằng giấy phép sử dụng phù hợp với mục đích của bạn, vì nhiều mô hình đi kèm với các điều khoản sử dụng rất cụ thể.

Comparison - So sánh

Trong phần này, chúng ta đã quan sát sự tiến hóa của các mô hình ngôn ngữ dựa trên **transformer**. Ban đầu, chúng được phát triển dưới dạng các kiến trúc **encoder-decoder** với hàng trăm triệu tham số, được huấn luyện trên hàng trăm triệu token, và đã phát triển thành các kiến trúc **decoder-only** khổng lồ với hàng tỷ tham số và được huấn luyện trên hàng nghìn tỷ token. **Bảng 1** cho thấy sự thay đổi của các siêu tham số quan trọng đối với tất cả các mô hình đã được thảo luận trong bài báo này qua thời gian. Việc mở rộng dữ liệu và tham số không chỉ giúp cải thiện hiệu suất của các **LLM** trên các tác vụ tiếp theo, mà còn dẫn đến các hành vi nổi lên và khả năng tổng quát **zero-shot** hoặc **few-shot** đối với các tác vụ mới. Tuy nhiên, ngay cả những mô hình **LLM** tốt nhất vẫn còn nhiều hạn chế. Ví dụ, chúng không thực sự giỏi trong việc tham gia vào các cuộc trò chuyện giống con người, khả năng toán học của chúng có hạn, và đôi khi chúng không hoàn toàn phù hợp với đạo đức của con người (chẳng hạn, chúng có thể bị thiên vị hoặc tạo ra các phản hồi độc hại (toxic)). Trong phần tiếp theo, chúng ta sẽ tìm hiểu cách nhiều vấn đề này đang được giải quyết.

	Model						
	Attention (2017)	GPT (2018)	GPT-2 (2019)	GPT-3 (2020)	LaMDA (2021)	Gopher (2021)	Chinchilla (2022)
Optimizer	ADAM	ADAM	ADAM	ADAM	ADAM	ADAM	ADAM-W
# Parameters	213M	117M	1.5B	175B	137B	280B	70B
Vocab size	~37K	~40K	~50K	~50K	~32K	~32K	~32K
Embedding dimension	1024	768	1600	12288	8192	16384	8192
Key dimension	64	64	64	128	128	128	128
# heads (H)	16	12	25	96	128	128	64
# encoder layers	6	N/A	N/A	N/A	N/A	N/A	N/A
# decoder layers	6	12	48	96	64	80	80
Feed forward dimension	4 * 1024	4 * 768	4 * 1600	4 * 12288	8 * 8192	4 * 16384	4 * 8192
Context Token Size	N/A	512	1024	2048	N/A	2048	2048
Pre-Training tokens	~160M[A]	~1.25B[A]	~10B	~300B	~168B	~300B	~1.4T

Bảng 1. Các siêu tham số quan trọng của các mô hình ngôn ngữ lớn dựa trên transformer

Fine-tuning LLM

Các mô hình ngôn ngữ lớn (Large Language Models - LLM) thường trải qua nhiều giai đoạn huấn luyện. Giai đoạn đầu tiên, thường được gọi là **pre-training**, là giai đoạn nền tảng, nơi một LLM được huấn luyện trên các tập dữ liệu văn bản lớn, đa dạng và không được gán nhãn, với nhiệm vụ dự đoán token tiếp theo dựa trên ngữ cảnh trước đó. Mục tiêu của giai đoạn này là tận dụng một phân phối dữ liệu lớn, tổng quát và xây dựng một mô hình có khả năng lấy mẫu hiệu quả từ phân phối này. Sau giai đoạn pre-training, LLM thường thể hiện khả năng hiểu và sinh ngôn ngữ ở mức độ hợp lý trên nhiều nhiệm vụ khác nhau, thường được kiểm tra thông qua **zero-shot** hoặc **few-shot prompting** (bổ sung hướng dẫn bằng một vài ví dụ hoặc minh họa). Pre-training là giai đoạn tốn kém nhất về thời gian (kéo dài từ vài tuần đến vài tháng tùy thuộc vào kích thước mô hình) và yêu cầu tài nguyên tính toán lớn (giờ GPU/TPU).

Sau khi hoàn thành giai đoạn huấn luyện ban đầu, mô hình có thể được tinh chỉnh chuyên sâu hơn thông qua **fine-tuning**, thường được gọi là **instruction-tuning**

hoặc đơn giản là **supervised fine-tuning (SFT)**. Quá trình SFT bao gồm việc huấn luyện LLM trên một tập dữ liệu minh họa cụ thể theo nhiệm vụ, đồng thời đo lường hiệu suất của mô hình trên các nhiệm vụ đặc thù trong từng lĩnh vực. Một số ví dụ về các hành vi có thể được cải thiện thông qua fine-tuning bao gồm:

- **Instruction-tuning/instruction following**: LLM được cung cấp các hướng dẫn đầu vào, chẳng hạn như tóm tắt một đoạn văn bản, viết mã, hoặc sáng tác một bài thơ theo phong cách nhất định.
- **Dialogue-tuning**: Đây là một trường hợp đặc biệt của instruction-tuning, trong đó LLM được tinh chỉnh dựa trên dữ liệu hội thoại dạng câu hỏi và phản hồi. Điều này thường được gọi là hội thoại đa lượt (**multi-turn dialogue**).
- **Safety tuning**: Đây là một bước quan trọng để giảm thiểu rủi ro liên quan đến bias, discrimination, and toxic outputs (thiên kiến, phân biệt đối xử và các đầu ra độc hại). Quá trình này bao gồm việc lựa chọn dữ liệu cẩn thận, kiểm tra thông qua vòng lặp có sự tham gia của con người (**human-in-the-loop validation**) và tích hợp các cơ chế an toàn. Các kỹ thuật như **reinforcement learning with human feedback (RLHF)** giúp LLM ưu tiên các phản hồi an toàn và đạo đức.

Fine-tuning ít tốn kém hơn đáng kể và hiệu quả hơn về dữ liệu so với pre-training. Có nhiều kỹ thuật tối ưu hóa chi phí được trình bày chi tiết trong tài liệu này.

Supervised fine-tuning

Như đã đề cập trong phần trước, SFT là quá trình cải thiện hiệu suất của mô hình ngôn ngữ lớn (LLM) trên một hoặc một nhóm nhiệm vụ cụ thể bằng cách tiếp tục huấn luyện trên dữ liệu được gán nhãn và chuyên biệt cho từng lĩnh vực.

Tập dữ liệu SFT thường nhỏ hơn đáng kể so với tập dữ liệu huấn luyện ban đầu và thường được tuyển chọn bởi con người để đảm bảo chất lượng cao.

Trong bối cảnh này, mỗi điểm dữ liệu bao gồm một đầu vào (**prompt**) và một kết quả mẫu (**target response**). Ví dụ: câu hỏi (**prompt**) và câu trả lời (**target response**), bản dịch từ một ngôn ngữ (**prompt**) sang ngôn ngữ khác (**target response**), một tài liệu để tóm tắt (**prompt**), và bản tóm tắt tương ứng (**target response**).

Điều quan trọng cần lưu ý là ngoài việc cải thiện hiệu suất trên các nhiệm vụ cụ thể, fine-tuning còn giúp mô hình cải thiện hành vi để trở nên an toàn hơn, ít toxic hơn, dễ hội thoại hơn, và tốt hơn trong việc thực hiện hướng dẫn.

Reinforcement learning from human feedback

Thông thường, sau khi thực hiện SFT, một giai đoạn fine-tuning thứ hai được gọi là **reinforcement learning from human feedback (RLHF)** sẽ được áp dụng. Đây là một kỹ thuật fine-tuning mạnh mẽ giúp mô hình LLM tốt hơn trong việc điều chỉnh các phản hồi theo hướng phù hợp với sở thích của con người (ví dụ: tạo ra các phản hồi hữu ích hơn, trung thực hơn, an toàn hơn, v.v.).

Khác với SFT, nơi LLM chỉ được tiếp xúc với các positive examples - ví dụ tích cực (dữ liệu minh họa chất lượng cao), RLHF cho phép tận dụng cả các đầu ra tiêu cực, qua đó phạt mô hình khi nó tạo ra các phản hồi không mong muốn. Việc phạt đầu ra tiêu cực làm giảm khả năng mô hình tạo ra các phản hồi không an toàn hoặc không hữu ích.

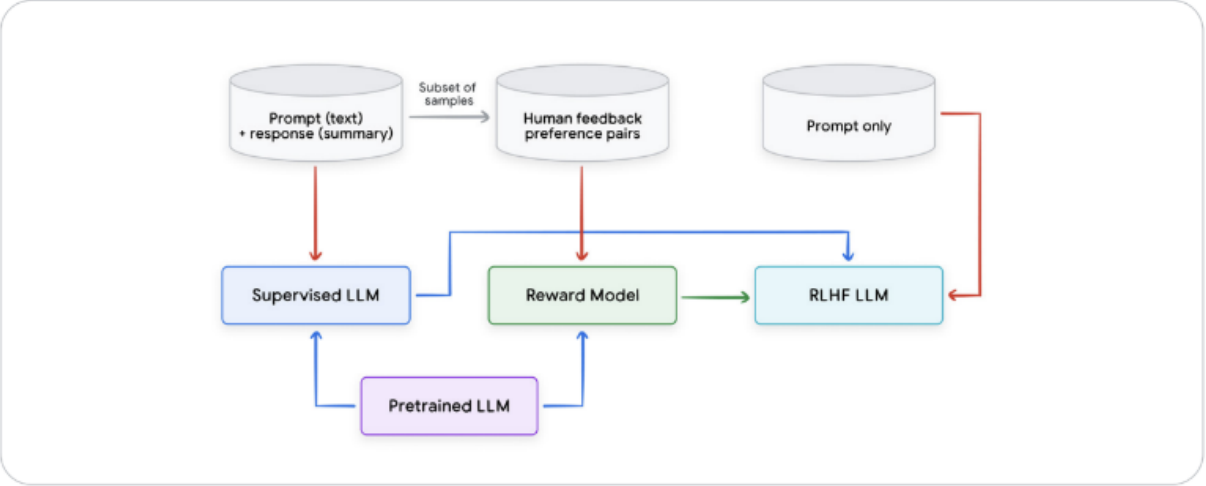


Figure 7. An example RLHF procedure

Để thực hiện RLHF, một **reward model (RM)** thường cần được huấn luyện với quy trình tương tự như trong **Figure 7**. RM thường được khởi tạo với một mô hình transformer đã được tiền huấn luyện, và thường cũng đã qua SFT. Sau đó, RM được tinh chỉnh trên dữ liệu sở thích của con người, bao gồm:

- Dữ liệu một chiều: chứa prompt, phản hồi và điểm số. (prompt, response, score)

- Hoặc dữ liệu cặp: bao gồm một prompt và một cặp response cùng với nhãn thể hiện phản hồi nào được ưa thích hơn.

Ví dụ, với hai bản tóm tắt A và B của cùng một bài viết, người đánh giá sẽ chọn bản tóm tắt được ưa thích dựa trên hướng dẫn chi tiết. Các nhãn sở thích này được gọi là **human feedback**. Các nhãn này có thể ở dạng nhị phân (ví dụ: "tốt" hoặc "xấu"), thang đo Likert, thứ tự xếp hạng (khi có hơn 2 ứng viên), hoặc đánh giá chi tiết hơn về chất lượng bản tóm tắt.

Tín hiệu sở thích này cũng có thể kết hợp nhiều chiều đánh giá, như: an toàn, hữu ích, công bằng, và trung thực.

Quy trình RLHF tiêu chuẩn (xem Figure 7) bao gồm:

1. Khởi tạo và tinh chỉnh RM trên các cặp sở thích.
2. Sử dụng RM trong một thuật toán **Reinforcement Learning (RL)** (như **policy gradient**) để tiếp tục tinh chỉnh LLM đã qua SFT, giúp mô hình tạo ra các phản hồi phù hợp hơn với sở thích của con người.

Để mở rộng quy mô RLHF, kỹ thuật **RL from AI Feedback (RLAIF)** tận dụng phản hồi từ AI thay vì phản hồi của con người để tạo nhãn sở thích. Ngoài ra, có thể loại bỏ nhu cầu huấn luyện RLHF bằng cách sử dụng các phương pháp như **Direct Preference Optimization (DPO)**. Cả RLHF và RLAIF đều có thể được triển khai trên Google Cloud.

Parameter Efficient Fine-Tuning

Cả **SFT** và **RLHF** đều rất tốn kém về thời gian tính toán và yêu cầu sử dụng các bộ accelerators (tăng tốc), đặc biệt khi tinh chỉnh toàn bộ các mô hình LLM với hàng tỷ tham số. May mắn thay, có những kỹ thuật hiệu quả và hữu ích giúp giảm đáng kể chi phí và tăng tốc quá trình fine-tuning so với pre-training và full fine-tuning. Một trong những nhóm phương pháp đó là các kỹ thuật **parameter efficient fine-tuning (PEFT)**.

Ở mức tổng quan, các phương pháp PEFT bổ sung một tập tham số nhỏ hơn đáng kể (ví dụ, chỉ vài nghìn tham số) để "điều chỉnh" trọng số của LLM đã được tiền huấn luyện. Việc điều chỉnh này giúp LLM được tinh chỉnh để thực hiện một nhiệm vụ hoặc một nhóm nhiệm vụ mới. Lợi ích của phương pháp này là chỉ cần huấn luyện một tập trọng số nhỏ hơn nhiều so với việc fine-tuning toàn bộ mô hình.

Một số kỹ thuật PEFT phổ biến bao gồm: **adapter-based fine-tuning**, **low-rank adaptation (LoRA)**, và **soft prompting**.

- **Adapter-based fine-tuning :** Kỹ thuật này sử dụng các module nhỏ, gọi là **adapters**, được thêm vào mô hình đã tiền huấn luyện. Chỉ các tham số của adapter được huấn luyện, do đó giảm đáng kể số lượng tham số cần tối ưu hóa so với SFT truyền thống.
- **Low-Rank Adaptation (LoRA):** LoRA giải quyết vấn đề hiệu quả (efficient**)** bằng cách sử dụng hai ma trận nhỏ để xấp xỉ cập nhật ma trận trọng số gốc, thay vì tinh chỉnh toàn bộ LLM. Kỹ thuật này giữ nguyên trọng số ban đầu và chỉ huấn luyện các ma trận cập nhật, từ đó giảm đáng kể yêu cầu tài nguyên mà chỉ làm tăng độ trễ suy luận ở mức tối thiểu.

  LoRA cũng có các biến thể cải tiến như **QLoRA**, sử dụng trọng số lượng tử hóa để tăng hiệu quả hơn nữa. Một ưu điểm nổi bật của các module LoRA là khả năng **plug-and-play**, nghĩa là bạn có thể huấn luyện một module LoRA chuyên biệt cho một nhiệm vụ và dễ dàng thay thế bằng một module LoRA khác được huấn luyện cho nhiệm vụ khác. Điều này cũng giúp việc chuyển giao mô hình trở nên dễ dàng hơn, vì người nhận chỉ cần có ma trận gốc và các ma trận cập nhật.
- **Soft prompting :** là một kỹ thuật dùng để điều chỉnh, điều kiện hóa các mô hình ngôn ngữ lớn (LLM) đã được **đóng băng** (**frozen**) thông qua các vector có thể học được thay vì sử dụng các prompt văn bản được thiết kế thủ công. Các vector này, được gọi là **soft prompts**, được tối ưu hóa trên dữ liệu huấn luyện và có thể chỉ bao gồm tối thiểu năm token, giúp giảm thiểu số lượng tham số cần thiết, đồng thời hỗ trợ suy luận trên nhiều nhiệm vụ khác nhau (**mixed-task inference**).

Đối với hầu hết các nhiệm vụ, **full fine-tuning** vẫn mang lại hiệu suất tốt nhất, sau đó là LoRA và soft prompting. Tuy nhiên, thứ tự này đảo ngược khi xét về chi phí. Cả ba phương pháp đều tiết kiệm bộ nhớ hơn so với fine-tuning truyền thống và đạt được hiệu suất tương đương.

Using large language models (Sử dụng các mô hình ngôn ngữ lớn (LLMs))

Prompt engineering (Kỹ thuật thiết kế prompt) và **sampling techniques** (phương pháp lấy mẫu) có ảnh hưởng mạnh mẽ đến hiệu suất của các mô hình ngôn ngữ lớn (*LLMs*). Prompt engineering là quá trình thiết kế và tinh chỉnh các đầu vào văn bản (*prompts*) mà bạn cung cấp cho LLM để đạt được đầu ra mong muốn và phù hợp. Các phương pháp lấy mẫu quyết định cách các *output tokens* được chọn, từ đó ảnh hưởng đến tính chính xác, tính sáng tạo và sự đa dạng của đầu ra. Tiếp theo, chúng tôi sẽ thảo luận các biến thể khác nhau của prompt engineering và các sampling techniques, cũng như đề cập đến một số tham số quan trọng có thể tác động đáng kể đến hiệu suất của LLM.

Prompt engineering

Các mô hình ngôn ngữ lớn (*LLMs*) rất mạnh mẽ, nhưng chúng vẫn cần được hướng dẫn để phát huy hết tiềm năng. Prompt engineering là một thành phần quan trọng trong việc định hướng LLM để đạt được các đầu ra mong muốn. Điều này có thể bao gồm việc định hướng mô hình để đưa ra các phản hồi thực tế hoặc khơi dậy khả năng sáng tạo của mô hình để kể một câu chuyện hoặc sáng tác một bài hát. Ví dụ về prompt engineering bao gồm cung cấp các hướng dẫn rõ ràng cho LLM, đưa ra các ví dụ, sử dụng từ khóa và định dạng để nhấn mạnh thông tin quan trọng, cung cấp thêm các chi tiết nền tảng, v.v.

Bạn thường sẽ nghe các thuật ngữ *zero-shot*, *few-shot* và *chain-of-thought prompting* trong ngữ cảnh của prompt engineering. Chúng tôi định nghĩa các thuật ngữ này dưới đây:

- **Few-shot prompting**: Đây là khi bạn cung cấp cho LLM một mô tả về nhiệm vụ, cũng như một vài (ví dụ: từ ba đến năm) ví dụ được chọn lọc kỹ lưỡng để giúp định hướng phản hồi của LLM. Ví dụ, bạn có thể cung cấp cho mô hình tên của một vài quốc gia và thủ đô của chúng, sau đó yêu cầu mô hình tạo ra thủ đô của một quốc gia mới không có trong các ví dụ.
- **Zero-shot prompting**: Đây là khi bạn cung cấp trực tiếp cho LLM một prompt cùng với các hướng dẫn. Bạn thường cung cấp cho LLM một mô tả nhiệm vụ, và LLM dựa nhiều vào kiến thức hiện có của mình để đưa ra phản hồi đúng. Phương pháp này không yêu cầu thêm dữ liệu hoặc ví dụ nào, do đó có tên là "Zero-shot", nhưng có thể kém tin cậy hơn so với *few-shot prompting*.

- **Chain-of-thought prompting**: Kỹ thuật này nhằm cải thiện hiệu suất trên các nhiệm vụ lý luận phức tạp. Thay vì chỉ đơn thuần hỏi LLM một câu hỏi, bạn cung cấp một prompt minh họa cách giải các vấn đề tương tự bằng cách suy luận từng bước. LLM sau đó tự tạo ra chuỗi suy nghĩ của mình cho vấn đề mới, phân chia thành các bước nhỏ hơn và giải thích quy trình lý luận của nó. Cuối cùng, nó cung cấp câu trả lời dựa trên quy trình lý luận này.

Prompt engineering là một lĩnh vực nghiên cứu đang phát triển mạnh mẽ.

Sampling Techniques and Parameters (Kỹ thuật Sampling và Các Tham số)

Một loạt các *sampling techniques* có thể được áp dụng để xác định cách mô hình chọn token tiếp theo trong một chuỗi. Chúng rất quan trọng trong việc kiểm soát chất lượng, tính sáng tạo, và sự đa dạng của đầu ra từ LLM. Dưới đây là phân tích các kỹ thuật sampling khác nhau và các tham số quan trọng của chúng:

- **Greedy search**: Lựa chọn token có xác suất cao nhất tại mỗi bước. Đây là lựa chọn đơn giản nhất nhưng có thể dẫn đến đầu ra lặp lại và dễ dự đoán.
- **Random sampling**: Lựa chọn token tiếp theo dựa theo phân phối xác suất, trong đó mỗi token được chọn tỷ lệ thuận với xác suất được dự đoán. Phương pháp này có thể tạo ra văn bản bất ngờ và sáng tạo hơn, nhưng cũng có khả năng cao sinh ra đầu ra phi logic.
- **Temperature sampling**: Điều chỉnh phân phối xác suất bằng một tham số *temperature*. *Temperature* cao thúc đẩy sự đa dạng, trong khi *temperature* thấp ưu tiên các token có xác suất cao.
- **Top-K sampling**: Lựa chọn ngẫu nhiên từ K token có xác suất cao nhất. Giá trị của K quyết định mức độ ngẫu nhiên trong quá trình chọn.
- **Top-P sampling (nucleus sampling)**: Lựa chọn từ một tập hợp động các token có tổng xác suất tích lũy bằng P. Phương pháp này cho phép mô hình điều chỉnh số lượng ứng viên tiềm năng tùy theo mức độ tự tin, ưu tiên đa dạng hơn khi không chắc chắn và tập trung vào một tập nhỏ các từ có xác suất cao khi tự tin.
- **Best-of-N sampling**: Tạo N phản hồi riêng biệt và chọn phản hồi tốt nhất theo một tiêu chí được xác định trước (ví dụ: mô hình phần thưởng hoặc kiểm tra

tính logic). Phương pháp này đặc biệt hữu ích cho các đoạn văn bản ngắn hoặc các tình huống yêu cầu logic và lập luận.

Bằng cách kết hợp prompt engineering với các kỹ thuật sampling và điều chỉnh đúng các siêu tham số (*hyperparameters*), bạn có thể tác động đáng kể đến phản hồi của LLM, giúp đầu ra phù hợp hơn, sáng tạo và nhất quán với nhu cầu cụ thể của bạn.

Đến đây, chúng ta đã tìm hiểu các loại kiến trúc LLM, công nghệ nền tảng, cũng như các phương pháp đào tạo, tinh chỉnh và điều chỉnh các mô hình này cho nhiều nhiệm vụ khác nhau. Tiếp theo, chúng ta sẽ xem xét một số nghiên cứu chính về cách tăng tốc quá trình giải mã trong LLM để tạo ra các phản hồi nhanh hơn.

Accelerating inference (Tăng tốc suy luận)

Scaling laws cho các mô hình ngôn ngữ lớn (*LLMs*), ban đầu được khám phá qua nghiên cứu của Kaplan và cộng sự, vẫn còn giá trị cho đến ngày nay. Các mô hình ngôn ngữ liên tục gia tăng kích thước, điều này đã góp phần trực tiếp vào việc cải thiện đáng kể chất lượng và độ chính xác của các mô hình này trong những năm gần đây. Việc tăng số lượng tham số đã nâng cao chất lượng của LLMs nhưng cũng đồng thời tăng tài nguyên tính toán cần thiết để vận hành chúng. Nhiều phương pháp đã được áp dụng nhằm cải thiện hiệu quả của LLMs cho các nhiệm vụ khác nhau, vì các nhà phát triển được khuyến khích giảm chi phí và độ trễ cho người dùng mô hình. Việc cân bằng chi phí triển khai mô hình về thời gian, tiền bạc và năng lượng được gọi là sự đánh đổi giữa chi phí và hiệu suất (cost-performance tradeoff), và thường cần điều chỉnh để phù hợp với từng trường hợp sử dụng cụ thể.

Hai tài nguyên chính được LLMs sử dụng là memory (bộ nhớ) và computation (khả năng tính toán). Các kỹ thuật nhằm cải thiện hiệu quả hoặc tốc độ suy luận chủ yếu tập trung vào hai tài nguyên này. Tốc độ kết nối giữa bộ nhớ và tính toán cũng rất quan trọng nhưng thường bị giới hạn bởi phần cứng. Khi kích thước của các LLMs đã tăng lên gấp 1000 lần, từ hàng triệu đến hàng tỷ tham số, số lượng tham số bổ sung này làm tăng cả kích thước bộ nhớ cần thiết để lưu trữ mô hình và khối lượng tính toán cần thiết để tạo ra kết quả của mô hình.

Với việc LLMs ngày càng được áp dụng cho các trường hợp sử dụng quy mô lớn và yêu cầu độ trễ thấp, tìm kiếm các phương pháp tối ưu hóa hiệu suất suy luận đã

trở thành một ưu tiên và là một chủ đề nghiên cứu sôi động với những tiến bộ đáng kể. Chúng ta sẽ khám phá một số phương pháp và đánh giá các sự đánh đổi liên quan để tăng tốc quá trình suy luận.

Trade offs (Sự đánh đổi)

Nhiều phương pháp tối ưu hóa suy luận hiệu quả cao yêu cầu đánh đổi một số yếu tố. Những đánh đổi này có thể được điều chỉnh tùy theo từng trường hợp, cho phép áp dụng các phương pháp được thiết kế phù hợp với các trường hợp sử dụng và yêu cầu suy luận khác nhau. Một số phương pháp tối ưu hóa được thảo luận sau đây nằm ở các mức độ khác nhau trên phổ của những sự đánh đổi này.

Việc đánh đổi một yếu tố này so với yếu tố khác (ví dụ: độ trễ so với chất lượng hoặc chi phí) không có nghĩa là hoàn toàn hy sinh yếu tố đó. Điều này chỉ đơn giản là chấp nhận một sự suy giảm nhỏ, không đáng kể về chất lượng, độ trễ hoặc chi phí để đổi lấy lợi ích cải thiện đáng kể ở một yếu tố khác.

The Quality vs Latency/Cost Tradeoff (Đánh đổi giữa chất lượng và độ trễ/chi phí)

Có thể cải thiện đáng kể tốc độ và chi phí suy luận bằng cách chấp nhận những sự giảm thiểu nhỏ, thậm chí không đáng kể, về độ chính xác của mô hình. Một ví dụ là sử dụng một mô hình nhỏ hơn để thực hiện nhiệm vụ. Một ví dụ khác là kỹ thuật *quantisation*, trong đó giảm độ chính xác của các tham số mô hình, từ đó dẫn đến các tính toán nhanh hơn và sử dụng ít bộ nhớ hơn.

Một điểm khác biệt quan trọng khi tiếp cận sự đánh đổi này là giữa khả năng mất chất lượng theo lý thuyết và khả năng thực tế của mô hình trong việc thực hiện nhiệm vụ mong muốn. Điều này phụ thuộc vào từng trường hợp sử dụng cụ thể, và việc khám phá sẽ thường dẫn đến những cải thiện đáng kể về tốc độ mà không ảnh hưởng đến chất lượng theo cách có ý nghĩa hoặc đáng chú ý. Ví dụ, nếu nhiệm vụ cần thực hiện đơn giản, thì một mô hình nhỏ hơn hoặc đã được *quantised* có khả năng thực hiện tốt. Giảm công suất tham số hoặc độ chính xác không đồng nghĩa với việc mô hình kém hiệu quả hơn ở nhiệm vụ cụ thể đó.

The Latency vs Cost Tradeoff (Đánh đổi giữa độ trễ và chi phí)

Một tên gọi khác cho sự đánh đổi này là đánh đổi giữa độ trễ và thông lượng (*throughput*), trong đó thông lượng đề cập đến khả năng của hệ thống xử lý nhiều yêu cầu một cách hiệu quả. Thông lượng tốt hơn trên cùng một phần cứng có nghĩa là chi phí suy luận LLM được giảm, và ngược lại.

Giống như các hệ thống phần mềm truyền thống, thường có nhiều cơ hội để đánh đổi giữa độ trễ và chi phí suy luận của LLM. Đây là một sự đánh đổi quan trọng vì suy luận LLM thường là thành phần chậm nhất và tốn kém nhất trong toàn bộ hệ thống. Việc cân bằng độ trễ và chi phí một cách có chủ ý là chìa khóa để đảm bảo rằng hiệu suất của LLM được điều chỉnh phù hợp với sản phẩm hoặc trường hợp sử dụng. Ví dụ, trong các trường hợp sử dụng suy luận hàng loạt (như gán nhãn ngoại tuyến), chi phí có thể là yếu tố quan trọng hơn độ trễ của bất kỳ yêu cầu cụ thể nào. Ngược lại, một sản phẩm chatbot dựa trên LLM sẽ đặt tầm quan trọng cao hơn vào độ trễ của yêu cầu.

Bây giờ khi đã đề cập đến một số sự đánh đổi quan trọng cần xem xét khi tối ưu hóa suy luận, chúng ta sẽ xem xét một số kỹ thuật tăng tốc suy luận hiệu quả nhất. Như đã thảo luận trong phần đánh đổi, một số kỹ thuật tối ưu hóa có thể ảnh hưởng đến đầu ra của mô hình. Vì vậy, các phương pháp này sẽ được chia thành hai loại: *output-approximating* (gần đúng đầu ra) và *output-preserving* (bảo toàn đầu ra).

Output-approximating methods (Các phương pháp gần đúng đầu ra)

Quantization (Lượng tử hóa)

Các mô hình ngôn ngữ lớn (LLMs) về cơ bản được cấu thành từ nhiều ma trận số (numerical matrices), hay còn gọi là trọng số mô hình (model weights). Trong quá trình suy luận, các phép toán ma trận được áp dụng lên các model weights để tạo ra các đầu ra số (activations). Quantization là quá trình giảm độ chính xác số mà các trọng số và đầu ra được lưu trữ, truyền tải và vận hành. Mặc định, các trọng số và đầu ra thường được biểu diễn dưới dạng số thực 32 bit, nhưng với quantization, độ chính xác có thể giảm xuống 8 bit hoặc thậm chí 4 bit số nguyên.

Quantization mang lại nhiều lợi ích về hiệu suất, bao gồm giảm dấu chân bộ nhớ (memory footprint) của mô hình, cho phép tải các mô hình lớn hơn trên cùng một phần cứng. Nó cũng giảm chi phí communication (truyền tải) của trọng số và đầu

ra trong một chip hoặc giữa các chip trong thiết lập suy luận phân tán, qua đó tăng tốc độ suy luận vì communication (viêc truyền tải) là một nguyên nhân chính gây độ trễ. Ngoài ra, giảm độ chính xác của trọng số/đầu ra có thể cho phép các phép toán số nhanh hơn vì một số phần cứng tăng tốc (ví dụ: TPUs/GPUs) hỗ trợ các phép nhân ma trận nhanh hơn cho một số biểu diễn độ chính xác thấp.

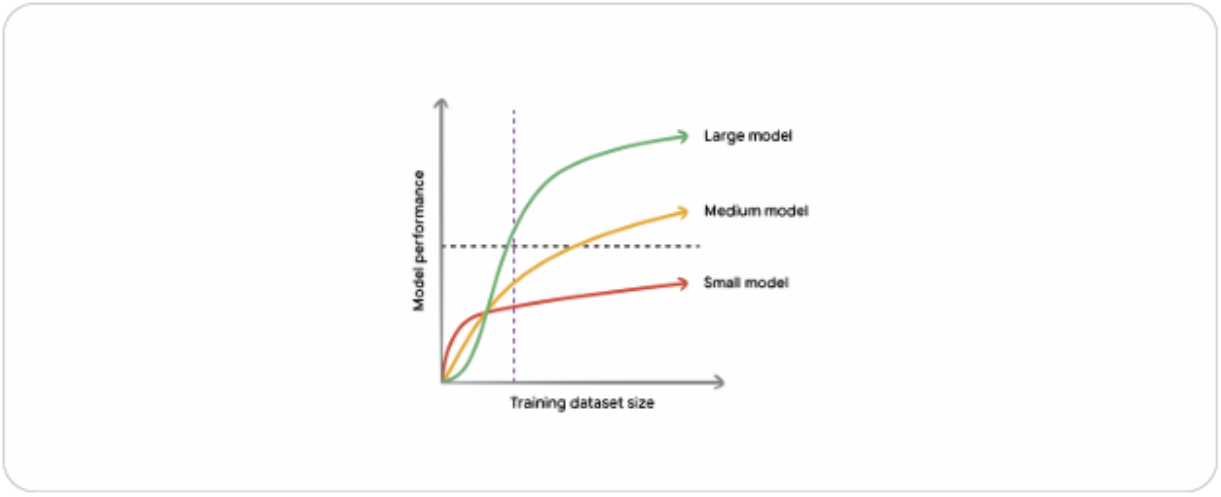
Tác động của quantization lên chất lượng có thể rất nhỏ hoặc không đáng kể tùy thuộc vào trường hợp sử dụng và mô hình. Hơn nữa, trong những trường hợp mà quantization có thể gây ra suy giảm chất lượng, sự suy giảm này thường nhỏ so với lợi ích về hiệu suất, cho phép đạt được một sự đánh đổi hiệu quả giữa chất lượng và độ trễ/chi phí (Quality vs Latency/Cost Tradeoff). Ví dụ, Benoit Jacob và cộng sự đã báo cáo rằng tốc độ tăng gấp 2 lần với mức giảm 2% độ chính xác trong nhiệm vụ phát hiện khuôn mặt (FaceDetection) trên MobileNet SSD.

Quantization có thể được áp dụng như một hoạt động chỉ dành cho suy luận (inference-only operation) hoặc được tích hợp vào quá trình huấn luyện, được gọi là Quantisation Aware Training (QAT). QAT thường được coi là một phương pháp bền vững hơn vì mô hình có khả năng phục hồi một phần các tổn thất chất lượng liên quan đến lượng tử hóa trong quá trình huấn luyện. Để đảm bảo sự đánh đổi tốt nhất giữa chi phí và chất lượng, cần điều chỉnh chiến lược lượng tử hóa, chẳng hạn như lựa chọn các độ chính xác khác nhau cho trọng số so với đầu ra, và áp dụng lượng tử hóa ở mức độ chi tiết như theo kênh hoặc theo nhóm (channel or group-wise).

Distillation (Chưng cất kiến thức)

Sử dụng một mô hình nhỏ hơn để thực hiện một nhiệm vụ là một trong những kỹ thuật tối ưu hóa suy luận hiệu quả nhất. Tuy nhiên, các mô hình nhỏ hơn có thể bị suy giảm đáng kể về chất lượng so với các mô hình lớn hơn. *Distillation* là một tập hợp các kỹ thuật huấn luyện nhằm cải thiện chất lượng của một mô hình nhỏ hơn (*student model*) bằng cách sử dụng một mô hình lớn hơn (*teacher model*). Phương pháp này hiệu quả vì các mô hình lớn thường vượt trội hơn các mô hình nhỏ ngay cả khi cả hai được huấn luyện trên cùng một tập dữ liệu, chủ yếu nhờ vào dung lượng tham số (*parametric capacity*) và động lực học huấn luyện (*training dynamics*). Khoảng cách về hiệu suất giữa hai loại mô hình này tiếp tục tăng khi kích thước tập dữ liệu huấn luyện tăng lên, như minh họa trong Hình 8.

Đáng chú ý là ngay cả khi với lượng dữ liệu huấn luyện thấp, các mô hình lớn đã cho thấy hiệu suất tốt hơn so với các mô hình nhỏ tương ứng được huấn luyện trên cùng dữ liệu. Thực tế này dẫn đến biến thể đầu tiên của *distillation* được gọi là *data distillation* hoặc *model compression*. Ở phương pháp này, chúng ta sử dụng một mô hình lớn đã được huấn luyện trên tập dữ liệu có sẵn để tạo ra nhiều dữ liệu tổng hợp (*synthetic data*) nhằm huấn luyện mô hình *student*. Việc tăng kích thước tập dữ liệu sẽ giúp mô hình *student* cải thiện chất lượng hơn so với chỉ huấn luyện trên dữ liệu ban đầu. Tuy nhiên, dữ liệu tổng hợp cần được xử lý cẩn thận để đảm bảo chất lượng cao, vì dữ liệu kém chất lượng có thể dẫn đến các tác động tiêu cực.



Hình 8. Minh họa hiệu suất của các mô hình với kích thước khác nhau phụ thuộc vào kích thước tập dữ liệu huấn luyện.

Các kỹ thuật *distillation* khác cố gắng đưa mô hình *student* tiệm cận với mô hình *teacher* ở mức độ chi tiết hơn so với chỉ tạo dữ liệu tổng hợp. Một kỹ thuật nổi bật là *knowledge distillation*, trong đó chúng ta cố gắng căn chỉnh phân phối token đầu ra của mô hình *student* với mô hình *teacher*. Phương pháp này có hiệu quả cao về mặt số lượng mẫu hơn so với *data distillation*. Một kỹ thuật khác là *on-policy distillation*, trong đó sử dụng phản hồi từ mô hình *teacher* trên mỗi chuỗi được tạo bởi mô hình *student* trong thiết lập học tăng cường (*reinforcement learning*).

Output-preserving methods (Các phương pháp bảo toàn đầu ra)

Các phương pháp này được đảm bảo không làm thay đổi chất lượng đầu ra của mô hình, nghĩa là không gây ra bất kỳ thay đổi nào đối với kết quả đầu ra của mô hình. Do đó, chúng thường là các bước đầu tiên rõ ràng để tối ưu hóa suy luận trước khi áp dụng các phương pháp gần đúng với những đánh đổi phức tạp hơn.

Flash Attention (Tăng tốc Attention)

Cơ chế *Scaled Dot-product Attention*, một phần quan trọng trong kiến trúc *Transformer*, là một phép toán bậc hai theo độ dài đầu vào. Việc tối ưu hóa tính toán *self-attention* có thể mang lại lợi ích đáng kể về độ trễ và chi phí.

Flash Attention, được giới thiệu bởi Tri Dao và cộng sự, tối ưu hóa tính toán attention bằng cách làm cho thuật toán attention nhận biết IO (*IO Aware*), đặc biệt là giảm lượng dữ liệu được di chuyển giữa bộ nhớ băng thông cao (HBM - *High Bandwidth Memory*) và các tầng bộ nhớ nhanh hơn (SRAM/VMEM) trong TPUs và GPUs. Trong khi tính toán attention, thứ tự của các phép toán được thay đổi và nhiều lớp được hợp nhất để sử dụng hiệu quả nhất các tầng bộ nhớ nhanh hơn.

Flash Attention là một thuật toán chính xác, giữ nguyên đầu ra số học của phép tính attention và có thể mang lại lợi ích đáng kể về độ trễ nhờ giảm chi phí IO. Tri Dao và cộng sự đã chứng minh rằng thuật toán này cải thiện độ trễ của tính toán attention từ 2 đến 4 lần.

Prefix Caching (Lưu trữ trước các giá trị tiền tố)

Một trong những phép toán tiêu tốn nhiều tài nguyên tính toán nhất, và do đó chậm nhất, trong suy luận của LLM là tính toán các điểm số attention key và value (*KV scores*) cho đầu vào. Phép toán này thường được gọi là *prefill*. Kết quả cuối cùng của *prefill* là cái gọi là *KV Cache*, bao gồm các điểm số attention key và value cho từng lớp của transformer trên toàn bộ đầu vào. *KV Cache* là yếu tố quan trọng trong giai đoạn encoding (giải mã), giúp tạo ra các token đầu ra. Nhờ *KV Cache*, mô hình không cần tính lại các điểm số attention cho đầu vào ở mỗi bước autoregressive decode (giải mã tự hồi quy).

Prefix Caching là quá trình lưu trữ *KV Cache* giữa các yêu cầu suy luận liên tiếp nhằm giảm độ trễ và chi phí của phép toán *prefill*. Cách cơ chế *self-attention* hoạt động làm cho việc tái sử dụng *KV Cache* trở nên khả thi, vì các token chỉ chú ý đến các token xuất hiện trước đó trong chuỗi. Nếu có đầu vào mới được thêm vào

đầu vào mà mô hình đã xử lý trước đó, chúng ta có thể tránh tính lại *prefill* cho phần đầu vào cũ.

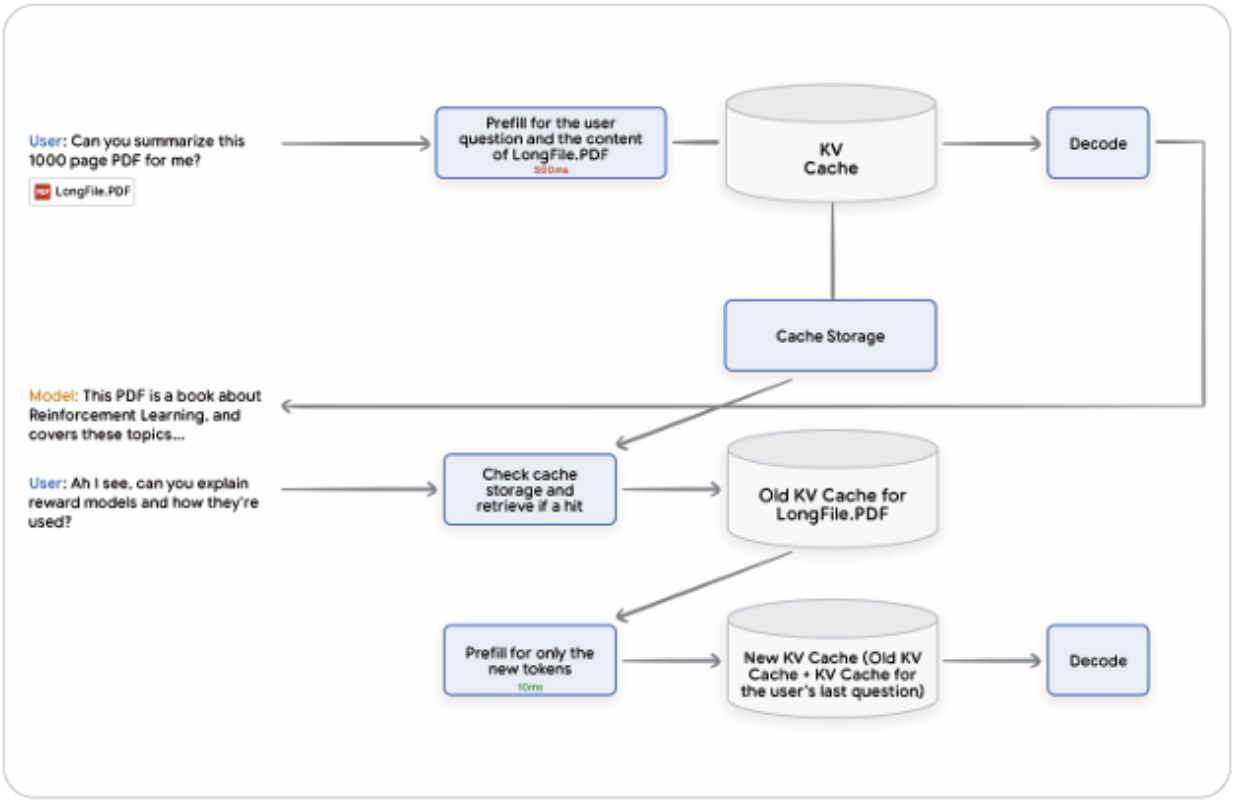


Figure 9. An illustration of Prefix Caching in a chat scenario (Hình 9. Minh họa về lưu trữ trước tiền tố trong kịch bản hội thoại)

Hình 9 minh họa cách hoạt động của *prefix caching* trong một kịch bản nhiều lượt với việc tải lên tài liệu. Trong lượt đầu tiên của người dùng, phép toán *prefill* phải xử lý toàn bộ tài liệu, mất 500ms. Kết quả là *KV cache* được lưu trữ, do đó trong lượt thứ hai của người dùng, chúng ta có thể lấy lại *cache* trực tiếp từ bộ lưu trữ và tránh tính toán lại cho tài liệu dài, qua đó tiết kiệm đáng kể tài nguyên tính toán và giảm độ trễ.

Các *prefix cache* có thể được lưu trữ trong bộ nhớ hoặc trên ổ đĩa và được truy xuất theo yêu cầu. Một điều quan trọng cần lưu ý là đảm bảo cấu trúc/schema của đầu vào phù hợp với *prefix caching*. Chúng ta nên tránh thay đổi tiền tố trong các yêu cầu tiếp theo, vì điều đó sẽ làm mất hiệu lực *cache* cho tất cả các token theo sau. Ví dụ, đặt một dấu thời gian mới ở đầu mỗi yêu cầu sẽ làm mất hiệu lực hoàn toàn *cache*, vì mỗi yêu cầu sau đó sẽ có một tiền tố mới.

Nhiều trường hợp sử dụng LLM phù hợp một cách tự nhiên với *prefix caching*. Ví dụ, các chatbot LLM, nơi người dùng thực hiện các cuộc hội thoại nhiều lượt có thể kéo dài đến hàng chục nghìn token, chúng ta có thể tránh tính toán lại *KV cache* cho các phần trước của cuộc hội thoại. Tải lên tài liệu hoặc mã lớn cũng là một trường hợp sử dụng khác, trong đó tài liệu mà người dùng tải lên sẽ không thay đổi từ yêu cầu này sang yêu cầu khác. Điều duy nhất thay đổi là các câu hỏi mà người dùng đặt, do đó lưu trữ *KV cache* cho tài liệu (đặc biệt đối với các tài liệu lớn) có thể mang lại lợi ích lớn về độ trễ và chi phí.

Prefix caching hiện có sẵn như một dịch vụ gọi là *Context Caching* trên Google AI Studio và Vertex AI trên Google Cloud.

Speculative Decoding (Giải mã suy đoán)

Giai đoạn đầu tiên trong suy luận của các mô hình ngôn ngữ lớn (LLM), được gọi là *prefill*, phụ thuộc vào khả năng tính toán vì phải thực hiện các phép toán ma trận lớn trên nhiều token song song. Giai đoạn thứ hai, được gọi là *decode*, thường phụ thuộc vào bộ nhớ vì các token được giải mã *autoregressive* từng bước một.

Không dễ để tận dụng năng lực tính toán song song bổ sung để tăng tốc giai đoạn *decode* do phải chờ token hiện tại được tạo ra trước khi có thể tính toán token tiếp theo (theo cơ chế *self-attention*), khiến quá trình giải mã mang tính tuần tự.

Speculative decoding (được đề xuất bởi Leviathan và cộng sự) nhằm vượt qua hạn chế này bằng cách tận dụng năng lực tính toán dư thừa để tăng tốc mỗi bước decode. Ý tưởng chính là sử dụng một mô hình thứ cấp nhỏ hơn nhiều (thường được gọi là *drafter*) để dự đoán trước các token (ví dụ: 4 token tiếp theo). Quá trình này diễn ra rất nhanh vì *drafter* nhỏ và nhanh hơn nhiều so với mô hình chính. Sau đó, mô hình chính được sử dụng để xác minh các giả thuyết của *drafter* song song qua từng bước (ví dụ: token đầu tiên, hai token đầu tiên, ba token đầu tiên, và cuối cùng là tất cả 4 token). Cuối cùng, chúng ta chọn giả thuyết được chấp nhận với số lượng token tối đa.

Hình 10. Minh họa giải mã suy đoán với 3 token

Lưu ý rằng cả 3 bước của mô hình chính được thực hiện song song. Vì giai đoạn *decode* không bị giới hạn bởi khả năng tính toán, chúng ta có thể tận dụng năng lực dư thừa để cải thiện đáng kể độ trễ. Trong ví dụ trên, giả sử mỗi bước của mô hình chính cần 10ms, trong khi *drafter* cần 1ms. Nếu không sử dụng *speculative decoding*, chúng ta cần 3 * 10ms = 30ms để tạo ra phản hồi. Với *speculative decoding*, do có sự song song hóa, chỉ có một bước của mô hình chính nằm trên đường dẫn quan trọng, vì vậy chúng ta chỉ cần 3 * 1ms + 10ms = 13ms. Đây là một cải thiện độ trễ đáng kể.

Kỹ thuật này hoàn toàn bảo toàn chất lượng; mô hình chính sẽ loại bỏ bất kỳ token nào mà nó không dự đoán được ngay từ đầu. Do đó, điều duy nhất mà *speculative decoding* làm là chạy trước và trình bày các giả thuyết để mô hình chính chấp nhận hoặc từ chối song song.

Một điều kiện quan trọng để *speculative decoding* hoạt động hiệu quả là mô hình *drafter* cần có mức độ căn chỉnh tốt với mô hình chính. Nếu không, không có token nào được chấp nhận. Vì vậy, việc đầu tư vào chất lượng huấn luyện của mô hình *drafter* rất đáng giá để đạt được độ trễ tốt hơn.

Bây giờ, khi đã xem qua các phương pháp tăng tốc phản hồi của LLM, chúng ta sẽ xem xét một số ví dụ về cách áp dụng các mô hình này vào các nhiệm vụ khác nhau để hiểu rõ cách sử dụng chúng.

Batching and Parallelization

Hầu hết các kỹ thuật tối ưu hóa mà chúng ta đã thảo luận đến nay đều liên quan đặc thù đến học máy và kiến trúc Transformer. Tuy nhiên, giống như bất kỳ hệ thống phần mềm nào, cũng có các cơ hội cải thiện thông lượng (*throughput*) và độ trễ (*latency*) thông qua việc kết hợp:

1. *Batching* các phép toán ít đòi hỏi tính toán hơn (ví dụ: chúng ta có thể chạy nhiều yêu cầu trên cùng một phần cứng đồng thời để tận dụng tốt hơn năng lực tính toán dư thừa).
2. *Parallelizing* các phần tính toán đòi hỏi nhiều tài nguyên hơn (ví dụ: chúng ta có thể chia nhỏ các phép toán và phân bổ chúng trên nhiều phần cứng hơn để tăng năng lực tính toán và cải thiện độ trễ).

Batching trong LLMs đặc biệt hữu ích ở giai đoạn giải mã (*decode*) của LLMs. Như đã giải thích trong phần *Speculative Decoding*, *decode* không bị giới hạn bởi khả năng tính toán, do đó có cơ hội để gộp thêm các yêu cầu. Chúng ta cần cẩn trọng trong việc gộp các phép toán theo cách tận dụng năng lực dư thừa trên các bộ tăng tốc (*accelerators*) như TPUs và GPUs. Đồng thời, cần đảm bảo không vượt quá giới hạn bộ nhớ, vì *decode* là một hoạt động sử dụng bộ nhớ lớn. Gộp nhiều yêu cầu hơn sẽ gây áp lực lớn hơn lên bộ nhớ trống. Batching đã trở thành một thành phần quan trọng trong hầu hết các thiết lập suy luận LLM hiệu suất cao.

Song song hóa (Parallelization) cũng là một kỹ thuật được sử dụng rộng rãi nhờ các cơ hội đa dạng để mở rộng ngang (*horizontal scaling*) trên nhiều phần cứng hơn trong kiến trúc Transformers. Có nhiều kỹ thuật song song hóa bao gồm:

- **Sequence parallelism**: Song song hóa trên đầu vào của mô hình.
- **Pipeline parallelism**: Song song hóa giữa các lớp của mô hình.
- **Tensor parallelism**: Song song hóa trong một lớp duy nhất.

Một trong những yếu tố quan trọng cần xem xét trong song song hóa là chi phí communication và đồng bộ hóa giữa các mảnh dữ liệu được phân phối cho các máy khác nhau. Communication là một chi phí lớn và có thể làm giảm lợi ích của việc bổ sung năng lực tính toán nếu không cẩn trọng trong việc lựa chọn chiến lược song song hóa. Tuy nhiên, việc chọn đúng chiến lược để cân bằng nhu cầu tính toán bổ sung và chi phí communication có thể mang lại cải thiện đáng kể về độ trễ.

Bây giờ, sau khi đã xem qua một số phương pháp giúp LLM tạo phản hồi nhanh hơn, chúng ta sẽ cùng xem xét một số ví dụ về cách áp dụng các mô hình này vào các nhiệm vụ khác nhau để hiểu rõ cách sử dụng chúng.

Applications (Ứng dụng)

Các mô hình ngôn ngữ lớn (*LLMs*) đang cách mạng hóa cách chúng ta tương tác và xử lý thông tin. Với khả năng chưa từng có trong việc hiểu ngữ cảnh và tạo nội dung, chúng đang thay đổi đáng kể nhiều ứng dụng trong các lĩnh vực văn bản, mã nguồn, hình ảnh, âm thanh và video.

Dưới đây là một số ví dụ về các lĩnh vực ứng dụng, nhưng độc giả nên lưu ý rằng đây không phải là danh sách đầy đủ. Rất nhiều ý tưởng mới đang liên tục xuất hiện về cách tốt nhất để tận dụng khả năng của các công cụ này. Để biết thêm thông tin về cách xây dựng và triển khai tối ưu các ứng dụng dựa trên các trường hợp sử dụng được đề cập dưới đây, vui lòng tham khảo các tài liệu chuyên sâu tiếp theo.

Python

```python
# Before you start run this command:
# pip install --upgrade --user --quiet google-cloud-aiplatform
# after running pip install make sure you restart your kernel

import vertexai
from vertexai.language_models import TextGenerationModel
from vertexai.preview.generative_models import GenerationConfig,GenerativeModel

# Set values as per your requirements
PROJECT_ID = '<project_id>' # set to your project_id
vertexai.init(project=PROJECT_ID, location='us-central1')

PROMPT= 'What is a LLM?' # set your prompt here
model = GenerativeModel('gemini-1.5-pro-002')

# call the Gemini API
response = model.generate_content(
   PROMPT)

print(response.text)

# google AI Studio SDK
import google.generativeai as genai
import os

# update with your API key
genai.configure(api_key=os.environ["GOOGLE_API_KEY"])

# choose the model
model = genai.GenerativeModel('gemini-pro')

response = model.generate_content('What is a LLM?') # set your prompt here

print(response.text)
```

Snippet 3. Using Vertex AI and Google AI studio SDKs for unimodal text gene

Việc tạo các phản hồi dựa trên văn bản cho các trường hợp sử dụng của bạn cũng rất đơn giản, thông qua Google Cloud Vertex AI SDK hoặc AI Studio hướng đến nhà phát triển. *Snippet 3* minh họa các ví dụ mã từ các SDK này để tạo phản hồi từ các văn bản *prompts* sử dụng mô hình Gemini. Lưu ý rằng các khía cạnh đa phương tiện của Gemini được trình bày chi tiết trong các tài liệu chuyên biệt tương ứng.

Code and mathematics (Mã nguồn và toán học)

Các mô hình sinh (*Generative models*) có khả năng hiểu và tạo ra mã nguồn cũng như thuật toán, hỗ trợ đáng kể các nhà phát triển trong nhiều lĩnh vực ứng dụng. Một số trường hợp sử dụng phổ biến cho mã nguồn bao gồm:

- **Code generation (Tạo mã nguồn)**: LLMs có thể được yêu cầu bằng ngôn ngữ tự nhiên để tạo mã nguồn bằng một ngôn ngữ lập trình cụ thể nhằm thực hiện các thao tác nhất định. Đầu ra có thể được sử dụng như một bản nháp.
- **Code completion (Hoàn thiện mã nguồn)**: LLMs có thể chủ động gợi ý các đoạn mã hữu ích khi người dùng đang gõ. Điều này giúp tiết kiệm thời gian và cải thiện chất lượng mã nguồn.
- **Code refactoring and debugging (Tái cấu trúc và gỡ lỗi mã nguồn)**: LLMs có thể giúp giảm nợ kỹ thuật bằng cách tái cấu trúc và gỡ lỗi mã nguồn, từ đó nâng cao chất lượng, hiệu suất và độ chính xác.
- **Code translation (Dịch mã nguồn)**: LLMs hỗ trợ chuyển đổi mã nguồn từ một ngôn ngữ lập trình này sang ngôn ngữ khác, giúp tiết kiệm thời gian và công sức. Ví dụ: một LLM có thể chuyển đổi mã Python sang Java.
- **Test case generation (Tạo trường hợp kiểm thử)**: LLMs có thể được yêu cầu tạo ra các kiểm thử đơn vị (*unit tests*) cho một mã nguồn đã cho, tiết kiệm đáng kể thời gian và giảm thiểu lỗi.
- **Code documentation and understanding (Tài liệu hóa và hiểu mã nguồn)**: LLMs có thể được sử dụng trong các cuộc hội thoại tự nhiên để giúp người dùng hiểu một mã nguồn cụ thể. Chúng cũng có thể tạo ra các nhận xét thích hợp, đánh giá trạng thái bản quyền, và viết ghi chú phát hành (*release notes*).

Gần đây, đã có nhiều tiến bộ đáng chú ý trong lĩnh vực lập trình cạnh tranh và toán học:

  - **AlphaCode 2**: Kết hợp khả năng lý luận của mô hình Gemini với tìm kiếm (*search*) và sử dụng các công cụ để giải các bài toán lập trình cạnh tranh. AlphaCode nhận mô tả bài toán làm đầu vào và xuất mã nguồn giải quyết bài toán đó. Hiện tại, nó nằm trong top 15% lập trình viên cạnh tranh hàng đầu trên nền tảng Codeforces.
  - **FunSearch**: Sử dụng một quy trình tiến hóa dựa trên việc ghép nối một LLM được huấn luyện trước với một bộ đánh giá có hệ thống (*systematic evaluator*). FunSearch đã giải quyết bài toán *cap set problem* – một bài toán

mở trong toán học – và phát hiện ra các thuật toán xếp gói (*bin-packing algorithms*) hiệu quả hơn, được sử dụng trong nhiều ứng dụng như tăng hiệu quả của trung tâm dữ liệu.

- **AlphaGeometry**: Giải quyết các bài toán tìm chứng minh cho các định lý hình học phức tạp. AlphaGeometry bao gồm một hệ thống *neuro-symbolic* kết hợp giữa mô hình ngôn ngữ thần kinh (*neural language model*) và bộ suy diễn ký hiệu (*symbolic deduction engine*). Hệ thống này đã giải được 25/30 bài toán hình học cấp độ Olympic, so sánh với mức điểm trung bình của huy chương vàng con người là 25.9.

Machine translation (Dịch máy)

Các mô hình ngôn ngữ lớn (*LLMs*) có khả năng tạo ra các bản dịch mượt mà, chất lượng cao và chính xác về mặt ngữ cảnh. Điều này đạt được nhờ khả năng hiểu sâu sắc về các sắc thái ngôn ngữ, thành ngữ và ngữ cảnh. Dưới đây là một số trường hợp sử dụng thực tế:

- **Instant messaging apps (Ứng dụng nhắn tin tức thời)**: Trong các nền tảng nhắn tin, LLMs có thể cung cấp các bản dịch theo thời gian thực (*on-the-fly*) một cách tự nhiên. Khác với các thuật toán trước đây chỉ dịch từng từ một (*word-for-word*), LLMs hiểu được tiếng lóng, thành ngữ và sự khác biệt vùng miền, từ đó nâng cao khả năng giao tiếp xuyên ngôn ngữ.
- **E-commerce (Thương mại điện tử)**: Trên các nền tảng toàn cầu như AliExpress, mô tả sản phẩm được tự động dịch. LLMs giúp đảm bảo các sắc thái văn hóa và các cụm từ thành ngữ trong mô tả sản phẩm được dịch phù hợp, giảm thiểu các hiểu lầm.
- **Travel apps (Ứng dụng du lịch)**: Trong các ứng dụng như Google Translate, khách du lịch nhận được các bản dịch nói theo thời gian thực. Với LLMs, các cuộc hội thoại được dịch trở nên trôi chảy hơn, giúp các tương tác ở nước ngoài trở nên dễ dàng hơn.

Text summarization (Tóm tắt văn bản)

Tóm tắt văn bản là một khả năng cốt lõi của nhiều mô hình ngôn ngữ lớn được đề cập trong tài liệu này. Dưới đây là một số trường hợp sử dụng tiềm năng:

- **News aggregators (Trình tổng hợp tin tức)**: LLMs có thể tạo các bản tóm tắt nắm bắt không chỉ các sự kiện chính mà còn cả cảm xúc và giọng điệu của bài viết, cung cấp cho người đọc sự hiểu biết toàn diện hơn.
- **Research databases (Cơ sở dữ liệu nghiên cứu)**: LLMs có thể hỗ trợ các nhà nghiên cứu tạo ra các bản tóm tắt nêu bật các phát hiện và ý nghĩa cốt lõi của các bài báo khoa học.
- **Chat management (Quản lý hội thoại)**: Trên các nền tảng như Google Chat, các hệ thống dựa trên LLM có thể tạo ra các bản tóm tắt chuỗi hội thoại phản ánh mức độ khẩn cấp và giọng điệu, hỗ trợ người dùng ưu tiên các phản hồi của họ.

Question-answering (Hỏi đáp)

Các hệ thống QA thế hệ cũ thường hoạt động dựa trên đối sánh từ khóa, thường bỏ qua chiều sâu ngữ cảnh của các truy vấn người dùng. Tuy nhiên, LLMs đi sâu vào ngữ cảnh. Chúng có thể suy luận ý định của người dùng, truy cập các ngân hàng thông tin khổng lồ và cung cấp các câu trả lời giàu ngữ cảnh và chính xác. Một số ví dụ về nơi có thể sử dụng điều này bao gồm:

- **Virtual assistants (Trợ lý ảo)**: LLMs có thể đưa ra các giải thích chi tiết về dự báo thời tiết dựa trên vị trí của người dùng, thời điểm trong năm và các xu hướng thời tiết gần đây.
- **Customer support (Hỗ trợ khách hàng)**: Trên các nền tảng kinh doanh, các chatbot dựa trên LLM có thể cung cấp các câu trả lời tính đến lịch sử mua hàng, các truy vấn trước đó và các vấn đề tiềm năng của người dùng, mang lại sự hỗ trợ cá nhân hóa.
- **Academic platforms (Nền tảng học thuật)**: Trên các nền tảng học thuật như Wolfram Alpha, LLMs có thể đáp ứng các truy vấn của người dùng bằng cách hiểu chiều sâu và ngữ cảnh của các câu hỏi học thuật, cung cấp các câu trả lời phù hợp cho mọi đối tượng từ học sinh trung học đến nghiên cứu sinh sau đại học.

Chất lượng của các câu trả lời được tạo ra, cũng như các trích dẫn và nguồn tương ứng, có thể được cải thiện đáng kể bằng cách sử dụng các hệ thống tìm kiếm nâng cao (như các kiến trúc Retrieval Augmented Generation - RAG) để mở rộng *prompt* với thông tin phù hợp, cũng như kiểm chứng sau khi phản hồi được tạo ra (*post-*

hoc grounding). Các hướng dẫn rõ ràng, vai trò về những gì nên và không nên sử dụng để trả lời câu hỏi, và các phương pháp *prompt engineering* nâng cao (như *chain of thought* và các kiến trúc tìm kiếm/RAG), kết hợp với giá trị *temperature* thấp và các yếu tố khác, cũng có thể cải thiện đáng kể.

Chatbots

Các chatbot trước đây hoạt động dựa trên các kịch bản cố định, dẫn đến các cuộc hội thoại "máy móc". LLMs đã thay đổi lĩnh vực này bằng cách mang lại các tương tác động và giống con người. Chúng có thể phân tích cảm xúc, ngữ cảnh, thậm chí cả sự hài hước, làm cho các cuộc hội thoại số trở nên chân thực hơn. Một số ví dụ về ứng dụng bao gồm:

- **Customer service (Dịch vụ khách hàng)**: Một chatbot trên các nền tảng bán lẻ như Zara không chỉ trả lời các câu hỏi liên quan đến sản phẩm mà còn có thể đưa ra lời khuyên về thời trang dựa trên xu hướng hiện tại.
- **Entertainment (Giải trí)**: Trên Media, các chatbot sử dụng LLM có thể tương tác động với người dùng, phản hồi các sự kiện trực tiếp trong luồng và kiểm duyệt các cuộc trò chuyện với sự hiểu biết về ngữ cảnh.

Content generation (Tạo nội dung)

Tạo văn bản không phải là điều mới mẻ, nhưng điều mà LLMs mang lại là khả năng chưa từng có trong việc tạo ra văn bản giống con người, có tính ngữ cảnh và chi tiết phong phú. Các mô hình trước đây thường mất ngữ cảnh hoặc mạch lạc khi xử lý các đoạn văn dài. LLMs, với kiến thức rộng lớn và sự hiểu biết tinh tế, có thể tạo ra văn bản với nhiều phong cách, giọng điệu và mức độ phức tạp khác nhau, kết hợp giữa tính thực tế và sáng tạo (tùy thuộc vào ngữ cảnh), thu hẹp khoảng cách giữa nội dung do máy tạo ra và do con người viết. Một số ví dụ thực tế bao gồm:

- **Content creation (Sáng tạo nội dung)**: Các nền tảng có thể sử dụng LLMs để hỗ trợ các nhà tiếp thị phát triển các quảng cáo. Thay vì tạo nội dung chung chung, LLMs có thể tạo ra các thông điệp sáng tạo, nhắm mục tiêu cụ thể và phù hợp với từng đối tượng.
- **Scriptwriting (Viết kịch bản)**: LLMs có thể hỗ trợ tạo ra các kịch bản cho phim hoặc chương trình truyền hình. Các nhà biên kịch có thể cung cấp chủ đề hoặc

các điểm cốt truyện, và mô hình có thể gợi ý các đoạn hội thoại hoặc mô tả cảnh, nâng cao quá trình sáng tạo.

Tạo văn bản là một nhiệm vụ rộng lớn bao gồm nhiều trường hợp sử dụng, từ những trường hợp mà độ chính xác của đầu ra được tạo ra là rất quan trọng, đến những trường hợp mà tính sáng tạo/đa dạng của ngôn ngữ được ưu tiên hơn. Các phương pháp lấy mẫu (*sampling methods*) và các tham số như *temperature* nên được điều chỉnh tương ứng. Để biết thêm thông tin, tham khảo các tài liệu kỹ thuật về *prompt engineering* và kiến trúc ứng dụng LLM.

Natural language inference (Suy diễn ngôn ngữ tự nhiên)

Suy diễn ngôn ngữ tự nhiên (NLI) là nhiệm vụ xác định xem một giả thuyết văn bản cụ thể có thể được suy diễn logic từ một tiền đề văn bản hay không.

Các mô hình truyền thống thường gặp khó khăn với các mối quan hệ phức tạp hoặc những mối quan hệ đòi hỏi sự hiểu biết sâu sắc hơn về ngữ cảnh. LLMs, với khả năng nắm bắt chi tiết về ngữ nghĩa và ngữ cảnh, vượt trội trong các nhiệm vụ như vậy, đưa độ chính xác lên gần mức hiệu suất của con người. Dưới đây là một số ví dụ thực tế:

- **Sentiment analysis (Phân tích cảm xúc)**: Các doanh nghiệp có thể sử dụng LLMs để suy diễn cảm xúc của khách hàng từ các đánh giá sản phẩm. Thay vì chỉ gắn thẻ cơ bản như "tích cực" hoặc "tiêu cực", LLMs có thể trích xuất các cảm xúc chi tiết hơn như "hài lòng", "thất vọng" hoặc "vui sướng".
- **Legal document review (Xem xét tài liệu pháp lý)**: Các công ty luật có thể áp dụng LLMs để suy diễn các hàm ý và ý định trong hợp đồng, đảm bảo không có các mâu thuẫn hoặc điều khoản có vấn đề tiềm ẩn.
- **Medical diagnoses (Chẩn đoán y khoa)**: Bằng cách phân tích mô tả và lịch sử bệnh nhân, LLMs có thể hỗ trợ bác sĩ suy diễn các chẩn đoán tiềm năng hoặc rủi ro sức khỏe, giúp đảm bảo can thiệp sớm.

Các tài liệu chuyên sâu về LLM theo từng lĩnh vực, kỹ thuật *prompt engineering*, và thiết kế kiến trúc ứng dụng LLM cung cấp thêm thông tin chi tiết về các trường hợp sử dụng này.

Text classification (Phân loại văn bản)

Phân loại văn bản là việc phân loại các văn bản vào các nhóm được định nghĩa trước. Trong khi các thuật toán truyền thống hoạt động hiệu quả, chúng thường gặp khó khăn với các danh mục mơ hồ hoặc chồng chéo. LLMs, nhờ khả năng hiểu sâu về ngữ cảnh, có thể phân loại văn bản với độ chính xác cao hơn, ngay cả khi đối mặt với những khác biệt tinh tế. Một số ví dụ thực tế bao gồm:

- **Spam detection (Phát hiện thư rác)**: Các dịch vụ email có thể sử dụng LLMs để phân loại email thành thư rác hoặc hợp pháp. Thay vì chỉ dựa vào từ khóa, các mô hình này hiểu ngữ cảnh và ý định, giúp giảm thiểu các kết quả dương tính giả.
- **News categorization (Phân loại tin tức)**: Các nền tảng tin tức có thể sử dụng LLMs để phân loại bài báo thành các chủ đề như "công nghệ", "chính trị", hoặc "thể thao", ngay cả khi các bài viết có sự giao thoa giữa các chủ đề.
- **Customer feedback sorting (Phân loại phản hồi khách hàng)**: Các doanh nghiệp có thể phân tích phản hồi của khách hàng thông qua LLMs để phân loại chúng vào các lĩnh vực như "thiết kế sản phẩm", "dịch vụ khách hàng", hoặc "giá cả", đảm bảo phản hồi được xử lý có mục tiêu.
- **Evaluating LLMs as autorater (Đánh giá LLMs như người tự động xếp hạng)**: LLMs có thể được sử dụng để xếp hạng, so sánh, và đánh giá các đầu ra được tạo ra bởi các LLMs khác.

Text analysis (Phân tích văn bản)

LLMs vượt trội trong việc phân tích sâu văn bản – trích xuất các mẫu, hiểu các chủ đề, và thu thập thông tin chi tiết từ các tập dữ liệu văn bản khổng lồ. Trong khi các công cụ truyền thống chỉ xử lý bề mặt, LLMs đi sâu hơn, cung cấp các thông tin phong phú và có thể hành động. Một số ví dụ thực tế bao gồm:

- **Market research (Nghiên cứu thị trường)**: Các công ty có thể tận dụng LLMs để phân tích các cuộc trò chuyện của người tiêu dùng trên mạng xã hội, trích xuất các xu hướng, sở thích, và nhu cầu mới nổi.
- **Literary analysis (Phân tích văn học)**: Các nhà nghiên cứu có thể sử dụng LLMs để hiểu các chủ đề, biểu tượng, và sự phát triển nhân vật trong các tác phẩm văn học, mang lại góc nhìn mới về các tác phẩm cổ điển và đương đại.

Multimodal applications (Ứng dụng đa phương thức)

Các mô hình ngôn ngữ lớn đa phương thức (*Multimodal LLMs*), có khả năng xử lý và tạo ra văn bản, hình ảnh, âm thanh và video, đã mở ra một biên giới mới trong AI, mang lại nhiều ứng dụng thú vị và đổi mới trên nhiều lĩnh vực. Dưới đây là một số ví dụ:

Creative content generation (Tạo nội dung sáng tạo):

- Storytelling (Kể chuyện): Một hệ thống AI có thể quan sát một hình ảnh hoặc video và tạo ra một câu chuyện hấp dẫn, tích hợp các chi tiết từ nội dung thị giác với cơ sở kiến thức của nó.
- Advertising and marketing (Quảng cáo và tiếp thị): Tạo ra các quảng cáo được nhắm mục tiêu và gợi cảm xúc dựa trên ảnh hoặc video sản phẩm.

Education and accessibility (Giáo dục và khả năng tiếp cận):

- Personalized learning (Học tập cá nhân hóa): Tùy chỉnh tài liệu giáo dục theo phong cách học tập cá nhân bằng cách kết hợp văn bản với các yếu tố hình ảnh và âm thanh tương tác.
- Assistive technology (Công nghệ hỗ trợ): Các LLM đa phương thức có thể cung cấp năng lượng cho các công cụ mô tả hình ảnh, video và âm thanh cho người khiếm thị hoặc khiếm thính.

Business and industry (Doanh nghiệp và công nghiệp):

- Document understanding and summarization (Hiểu và tóm tắt tài liệu): Tự động trích xuất thông tin quan trọng từ các tài liệu phức tạp, kết hợp văn bản và hình ảnh như hóa đơn và hợp đồng.
- Customer service (Dịch vụ khách hàng): Chatbot đa phương thức có thể hiểu và phản hồi các truy vấn của khách hàng bằng cách kết hợp văn bản và hình ảnh, mang lại trải nghiệm phong phú và cá nhân hóa hơn.

Science and research (Khoa học và nghiên cứu):

- Medical diagnosis (Chẩn đoán y khoa): Phân tích các bản chụp y khoa và báo cáo cùng nhau, xác định các vấn đề tiềm ẩn và cung cấp thông tin chi tiết cho bác sĩ.

- Bioinformatics and drug discovery (Tin sinh học và khám phá thuốc): Tích hợp kiến thức từ nhiều nguồn dữ liệu khác nhau như hình ảnh y khoa, cấu trúc protein và các bài báo nghiên cứu để thúc đẩy tiến bộ trong nghiên cứu.

Những ví dụ trên chỉ là bề nổi của tảng băng chìm. Khi nghiên cứu tiến xa hơn, các ứng dụng của LLM đa phương thức được kỳ vọng sẽ ngày càng phát triển, biến đổi cuộc sống hàng ngày của chúng ta theo những cách đa dạng và sâu sắc. Các LLM đa phương thức cũng hưởng lợi đáng kể từ các phương pháp sẵn có của LLM đơn phương thức (ví dụ: LLM dựa trên văn bản).

Nhờ khả năng hiểu và xử lý ngôn ngữ, LLMs đang định hình lại cách chúng ta tương tác, tạo ra và phân tích văn bản trên nhiều lĩnh vực khác nhau. Khi tiếp tục phát triển, các ứng dụng của chúng sẽ ngày càng mở rộng, tăng cường khả năng tương tác ngôn ngữ tự nhiên giữa máy móc và con người.

Summary (Tóm tắt)

Trong tài liệu này, chúng ta đã thảo luận về những khái niệm cơ bản của kiến trúc *Transformer*, nền tảng của tất cả các mô hình ngôn ngữ lớn (LLMs) hiện đại. Chúng ta đã trình bày chi tiết về sự phát triển của các kiến trúc mô hình LLM khác nhau và các thành phần của chúng. Ngoài ra, các phương pháp huấn luyện và tinh chỉnh mô hình hiệu quả cũng được đề cập. Chúng ta đã thảo luận ngắn gọn về *prompt engineering* và các kỹ thuật lấy mẫu (*sampling techniques*) ảnh hưởng mạnh mẽ đến đầu ra của LLM, đồng thời chạm đến các ứng dụng tiềm năng của công nghệ này. Một số điểm chính cần ghi nhớ:

- **Kiến trúc Transformer** là nền tảng của tất cả các LLM hiện đại. Qua các kiến trúc khác nhau được đề cập, chúng ta nhận thấy không chỉ việc tăng số lượng tham số của mô hình mà cả thành phần của tập dữ liệu cũng đóng vai trò quan trọng.
- **Thứ tự và chiến lược fine tuning (tinh chỉnh)** đóng vai trò quan trọng và có thể bao gồm nhiều bước như *Instruction Tuning*, *Safety Tuning*,... *Supervised Fine Tuning* (SFT) giúp nắm bắt bản chất của một nhiệm vụ. *RLHF* và có thể cả *RLAIF* có thể được sử dụng để chuyển phân phối từ giai đoạn tiền huấn luyện sang phân phối mong muốn hơn thông qua sức mạnh của reward function (hàm thưởng), giúp khuyến khích hành vi mong muốn và phạt hành vi không mong muốn.

- **Tối ưu hóa suy luận** từ các mô hình neural để đạt hiệu quả là một vấn đề quan trọng và là một lĩnh vực nghiên cứu sôi động. Nhiều phương pháp đã được phát triển để giảm chi phí phục vụ và độ trễ với tác động tối thiểu đến hiệu suất mô hình, và một số phương pháp tăng tốc chính xác đảm bảo đầu ra của mô hình không thay đổi.
- **Các mô hình ngôn ngữ lớn** có thể được sử dụng cho nhiều nhiệm vụ khác nhau bao gồm tóm tắt, dịch thuật, hỏi đáp, trò chuyện, tạo mã nguồn và nhiều hơn nữa. Bạn có thể tạo ra các nhiệm vụ riêng của mình bằng cách sử dụng các dịch vụ tạo văn bản như Vertex AI và Makersuite dựa trên các mô hình ngôn ngữ mới nhất của Google. Sau khi mô hình đã được huấn luyện và tinh chỉnh, việc thử nghiệm với *prompt engineering* là rất quan trọng. Bạn nên sử dụng kỹ thuật phù hợp nhất với nhiệm vụ cụ thể, vì LLM có thể nhạy cảm với *prompt*. Hơn nữa, cũng có thể cải thiện hiệu suất nhiệm vụ cụ thể, sự sáng tạo và đa dạng bằng cách điều chỉnh các tham số liên quan đến kỹ thuật lấy mẫu như *Top-K*, *Top-P*, và các bước v để tìm ra Max decoding kết hợp tối ưu giữa tính đúng đắn, sự đa dạng, và tính sáng tạo cần thiết cho nhiệm vụ.