AI VIET NAM – RESEARCH TEAM

Understanding LLMs: A Comprehensive Overview from Training to Inference

March 30, 2024

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

I. Article Overview:

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

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

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

II. Detailed Article Summary

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

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

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

PLM with significantly larger parameter sizes (6-10 billion parameters) and extensive training data is called Large Language Models (LLM). Having the knowledge to develop LLMs independently can play a role in replacing ChatGPT when it is no longer open source, and developing LLMs for specific domains has become very necessary. And an important part of that research is researching effective training and reasoning methods for LLMs.

1.2. Background Knowledge about LLMs:

a) Transformer:

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

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

Self-Attention Mechanism:

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

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

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

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

***Encoder:**

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

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

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

***Decoder:**

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

***Positional Embedding:**

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

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

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

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

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

b) Prompt learning:

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

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

Prompt learning includes the following basic components:

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

  The choice of prompt template plays a crucial role in prompt learning.

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

  Combining multiple type of templates can improve model performance.

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

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

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

***Learning Strategies of Prompt Learning:**

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

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

- Fix prompt LM tuning.

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

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

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

2. Technical Aspects of LLM Training:

The LLM training can be divided into three steps:

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

2.1. Data Collection and Preprocessing:

a) Dataset Preparation:

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

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

b) Data Preprocessing:

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

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

2.2. Model Architecture:

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

a) Encoder-Decoder Architecture:

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

b) Decoder-only Architecture:

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

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

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

2.3. Pre-training Tasks:

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

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

2.4. Model training:

a) Parallel Training:

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

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

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

There are 4 main methods in parallel training:

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

b) Mixed Precision Training:

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

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

c) Offloading:

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

d) Overlapping:

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

e) Checkpoint:

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

2.5. Fine-tuning:

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

a) Supervised Fine-tuning:

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

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

b) Alignment Tuning:

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

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

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

c) Parameter-efficient Tuning:

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

d) Safe Fine-tuning:

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

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

2.6. Evaluation:

a) Static testing dataset:

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

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

b) Open domain Q&A evaluation:

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

c) Security evaluation:

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

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

d) Evaluation method:

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

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

2.7. LLM Framework:

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

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

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

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

3. Inference with Large Language Models:

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

3.1. Model Compression:

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

3.2. Memory Scheduling:

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

3.3. Parallelism:

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

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

3.4. Structural Optimization:

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

3.5. Inference Framework:

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

4. Utilization of LLMs:

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

LLM Deployment Strategies:

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

5. Future Development Directions and Their Implications:

Future Development Trends of LLMs:

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

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