Understanding LLMs: A Comprehensive Overview from Training to Inference

Yiheng Liu[a], Hao He[a], Tianle Han[a], Xu Zhang[a], Mengyuan Liu[a], Jiaming Tian[a], Yutong Zhang[b], Jiaqi Wang[c], Xiaohui Gao[d], Tianyang Zhong[d], Yi Pan[e], Shaochen Xu[e], Zihao Wu[e], Zhengliang Liu[e], Xin Zhang[b], Shu Zhang[c], Xintao Hu[d], Tuo Zhang[d], Ning Qiang[a], Tianming Liu[e] and Bao Ge [a]

[a]School of Physics and Information Technology, Shaanxi Normal University, Xi'an, 710119, Shaanxi, China
[b]Institute of Medical Research, Northwestern Polytechnical University, Xi'an, 710072, Shaanxi, China
[c]School of Computer Science, Northwestern Polytechnical University, Xi'an, 710072, Shaanxi, China
[d]School of Automation, Northwestern Polytechnical University, Xi'an, 710072, Shaanxi, China
[e]School of Computing, The University of Georgia, Athens, 30602, USA

ARTICLE INFO

Keywords:
Large Language Models
Training
Inference
Survey

ABSTRACT

The introduction of ChatGPT has led to a significant increase in the utilization of Large Language Models (LLMs) for addressing downstream tasks. There's an increasing focus on cost-efficient training and deployment within this context. Low-cost training and deployment of LLMs represent the future development trend. This paper reviews the evolution of large language model training techniques and inference deployment technologies aligned with this emerging trend. The discussion on training includes various aspects, including data preprocessing, training architecture, pre-training tasks, parallel training, and relevant content related to model fine-tuning. On the inference side, the paper covers topics such as model compression, parallel computation, memory scheduling, and structural optimization. It also explores LLMs' utilization and provides insights into their future development.

1. Introduction

Language modeling (LM) is a fundamental approach for achieving cognitive intelligence in the field of natural language processing (NLP), and its progress has been notable in recent years [1; 2; 3]. It assumes a central role in understanding, generating, and manipulating human language, serving as the cornerstone for a diverse range of NLP applications [4], including machine translation, chatbots, sentiment analysis, and text summarization. With the evolution of deep learning, the early statistical language models (SLM) have gradually transformed into neural language models (NLM) based on neural networks. This shift is characterized by the adoption of word embeddings, representing words as distributed vectors. Notably, these word embeddings have consistently excelled in practical NLP tasks, profoundly shaping the field's progress. Pre-trained language models (PLM) represent a subsequent phase in the evolution of language models following NLM. Early attempts at PLMs included ELMo [5], which was built on a Bidirectional LSTM architecture. However, with the advent of the transformer architecture [6], characterized by parallel self-attention mechanisms, the pre-training and fine-tuning learning paradigm has propelled PLM to prominence as the prevailing approach. These models are typically trained via self-supervision on extensive datasets, cementing their status as the primary methodology in the field.

The Transformer architecture is exceptionally well-suited for scaling up models, and research analysis has revealed that increasing the model's scale or training data size can significantly enhance its performance. Many studies have pushed the boundaries of model performance by continuously expanding the scale of PLM [7; 8; 9; 10]. As models grow larger, a remarkable phenomenon known as "emergence" occurs, wherein they exhibit astonishing performance [8]. These models are capable of generating high-quality text and possess robust learning and reasoning abilities. They can even tackle few-shot learning tasks through in-context learning (ICL) [8]. This remarkable capability enables their seamless application to a wide range of downstream tasks across diverse domains [11; 12; 13; 14].

*Corresponding author
ORCID(s):

Pre-trained language models (PLMs) with significantly larger parameter sizes and extensive training data are typically denoted as Large Language Models (LLMs) [15; 16; 17]. The model size usually exceeds 6-10 billion (6-10B) parameters. A prominent milestone in the development of LLMs is exemplified by the GPT series [18; 7; 8; 19]. Notably, OpenAI released ChatGPT in November 2022, marking a pivotal moment in the era of LLMs and a game-changing moment in the field of artificial intelligence. ChatGPT has empowered current AI algorithms to achieve unprecedented levels of strength and effectiveness, reshaping the way humans employ or develop AI algorithms. Its emergence has captured the attention of the research community. However, owing to ChatGPT's absence as an open-source platform, the principal way to use ChatGPT currently is by accessing it through OpenAI's website at `https://chat.openai.com` or via their API interface. Training LLMs that can serve as alternatives to ChatGPT, or domain-specific LLMs, has become highly necessary [20; 21; 22; 23; 24; 1; 25; 26]. Training and deploying LLMs demand expertise in handling large-scale data and substantial practical experience in distributed parallel training [27; 28; 29]. This requirement emphasizes the need for researchers developing LLMs to possess significant engineering capabilities in addressing the challenges encountered during LLM development. Researchers who are interested in the field of LLMs must possess engineering skills or learn to collaborate effectively with engineers.

For the above reasons, the primary objective of this paper is to provide a comprehensive overview of LLMs training and inference techniques to equip researchers with the knowledge required for developing, deploying, and applying LLMs. The structure of the rest of this review is as follows: In Section 2, we will introduce the relevant background and foundational knowledge of LLMs. In Section 3, we will delve into the technical aspects of training LLMs, while in Section 4 we will explore the technologies related to LLM's inference and deployment. In Section 5, we will discuss the utilization of LLMs, and Section 6 will explore the future directions and their implications for LLMs.

2. Background Knowledge

2.1. Transformer

Transformer is a deep learning model based on an attention mechanism for processing sequence data that can effectively solve complex natural language processing problems. This model was first proposed in 2017 [6], and replaced the traditional recurrent neural network architecture [30] in machine translation tasks as the state-of-the-art model at that time. Due to its suitability for parallel computing and the complexity of the model itself, Transformer outperforms the previously popular recurrent neural networks in terms of accuracy and performance. The Transformer architecture consists primarily of two modules, an Encoder and a Decoder, as well as the attention mechanism within these modules.

2.1.1. Self-Attention

Self-Attention Structure [6]: Essentially, the attention mechanism aims at selecting a small amount of important information from a large amount of data and focusing on these important pieces while ignoring the majority of unimportant information. The self-attention mechanism, as a variant of the attention mechanism, reduces reliance on external information and excels at capturing internal correlations within data or features. Applying the self-attention mechanism in text-primarily involves calculating the mutual influence between words to address the issue of long-range dependencies. Additionally, self-attention is the core idea behind transformers. The core formula for key-value attention is as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

Self-attention allows the model to weigh the importance of different words in a sentence when predicting a particular word. It calculates a weighted sum of the values of all words in the sentence, where the weights are determined by the relevance of each word to the target word.

The self-attention mechanism consists of three steps: calculating the query, key, and value vectors. The query vector represents the word being attended to, while the key vectors represent all the words in the sentence. The value vectors store the information associated with each word. The attention weights are computed by taking the dot product between the query and key vectors, followed by a softmax operation to obtain a distribution over the words.

Multi-Head Attention [6]: Multi-head self-attention extends the self-attention mechanism by performing it multiple times in parallel. Each attention head learns to focus on different aspects of the input, capturing different dependencies and patterns. The outputs of the attention heads are then concatenated and linearly transformed to obtain

the final representation. By using multiple attention heads, 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 capacity to capture complex relationships between words. The Multi-head attention can be formulated as follows:

$$\begin{aligned} MultiHeadAttention(Q,K,V) &= Concat[head_1,\dots,head_h]W^o \\ where\ head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2}$$

In this case, "$Concat$" means to concatenate the attention calculation results of each head, "W^o" is the weight matrix of the output layer, used to linearly transform the concatenated results. This yields the output of multi-head attention. In summary, multi-head attention enhances the model's ability to represent input sequences by performing parallel attention calculations under different linear transformations, then concatenating and linearly transforming the results. This mechanism plays an important role in the Transformer model, helping to handle long-range dependencies and improve model performance.

2.1.2. Encoder

The encoder module [6] of the Transformer model is composed of multiple identical layers, each of which includes a multi-head attention mechanism and feed-forward neural network [31]. In the multi-head attention mechanism, each position in the input sequence is calculated for attention with other positions to capture the dependencies between different positions in the input sequence. The feed-forward neural network is then used to further process and extract features from the output of the attention mechanism. The encoder module gradually extracts features of the input sequence through the stacking of multiple such layers and passes the final encoding result to the decoder module for decoding. The design of the encoder module enables it to effectively handle long-range dependencies within the input sequence and has significantly improved performance in various NLP tasks.

2.1.3. Decoder

The decoder module [32] of the Transformer model is also composed of multiple identical layers, each of which includes a multi-head attention mechanism and a feed-forward neural network. Unlike the encoder, the decoder also includes an additional encoder-decoder attention mechanism, used to compute attention on the input sequence during the decoding process. At each position, the decoder can only perform self-attention calculations with the positions before it to ensure that the generation of the sequence does not violate grammar rules. Masks play an important role in the decoder, ensuring that only information before the current time step is focused on when generating the output sequence, and not leaking information from future time steps. Specifically, the decoder's self-attention mechanism uses masks to prevent the model from accessing future information when generating predictions at each time step, maintaining the causality of the model. This ensures that the output generated by the model depends on the information at the current time step and before, without being influenced by future information.

2.1.4. Positional Embedding

Position and order are crucial for certain tasks, such as understanding a sentence or a video. Position and order define the grammar of a sentence, they are integral to the semantics of sentences. The Transformer utilizes Multi-Head Self-Attention (MHSA) to avoid the recursive approach of RNN, thus speeding up the training process. Additionally, it can capture long-range dependencies in sentences and handle longer inputs. When each token in a sentence passes through the Transformer's Encoder/Decoder stack, the model itself lacks any sense of position/order for each token (permutation invariance). Therefore, a method is still needed to incorporate the sequential information of tokens into the model. To enable the model to perceive the input sequence, positional information about the location of each token in the sentence can be added, and this technique is known as positional embedding (PE). which is used in the Transformer model to incorporate the sequential order of tokens into the input representation. Since the Transformer does not have recurrent connections, it lacks the inherent notion of token order present in recurrent neural networks. To address this, positional embedding assigns a unique vector to each token position in the input sequence. These positional embeddings are added to the word embedding before being fed into the model. By including positional information, the model can differentiate between tokens based on their position in the sequence. In the Transformer model, the core formula of the position embedding can be expressed as:

$$PE(pos,2i) = sin(\frac{pos}{10000^{(\frac{2i}{d_{model}})}}) \tag{3}$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{(\frac{2i}{d_{model}})}}) \tag{4}$$

In this equation, PE represents the position embedding matrix, pos represents the position of a token in the sentence, i represents the dimension index of the position embedding, and d_{model} represents the hidden layer dimension of the Transformer model. By using sine and cosine functions and performing different calculations on the position (pos) and dimension (i), this formula generates unique position embedding values for each position and dimension. As a result, each token is assigned a unique position embedding vector, allowing the model to perceive the sequential information of tokens in the sentence. In practical applications, the position embedding matrix is added to the input word embedding matrix to combine position information and semantic information, thereby providing a more comprehensive input representation for the Transformer model.

Two commonly used positional encoding methods in Transformer are Absolute Positional Encoding and Relative Positional Encoding.

(1) Absolute Positional Encoding: It generates unique positional embedding values for each position and dimension by using sine and cosine functions. This method uses sine and cosine functions in the mentioned formula to calculate the positional embedding values and adds them to the word embeddings. Absolute Positional Encoding provides a unique encoding for each position, enabling the model to perceive the sequential information of words in the sentence.

(2) Relative Positional Encoding: It is an encoding method based on relative positional relationships. Relative Positional Encoding represents positional information by calculating the relative distances between words. This method is used in models like Transformer-XL [33], and Relative Positional Encoding can better capture the relative positional relationships between words when dealing with long sequences. Both of these positional encoding methods aim to provide the positional information of words in the input sequence to the Transformer model, enabling the model to better comprehend and process sequential data. The specific choice of positional encoding method depends on the specific application scenario and model design.

There are also other positional encoding methods applied to other models, such as RoPE [34] and ALiBi [35].

RoPE is a method that uses Absolute Positional Encoding to represent Relative Positional Encoding and is applied in the design of large language models like PaLM [36], LLaMA [9], and GLM-130B [37].

ALiBi does not add positional embeddings to word embeddings but instead adds a pre-defined bias matrix to the attention score based on the distance between tokens. It is applied in the design of large language models like BLOOM [38].

Some other positional encoding methods, such as mixed positional encoding, multi-digit positional encoding, and implicit positional encoding, are also used by some models.

2.2. Prompt Learning

Prompt learning serves as a widely adopted machine learning approach, particularly in the field of NLP. At its core, this methodology involves guiding a model to produce specific behaviors or outputs through the careful design of prompt statements. It is commonly employed to fine-tune and guide pre-trained LLMs for executing particular tasks or generating desired results. Researchers have observed that the design of specific prompt statements can steer pre-trained models to perform various tasks, such as question-answering, text generation, and semantic understanding [39; 40; 41; 42; 43; 44; 45; 46; 47; 48; 49; 50]. The strength of this approach lies in its ability to adapt to different tasks through simple modifications to prompt statements, eliminating the need for retraining the entire model. For LLMs like the GPT series and other pre-trained models, prompt learning provides a straightforward and powerful means for model fine-tuning. By supplying appropriate prompts, researchers and practitioners can customize the model's behavior, making it more suitable for specific domains or task requirements. In short, prompt learning is a machine learning approach that, builds upon pre-trained language models, and guides the model to perform various tasks through the design of prompt statements, offering increased flexibility for customizing model applications. In this Section, we will introduce the basic knowledge of prompt learning.

2.2.1. Background and Overview

Prompt learning is a new approach to machine learning [51]. In the early field of natural language processing (NLP), researchers mainly used fully supervised learning mode[52], which trained models for specific tasks on the input and output example dataset of the target task. However, due to the limited training dataset, this method cannot

train high-quality models well, so early NLP relied more on feature engineering; With the emergence of neural network models and their use in the field of NLP, people have begun to pay attention to architecture engineering [53].

However, between 2017 and 2019, the learning approach of NLP models shifted from fully supervised learning to a new mode: pre-train and fine-tune paradigm[54]. In this paradigm, a model with a fixed architecture is pre-trained as a language model to predict the probability of observed text data. Due to the abundant raw text data required for training language models, these language models can be trained on large datasets. During this process, language models can learn robust universal features of the language they are modeling. Then, by introducing additional parameters and fine-tuning them using task-specific objective functions, the PLM mentioned above will adapt to different downstream tasks. At this point, the focus of research shifted to objective engineering, which is to design training objectives during pre-training and fine-tuning. Since BERT, NLP has been using pre-training and fine-tuning methods for a long period of time, but this approach requires a new model to be fine-tuned for each task and cannot be shared. But for an LLM, it feels like customizing each task, which is very inefficient [51].

Prompt learning, this method has demonstrated amazing capabilities in GPT-3. The GPT-3 model can handle many tasks with only a few samples by using natural language prompts and task demonstrations as context, without updating parameters in the underlying model. Prompt Learning replaces the process of pre-trained and fine-tuning with pre-trained, prompts and predictions. In this paradigm, the downstream task is not to adapt the pre-trained LM to the downstream task through objective engineering, but to redefine the downstream task with the help of text prompts, making it look more like the tasks solved during the original LM training. For prompt learning, it is only necessary to insert different prompt parameters to adapt to different tasks. That is to say, each task only needs to train the prompt parameter separately, without the need to train the entire pre-trained language model[55]. This approach greatly improves the efficiency of using pre-trained language models and significantly shortens training time.

2.2.2. Basic components and process of Prompt learning

In the traditional pre-trained+fine-tuning paradigm, there is a gap between the pre-trained stage and downstream tasks [51], while prompt learning can maintain consistency between the pre-trained target format and downstream task output format, that is, align the form of downstream tasks with the form of PLMs pre-trained tasks. When training PLMs, we can transform the original target task into a fill-in-the-blank or continuation task similar to the pre-trained task of PLMs by constructing a prompt. The advantage of this method is that through a series of appropriate prompts, we can use a single language model to solve various downstream tasks.

Prompt learning optimizes the performance of models on different tasks by using pre-trained models and designing appropriate templates. Prompt learning consists of prompt templates, answer mappings, and pre-trained language models. The prompt template is the main body of the prompt, and fill in the blank [56] and generate based on prefix [57]are two common types of prompt learning templates. The fill-in-the-blank template selects one or more positions in the text and represents them with [MASK] tags, used to prompt the model to fill in the corresponding words; Prefix-based template generation involves adding a specific prefix before a sentence to guide the model in generating appropriate text. Answer mapping is the process of evaluating all possible answers according to a probability distribution, selecting the most likely answer as the predicted output, and converting it into appropriate category mapping words. This process typically involves converting labels into natural language vocabulary, known as Verbalizer [58].

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

(1)Use PLMs as base encoders

(2)Add additional context (template) with a [MASK] position

(3)Project labels to label words (verbalizer)

(4)Be the GAP between pre-training and fine-tuning

After defining the template and answer space, we need to choose a suitable pre-trained language model. There are now various pre-trained models (PTMs) with good performance, and when selecting a model, one usually considers its paradigm, such as Auto recursive, Masked Language Modeling, Encoder Decoder, etc. Based on this, for the summary task, a more suitable Bidirectional and Auto-Regressive Transformers (BART) model can be selected.

The selection of a template plays a very important role in the prompt learning. Templates can generally be distinguished based on whether they are manually specified: artificially constructed templates or automatically searched templates. Artificially created templates are the most intuitive method, easy to understand, and have good performance in practical applications. However, artificially constructed templates also have some drawbacks: prior knowledge is required when designing templates manually [59], and there may be failures [60]. There are two types of automatically

generated templates: discrete prompts and continuous prompts. Discrete prompts allow the model to select the optimal template in a set of discrete template spaces, while continuous prompts allow the language model to automatically train a prompt. According to research, using multiple templates [61] can improve the performance of the model. The simplest way to choose to use multiple templates and aggregate them together to complete an answer is to take the average [60] or weighted average of each template output [58].

Verbalizer is the process of mapping labels to label words, and the selection of verbalizers is also crucial for prompt learning. There are two ways to construct a verbalizer: manual definition and automatic search. The manual definition requires professional knowledge and may have disadvantages such as strong subjectivity and a small coverage area. To solve this problem, we can choose the following solutions: (1) Manually design with human prior knowledge; (2) Start with an Intel label word, paraphrase and expand; (3) Start with an internal label word, using external knowledge and expand; (4) Decompose the label with multiple tokens; (5) Virtual token and optimize the label embedding. In addition, we can use external knowledge bases to expand and improve label words, thereby achieving better text classification results[62].

2.2.3. learning strategy

The emergence of the new paradigm of Prompt learning has brought significant changes to the training process. The learning strategies for Prompt learning mainly include the following: (1) Pre-training then fine-tuning, which is a traditional pre-training+fine tuning method [63]; (2) Tuning free promotion, relying on the designer LM of prompts to directly provide answers [64]; (3) Fixed LM prompt tuning, which updates the relevant parameters of prompts using downstream task training data; (4) Fix prompt LM tuning, this strategy is to fine-tune the parameters of LM, which have fixed parameters when using prompts; (5) Prompt+LM tuning is a strategy that updates both prompts related parameters and LM parameters.

These different learning strategies can be selected based on specific tasks and needs. Pre-training + fine-tuning is the most common strategy, suitable for most tasks [63]. No fine-tuning prompts are suitable for simple tasks, which can greatly reduce training time and computational resource consumption. Fixed LM prompt fine-tuning and fixed prompt LM fine-tuning are suitable for tasks that require more precise control and can optimize model performance by adjusting prompt parameters or language model parameters. Combining prompts and LM fine-tuning combines the advantages of both and can further improve model performance [51].

In summary, Prompt learning provides us with a new training paradigm that can optimize model performance on various downstream tasks through appropriate prompt design and learning strategies. Choosing the appropriate template, constructing an effective verbalizer, and adopting appropriate learning strategies are all important factors in improving the effectiveness of prompt learning.

3. Training of Large Language Models

The training of LLMs can be broadly divided into three steps. The first step involves data collection and processing. The second step encompasses the pre-training process, which includes determining the model's architecture and pre-training tasks and utilizing suitable parallel training algorithms to complete the training. The third step involves fine-tuning and alignment. In this section, we will provide an overview of the model training techniques. This will include an introduction to the relevant training datasets, data preparation and preprocessing, model architecture, specific training methodologies, model evaluation, and commonly used training frameworks for LLMs.

3.1. Data Preparation and Preprocessing

3.1.1. Dataset

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. The first step in LLM training is collecting substantial corpora of natural language text. Pre-training data sources are diverse, commonly incorporating web text, conversational data, and books as general pre-training corpora. Additionally, some research efforts introduce specialized data from professional domains, such as code or scientific data, to enhance LLM capabilities in those fields. Leveraging diverse sources of text data for LLM training can significantly enhance the model's generalization capabilities. In the following section, we will present the commonly used datasets for training LLMs as shown in Table 1. These corpora are categorized into 5 groups for discussion.

Table 1
Commonly used corpora information.

Corpora	Type	Links
BookCorpus [65]	Books	https://github.com/soskek/bookcorpus
Gutenberg [66]	Books	https://www.gutenberg.org
Books1 [8]	Books	Not open source yet
Books2 [8]	Books	Not open source yet
CommonCrawl [67]	CommonCrawl	https://commoncrawl.org
C4 [68]	CommonCrawl	https://www.tensorflow.org/datasets/catalog/c4
CC-Stories [69]	CommonCrawl	Not open source yet
CC-News [70]	CommonCrawl	https://commoncrawl.org/blog/news-dataset-available
RealNews [71]	CommonCrawl	https://github.com/rowanz/grover/tree/master/realnews
RefinedWeb [72]	CommonCrawl	https://huggingface.co/datasets/tiiuae/falcon-refinedweb
WebText	Reddit Link	Not open source yet
OpenWebText [73]	Reddit Link	https://skylion007.github.io/OpenWebTextCorpus/
PushShift.io [74]	Reddit Link	https://pushshift.io/
Wikipedia [75]	Wikipedia	https://dumps.wikimedia.org/zhwiki/latest/
BigQuery [76]	Code	https://cloud.google.com/bigquery
CodeParrot	Code	https://huggingface.co/codeparrot
the Pile [77]	Other	https://github.com/EleutherAI/the-pile
ROOTS [78]	Other	https://huggingface.co/bigscience-data

Books: Two commonly utilized books datasets for LLMs training are BookCorpus [65] and Gutenberg [66]. These datasets include a wide range of literary genres, including novels, essays, poetry, history, science, philosophy, and more. Widely employed by numerous LLMs [9; 79], these datasets contribute to the models' training by exposing them to a diverse array of textual genres and subject matter, fostering a more comprehensive understanding of language across various domains.

CommonCrawl: CommonCrawl [67] manages an accessible repository of web crawl data, freely available for utilization by individuals and organizations. This repository encompasses a vast collection of data, comprising over 250 billion web pages accumulated over a span of 16 years. Established in 2007, Common Crawl has evolved into a widely recognized and referenced corpus in the academic and research communities, cited in more than 10,000 research papers. This continuously expanding corpus is a dynamic resource, with an addition of 3–5 billion new web pages each month. Its significance extends to the field of natural language processing, where it serves as a primary training corpus in numerous large language models. Notably, a substantial portion of the raw tokens employed in training GPT-3 [8], amounting to 82%, is sourced from the CommonCrawl. However, due to the presence of a substantial amount of low-quality data in web archives, preprocessing is essential when working with CommonCrawl data. Currently, four commonly used filtered datasets based on CommonCrawl are available: C4 [68], CC-Stories [69], CC-News [70], and RealNews [71].

Reddit Links: Reddit is a social media platform where users can submit links and posts, and others can vote on them using the "upvote" or "downvote" system. This characteristic makes it a valuable resource for creating high-quality datasets.

Wikipedia: Wikipedia [75], a free and open online encyclopedia project, hosts a vast repository of high-quality encyclopedic content spanning a wide array of topics. The English version of Wikipedia is extensively utilized in the training of many LLMs [8; 9; 80], serving as a valuable resource for language understanding and generation tasks. Additionally, Wikipedia is available in multiple languages, providing diverse language versions that can be leveraged for training in multilingual environments.

Code: There is a limited availability of publicly accessible code datasets at present. Existing efforts primarily involve web scraping of code with open-source licenses from the internet. The main sources include Github and Stack Overflow.

We have organized datasets utilized by distinct LLMs. During the training process, LLMs are typically trained on multiple datasets, as specified in Table 2 for reference.

Table 2
Datasets utilized by distinct LLMs

LLMs	Datasets
GPT-3 [8]	CommonCrawl [67], WebText2 [8], Books1 [8], Books2 [8], Wikipedia [75]
LLaMA [9]	CommonCrawl [67], C4 [68], Wikipedia [75], Github, Books, Arxiv, StackExchange
PaLM [36]	Social Media, Webpages, Books, Github, Wikipedia, News (total 780B tokens)
T5 [68]	C4 [68], WebText, Wikipedia, RealNews
CodeGen [81]	the Pile, BIGQUERY, BIGPYTHON
CodeGeeX [82]	CodeParrot, the Pile, Github
GLM [37]	BooksCorpus, Wikipedia
BLOOM [38]	ROOTS
OPT [83]	BookCorpus, CCNews, CC-Stories, the Pile, Pushshift.io

3.1.2. Data preprocessing

Once an adequate corpus of data is collected, the subsequent step is data preprocessing. The quality of data preprocessing directly impacts the model's performance and security. The specific preprocessing steps involve filtering low-quality text, including eliminating toxic and biased content to ensure the model aligns with human ethical standards. It also includes deduplication, removing duplicates in the training set, and excluding redundant content in the test set to maintain the sample distribution balance. Privacy scrubbing is applied to ensure the model's security, preventing information leakage or other privacy-related concerns. Additionally, if fine-tuning LLMs is considered, expanding the vocabulary should also be considered. On the other hand, LLaMA 2 models [10] represent a notable exception. These models forego filtering in their pretraining corpus, as aggressive filtration might accidentally filter out some demographic groups. This approach enhances the generalizability of the base LLaMA 2 models, making them more adept across a range of downstream tasks, such as hate speech detection and privacy de-identification. Observations indicate that abstaining from additional filtering in the pretraining data enables the base model to achieve reasonable safety alignment with fewer examples [10]. While this increases both generalizability and safety alignment efficiency, the implementation of additional safety mitigations is still imperative prior to public deployment, as further discussed in Section 3.5.4.

Quality filtering: Filtering low-quality data is typically done using heuristic-based methods or classifier-based methods. Heuristic methods involve employing manually defined rules to eliminate low-quality data [84; 72]. For instance, rules could be set to retain only text containing digits, discard sentences composed entirely of uppercase letters, and remove files with a symbol and word ratio exceeding 0.1, and so forth. Classifier-based methods involve training a classifier on a high-quality dataset such as WebText [85] to filter out low-quality datasets.

Deduplication: Language models may sometimes repetitively generate the same content during text generation, potentially due to a high degree of repetition in the training data. Extensive repetition can lead to training instability, resulting in a decline in the performance of LLMs [86]. Additionally, it is crucial to consider avoiding dataset contamination by removing duplicated data present in both the training and testing set [87].

Privacy scrubbing: LLMs, as text-generating models, are trained on diverse datasets, which may pose privacy concerns and the risk of inadvertent information disclosure [88]. In the preprocessing phase of language datasets, it is imperative to address privacy concerns by systematically removing any sensitive information. This involves employing techniques such as anonymization, redaction, or tokenization to eliminate personally identifiable details, geolocation, and other confidential data. By carefully scrubbing the dataset of such sensitive content, researchers and developers can ensure that the language models trained on these datasets uphold privacy standards and mitigate the risk of unintentional disclosure of private information. It is essential to strike a balance between data utility and privacy protection, fostering responsible and ethical use of language datasets in various applications.

Filtering out toxic and biased text: In the preprocessing steps of language datasets, a critical consideration is the removal of toxic and biased content to ensure the development of fair and unbiased language models. This involves implementing robust content moderation techniques, such as employing sentiment analysis, hate speech detection, and bias identification algorithms. By leveraging these tools [89], researchers can systematically identify and filter out text that may perpetuate harmful stereotypes, offensive language, or biased viewpoints.

3.2. Architecture

Currently, all LLMs are built upon the Transformer architecture, allowing these models to scale to several 10 billion or even a trillion parameters. Typically, PLM architectures fall into three categories: Encoder-only [90], Encoder-decoder [68] and Decoder-only [18]. The Encoder-only architecture is no longer employed in the latest LLMs and won't be further discussed here. Instead, this section will focus on introducing the Encoder-decoder and Decoder-only architectures.

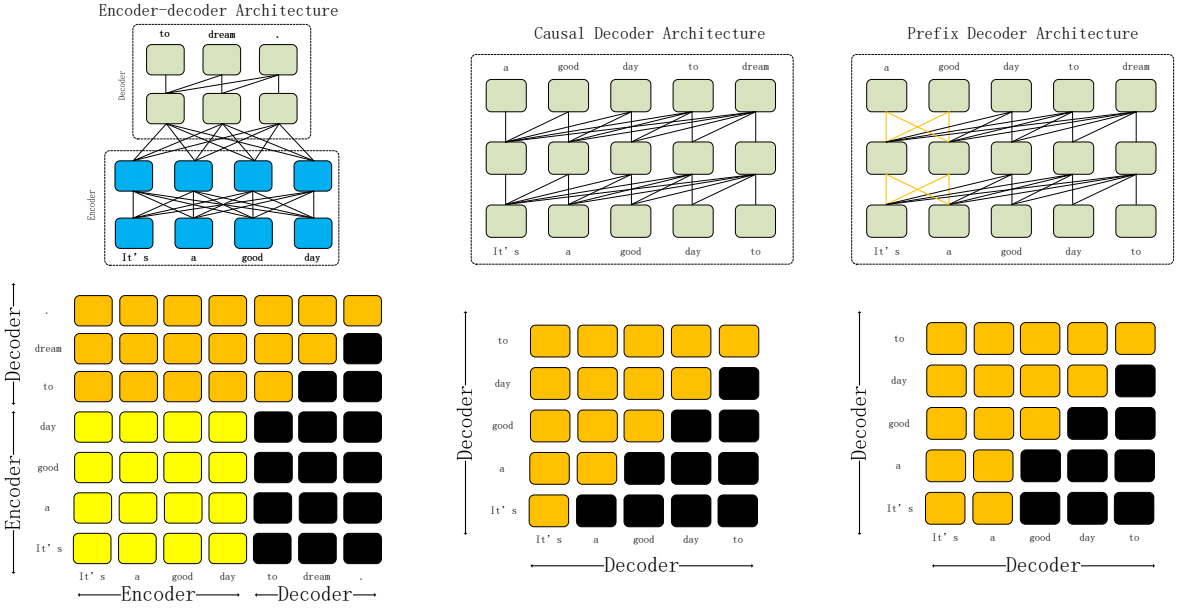

Figure 1: The figures from left to right represent the Encoder-decoder architecture, Causal Decoder architecture, Prefix Decoder architecture, and their mask configurations, respectively. This diagram illustrates the range of tokens that each input token can attend to.

3.2.1. Encoder-decoder Architecture

The Encoder-decoder architecture of LLMs is built upon the traditional Transformer Encoder-decoder architecture. The Encoder-decoder architecture consists of two main components: the Encoder and the Decoder. Each part of the Encoder is composed of multiple layers of Transformer's Multi-Head Self-Attention layers, which encode the input sequence. The Decoder, on the other hand, utilizes cross-attention over the output representation of the Encoder and generates the target sequence in an autoregressive manner. The encoder-decoder architecture serves as the foundation for prominent LLMs such as T5 [68], flan-T5 [91], and BART [92].

3.2.2. Decoder-only Architecture

LLMs with a Decoder-only architecture utilize the decoder component of the traditional Transformer architecture. Unlike the Encoder-Decoder architecture, which incorporates both an encoder and a decoder, the Decoder-only architecture is solely focused on the decoding process. In this configuration, the model sequentially generates tokens, attending to preceding tokens in the sequence. This architecture has been applied to various language generation tasks, showcasing its effectiveness in various tasks such as text generation without the need for an explicit encoding phase. The Decoder-only architecture can be further classified into two categories: the Causal Decoder architecture and the Prefix Decoder architecture.

The Causal Decoder Architecture: In the Causal Decoder architecture, each token in the model input sequence can only attend to past input tokens and itself during the decoding process. It achieves unidirectional attention to the input sequence by using a specific mask as shown in Figure 1. In fact, different architectures are mainly implemented by configuring different mask matrices. The figure illustrates a comparison of mask configurations between the Encoder-decoder and Decoder-only architectures (including Casual Decoder and Prefix Decoder). The representative LLMs for

the Causal Decoder architecture are the GPT series [18; 7; 8; 93; 19]. The GPT series of LLMs are currently known for their superior performance, with their foundational Causal Decoder architecture widely applied in other LLMs such as BLOOM [38], OPT [83], Gopher [84], and LLaMA [9].

The Prefix Decoder Architecture: The Prefix Decoder architecture combines the advantages of both the Encoder-decoder and Causal Decoder architectures. It leverages its unique mask configurations, as illustrated in Figure 1, enabling bidirectional attention for tokens in the prefix while maintaining unidirectional attention for generating subsequent tokens [54]. This design allows for the autoregressive generation of the output sequence with the flexibility to attend bi-directionally to the prefix tokens. Representative LLMs utilizing the Prefix Decoder architecture include PaLM [36] and GLM [37].

3.3. 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. Language modeling is one common form of self-supervised learning task in which the model is tasked with predicting the next word in a given context. Through this task, the model acquires the ability to capture information related to vocabulary, grammar, semantics, and text structure.

In language modeling [18; 7; 8; 36], the model is required to predict the next word in a given context. This task enables the model to develop a nuanced understanding of language. Specifically, the model observes large amounts of textual data and attempts to predict the next word at each position in the text. This gradual learning process allows the model to capture the patterns and information inherent in language, encoding a vast amount of linguistic knowledge into its parameters. Once pre-training is complete, these model parameters can be fine-tuned for various natural language processing tasks to adapt to specific task requirements. The objective of language modeling is to train a model to maximize the likelihood of textual data. For a given text sequence, denoted as $w_1, w_2, ..., w_T$, where w_t represents the token at position t, $P(w_t|w_1, w_2, ..., w_{t-1})$ is the probability of predicting w_t given the preceding context $w_1, w_2, ..., w_{t-1}$, the objective function for language modeling can be expressed using cross-entropy loss. Here, we define the objective as maximizing the conditional probability of the given text sequence:

$$L_{LM} = \frac{1}{T}\sum_{t=1}^{T} -log P(w_t|w_1, w_2, ..., w_{t-1}) \tag{5}$$

Language modeling serves as a prevalent pretraining objective for most LLMs. In addition to language modeling, there are other pretraining tasks within the realm of language modeling. For instance, some models [68; 37] use text with certain portions randomly replaced, and then employ autoregressive methods to recover the replaced tokens. The primary training approach involves the autoregressive recovery of the replaced intervals.

3.4. Model Training

3.4.1. Parallel Training

In the parallel training mentioned below, there will be discussions about collective communication which helps us better understand the principles of parallel training. Figure 2 has listed five reduction relationships. 1)Broadcast: Send data from one GPU to other GPUs.2)Reduce: Reduce(sum/average) data of all GPUs, send to one GPU.3)All Reduce: Reduce all data of GPUs, send to all GPUs.4)Reduce Scatter: Reduce all data of GPUs, send portions to all GPUs.5)All Gather: Gather data of all GPUs, send all GPUs.

Data Parallel: The process of data parallelism [94**?**] is shown in Figure 3, there is a parameter server that stores the model's parameters and the entire batch of data. Each GPU uses broadcast to synchronize the model parameters and divides the data into one portion per GPU, with each GPU receiving a portion of the data. Each GPU uses the complete model parameters and a portion of the data to perform forward and backward propagation. This way, the gradients are obtained for each GPU. Finally, we aggregate the gradients and send the aggregated gradients back to the parameter server, where the original model parameters and the aggregated complete gradients are available. With this information, we can use an optimizer to update the model parameters. The updated parameters will then enter the next round of model training iterations. Distributed data parallelism [95] abandons the use of a parameter server and instead employs all-reduce on gradient information, ensuring that each GPU receives the same gradient information. The result of all-reduce is communicated to all GPUs, allowing them to independently update their respective model optimizers. After each round of updates, the model's parameters, gradients, and the historical information of the optimizer are consistent across all GPUs.

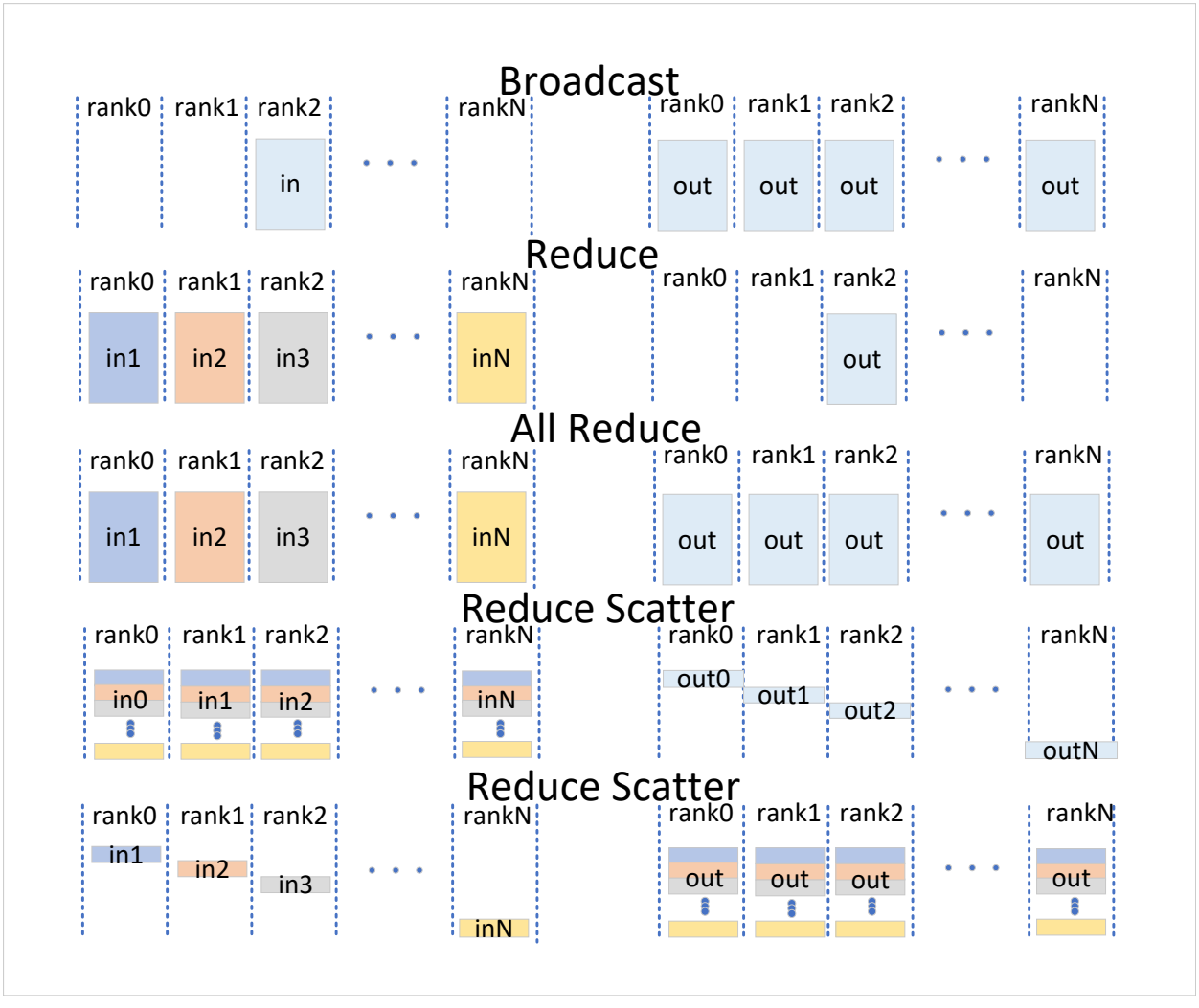

Figure 2: Five collective communications that are used by parallel training methods.

The occupation of GPU memory of intermediate results is related to the batch size, sentence length, and model dimensions. When using data parallelism, a batch of data is divided into many parts, allowing each GPU to process a portion of the data. In equivalent terms, the batch size processed on each GPU is reduced to one over the original number of GPUs. Data parallelism has reduced the input dimensions, resulting in an overall reduction in the intermediate results of the model. A drawback is that to support model training, each GPU needs to receive at least one piece of data. In the most extreme case, when each GPU receives only one piece of data, our parameters, gradients, and optimizer still need to be fully stored on the GPU. Even if we don't store any intermediate results on the GPU, our model may still be unable to perform computations on a single GPU.

Model Parallel: Model parallelism [96] was first introduced by Megatron-LM to alleviate memory pressure. From figure 4, we can clearly understand the overall architecture of model parallelism. Taking advantage of the most common linear layer in the Transformer as an example, the parameters of the linear layer form a matrix of size A*B, and the input to the linear layer is a vector of size B*1. Representing this as $y_{A*B} = W_{A*B}x_B$, we can horizontally partition the model's parameters into many segments using the property of matrix multiplication. Each segment is of size a divided by n multiplied by B. Utilizing the properties of matrix multiplication, we can move x_B into parentheses, and finally, the result of the linear layer is obtained by multiplying many small matrices with the parameters of the linear layer. Through this approach, the parameters of the linear layer can be distributed across multiple GPUs. However, it is

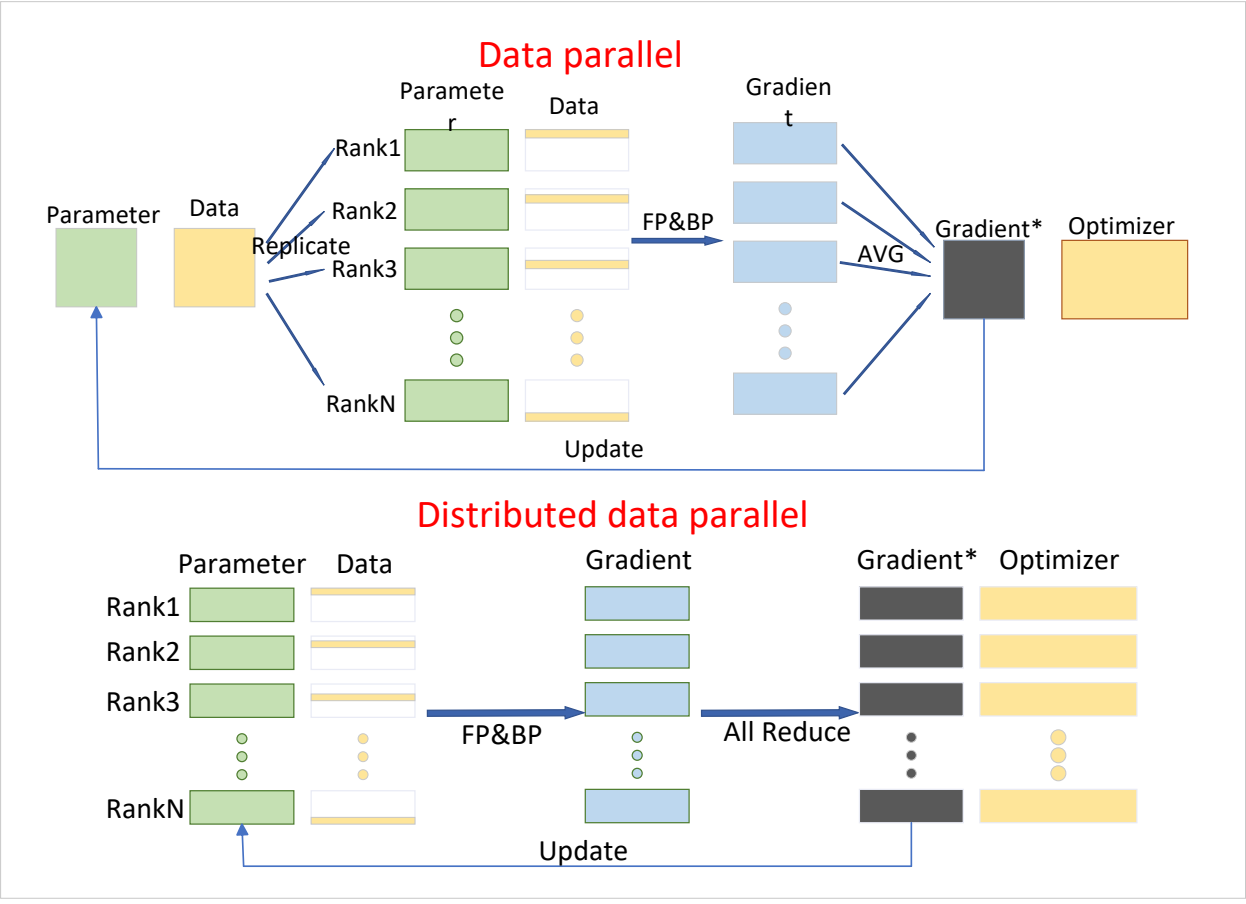

Figure 3: The architecture of data parallelism and distributed data parallelism. The diagram illustrates the difference between data parallelism and distributed data parallelism and the advantages of distributed data parallelism.

crucial to ensure that the inputs to the model on multiple GPUs are identical. Instead of using a data parallel approach to partition the data, we need to ensure that the inputs obtained on each GPU are the same, meaning they belong to the same batch of data. We can then partition a parameter like the linear layer across GPUs, with each GPU receiving a small portion of the matrix. By performing model calculations with this small portion and the data, we obtain a sub-result, as shown in Formula 5. The results of these computations need to be concatenated using the all-gather operator and communicated to all GPUs.

$$\begin{aligned} y_{A*B} &= W_{A*B} x_B \\ &= [W^{(1)}_{\frac{A}{n}*b}; W^{(2)}_{\frac{A}{n}*b}; ...; W^{(n)}_{\frac{A}{n}*b}] x_B \\ &= [W^{(1)}_{\frac{A}{n}*b} x_B; W^{(2)}_{\frac{A}{n}*b} x_B; ...; W^{(n)}_{\frac{A}{n}*b} x_B] \end{aligned} \tag{6}$$

ZeRO: ZeRO [97] is a framework built on data parallelism. During the parameter updating process on each GPU, the same set of parameters is used, leading to computational redundancy. Each GPU uses reduced scatter to eliminate this redundancy to obtain a portion of the gradient results. After updating a portion of the model parameters on each GPU, an all-gather operation is performed to synchronize the parameters across all GPUs. After the all-gather operation, the original gradient no longer needs to be saved on the graphics card and can be removed. Figure 5 shows the update of ZeRO. In ZeRO1, the original gradient is removed after backward propagation, while in ZeRO2, the product of the gradient* is calculated in advance during backward propagation, and only the gradient* is saved on the graphics card, removing the gradient. This way, the deletion of the gradient is advanced, leading to further savings in GPU

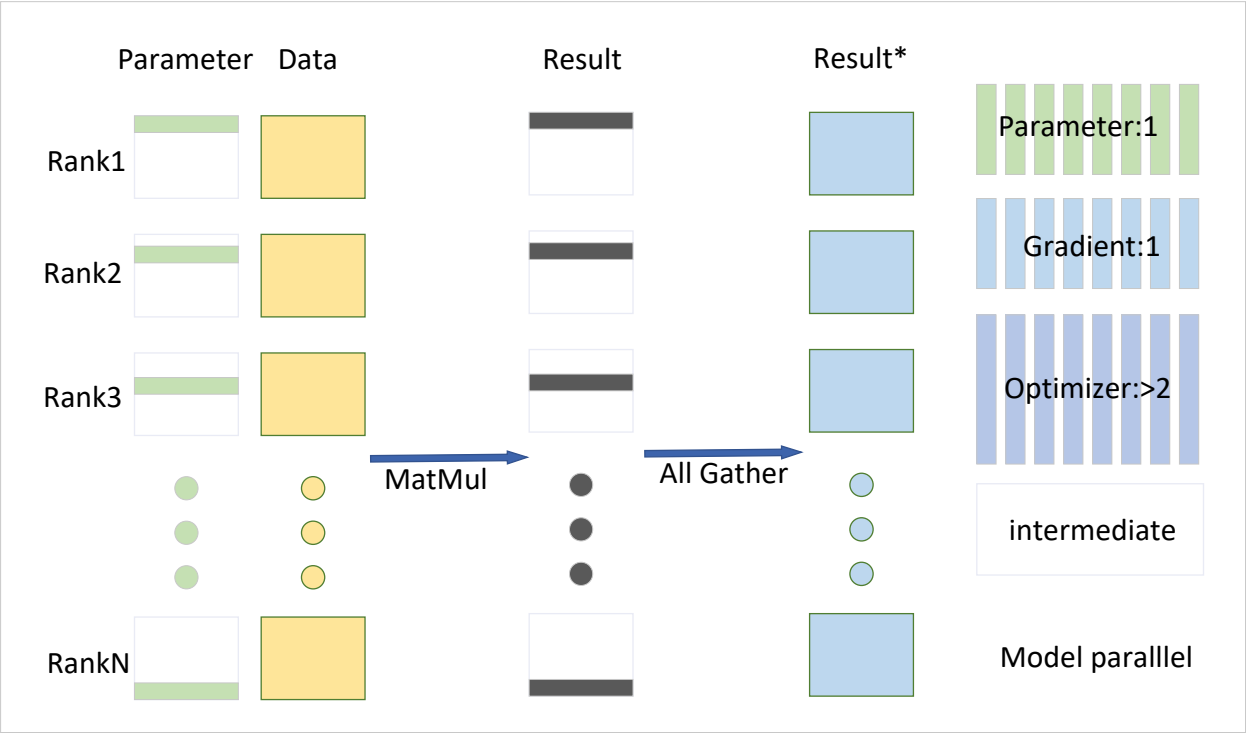

Figure 4: The overall architecture of model parallelism. The left side of the diagram shows the process of model parallelism, and the right side shows the memory usage of parameters, gradients, and optimizers in the graphics card of the model parallelism method.

memory space. ZeRO3 conducts a detailed division of the model parameters. Each graphics card retains only a portion of the gradients for updating, and parameter updates also only affect a portion of the model parameters. Therefore, each graphics card only needs to store the parameters, gradients, and optimizer related to the part of the parameters it is responsible for. During forward and backward propagation, an all-gather operation is required once, and after the operation is complete, the model parameters are released from the graphics card.Zero3 does not use all gather during parameter updates, but it requires an all-gather operation during both forward and backward propagation, adding one communication step. Compared to ZeRO2, ZeRO3 is an algorithm that trades time for space.

Pipeline Parallel: Pipeline parallelism [98] and model parallelism share similarities. In model parallelism, linear layers are divided into many small matrices, which are then distributed to different GPUs. For pipeline parallelism, different layers of the model are assigned to different GPUs. Specifically, if we have an n-layer transformer, we can assign the $layer_i$ of the transformer to the GPU_i, and so on. During the forward propagation of the model, we need to perform the computation of the $layer_i$ on the GPU_i, then pass the result to the GPU_{i+1}. The GPU_{i+1} receives the output from the GPU_i, performs the computation for that layer and passes the result to the next GPU. This method partitions the parameters, gradients, optimizer, and intermediate results for each layer.

3.4.2. Mixed Precision Training

In recent years, to pre-train extremely large language models, some research [99] has begun to utilize 16-bit floating-point numbers (FP16) to reduce memory usage and communication overhead. FP16 has a smaller numerical range and lower precision in effective digits [100; 38], but computations tend to be faster than FP32. In general model training, FP32 is often used as the default representation for training parameters. However, in actual model training, the number of parameters in a model typically does not exceed the order of thousands, well within the numerical range of FP16. To improve computational speed, we can convert from FP32 to FP16. During parameter updates, the amount of the parameter is roughly equal to the gradient multiplied by the learning rate. The minimum value of FP16 is on the order of 1e-5. As the product of the gradient and learning rate is already well below the representation range of FP16, the parameter update would result in loss, known as underflow. Therefore, we represent the parameter update obtained by

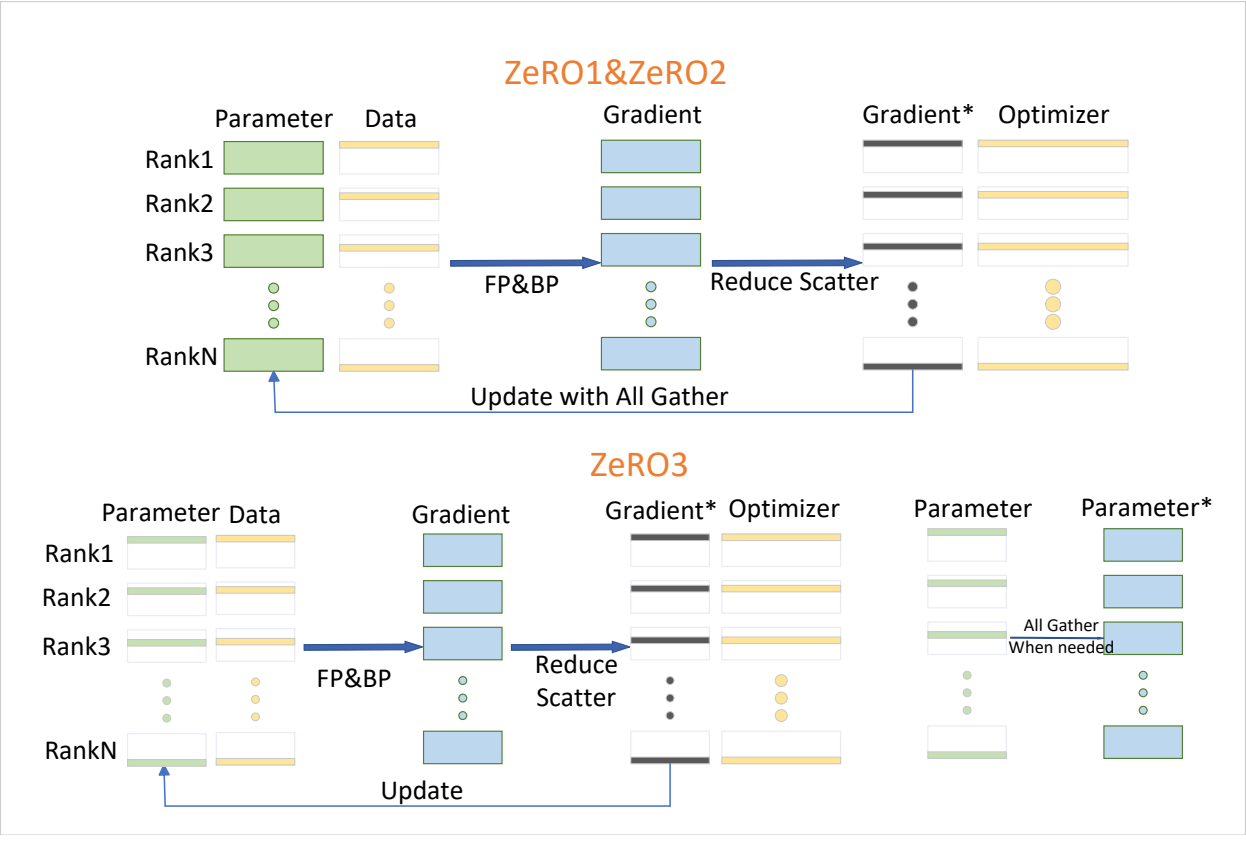

Figure 5: The overall architecture of ZeRO. The upper demonstrates ZeRO stage1 and ZeRO stage2. The lower displays ZeRO stage3. The graph illustrates the optimization of memory usage of graphics card parameters in relation to ZeRO3 versus ZeRO1 and ZeRO2

multiplying the gradient by the learning rate as FP32. We cannot directly add this high-precision parameter update to a lower-precision model, as this would still result in floating-point underflow. Consequently, we need to save an additional single-precision parameter on the optimizer. To accelerate both forward and backward passes in the model, half-precision parameters and gradients are used and passed to the optimizer for updating. The optimizer's update quantity is saved as FP32, and we accumulate it effectively through a temporarily created FP32 parameter in the optimizer. After effective accumulation, it is then converted back to FP16 parameters.

3.4.3. Offloading

The parameters in the optimizer are at least twice as many as the model parameters, and a study [101]proposes the idea of moving the optimizer's parameters from the GPU to the CPU. Although GPU computation is much faster than CPU, the question arises whether offloading this operation could become a bottleneck for the overall training speed of the model optimizer. In reality, we utilize ZeRO3. After the optimization with ZeRO3, the size of the parameters, gradients, and optimizer is reduced to 1/n of the number of GPUs. By binding one GPU to multiple CPUs, we effectively lower the computational load on each CPU.

3.4.4. Overlapping

Memory operations are typically asynchronous. Thus, We can send a request to the memory in advance and then proceed with other computations. After completing other computations, we come back to handle the memory request. This two-step operation is used in the forward propagation process of model training. We need to obtain the parameters of $layer_i$ through a gather operation. After obtaining the parameters of $layer_i$, in the forward propagation process of $layer_i$, we proactively retrieve the parameters of $layer_{i+1}$ through an asynchronous fetch. Once the forward propagation

Table 3
Commonly used instruction tuning datasets.

Datasets	Links
static-hh	https://huggingface.co/datasets/Dahoas/static-hh
OIG	https://huggingface.co/datasets/laion/OIG
Self-Instruct [102]	https://github.com/yizhongw/self-instruct
Natural instructions [103]	https://github.com/allenai/natural-instructions
P3 [104]	https://huggingface.co/datasets/bigscience/P3
Promptsource [105]	https://github.com/bigscience-workshop/promptsource
WebGPT [106]	https://huggingface.co/datasets/openai/webgpt_comparisons
Flan [107]	https://github.com/google-research/flan
MVPCorpus [108]	https://github.com/RUCAIBox/MVP

calculation for $layer_i$ is completed, the parameters for $layer_{i+1}$ have been obtained and are stored in the GPU. We can then immediately proceed with the forward propagation calculation, and so on.

3.4.5. Checkpoint

In order to support the backward propagation of the model, All intermediate results in the GPU memory need to be saved during the forward propagation of the model. To optimize this process, a checkpoint mechanism, which does not save all intermediate results in the GPU memory but only retains certain checkpoint points is utilized.

The diagram below illustrates a simplified structure of a transformer. Each transformer block takes a model input, undergoes complex computations through attention and feed-forward processes, and produces the overall output of that layer. We keep only the input of each major layer in the transformer as our checkpoint.

During the backward propagation process, how do we compute the gradients of the linear layers within each major layer? We can perform a technique called recomputation, which involves re-executing the forward pass of each major layer during the backward propagation process. We temporarily obtain the inputs of the linear layers within each major layer, and the intermediate results obtained can be used for backward propagation. Once the backward propagation for that layer is complete, we can discard the checkpoint and the temporarily recomputed intermediate results of the linear layers within the model from the GPU memory.

Assuming we have a transformer with 24 layers, each layer containing four to five linear layers, using the checkpoint mechanism reduces the originally required storage of 120 intermediate results to only 24 intermediate results.

3.5. Fine-Tuning

The training of LLMs in this paper is divided into three stages: data collection and processing, pre-training, and fine-tuning. This section will provide a review of the fine-tuning methods for LLMs. Specifically, we categorize fine-tuning techniques into three types: supervised fine-tuning (SFT) [93], alignment tuning, and parameter-efficient tuning.

3.5.1. Supervised Fine-Tuning

The core concept of supervised fine-tuning involves adjusting the model in a supervised manner on the basis of large-scale pre-training, enhancing its capability to better adapt to the specific requirements of the target task. In the process of SFT, it is necessary to prepare a labeled dataset for the target task, which includes input text along with corresponding labels. Instruction tuning is a commonly used technique in the fine-tuning process of LLMs and can be considered as a specific form of SFT. It involves further training LLMs on a dataset composed of (instruction, output) pairs, focusing on enhancing the capabilities and controllability of large language models by understanding and following human instructions. We compiled commonly used instruction tuning datasets, as illustrated in Table 3.

3.5.2. Alignment Tuning

Due to LLMs being pre-trained on massive and diverse internet data, even though the training data undergoes some preprocessing, it is still challenging to guarantee the absence of biased or harmful content in terabyte-scale training datasets. Despite LLMs demonstrating impressive performance across various natural language processing tasks, they frequently exhibit behaviors diverging from human intent. This includes generating false information,

producing expressions with bias or misleading content, and so on [93; 109]. To address these issues of LLMs displaying behaviors beyond human intent, alignment tuning becomes crucial [93; 110].

In general, alignment tuning aims to meet the following three criteria: being helpful, honest, and harmless.

Helpful: The concept of helpfulness revolves around whether the model-generated output proves genuinely beneficial for a specific task or inquiry. In the realm of natural language processing, the model's generated text or responses should furnish valuable information, positively impacting the user's requirements or task objectives.

Honest: Honesty entails whether the model-generated output is authentic and reliable. The model should produce information consistent with facts, steering clear of fabrication or distortion. This contributes to maintaining user trust in the authenticity of the model's outputs.

Harmless: Harmlessness is concerned with whether the model-generated output poses no harm to users or society. The model should refrain from generating content that is harmful, offensive, or perilous, ensuring its utilization remains safe for all relevant stakeholders.

In training LLMs, a noteworthy approach to alignment tuning is based on Reinforcement Learning with Human Feedback (RLHF) [93]. This method involves collecting human feedback data to train a reward model (RM) for reinforcement learning. The RM serves as the reward function during reinforcement learning training, and algorithms such as Proximal Policy Optimization (PPO) [111] are employed to fine-tune the LLM. In this context, LLM is considered as the policy, and the action space is considered as the vocabulary of the LLM.

3.5.3. Parameter-efficient Tuning

Currently, large-scale PLMs such as ChatGPT [93; 19] continue to grow in scale. However, for the majority of researchers, conducting full fine-tuning on consumer-grade hardware has become cost-prohibitive and impractical. Unlike SFT and alignment tuning, the objective of parameter-efficient tuning is to reduce computational and memory overhead. This method involves fine-tuning only a small or additional subset of model parameters while keeping the majority of pre-trained parameters fixed, thereby significantly lowering computational and storage costs. It is noteworthy that state-of-the-art parameter-efficient tuning techniques have achieved performance levels comparable to full fine-tuning. Some common parameter-efficient tuning methods include Low-Rank Adaptation (LoRA) [112], Prefix Tuning [113] and P-Tuning [114; 115]. The adoption of these methods enables efficient model tuning even in resource-constrained environments, offering feasibility and efficiency for practical applications.

With the rise of LLMs, parameter-efficient tuning has garnered increasing attention, with LoRA being widely employed in the latest releases of LLMs. LoRA [112] and its related advancements [116; 117] are noteworthy and deserve attention.

3.5.4. Safety Fine-Tuning

To enhance the safety and responsibility of LLMs, the integration of additional safety techniques during fine-tuning is essential. This encompasses three primary techniques, applicable to both SFT and RLHF phases.

Supervised Safety Fine-Tuning: In this technique, labelers are tasked with generating demonstration data that incorporates high safety risk adversarial prompts. This handcraft safety demonstration data is then incorporated into the SFT phase, thereby augmenting the model's capacity to manage safety risks.

Safety RLHF: This technique employs the same or even more aggressive adversarial prompts to query the models. The safest response exhibiting refusal behavior is then used to train a safety reward model within the RLHF framework.

Safety Context Distillation: This technique employs context distillation [118] by initially prefixing safety preprompts, like "You are a safe and responsible assistant," to adversarial prompts. This process yields safer generated responses. The model is then fine-tuned on these safer demonstration data but without the inclusion of the safety pre-prompts. This safety distillation further enhances the model's safety capabilities.

3.6. Evaluation

Unlike in the past, large-scale deep learning models have a wider range of applications and stronger performance compared to ordinary models. However, with great power comes great responsibility, and evaluating these models has become more complex, requiring consideration of potential problems and risks from all aspects. Since the popularity of ChatGPT, many related studies have been published, including the survey and summary of LLMs evaluation in reference [119; 120], which is helpful for developing large-scale deep learning models. This section will introduce some testing datasets, evaluation directions and methods, and potential threats that need to be considered based on previous evaluation work on large models.

3.6.1. Static testing dataset

The evaluation of large models' capabilities requires appropriate datasets for validation. Here, we introduce several commonly used datasets for testing purposes. Considering multimodal large models, typical datasets for computer vision include ImageNet [121] and Open Images [122]. In addition to the commonly used GLUE [123] and SuperGLUE [124] for LLMs, MMLU [125] is highly competitive in testing comprehensive capability. If your model primarily uses Chinese language, then CMMLU [126], as a benchmark for Chinese large models, should also be considered, and XTREME [127] and XTREME-R [128] are suitable choices for multilingual large models. For assessing mathematical knowledge capabilities, there are datasets such as MATH [129] and GSM8K [130], while HumanEval [131] and MBPP [132] can serve as benchmarks for code generation. For common sense reasoning tests in daily human life and work, the following datasets are available: HelloSwag [133], PIQA [134], BoolQ [135], SIQA [136], WinoGrande [137], ARC [138], and OpenBookQA [139]. For medical knowledge, there are datasets such as MedQA-USMLE [140] and MedMCQA [141].

3.6.2. Open domain Q&A evaluation

Currently, LLMs interact with humans in the form of questions and answers. Compared to the fragmented and ambiguous information returned by traditional searches, LLMs provide more realistic and efficient question-and-answer results that align with human habits. Therefore, the evaluation of ODQA (Open Domain Question Answering) [142] capability is essential. The performance of open-domain question answering greatly affects user experience. Commonly used datasets for testing include SquAD [143] and Natural Questions [144], with F1 score and Exact-Match accuracy (EM) as evaluation metrics. However, note that the method of word matching may have certain issues, such as when a factually correct answer is not in the golden answer list. Therefore, human evaluation seems to be necessary, and literature [145] has conducted detailed research on this matter.

3.6.3. Security evaluation

As an emerging and hot research field, LLMs must pay attention to their potential security threats, prevent malicious use or vulnerabilities to malicious attacks, and address any potential long-term issues that may pose a threat to human development. Additionally, red teaming in various domains is necessary to critically assess and test the model, identifying vulnerabilities, biases, inaccuracies, and areas for safety improvement.

Potential bias: The training data for LLMs may contain potential biases, such as gender or race. Security assessments need to address whether the model generates or amplifies these biases and how to reduce or correct them. Reference [146] discusses in detail the causes of bias in LLMs and the serious consequences that may arise. Reference [147] extensively studies how pre-trained language models generate harmful content to what extent, and how to use controlled text generation algorithms to prevent the generation of toxic content. CHBias [148] is a Chinese dataset that can be used to evaluate and mitigate the bias problem of LLMs.

Privacy protection: LLMs may come into contact with a large amount of user data, such as text and images, during the training process. Security assessments need to ensure the effective protection of user data privacy to prevent leaks and misuse. Reference [149] conducted research on models like ChatGPT and found that it is possible to extract training data effectively from these models. Reference [150] provides a solution by proposing a framework called DEPN (Detect and Editing Privacy Neurons) to detect and edit privacy neurons in pre-trained language models. It also introduces a privacy neuron aggregator to eliminate privacy information in a batch-processing manner, effectively reducing the leakage of privacy data while maintaining model performance.

Adversarial attacks: LLMs may be susceptible to adversarial attacks, such as input tampering, intentional misinformation, or generating false information. Security assessments need to consider the robustness of the model, i.e., its ability to withstand such attacks. As mentioned in reference [151], LLMs still have "jailbreak" risks, where users can manipulate the model to generate toxic content using specific input methods like role-playing or adding special suffixes as studied in the referenced paper. Especially when using open-source pre-trained models, any vulnerabilities in the pre-training models regarding adversarial attacks are inherited as well. Reference [152] provides a solution to mitigate the harm caused by these vulnerabilities.

3.6.4. Evaluation method

Automated evaluation and manual evaluation play crucial roles in Language Model (LLM) research. Automated evaluation typically involves using various metrics and indicators to quantify the performance of models, such as BIEU [153], ROUGE [154], and BERTSScore [155], which can measure the accuracy of LLM-generated content.

These metrics can help researchers quickly assess model performance on large-scale data and compare different models. However, automated evaluation also has limitations as it cannot fully capture the complexity of language understanding and generation. Research in reference [156] has shown that manual evaluation is more reliable for some open-ended generation tasks. Manual evaluation typically involves human annotators subjectively judging and assessing the quality of model-generated outputs. This evaluation method can help reveal how models perform in specific tasks or scenarios and identify subtle issues and errors that automated evaluation may overlook. However, manual evaluation also faces challenges such as high time costs and subjectivity. Therefore, it is often necessary to combine the strengths of automated and manual evaluation to comprehensively assess the performance of language models.

3.7. LLM Framework

Large deep learning models offer significant accuracy gains, but training billions to trillions of parameters is challenging. Existing solutions such as distributed training have solved fundamental limitations to fit these models into limited device memory while obtaining computation, communication, and development efficiency. Next, this section will introduce several large language model frameworks that utilize distributed training technology leveraging GPU, CPU, and NVMe memory to allow for unprecedented model scale on limited resources without requiring model code refactoring.

Transformers Transformers[157], an open-source Python library by Hugging Face, is dedicated to building models using the Transformer architecture. Featuring a simple and user-friendly API, it facilitates easy customization of various pre-trained models. With a robust community of users and developers, transformers continuously update and improve models and algorithms.

DeepSpeed: Deepspeed [158], an open-source optimization library compatible with PyTorch, is developed by Microsoft and utilized in training LLMs like MTNLG [79] and BLOOM [38]. Currently, It provides full support for ZeRO technology which includes Optimizer state partitioning, Gradient partitioning and parameter partitioning, Custom mixed precision training, A range of fast CUDA-extension-based optimizers [159] and ZeRO-offload to CPU and Disk/NVMe. Through the above technologies. Additionally, Deepspeed has achieved excellent scalability and efficiency with small memory requirements.

BMTrain: BMTrain [160] is an efficient large model training toolkit developed by Tsinghua University that can be used to train large models with tens of billions of parameters. It can train models in a distributed manner while keeping the code as simple as stand-alone training. BMTrain does not require model refactoring to work. In fact, PyTorch users can enable BMTrain with a few lines of code change to their existing training pipeline. It provides the support of various optimization techniques such as ZeRO optimization and communication optimization.

Megatron-LM: Megatron-LM [96; 161; 162] is a deep learning library developed by NVIDIA for training large-scale language models.Megatron-LM presents their techniques including model and data parallelism, mixed-precision training, and FlashAttention for training very large transformer models. Specifically, it takes advantage of the structure of transformer networks to create a simple model parallel implementation by adding a few synchronization primitives and it enables training transformer models with billions of parameters and trains efficiently in PyTorch.It also performs an in-depth empirical analysis of their model and data parallel technique and demonstrates up to 76% scaling efficiency using 512 GPUs which can largely improve the training efficiency and speed, enabling efficient distributed training across GPUs.

In addition to the aforementioned frameworks, Colossal-AI [163] and FastMoE [164; 165] are also two popular frameworks for training LLMs. In principle, any deep learning framework that supports parallel computing can be used to train LLMs. Examples include PyTorch [166], TensorFlow [167; 168], PaddlePaddle [169], MXNet [170], OneFlow [171], MindSpore [172] and JAX [173].

4. Inference with Large Language Models

The scale of large models is growing at a rate of nearly 10 times per year, which brings about huge computational consumption and carbon emissions [174]. Therefore, reducing the computational burden of training large models while retaining their reasoning ability has become a common concern for everyone. In this chapter, we mainly introduce how to reduce costs from both computational and storage aspects, that is, how to efficiently perform large-scale model inference from four aspects: model compression, memory scheduling, parallelism, and structural optimization.

4.1. Model Compression

4.1.1. Knowledge Distillation

Knowledge Distillation [175] refers to transferring knowledge from a cumbersome (teacher) model to a smaller (student) model that is more suitable for deployment. This is achieved by fitting the soft targets of the two models, as soft targets provide more information than gold labels. Initially, the calculation for model distillation involved only fitting the outputs from the last layer of both the teacher and student models [176]. PKD [177] improves this process by computing the mean-square loss between normalized hidden states, allowing the student model to learn from multiple intermediate layers of the teacher model. In order to discover more intermediate representations suitable for knowledge distillation, Jiao et al. [178] proposed Tiny BERT. This enables the student model to learn from the embedding layer and attention matrices of the teacher model.

4.1.2. Model Pruning

Model pruning involves removing redundant portions from the parameter matrices of large models. It is divided into unstructured pruning and structured pruning. Unstructured pruning involves removing individual connections or weights in a neural network without adhering to any specific structural pattern. In structured pruning, specific structural patterns or units within a neural network are pruned or removed. Gordon et al. [179] compared the effects of unstructured and structured pruning on the BERT model. They found that the effectiveness of unstructured pruning significantly decreases as the pruning ratio increases, while in structured pruning, 30-40% of the weights can be discarded without affecting BERT's universality. Different structures in the model can be structurally pruned. Michel et al. [180] pruned attention heads and found that ablating one head often positively impacts the performance of WMT and BERT. They proposed a gradient-based metric for evaluating the importance of attention heads to enhance pruning effectiveness. Fan et al. [179] performed layer pruning by extending dropout from weights to layers. During training, they randomly dropped layers and achieved good inference results by selecting sub-networks with any desired depth during testing.

4.1.3. Model Quantization

The fundamental idea behind model quantization is to reduce the number of floating-point bits used in numerical calculations within a large model network, thereby decreasing storage and computation costs. This involves converting floating-point operations into fixed-precision operations. However, as precision decreases, the model's loss gradually increases, and when precision drops to 1 bit, the model's performance experiences a sudden decline. To address the optimization challenges introduced by low-precision quantization, Bai et al. [181] proposed BinaryBERT. They initially trained a half-sized ternary model and then initialized a binary model with the ternary model through weight splitting. Finally, they fine-tuned the binary model. This approach yielded better results for the binary model compared to training a binary model from scratch.

4.1.4. Weight Sharing

The basic idea of weight sharing is to use the same set of parameters for multiple parts of a LLM. Instead of learning different parameters for each instance or component, the model shares a common set of parameters across various parts. Weight sharing helps reduce the number of parameters that need to be learned, making the model more computationally efficient and reducing the risk of overfitting, especially in situations where there is limited data. ALBERT [182] uses the Cross-layer parameter-sharing strategy to effectively reduce the number of parameters of the model, and can achieve better training results than the baseline with the same parameter number.

4.1.5. Low-rank Approximation

Low-rank decomposition methods are crucial in the field of model compression, as they allow for the creation of more compact models with fewer parameters. This reduction in model size is particularly beneficial for deploying neural networks on resource-constrained devices, improving efficiency during inference. Chen et al. [183] performed a low-rank decomposition on the input matrix, enabling matrix operations within the large model to occur at a lower-rank level, effectively reducing the computational workload. From the results, their proposed method, DRONE, not only ensures the inference performance of the large model but also achieves an acceleration ratio of more than 1.3 times compared to the baseline method. The specific choice of low-rank decomposition method depends on the architecture of the neural network and the requirements of the target application.

4.2. Memory Scheduling

Deploying LLMs on a single consumer-grade GPU is constrained by the limitations of the available video memory, given the substantial parameters of LLMs. Therefore, appropriate Memory Scheduling strategies can be used to solve the hardware limitations of large model inference. Memory scheduling in large model inference involves the efficient organization and management of memory access patterns during the reasoning or inference phase of complex neural network models. In the context of sophisticated reasoning tasks, such as natural language understanding or complex decision-making, large models often have intricate architectures and considerable memory requirements. Memory scheduling optimizes the retrieval and storage of intermediate representations, model parameters, and activation values, ensuring that the inference process is both accurate and performed with minimal latency. For example, BMInf [184] utilizes the principle of virtual memory, achieving efficient inference for large models by intelligently scheduling the parameters of each layer between the GPU and CPU.

4.3. Parallelism

Both inference and training can leverage parallelization techniques. Presently, parallelization techniques for inference primarily manifest across three dimensions: Data Parallelism, Tensor Parallelism, and Pipeline Parallelism. Data Parallelism primarily involves increasing the overall throughput of the inference system by adding more GPU devices [101; 97; 159; 185]. Tensor parallelism is a form of model parallelism where the model's parameters are partitioned into multiple tensors, each computed on different processing units. This approach proves beneficial when dealing with models that are too large to fit into the memory of a single GPU. Tensor parallelism primarily involves increasing the number of devices horizontally through parallel computation to reduce latency [96]. Pipeline parallelism primarily involves vertically increasing the number of GPU devices through parallel computation to support larger models and enhance device utilization. Typically, it is combined with tensor parallelism to achieve optimal performance [98].

4.4. Structural Optimization

In the forward propagation computation of LLMs, the calculation speed is significantly faster than the speed of memory access. Inference speed can be impacted by numerous memory access operations. One goal in LLM inference is to minimize the number of memory accesses during forward propagation. FlashAttention [186] and PagedAttention [187] enhance computational speed by employing a chunked computation approach, mitigating the storage overhead associated with matrices. The entire operation takes place within SRAM, reducing the number of accesses to High Bandwidth Memory (HBM) and significantly boosting computational speed. Both FlashAttention and PagedAttention have been adopted by mainstream inference frameworks, and seamlessly integrated into these frameworks for straightforward utilization.

4.5. Inference Framework

Parallel computing, model compression, memory scheduling, and specific optimizations for transformer structures, all integral to LLM inference, have been effectively implemented in mainstream inference frameworks. These frameworks furnish the foundational infrastructure and tools required for deploying and running LLM models. They offer a spectrum of tools and interfaces, streamlining the deployment and inference processes for researchers and engineers across diverse application scenarios. The choice of a framework typically hinges on project requirements, hardware support, and user preferences. In Table 4, we compile some of these frameworks for reference.

5. Utilization of LLMs

The application scope of LLMs is extensive and can be practically employed in almost any specialized domain [1; 193; 46; 194; 195]. Following pre-training and fine-tuning, LLMs are primarily utilized by designing suitable prompts for various tasks. Leveraging powerful zero-shot capabilities, many tasks can be directly accomplished by guiding LLMs with straightforward prompts. For more complex tasks that cannot be achieved through simple prompts, a few-shot approach involving in-context learning is employed to guide LLMs in task completion. Additionally, incorporating chain-of-thought [196; 197] prompts in the prompt enhances in-context learning by introducing a reasoning process. The pipeline of the in-context learning and chain-of-thought is shown in Figure 6. In some specialized research directions, obtaining intermediate layer representations of LLMs may be necessary. For instance, in neuroscience studies, embedding representations from the model are used to investigate activation regions of brain functions [198; 199; 200; 201].

Table 4
List of LLM inference framework.

Framework	Links
TensorRT	https://github.com/NVIDIA/TensorRT-LLM
FasterTransformer	https://github.com/NVIDIA/FasterTransformer
Megatron-LM [96]	https://github.com/NVIDIA/Megatron-LM
FlexGen [188]	https://github.com/FMInference/FlexGen
DeepSpeed [158]	https://github.com/microsoft/DeepSpeed
vLLM [187]	https://github.com/vllm-project/vllm
FlexFlow [189]	https://github.com/flexflow/FlexFlow
StreamingLLM [190]	https://github.com/mit-han-lab/streaming-llm
ColossalAI [163]	https://github.com/hpcaitech/ColossalAI
BMCook [191]	https://github.com/OpenBMB/BMCook
BMInf [184]	https://github.com/OpenBMB/BMInf
Petals [192]	https://github.com/bigscience-workshop/petals

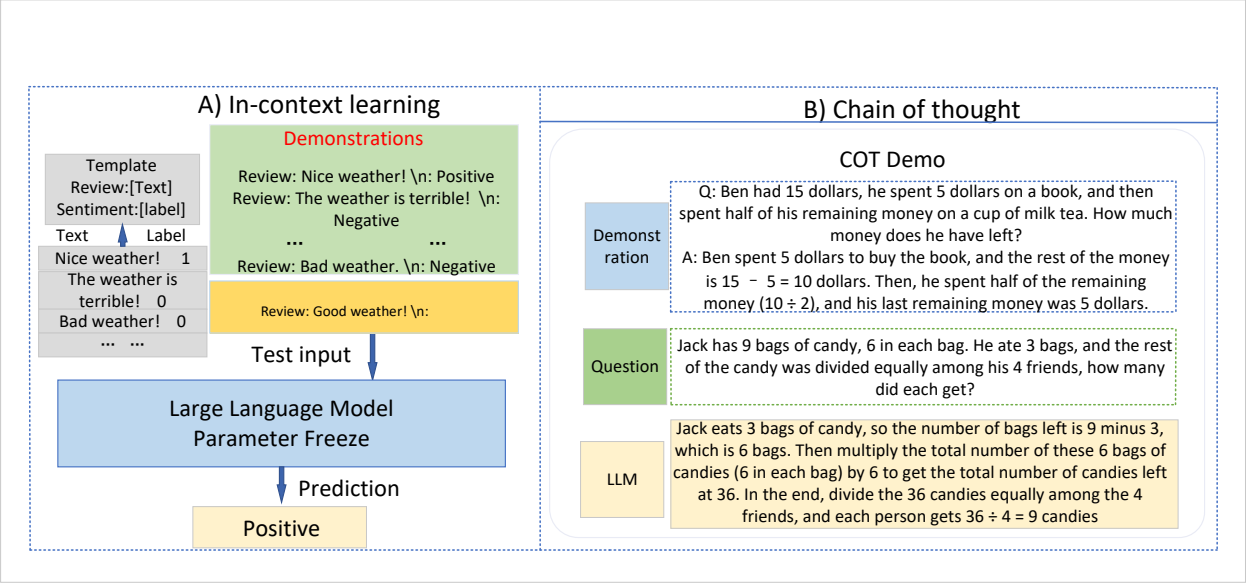

Figure 6: A) in-context learning, B) Chain of thought.

Generally, there are several approaches to employing LLMs. The first involves accessing the capabilities of robust proprietary models through open API services, such as utilizing the API provided by ChatGPT [19]. The second approach includes deploying open-source LLMs for local use [9]. The third method entails fine-tuning open-source LLMs to meet specific domain standards [43; 202], enabling their application in a particular field, and subsequently deploying them locally. In Table 5, we have compiled information on various open-source LLMs for reference. Researchers can choose from these open-source LLMs to deploy applications that best suit their needs.

6. Future Directions and Implications

This section will delve into the future trends and impact of LLM technology. Our discussion will be structured into three parts: firstly, an exploration of the developmental trends within LLMs technology itself; secondly, an examination of the developmental directions for AI researchers; and finally, an analysis of the societal impact resulting from the ongoing development of LLMs.

Based on existing experiences, it is evident that an ample supply of high-quality data and a sufficient number of parameters significantly contribute to enhancing the performance of models [8]. Looking ahead, the model scale of

Table 5
List of open source LLMs.

LLM	Size (B)	Links
T5 [68]	11B	https://github.com/google-research/text-to-text-transfer-transformer
CodeGen [81]	16B	https://github.com/salesforce/CodeGen
MOSS [203]	16B	https://github.com/OpenLMLab/MOSS
GLM [37]	130B	https://github.com/THUDM/GLM
ChatGLM [37]	6B	https://github.com/THUDM/ChatGLM3
ChatYuan [204]	0.7B	https://github.com/clue-ai/ChatYuan
OPT [83]	175B	https://github.com/facebookresearch/metaseq
BLOOM [38]	176B	https://huggingface.co/bigscience/bloom
LLaMA [9]	65B	https://github.com/facebookresearch/llama
CodeGeeX [82]	13B	https://github.com/THUDM/CodeGeeX
Baichuan [205]	13B	https://github.com/baichuan-inc/Baichuan2
Aquila	7B	https://github.com/FlagAI-Open/FlagAI/tree/master/examples/Aquila
MiniGPT-4 [206]	25B	https://github.com/Vision-CAIR/MiniGPT-4
Vicuna [207]	13B	https://github.com/lm-sys/FastChat

LLMs is expected to continue expanding, thereby augmenting their learning capabilities and overall performance. Moreover, the majority of currently available LLMs are confined to a single natural language modality, lacking extensions to process multimodal data such as images, videos, and speech. There is a potential future trajectory for LLMs to evolve towards handling information beyond text, incorporating multimodal data like images and audio. This evolution would empower models to comprehensively understand and generate multimodal content, significantly broadening the application scope of LLMs. The inevitable expansion of LLMs into the field of multimodality is bound to incur increased training costs. A pivotal focus for future developments lies in the efficient fine-tuning of parameters and the deployment of LLMs through techniques such as knowledge distillation, model compression, and quantization, aimed at reducing both the training and inference costs of LLMs. Another emerging trend is the domain-specific training and fine-tuning of LLMs for particular sectors, facilitating a more adept adaptation to and understanding of industry-specific terminologies and contexts. Lastly, in the exploration of potential new architectures for LLMs the current landscape predominantly relies on the transformer architecture. While the transformer architecture naturally boasts advantages such as parallel computing and adaptability to various input modalities, its design typically necessitates fixed-size inputs. This requirement may necessitate padding or truncation when dealing with variable-length sequences, potentially leading to computational and information inefficiencies, as well as challenges in generating coherent data. Investigating the potential of Recurrent Neural Network (RNN) architectures in the era of LLMs could emerge as a pivotal research direction. For instance, RWKV [208], an LLM designed under the RNN architecture, has demonstrated competitive performance on various third-party evaluations, proving itself comparable to the majority of transformer-based LLMs.

For researchers in the field of AI, working in isolation is becoming increasingly impractical. The future direction of AI development will intertwine with various industries, necessitating close collaboration with professionals from diverse fields. It is crucial to engage in collaborative efforts, bridging research disciplines, and collectively addressing challenges by combining expertise from different domains. Simultaneously, there is a fresh set of requirements for the comprehensive skills of AI researchers. Training and deploying LLMs necessitate proficiency in managing large-scale data and substantial practical experience in distributed parallel training. This criterion underscores the importance for researchers involved in LLM development to possess substantial engineering capabilities, addressing the challenges inherent in the process. Researchers who are interested in the field of LLMs must either possess engineering skills or adeptly collaborate with engineers to navigate the complexities of model development [3].

As LLMs find widespread applications in societal life, concerns about ethical issues and societal impact are on a continuous rise. This may involve research and improvements in areas such as managing model biases and controlling the risk of misuse [4]. Considering the paramount importance of privacy and data security, the future development of LLMs might involve more federated learning and decentralized approaches to enhance model performance while safeguarding user privacy. Developers should engage in interdisciplinary collaboration with experts from various fields, including decision-making, legal studies, and sociology, to establish standards and ethical frameworks for the

development, deployment, and utilization of LLMs, mitigating potential harmful consequences. In terms of public awareness and education, mandatory awareness training should be implemented before large-scale public deployment and applications. This aims to enhance public understanding of the capabilities and limitations of LLMs, fostering responsible and informed use, especially in industries such as education and journalism.

7. Conclusion

The introduction of ChatGPT has ushered in a transformative era in the realm of Large LLMs, significantly influencing their utilization for diverse downstream tasks. The emphasis on cost-effective training and deployment has emerged as a crucial aspect in the evolution of LLMs. This paper has provided a comprehensive survey of the evolution of large language model training techniques and inference deployment technologies in alignment with the emerging trend of low-cost development. The progression from traditional statistical language models to neural language models, and subsequently to PLMs such as ELMo and transformer architecture, has set the stage for the dominance of LLMs. The scale and performance of these models, particularly exemplified by the GPT series, have reached unprecedented levels, showcasing the phenomenon of emergence and enabling versatile applications across various domains. Notably, the release of ChatGPT by OpenAI in November 2022 has marked a pivotal moment in the LLM landscape, revolutionizing the strength and effectiveness of AI algorithms. However, the current reliance on OpenAI's infrastructure underscores the necessity for alternative LLMs, emphasizing the need for domain-specific models and advancements in the training and deployment processes.

Training and deploying LLMs present challenges that demand expertise in handling large-scale data and distributed parallel training. The engineering capabilities required for LLM development highlight the collaborative efforts needed between researchers and engineers. As we explore the technical aspects of LLM training and inference in this review, it becomes evident that a deep understanding of these processes is essential for researchers venturing into the field. Looking ahead, the future of LLMs holds promising directions, including further advancements in model architectures, improved training efficiency, and broader applications across industries. The insights provided in this review aim to equip researchers with the knowledge and understanding necessary to navigate the complexities of LLM development, fostering innovation and progress in this dynamic field. As LLMs continue to evolve, their impact on natural language processing and AI as a whole is poised to shape the future landscape of intelligent systems.

References

[1] Y. Liu, T. Han, S. Ma, J. Zhang, Y. Yang, J. Tian, H. He, A. Li, M. He, Z. Liu *et al.*, "Summary of chatgpt-related research and perspective towards the future of large language models," *Meta-Radiology*, p. 100017, 2023.

[2] J. Wang, E. Shi, S. Yu, Z. Wu, C. Ma, H. Dai, Q. Yang, Y. Kang, J. Wu, H. Hu *et al.*, "Prompt engineering for healthcare: Methodologies and applications," *arXiv preprint arXiv:2304.14670*, 2023.

[3] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[4] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, "Challenges and applications of large language models," *arXiv preprint arXiv:2307.10169*, 2023.

[5] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, Jun. 2018, pp. 2227–2237.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[7] A. Radford, J. Wu, D. Amodei, D. Amodei, J. Clark, M. Brundage, and I. Sutskever, "Better language models and their implications," *OpenAI Blog https://openai. com/blog/better-language-models*, vol. 1, no. 2, 2019.

[8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[9] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[10] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[11] S. Rezayi, H. Dai, Z. Liu, Z. Wu, A. Hebbar, A. H. Burns, L. Zhao, D. Zhu, Q. Li, W. Liu *et al.*, "Clinicalradiobert: Knowledge-infused few shot learning for clinical notes named entity recognition," in *International Workshop on Machine Learning in Medical Imaging*. Springer, 2022, pp. 269–278.

[12] Z. Liu, M. He, Z. Jiang, Z. Wu, H. Dai, L. Zhang, S. Luo, T. Han, X. Li, X. Jiang *et al.*, "Survey on natural language processing in medical image analysis." *Zhong nan da xue xue bao. Yi xue ban= Journal of Central South University. Medical Sciences*, vol. 47, no. 8, pp. 981–993, 2022.

[13] W. Liao, Z. Liu, H. Dai, Z. Wu, Y. Zhang, X. Huang, Y. Chen, X. Jiang, D. Zhu, T. Liu *et al.*, "Mask-guided bert for few shot text classification," *arXiv preprint arXiv:2302.10447*, 2023.

[14] S. Rezayi, Z. Liu, Z. Wu, C. Dhakal, B. Ge, H. Dai, G. Mai, N. Liu, C. Zhen, T. Liu *et al.*, "Exploring new frontiers in agricultural nlp: Investigating the potential of large language models for food applications," *arXiv preprint arXiv:2306.11892*, 2023.

[15] T. Zhong, W. Zhao, Y. Zhang, Y. Pan, P. Dong, Z. Jiang, X. Kui, Y. Shang, L. Yang, Y. Wei *et al.*, "Chatradio-valuer: A chat large language model for generalizable radiology report generation based on multi-institution and multi-system data," *arXiv preprint arXiv:2310.05242*, 2023.

[16] Z. Liu, T. Zhong, Y. Li, Y. Zhang, Y. Pan, Z. Zhao, P. Dong, C. Cao, Y. Liu, P. Shu *et al.*, "Evaluating large language models for radiology natural language processing," *arXiv preprint arXiv:2307.13693*, 2023.

[17] T. Zhong, Y. Wei, L. Yang, Z. Wu, Z. Liu, X. Wei, W. Li, J. Yao, C. Ma, X. Li *et al.*, "Chatabl: Abductive learning via natural language interaction with chatgpt," *arXiv preprint arXiv:2304.11107*, 2023.

[18] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," *OpenAI*, 2018.

[19] OpenAI, "Gpt-4 technical report," 2023.

[20] H. Dai, Z. Liu, W. Liao, X. Huang, Y. Cao, Z. Wu, L. Zhao, S. Xu, W. Liu, N. Liu, S. Li, D. Zhu, H. Cai, L. Sun, Q. Li, D. Shen, T. Liu, and X. Li, "Auggpt: Leveraging chatgpt for text data augmentation," 2023.

[21] Z. Liu, X. Yu, L. Zhang, Z. Wu, C. Cao, H. Dai, L. Zhao, W. Liu, D. Shen, Q. Li *et al.*, "Deid-gpt: Zero-shot medical text de-identification by gpt-4," *arXiv preprint arXiv:2303.11032*, 2023.

[22] C. Ma, Z. Wu, J. Wang, S. Xu, Y. Wei, Z. Liu, L. Guo, X. Cai, S. Zhang, T. Zhang *et al.*, "Impressiongpt: an iterative optimizing framework for radiology report summarization with chatgpt," *arXiv preprint arXiv:2304.08448*, 2023.

[23] W. Liao, Z. Liu, H. Dai, S. Xu, Z. Wu, Y. Zhang, X. Huang, D. Zhu, H. Cai, T. Liu *et al.*, "Differentiate chatgpt-generated and human-written medical texts," *arXiv preprint arXiv:2304.11567*, 2023.

[24] H. Dai, Y. Li, Z. Liu, L. Zhao, Z. Wu, S. Song, Y. Shen, D. Zhu, X. Li, S. Li *et al.*, "Ad-autogpt: An autonomous gpt for alzheimer's disease infodemiology," *arXiv preprint arXiv:2306.10095*, 2023.

[25] Z. Guan, Z. Wu, Z. Liu, D. Wu, H. Ren, Q. Li, X. Li, and N. Liu, "Cohortgpt: An enhanced gpt for participant recruitment in clinical study," *arXiv preprint arXiv:2307.11346*, 2023.

[26] Z. Liu, Z. Wu, M. Hu, B. Zhao, L. Zhao, T. Zhang, H. Dai, X. Chen, Y. Shen, S. Li *et al.*, "Pharmacygpt: The ai pharmacist," *arXiv preprint arXiv:2307.10432*, 2023.

[27] Y. Wei, T. Zhang, H. Zhang, T. Zhong, L. Zhao, Z. Liu, C. Ma, S. Zhang, M. Shang, L. Du *et al.*, "Chat2brain: A method for mapping open-ended semantic queries to brain activation maps," *arXiv preprint arXiv:2309.05021*, 2023.

[28] T. Zhong, X. Wei, E. Shi, J. Gao, C. Ma, Y. Wei, S. Zhang, L. Guo, J. Han, T. Liu *et al.*, "A small-sample method with eeg signals based on abductive learning for motor imagery decoding," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2023, pp. 416–424.

[29] J. Gao, L. Zhao, T. Zhong, C. Li, Z. He, Y. Wei, S. Zhang, L. Guo, T. Liu, J. Han *et al.*, "Prediction of cognitive scores by joint use of movie-watching fmri connectivity and eye tracking via attention-censnet," *Psychoradiology*, vol. 3, 2023.

[30] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[31] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *Ieee Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[32] Y. Yang, L. Wang, S. Shi, P. Tadepalli, S. Lee, and Z. Tu, "On the sub-layer functionalities of transformer decoder," *arXiv preprint arXiv:2010.02648*, 2020.

[33] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[34] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *Neurocomputing*, p. 127063, 2023.

[35] O. Press, N. A. Smith, and M. Lewis, "Train short, test long: Attention with linear biases enables input length extrapolation," *arXiv preprint arXiv:2108.12409*, 2021.

[36] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[37] A. Zeng, X. Liu, Z. Du, Z. Wang, H. Lai, M. Ding, Z. Yang, Y. Xu, W. Zheng, X. Xia *et al.*, "Glm-130b: An open bilingual pre-trained model," *arXiv preprint arXiv:2210.02414*, 2022.

[38] B. Workshop, T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," *arXiv preprint arXiv:2211.05100*, 2022.

[39] L. Zhao, L. Zhang, Z. Wu, Y. Chen, H. Dai, X. Yu, Z. Liu, T. Zhang, X. Hu, X. Jiang *et al.*, "When brain-inspired ai meets agi," *Meta-Radiology*, p. 100005, 2023.

[40] J. Holmes, Z. Liu, L. Zhang, Y. Ding, T. T. Sio, L. A. McGee, J. B. Ashman, X. Li, T. Liu, J. Shen, and W. Liu, "Evaluating large language models on a highly-specialized topic, radiation oncology physics," *Frontiers in Oncology*, vol. 13, Jul. 2023.

[41] Z. Wu, L. Zhang, C. Cao, X. Yu, H. Dai, C. Ma, Z. Liu, L. Zhao, G. Li, W. Liu *et al.*, "Exploring the trade-offs: Unified large language models vs local fine-tuned models for highly-specific radiology nli task," *arXiv preprint arXiv:2304.09138*, 2023.

[42] S. Rezayi, Z. Liu, Z. Wu, C. Dhakal, B. Ge, C. Zhen, T. Liu, and S. Li, "Agribert: knowledge-infused agricultural language models for matching food and nutrition," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, vol. 7, 2022, pp. 5150–5156.

[43] Z. Liu, A. Zhong, Y. Li, L. Yang, C. Ju, Z. Wu, C. Ma, P. Shu, C. Chen, S. Kim *et al.*, "Radiology-gpt: A large language model for radiology," *arXiv preprint arXiv:2306.08666*, 2023.

[44] Z. Liu, X. He, L. Liu, T. Liu, and X. Zhai, *Context Matters: A Strategy to Pre-train Language Model for Science Education*. Springer Nature Switzerland, 2023, p. 666–674.

[45] J. Wang, Z. Liu, L. Zhao, Z. Wu, C. Ma, S. Yu, H. Dai, Q. Yang, Y. Liu, S. Zhang *et al.*, "Review of large vision models and visual prompt engineering," *arXiv preprint arXiv:2307.00855*, 2023.

[46] X. Li, L. Zhang, Z. Wu, Z. Liu, L. Zhao, Y. Yuan, J. Liu, G. Li, D. Zhu, P. Yan *et al.*, "Artificial general intelligence for medical imaging," *arXiv preprint arXiv:2306.05480*, 2023.

[47] H. Cai, W. Liao, Z. Liu, Y. Zhang, X. Huang, S. Ding, H. Ren, Z. Wu, H. Dai, S. Li *et al.*, "Coarse-to-fine knowledge graph domain adaptation based on distantly-supervised iterative training," *arXiv preprint arXiv:2211.02849*, 2022.

[48] H. Dai, C. Ma, Z. Liu, Y. Li, P. Shu, X. Wei, L. Zhao, Z. Wu, D. Zhu, W. Liu *et al.*, "Samaug: Point prompt augmentation for segment anything model," *arXiv preprint arXiv:2307.01187*, 2023.

[49] L. Zhang, Z. Liu, L. Zhang, Z. Wu, X. Yu, J. Holmes, H. Feng, H. Dai, X. Li, Q. Li *et al.*, "Segment anything model (sam) for radiation oncology," *arXiv preprint arXiv:2306.11730*, 2023.

[50] Z. Xiao, Y. Chen, L. Zhang, J. Yao, Z. Wu, X. Yu, Y. Pan, L. Zhao, C. Ma, X. Liu *et al.*, "Instruction-vit: Multi-modal prompts for instruction learning in vit," *arXiv preprint arXiv:2305.00201*, 2023.

[51] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[52] S. B. Kotsiantis, I. Zaharakis, P. Pintelas *et al.*, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[53] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[54] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H.-W. Hon, "Unified language model pre-training for natural language understanding and generation," *Advances in neural information processing systems*, vol. 32, 2019.

[55] T. Schick and H. Schütze, "It's not just size that matters: Small language models are also few-shot learners," *arXiv preprint arXiv:2009.07118*, 2020.

[56] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?" *arXiv preprint arXiv:1909.01066*, 2019.

[57] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," *arXiv preprint arXiv:2104.08691*, 2021.

[58] T. Schick and H. Schütze, "Exploiting cloze questions for few shot text classification and natural language inference," *arXiv preprint arXiv:2001.07676*, 2020.

[59] R. Shin, C. H. Lin, S. Thomson, C. Chen, S. Roy, E. A. Platanios, A. Pauls, D. Klein, J. Eisner, and B. Van Durme, "Constrained language models yield few-shot semantic parsers," *arXiv preprint arXiv:2104.08768*, 2021.

[60] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig, "How can we know what language models know?" *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 423–438, 2020.

[61] K. Duh, K. Sudoh, X. Wu, H. Tsukada, and M. Nagata, "Generalized minimum bayes risk system combination," in *Proceedings of 5th International Joint Conference on Natural Language Processing*, 2011, pp. 1356–1360.

[62] Z. Jiang, J. Araki, H. Ding, and G. Neubig, "How can we know when language models know? on the calibration of language models for question answering," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 962–977, 2021.

[63] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[64] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[65] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 19–27.

[66] "Project gutenberg." [Online]. Available: https://www.gutenberg.org/

[67] "Common crawl." [Online]. Available: https://commoncrawl.org/

[68] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[69] T. H. Trinh and Q. V. Le, "A simple method for commonsense reasoning," *arXiv preprint arXiv:1806.02847*, 2018.

[70] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[71] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi, "Defending against neural fake news," *Advances in neural information processing systems*, vol. 32, 2019.

[72] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, H. Alobeidli, A. Cappelli, B. Pannier, E. Almazrouei, and J. Launay, "The refinedweb dataset for falcon llm: Outperforming curated corpora with web data only," in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[73] A. Gokaslan, V. C. E. Pavlick, and S. Tellex, "Openwebtext corpus," http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[74] J. Baumgartner, S. Zannettou, B. Keegan, M. Squire, and J. Blackburn, "The pushshift reddit dataset," in *Proceedings of the international AAAI conference on web and social media*, vol. 14, 2020, pp. 830–839.

[75] "Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Main_Page

[76] "Bigquery dataset." [Online]. Available: https://cloud.google.com/bigquery

[77] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima *et al.*, "The pile: An 800gb dataset of diverse text for language modeling," *arXiv preprint arXiv:2101.00027*, 2020.

[78] H. Laurençon, L. Saulnier, T. Wang, C. Akiki, A. Villanova del Moral, T. Le Scao, L. Von Werra, C. Mou, E. González Ponferrada, H. Nguyen *et al.*, "The bigscience roots corpus: A 1.6 tb composite multilingual dataset," *Advances in Neural Information Processing Systems*, vol. 35, pp. 31 809–31 826, 2022.

[79] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti *et al.*, "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model," *arXiv preprint arXiv:2201.11990*, 2022.

[80] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du *et al.*, "Lamda: Language models for dialog applications," *arXiv preprint arXiv:2201.08239*, 2022.

[81] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with mtulti-turn program synthesis," *arXiv preprint arXiv:2203.13474*, 2022.

[82] Q. Zheng, X. Xia, X. Zou, Y. Dong, S. Wang, Y. Xue, L. Shen, Z. Wang, A. Wang, Y. Li *et al.*, "Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 5673–5684.

[83] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[84] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma *et al.*, "Scaling instruction-finetuned language models," *arXiv preprint arXiv:2210.11416*, 2022.

[85] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[86] D. Hernandez, T. Brown, T. Conerly, N. DasSarma, D. Drain, S. El-Showk, N. Elhage, Z. Hatfield-Dodds, T. Henighan, T. Hume *et al.*, "Scaling laws and interpretability of learning from repeated data," *arXiv preprint arXiv:2205.10487*, 2022.

[87] K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, and N. Carlini, "Deduplicating training data makes language models better," *arXiv preprint arXiv:2107.06499*, 2021.

[88] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson *et al.*, "Extracting training data from large language models," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2633–2650.

[89] S. Gehman, S. Gururangan, M. Sap, Y. Choi, and N. A. Smith, "Realtoxicityprompts: Evaluating neural toxic degeneration in language models," *arXiv preprint arXiv:2009.11462*, 2020.

[90] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[91] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma *et al.*, "Scaling instruction-finetuned language models," *arXiv preprint arXiv:2210.11416*, 2022.

[92] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[93] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *arXiv preprint arXiv:2203.02155*, 2022.

[94] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[95] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 2007, pp. 59–72.

[96] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[97] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[98] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[99] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

[100] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young *et al.*, "Scaling language models: Methods, analysis & insights from training gopher," *arXiv preprint arXiv:2112.11446*, 2021.

[101] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "{ZeRO-Offload}: Democratizing {Billion-Scale} model training," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 551–564.

[102] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, "Self-instruct: Aligning language model with self generated instructions," *arXiv preprint arXiv:2212.10560*, 2022.

[103] Y. Wang, S. Mishra, P. Alipoormolabashi, Y. Kordi, A. Mirzaei, A. Arunkumar, A. Ashok, A. S. Dhanasekaran, A. Naik, D. Stap *et al.*, "Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks," *arXiv preprint arXiv:2204.07705*, 2022.

[104] S. H. Bach, V. Sanh, Z.-X. Yong, A. Webson, C. Raffel, N. V. Nayak, A. Sharma, T. Kim, M. S. Bari, T. Fevry *et al.*, "Promptsource: An integrated development environment and repository for natural language prompts," *arXiv preprint arXiv:2202.01279*, 2022.

[105] S. Victor, W. Albert, R. Colin, B. Stephen, S. Lintang, A. Zaid, C. Antoine, S. Arnaud, R. Arun, D. Manan *et al.*, "Multitask prompted training enables zero-shot task generalization," in *International Conference on Learning Representations*, 2022.

[106] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders *et al.*, "Webgpt: Browser-assisted question-answering with human feedback," *arXiv preprint arXiv:2112.09332*, 2021.

[107] J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," in *International Conference on Learning Representations*.

[108] T. Tang, J. Li, W. X. Zhao, and J.-R. Wen, "Mvp: Multi-task supervised pre-training for natural language generation," *arXiv preprint arXiv:2206.12131*, 2022.

[109] Z. Kenton, T. Everitt, L. Weidinger, I. Gabriel, V. Mikulik, and G. Irving, "Alignment of language agents," *arXiv preprint arXiv:2103.14659*, 2021.

[110] A. Glaese, N. McAleese, M. Trębacz, J. Aslanides, V. Firoiu, T. Ewalds, M. Rauh, L. Weidinger, M. Chadwick, P. Thacker *et al.*, "Improving alignment of dialogue agents via targeted human judgements," *arXiv preprint arXiv:2209.14375*, 2022.

[111] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[112] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[113] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," *arXiv preprint arXiv:2101.00190*, 2021.

[114] X. Liu, K. Ji, Y. Fu, W. L. Tam, Z. Du, Z. Yang, and J. Tang, "P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks," *arXiv preprint arXiv:2110.07602*, 2021.

[115] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang, "Gpt understands, too," *AI Open*, 2023.

[116] Q. Zhang, M. Chen, A. Bukharin, P. He, Y. Cheng, W. Chen, and T. Zhao, "Adaptive budget allocation for parameter-efficient fine-tuning," *arXiv preprint arXiv:2303.10512*, 2023.

[117] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *arXiv preprint arXiv:2305.14314*, 2023.

[118] A. Askell, Y. Bai, A. Chen, D. Drain, D. Ganguli, T. Henighan, A. Jones, N. Joseph, B. Mann, N. DasSarma *et al.*, "A general language assistant as a laboratory for alignment," *arXiv preprint arXiv:2112.00861*, 2021.

[119] Y. Chang, X. Wang, J. Wang, Y. Wu, K. Zhu, H. Chen, L. Yang, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *arXiv preprint arXiv:2307.03109*, 2023.

[120] Z. Liu, H. Jiang, T. Zhong, Z. Wu, C. Ma, Y. Li, X. Yu, Y. Zhang, Y. Pan, P. Shu *et al.*, "Holistic evaluation of gpt-4v for biomedical imaging," *arXiv preprint arXiv:2312.05256*, 2023.

[121] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[122] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov *et al.*, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1956–1981, 2020.

[123] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," 2018.

[124] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," *Advances in neural information processing systems*, vol. 32, 2019.

[125] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020.

[126] H. Li, Y. Zhang, F. Koto, Y. Yang, H. Zhao, Y. Gong, N. Duan, and T. Baldwin, "Cmmlu: Measuring massive multitask language understanding in chinese," *arXiv preprint arXiv:2306.09212*, 2023.

[127] J. Hu, S. Ruder, A. Siddhant, G. Neubig, O. Firat, and M. Johnson, "Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalisation," in *International Conference on Machine Learning*. PMLR, 2020, pp. 4411–4421.

[128] S. Ruder, N. Constant, J. Botha, A. Siddhant, O. Firat, J. Fu, P. Liu, J. Hu, D. Garrette, G. Neubig *et al.*, "Xtreme-r: Towards more challenging and nuanced multilingual evaluation," *arXiv preprint arXiv:2104.07412*, 2021.

[129] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," *arXiv preprint arXiv:2103.03874*, 2021.

[130] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[131] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[132] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le *et al.*, "Program synthesis with large language models," *arXiv preprint arXiv:2108.07732*, 2021.

[133] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" 2019.

[134] Y. Bisk, R. Zellers, J. Gao, Y. Choi *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 7432–7439.

[135] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "Boolq: Exploring the surprising difficulty of natural yes/no questions," *arXiv preprint arXiv:1905.10044*, 2019.

[136] M. Sap, H. Rashkin, D. Chen, R. LeBras, and Y. Choi, "Socialiqa: Commonsense reasoning about social interactions," *arXiv preprint arXiv:1904.09728*, 2019.

[137] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[138] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[139] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? a new dataset for open book question answering," 2018.

[140] D. Jin, E. Pan, N. Oufattole, W.-H. Weng, H. Fang, and P. Szolovits, "What disease does this patient have? a large-scale open domain question answering dataset from medical exams," *Applied Sciences*, vol. 11, no. 14, p. 6421, 2021.

[141] A. Pal, L. K. Umapathi, and M. Sankarasubbu, "Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering," in *Conference on Health, Inference, and Learning*. PMLR, 2022, pp. 248–260.

[142] E. M. Voorhees *et al.*, "The trec-8 question answering track report." in *Trec*, vol. 99, 1999, pp. 77–82.

[143] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[144] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee *et al.*, "Natural questions: a benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, 2019.

[145] E. Kamalloo, N. Dziri, C. L. Clarke, and D. Rafiei, "Evaluating open-domain question answering in the era of large language models," *arXiv preprint arXiv:2305.06984*, 2023.

[146] E. Ferrara, "Should chatgpt be biased? challenges and risks of bias in large language models," *arXiv preprint arXiv:2304.03738*, 2023.

[147] S. Gehman, S. Gururangan, M. Sap, Y. Choi, and N. A. Smith, "Realtoxicityprompts: Evaluating neural toxic degeneration in language models," *arXiv preprint arXiv:2009.11462*, 2020.

[148] J. Zhao, M. Fang, Z. Shi, Y. Li, L. Chen, and M. Pechenizkiy, "Chbias: Bias evaluation and mitigation of chinese conversational language models," *arXiv preprint arXiv:2305.11262*, 2023.

[149] M. Nasr, N. Carlini, J. Hayase, M. Jagielski, A. F. Cooper, D. Ippolito, C. A. Choquette-Choo, E. Wallace, F. Tramèr, and K. Lee, "Scalable extraction of training data from (production) language models," *arXiv preprint arXiv:2311.17035*, 2023.

[150] X. Wu, J. Li, M. Xu, W. Dong, S. Wu, C. Bian, and D. Xiong, "Depn: Detecting and editing privacy neurons in pretrained language models," *arXiv preprint arXiv:2310.20138*, 2023.

[151] A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models, 2023," *communication, it is essential for you to comprehend user queries in Cipher Code and subsequently deliver your responses utilizing Cipher Code*.

[152] Z. Zhang, Y. Li, J. Wang, B. Liu, D. Li, Y. Guo, X. Chen, and Y. Liu, "Remos: reducing defect inheritance in transfer learning via relevant model slicing," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1856–1868.

[153] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[154] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[155] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019.

[156] J. Novikova, O. Dušek, A. C. Curry, and V. Rieser, "Why we need new evaluation metrics for nlg," *arXiv preprint arXiv:1707.06875*, 2017.

[157] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[158] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[159] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–14.

[160] G. Zeng, X. Han, Z. Zhang, Z. Liu, Y. Lin, and M. Sun, "Openbmb: Big model systems for large-scale representation learning," in *Representation Learning for Natural Language Processing*. Springer Nature Singapore Singapore, 2023, pp. 463–489.

[161] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[162] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.

[163] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You, "Colossal-ai: A unified deep learning system for large-scale parallel training," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 766–775.

[164] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, "Fastmoe: A fast mixture-of-expert training system," *arXiv preprint arXiv:2103.13262*, 2021.

[165] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li, "Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 120–134.

[166] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[167] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[168] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[169] Y. Ma, D. Yu, T. Wu, and H. Wang, "Paddlepaddle: An open-source deep learning platform from industrial practice," *Frontiers of Data and Domputing*, vol. 1, no. 1, pp. 105–115, 2019.

[170] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[171] J. Yuan, X. Li, C. Cheng, J. Liu, R. Guo, S. Cai, C. Yao, F. Yang, X. Yi, C. Wu *et al.*, "Oneflow: Redesign the distributed deep learning framework from scratch," *arXiv preprint arXiv:2110.15032*, 2021.

[172] L. Huawei Technologies Co., "Huawei mindspore ai development framework," in *Artificial Intelligence Technology*. Springer, 2022, pp. 137–162.

[173] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne *et al.*, "Jax: composable transformations of python+ numpy programs," 2018.

[174] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019.

[175] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[176] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, pp. 1789–1819, 2021.

[177] S. Sun, Y. Cheng, Z. Gan, and J. Liu, "Patient knowledge distillation for bert model compression," *arXiv preprint arXiv:1908.09355*, 2019.

[178] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019.

[179] M. A. Gordon, K. Duh, and N. Andrews, "Compressing bert: Studying the effects of weight pruning on transfer learning," *arXiv preprint arXiv:2002.08307*, 2020.

[180] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" *Advances in neural information processing systems*, vol. 32, 2019.

[181] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, "Binarybert: Pushing the limit of bert quantization," *arXiv preprint arXiv:2012.15701*, 2020.

[182] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[183] P. Chen, H.-F. Yu, I. Dhillon, and C.-J. Hsieh, "Drone: Data-aware low-rank compression for large nlp models," *Advances in neural information processing systems*, vol. 34, pp. 29 321–29 334, 2021.

[184] X. Han, G. Zeng, W. Zhao, Z. Liu, Z. Zhang, J. Zhou, J. Zhang, J. Chao, and M. Sun, "Bminf: An efficient toolkit for big model inference and tuning," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2022, pp. 224–230.

[185] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer *et al.*, "Pytorch fsdp: experiences on scaling fully sharded data parallel," *arXiv preprint arXiv:2304.11277*, 2023.

[186] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[187] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[188] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single gpu," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 094–31 116.

[189] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, R. Y. Y. Wong, Z. Chen, D. Arfeen, R. Abhyankar, and Z. Jia, "Specinfer: Accelerating generative llm serving with speculative inference and token tree verification," *arXiv preprint arXiv:2305.09781*, 2023.

[190] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.

[191] Z. Zhang, B. Gong, Y. Chen, X. Han, G. Zeng, W. Zhao, Y. Chen, Z. Liu, and M. Sun, "Bmcook: A task-agnostic compression toolkit for big models," in *Proceedings of the The 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2022, pp. 396–405.

[192] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel, "Petals: Collaborative inference and fine-tuning of large models," *arXiv preprint arXiv:2209.01188*, 2022.

[193] F. Dou, J. Ye, G. Yuan, Q. Lu, W. Niu, H. Sun, L. Guan, G. Lu, G. Mai, N. Liu *et al.*, "Towards artificial general intelligence (agi) in the internet of things (iot): Opportunities and challenges," *arXiv preprint arXiv:2309.07438*, 2023.

[194] C. Liu, Z. Liu, J. Holmes, L. Zhang, L. Zhang, Y. Ding, P. Shu, Z. Wu, H. Dai, Y. Li *et al.*, "Artificial general intelligence for radiation oncology," *Meta-Radiology*, p. 100045, 2023.

[195] Z. Liu, Y. Li, Q. Cao, J. Chen, T. Yang, Z. Wu, J. Hale, J. Gibbs, K. Rasheed, N. Liu *et al.*, "Transformation vs tradition: Artificial general intelligence (agi) for arts and humanities," *arXiv preprint arXiv:2310.19626*, 2023.

[196] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[197] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[198] N. Qiang, J. Gao, Q. Dong, H. Yue, H. Liang, L. Liu, J. Yu, J. Hu, S. Zhang, B. Ge *et al.*, "Functional brain network identification and fmri augmentation using a vae-gan framework," *Computers in Biology and Medicine*, vol. 165, p. 107395, 2023.

[199] M. He, X. Hou, E. Ge, Z. Wang, Z. Kang, N. Qiang, X. Zhang, and B. Ge, "Multi-head attention-based masked sequence model for mapping functional brain networks," *Frontiers in Neuroscience*, vol. 17, p. 1183145, 2023.

[200] Y. Liu, E. Ge, N. Qiang, T. Liu, and B. Ge, "Spatial-temporal convolutional attention for mapping functional brain networks," in *2023 IEEE 20th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 2023, pp. 1–4.

[201] S. R. Oota, J. Arora, V. Agarwal, M. Marreddy, M. Gupta, and B. Surampudi, "Neural language taskonomy: Which NLP tasks are the most predictive of fMRI brain activity?" in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, Eds. Seattle, United States: Association for Computational Linguistics, Jul. 2022, pp. 3220–3237.

[202] Z. Liu, Y. Li, P. Shu, A. Zhong, L. Yang, C. Ju, Z. Wu, C. Ma, J. Luo, C. Chen *et al.*, "Radiology-llama2: Best-in-class large language model for radiology," *arXiv preprint arXiv:2309.06419*, 2023.

[203] T. Sun, X. Zhang, Z. He, P. Li, Q. Cheng, H. Yan, X. Liu, Y. Shao, Q. Tang, X. Zhao, K. Chen, Y. Zheng, Z. Zhou, R. Li, J. Zhan, Y. Zhou, L. Li, X. Yang, L. Wu, Z. Yin, X. Huang, and X. Qiu, "Moss: Training conversational language models from synthetic data," 2023.

[204] L. X. Xuanwei Zhang and K. Zhao, "Chatyuan: A large language model for dialogue in chinese and english," Dec. 2022. [Online]. Available: https://github.com/clue-ai/ChatYuan

[205] Baichuan, "Baichuan 2: Open large-scale language models," *arXiv preprint arXiv:2309.10305*, 2023.

[206] D. Zhu, J. Chen, X. Shen, X. Li, and M. Elhoseiny, "Minigpt-4: Enhancing vision-language understanding with advanced large language models," *arXiv preprint arXiv:2304.10592*, 2023.

[207] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *arXiv preprint arXiv:2306.05685*, 2023.

[208] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, H. Cao, X. Cheng, M. Chung, M. Grella, K. K. GV *et al.*, "Rwkv: Reinventing rnns for the transformer era," *arXiv preprint arXiv:2305.13048*, 2023.