applied sciences

MDPI

Review

A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs)

Rajvardhan Patil [1,*] and Venkat Gudivada [2]

[1] School of Computing, Grand Valley State University, Allendale Charter Township, MI 49401, USA
[2] Computer Science Department, East Carolina University, Greenville, NC 27858, USA; gudivadav15@ecu.edu
* Correspondence: patilr@gvsu.edu; Tel.: +1-616-331-4375

Abstract: Natural language processing (NLP) has significantly transformed in the last decade, especially in the field of language modeling. Large language models (LLMs) have achieved SOTA performances on natural language understanding (NLU) and natural language generation (NLG) tasks by learning language representation in self-supervised ways. This paper provides a comprehensive survey to capture the progression of advances in language models. In this paper, we examine the different aspects of language models, which started with a few million parameters but have reached the size of a trillion in a very short time. We also look at how these LLMs transitioned from task-specific to task-independent to task-and-language-independent architectures. This paper extensively discusses different pretraining objectives, benchmarks, and transfer learning methods used in LLMs. It also examines different *finetuning* and *in-context learning* techniques used in downstream tasks. Moreover, it explores how LLMs can perform well across many domains and datasets if sufficiently trained on a large and diverse dataset. Next, it discusses how, over time, the availability of cheap computational power and large datasets have improved LLM's capabilities and raised new challenges. As part of our study, we also inspect LLMs from the perspective of scalability to see how their performance is affected by the model's depth, width, and data size. Lastly, we provide an empirical comparison of existing trends and techniques and a comprehensive analysis of where the field of LLM currently stands.

Keywords: language models; PLMs; large language model; LLMs; natural language processing; NLP; literature review; survey; review

Citation: Patil, R.; Gudivada, V. A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs). *Appl. Sci.* **2024**, *14*, 2074. https://doi.org/10.3390/app14052074

Academic Editors: Affan Yasin, Javed Ali Khan and Lijie Wen

Received: 2 February 2024
Revised: 21 February 2024
Accepted: 28 February 2024
Published: 1 March 2024

1. Introduction

1.1. Background

Most feature-engineering methods before generative pretrained transformer (GPT) relied on manually curated labeled data and were time-consuming and expensive. Additionally, not all applications had annotated or labeled datasets. To address these issues, statistical methods such as one-hot encoding [1], bag of words, N-grams [2], term frequency [3], and inverse document frequency [4,5] were proposed. In these approaches, word- or phrase-level statistics were computed and used as features in supervised models. However, such discrete space representations lacked contextual information and resulted in dimensionality curse, making them computationally inefficient. Although techniques such as dimensionality reduction technique [6] and independent component analysis [7] were applied, these techniques failed to capture a deeper understanding of concepts such as polysemy or identifying analogies, synonyms, antonyms, etc.

An alternative to using unlabeled data in a self-supervised manner to extract and leverage linguistic information emerged as a more effective and valuable approach. For making predictions, language models started incorporating contexts of increasingly larger scope. The self-supervised approach started with individual words, followed by surrounding words, sentences, and paragraphs [8]. Word embeddings like Word2Vec [9,10], Glove [11], and

FastText [12] were generated from the unlabeled corpora using the self-supervised approach. They improved performance across a variety of NLP tasks.

1.2. Static Embeddings

Due to the accurate representation of words as real-valued numeric vectors, continuous vector space representation soon became a viable alternative to discrete space- and density-based [13,14] representations. In a continuous vector space, shallow feed-forward networks were used to generate the word embeddings or word vectors. As neural networks are differentiable, gradient computation with respect to model parameters became possible. They were further optimized using techniques such as stochastic gradient descent. This approach used objectives such as continuous bag of words (CBOW) [10] and skip-gram [10] during training. Unlike statistical approaches, in this approach, the network automatically discovers the embeddings. The task of explicit feature engineering was therefore alleviated as the features were automatically deduced in neural network models.

1.3. Dynamic Embeddings

The word vector embeddings derived from shallow networks, and, although they captured the semantics of the words, they were static and context-insensitive. Their meaning did not change as per the change in context. Subsequently, deep neural network (DNN) models were implemented to derive dynamic embeddings. The dynamic embeddings, such as C2V [15], CoVe [16], ELMo [17], ULMFiT [18], UNILM [19], etc., being context-sensitive, were able to address the polysemy aspect of words. However, capturing long-term dependencies between words was still a challenge.

1.4. Task-Dependent Architectures

Recurrent neural networks (RNNs) or their variants were used to capture the long-term dependencies between words. In an RNN-based network, the encoder generated one single vector of fixed dimension for the entire input sequence. For example, in [20], the decoder received one single encoded hidden state from the encoder, representing the numerical summary of the input sequence. The information of the entire sequence is compressed into a single vector, making it difficult for the decoder to decode information, especially for longer sequences. Additionally, although RNNs could capture long-term dependencies, they had vanishing and exploding gradient issues. RNN variants, such as long short-term memory (LSTM) and gated recurrent unit (GRU), could overcome the vanishing and exploding gradient problem encountered in RNNs for sequence modeling. For instance, Ref. [21] used LSTM and achieved state-of-the-art (SOTA) performance on translation tasks. However, for neural machine translation (NMT) tasks requiring a sequence-to-sequence (seq2seq) model, the performance of LSTMs and GRUs decreased as the input sequence size increased. Self-attention-based transformer models addressed these issues of long-range dependencies encountered in RNN and its variant models.

1.5. Task-Agnostic Architecture

In the last decade, neural networks have been extensively used in language modeling tasks, starting from shallow feed-forward networks, RNNs, LSTMs, and deep neural networks to self-attention-based transformer networks. The shallow feed-forward networks deduced single-layer representation called 'word vectors', which were learned and then transferred to task-specific architectures. Then, RNNs with deep or multiple layers (DNNs) were used to generate context-sensitive and more robust (deep) representations that were transferred or applied to task-specific architectures. To overcome task-specific architectures, transformer-based pretrained language models (PLMs) came into play. Recent work using transformers has focused on a task-independent approach, where transfer and finetuning of the self-attention block are sufficient. These transformer-based language models are flexible and task-agnostic. They can be finetuned on different downstream tasks without

requiring architecture modifications. They have also led to significant improvement boosts, especially in capturing long-range dependencies.

As shown in Figure 1, the phases in these transformer-based LLMs can broadly be classified into pretraining, transfer learning, and/or in-context learning. In the sections to come, we explore in detail different attention mechanism masks, architectures, objectives used during pretraining, transfer and in-context learning techniques, scalability factors, and challenges regarding LLMs.

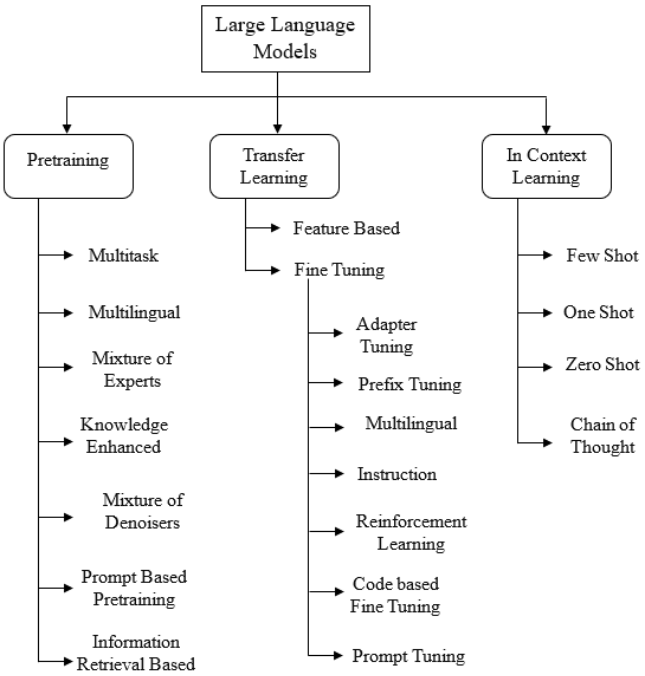

Figure 1. Large language model phases.

The outline of this survey paper is as follows. In Section 2, we look at the language model definition and the attention layer mechanism in detail. In Section 3, we describe the types of architectures and attention masks used in transformers. Section 4 elaborates on the pretraining objectives and different learning strategies used by the LLMs. Section 5 discusses transfer learning strategies, followed by in-context learning in Section 6. Section 7 describes different scale factors, such as model width, depth, datasets, architecture, and how they affect the performance of LLMs. Section 8 enumerates the challenges encountered by LLMs, followed by future directions and development trends in Section 9. Section 10 concludes the paper.

2. Language Models and Attention Mechanism

2.1. Language Models

Language models primarily have two main steps: pretraining and transfer learning. In the pretraining phase, some objective function is used to learn the network's initial parameters (language representation). Pretraining is then followed by the transfer learning phase, where the initial learned parameters are adapted or finetuned on a target downstream task. The pretraining is conducted in a self-supervised manner on the unlabeled corpus. Transfer learning, on the other hand, follows a supervised approach, where each of the downstream tasks, although having separate finetuned models, are initialized with the same pretrained parameters. This approach helps in learning a universal representation of language, which requires little adaptation when transferred to downstream tasks. Therefore, the target tasks do not need to be from the same domain as the unlabeled corpus. Unlike task-specific techniques, no architectural modifications are required for PLMs when applied to downstream tasks. However, the PLMs depend on a large corpus of unlabeled data to be effective across various tasks.

As stated in Bloom [22], language modeling refers to the task of modeling the probability of a sequence of tokens in a text, where a token can be a unit of text, such as a word, subword, character, byte, etc. Normally, in the pretraining phase of language models, a next-word prediction objective is used, which is conditioned on the previous tokens as context. So, for a given input or source sequence $S = (s_1, s_2, \ldots, s_n)$, the model predicts the joint probability of the output or target sequence $T = (t_1, t_2, \ldots, t_n)$, shown in Equation (1).

$$P(t) = \prod_{i=1}^{n} p(t_i | s_1, s_2, \ldots, s_{i-1}) \tag{1}$$

This approach, where the probability of the next token is iteratively predicted, is referred to as autoregressive language modeling and is represented using Equation (2).

$$\begin{aligned} p(x) &= p(x_1, x_2, \ldots, x_T) \\ &= \prod_{t=1}^{T} p(x_t | x_1, x_2, \ldots, x_{t-1}) \end{aligned} \tag{2}$$

Here, to deal with different downstream tasks (question answering, translation, summarizing, etc.), each task is casted or converted into a text-to-text framework. In this way, the language model can be applied or used to handle different downstream tasks. The pretrained model with parameters θ is then adapted during finetuning of dataset D to minimize the loss over the target tokens conditioned on the source tokens and previously seen target tokens. Equation (3) highlights this loss function 'L'.

$$L(D; \theta) = -\sum_{i} \sum_{j} log\ p_\theta(t_{ij} \mid s_i, t_{i,<j}) \tag{3}$$

LLMs follow a similar mechanism of pretraining and finetuning to language models, except the parameter size of LLMs is in billions and/or trillions.

2.2. Attention Layer

To be able to align the input and output words correctly, the attention layer helps the decoder understand which inputs are more important. This enables the decoder to focus on the right place or context during the prediction of each output token. The inputs and targets are first converted to embeddings or initial representations of the words. The attention mechanism then uses these encoded representations. Query vectors (Q) represent the decoder hidden states, and the key (K) and value (V) vector pairs come from the encoder hidden states. To compute the similarity score between queries and keys, the dot products between the query and key vectors are computed. If the key (K) and query (Q) are similarly aligned, then their dot product will yield a higher score. Therefore, higher scores of a particular key indicate that it is relatively more important to query than others with lower scores. To obtain the probabilities of the match between keys and queries, the scores are run through softmax to fit a distribution between 0 and 1. These probabilities act as an indexing mechanism regarding the value (V) vector. So, these probabilities are further multiplied with the value vectors (V), which results in alignment vectors. Equation (4) represents this computation from the attention layer.

$$\begin{aligned} Z &= attention(Q, K, V) = W_A V \\ &= softmax\left(\frac{QK^T}{\sqrt{(d_k)}}\right) V \end{aligned} \tag{4}$$

To help speed up the pretraining process, the teacher forcing technique is used, which leads to faster convergence and higher accuracy. In teacher forcing, instead of the model's output from the previous timestep, the ground truth (correct answer) is fed as an input

at each timestep. To enable this teacher forcing, the preattention decoder takes the target tokens and shifts them one place to the right.

2.3. Multihead Attention

Instead of recurrent layers, transformers differ from sequence to sequence by using multihead attention layers; hence, they do not suffer from vanishing gradients problems that are related to the length of the sequences. Figure 2 highlights the multihead attention mechanism in transformers. In the multihead attention mechanism, a set of parallel self-attention layers are added, which are called heads.

$$\begin{aligned} Z &= MultiHead(Q, K, V) \\ &= Concat(z_1, z_2, \ldots, z_n)\ W^0 \\ z_i &= attention(Q(W_i)^Q, K(W_i)^K, V(W_i)^V) \end{aligned} \tag{5}$$

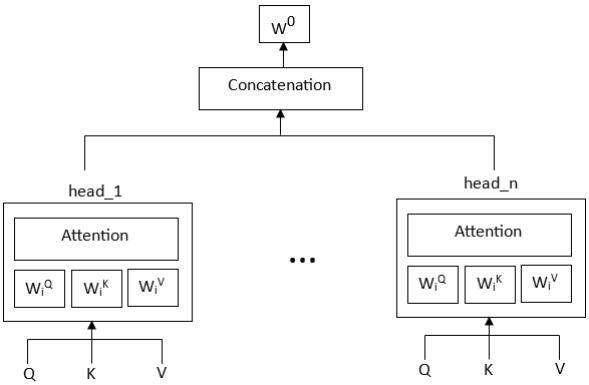

Figure 2. Multihead attention mechanism.

As shown in Figure 2, the output of these heads is further concatenated to produce a single output. This multihead attention mechanism emulates the recurrence sequence effect but with attention. Each head uses different linear transformations to represent words, and therefore different heads can learn different relationships between words. The multihead attention mechanism executes the attention of the scaled dot product in parallel. The multihead model is therefore able to jointly attend to information from different representations at different positions over the projected versions of queries, keys, and values. As shown in Equation (5), these output values are then concatenated and weighted, where each head z_i is the attention function of query, key, and value with trainable parameters $(W_i)^Q, (W_i)^K, (W_i)^V$.

2.4. Attention-Based RNN Models

As stated in [23], using an RNN encoder–decoder architecture for NMT tasks, the fixed-length encoded vector generated by the encoder became a performance bottleneck. The performance of such an RNN-based encoder–decoder model deteriorated rapidly as the length of an input sentence increased. It also assigned more importance to the later tokens in the input sequence than the ones appearing earlier in the sequence. Such an architecture, also called global attention, failed to capture local context and longer dependencies adequately. Additionally, as RNNs are sequential, they prohibited parallelization, resulting in longer training time. To overcome this issue, Ref. [23] proposed *attention mechanism* in the decoder, which automatically soft-searches relevant context from the input sentence required to predict the target word. The vectors of these context words and the previously

generated (target) words are used to predict the current target word. As a result, the attention mechanism helped align and translate the input and output jointly.

Unlike the traditional encoder–decoder RNN model, the self-attention mechanism does not encode the entire input sequence into a fixed single vector. The input sentence is therefore not squashed into a single fixed-length vector, where the decoder has flexibility to attend to more than one hidden state of the encoder. Additionally, in the attention mechanism, only a subset of encoded vectors of the input sequence are chosen adaptively during the decoding. The attention mechanism gives more weight or attention to the part of the input sequence that is relevant to the target. As a result, it allows capturing dependencies from the information spread throughout the sequence irrespective of the distance between the tokens. Furthermore, as the decoder is empowered with the attention mechanism, the encoder is relieved from the burden of encoding the input into a fixed-size vector. Paper [23] shows how this joint learning of alignment and translation improves performance over the basic encoder–decoder approach, especially over longer sentences.

2.5. Attention-Based Transformer Models

Although the attention mechanism from [23] improved significantly, it used bidirectional RNN as an encoder. RNN, being sequential, prohibits parallelization, leading to more computational time. Ref. [24] proposed transformer architecture that relied solely on the attention mechanism, altogether eliminating RNN and CNN components. Transformers handled long-term dependencies much better than RNNs, which resulted in robust transfer performance across several diverse tasks. Unlike RNNs, the transformer architecture reduced sequential computation and enabled parallelization, requiring less training time and achieving new state-of-the-art results. Unlike RNNs, it enables every position in the decoder to attend to all the positions in the input sequence. Being autoregressive, it considers the previously generated token as an additional input to generate the next target token. As shown in Figure 3, it uses stacked self-attention for both the encoder and decoder and has a masking mechanism in the decoder to preserve its autoregressive property.

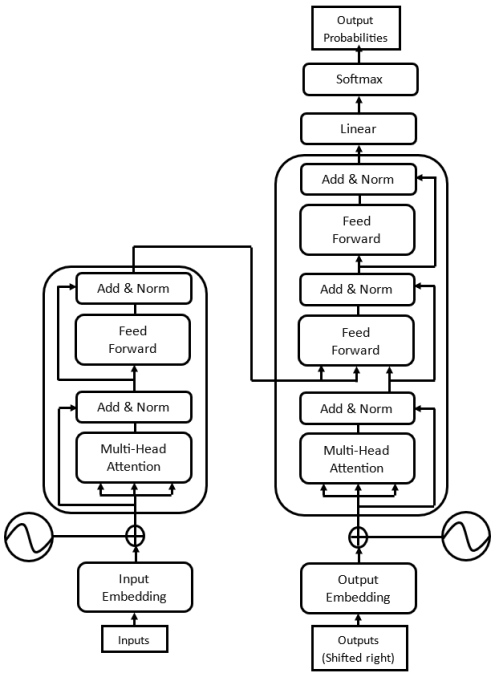

Figure 3. Transformer's encoder–decoder-based model.

3. Transformer

After its inception, transformer soon became the de facto standard for natural language tasks. Below, we discuss several variants of the original transformer-based model that were proposed to deal with NLU and NLGU tasks.

3.1. Encoder–Decoder-Based Model

An example of encoder–decoder architecture is the transformer model proposed in [24]. Its encoder and decoder blocks are stacked with multiple layers. As shown in Figure 3, the transformer encoder layer consists of a self-attention layer and a position-wise feed-forward layer. In addition to these two layers, decoder consists of a third cross-attention layer, which is responsible for attending to encoder output.

Encoder–decoder models adopt bidirectional attention for the encoder, unidirectional attention for the decoder, and a cross-attention mechanism between them. Cross-attention in the decoder has access only to the fully processed encoder output and is responsible for connecting input tokens to target tokens. The encoder–decoder-based models are pretrained for seq2seq tasks. They can also be pretrained on conditional generation tasks, where the output is generated in regard to the given input, for example in summarizing, question answering, and translation tasks. T5 [25] uses encoder–decoder architecture. As stated in T5, using encoder–decoder structure helped to achieve good performance regarding classification as well as for generative tasks.

Although encoder–decoder models end up having twice as many parameters as their decoder-only or encoder-only counterparts, they still have similar computational cost. Compared to PrefixLM models where the parameters are shared, here, the input and target are independently processed and use separate sets of parameters. Unlike decoder-only language models that are trained to generate the input, encoder–decoder models output target tokens.

The original transformer consisted of encoder–decoder blocks and was initially used for sequence-to-sequence tasks, such as NMT. However, it was discovered that, with the change in how the input is fed to the model, the single-stack (decoder or encoder) could also complete sequence–sequence model tasks. As a result, the subsequent models started containing either an encoder or decoder architecture. Below, we discuss these architectural variants of the original transformer model.

3.2. Encoder-Only-Based Model

Encoder-only models use bidirectional attention, where the target token can attend to the previous and next tokens. Encoder-only-based models, for instance, BERT [26], produce a single prediction for a given input sequence. As a result, they are more fit for classification and understanding tasks rather than NLG tasks, such as translation and summarizing.

3.3. Decoder-Only (Causal)-Based Model

In decoder-only models, the goal is to predict the next token in the sequence; therefore, such models are autoregressive. These models are trained solely for next-step prediction, so decoder-only models are well-suited for NLG tasks. In decoder-only models, the input and target tokens are concatenated before processing. As a result, the representations of inputs and targets are simultaneously built layer by layer as they propagate concurrently through the network. In the encoder–decoder model, the input and target tokens are processed separately and rely on cross-attention components to connect them. GPT [27] was one of the first models that relied solely on decoder-based architecture. However, as decoder-only models use a unidirectional attention mechanism, their performance might be hindered for tasks involving longer sequences, such as summarizing.

3.4. Prefix (Non-Causal) Language Model

Prefix language models are also decoder-only-based models but differ in the masking mechanism. Instead of a causal mask, a fully visible mask is used for the prefix part of the input sequence, and a causal mask is used for the target sequence.

For example, to translate an English sentence “I am doing well” to French, the model would apply a fully visible mask to the prefix “translate English to French: I am doing well. Target:”, followed by causal masking while predicting the target “je vais bien”. Also, unlike causal language models where the targets-only paradigm is used, the prefix language model uses the input-to-target paradigm. Both causal and prefix model architectures are

autoregressive as the objective is to predict the next token. However, the causal model uses a unidirectional attention mask, while the prefix model modifies the masking mechanism to employ bidirectional attention over prefix tokens. Figure 4 demonstrates the mechanism of the above architectures. The lines represent the attention visibility. Dark lines represent the fully visible masking (bidirectional attention), and light gray lines represent causal masking (unidirectional attention).

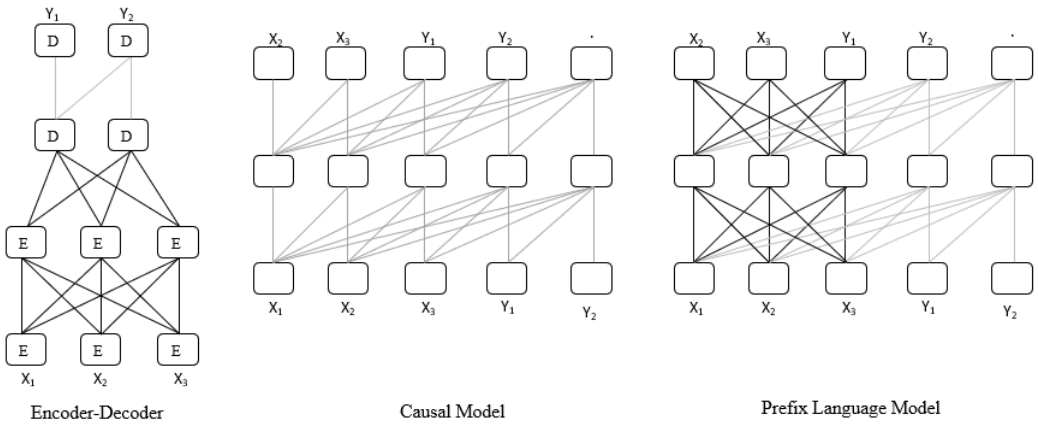

Figure 4. Different transformer architectures in LLMs.

As shown in Figure 4, in the encoder–decoder architecture, fully visible masking is used in the encoder and causal masking is used in the decoder. In a decoder-only model, the input and target are concatenated, and then a causal mask is used throughout. A decoder-only model with a prefix allows fully visible masking over part of the input token (prefix), followed by causal masking on the rest of the sequence. In general, autoencoding models learn bidirectional contextualized representation suited for NLU tasks, whereas autoregressive models learn to generate the next token and hence are suited for NLG tasks. Table 1 details architectural information of prominent LLM models, such as their parameter size, hardware used, number of encoder (E) and decoder (D) layers, attention heads, etc.

Table 1. Architecture details of LLMs.

Model	Param Size	Layers	d-Model	Attention Heads	Hardware
Transformer-base [24]	-	6 E, 6 D	512	8	8 NVIDIA P100 GPUs
Transformer-big [24]	-	12 E, 12 D	1024	16	8 NVIDIA P100 GPUs
BERT-base [26]	110 M	12 E	768	12	4 Cloud TPUs
BERT-large [26]	340 M	24 E	1024	16	16 Cloud TPUs (64 TPU chips)
GPT-1 [27]	117 M	12 D	768	12	-
GPT-2 [28]	117 M to 1.5 B	24 D to 48 D	1600	48	-
GPT-3 [29]	175 B	96	12,288	96	V 100 GPUs (285 K CPU cores, 10 K GPUs)
T5 [25]	220 M–11 B	(12 E, 12 D)	-	-	1024 TPU v3
REALM [30]	330 M	-	-	-	64 Google Cloud TPUs, 12 GB GPU
Jurassic-1 [31]	178 B	76	13,824	96	
mT5 [32]	13 B	-	-	-	-
Pangu-Alpha [33]	207 B	64	16,384	128	2048 Ascend 910 AI processors
CPM-2 [34]	198 B	24	4096	64	-
Yuan 1.0 [35]	245 B	-	-	-	-
HyperClova [36]	82B	64	10,240	80	128 DGX servers with 1024 A100 GPUs
GLaM [37]	1.2 T (96.6)	64 MoE	8192	128	1024 Cloud TPU-V4 chips (Single System)
ERNIE 3.0 [38]	10 B	48, 12	4096, 768	64, 12	384 NVDIA v100 GPU cards
Gopher [39]	280 B	80	16,384	128	4 DCN-connected TPU v3 Pods (each with 1024 TPU v3 chips)
Chinchilla [40]	70 B	80	8192	64	-
AlphaCode [41]	41.1 B	8 E, 56 D	6144	48, 16	-

Table 1. *Cont.*

Model	Param Size	Layers	d-Model	Attention Heads	Hardware
CodeGEN [42]	16.1 B	34	256	24	-
CodeGeeX [43]	13 B	39	5120	40	1536 Ascend 910 AI Processors
FLAN [44]	137 B	-	-	-	TPUv3 with 128 cores
InstructGPT [45]	175 B	96	12,288	96	V 100 GPUs
LaMDA [46]	137 B	64	8192	128	1024 TPU-v3 chips
T0 [47]	11 B	12	-	-	-
GPT NeoX 20B [48]	20 B	44	6144	64	12 AS-4124GO-NART servers (each with 8 NVIDIA A100-SXM4-40GB GPUs)
OPT [49]	175B	96	12,288	96	992 80GB A100 GPUs
MINERVA [50]	540.35 B	118	18,432	48	-
AlexaTM 20B [51]	20 B (19.75 B)	46 E, 32 D	4096	32	128 A100 GPUs
GLM-130 B [52]	130 B	70	12,288	96	96 NVIDIA DGX-A100 (8×40 G)
XGLM [53]	7.5 B	32	4096		
PaLM [54]	540.35 B	118	18,432	48	6144 TPU v4 chips (2 Pods)
Galactica [55]	120 B	96	10,240	80	128 NVIDIA A100 80 GB nodes
Pali [56]	16.9 (17) B	-	-	-	-
LLaMA [57]	65B	80	8192	64	2048 A100 GPU (80 GB RAM)
UL2 [58]	20 B	32 E, 32 D	4096	16	64 to 128 TPUv4 chips
Pythia [59]	12 B	36	5120	40	-
WeLM [60]	10 B	32	5120	40	128 A100-SXM4-40 GB GPUs
BLOOM [22]	176 B	70	14,336	112	48 nodes having 8 NVIDIA A100 80GB GPUs (384 GPUs)
GLM [61]	515 M	30	1152	18	64 V100 GPUs
GPT-J	6 B	28	4096	16	TPU v3-256 pod
YaLM	100 B				800 A100
Alpaca	7 B				8 80 GB A100s
Falcon	40 B	-	-	-	-
(Xmer) XXXL [62]	30 B	28	1280	256	64 TPU-v3 chips
[63]	1.1 T	32	4096	512 (experts)	
XLM-R [64]	550 M	24	1024	16	

3.5. Mask Types

Self-attention is the variant of the attention mechanism proposed in [23]. It generates an output sequence, which has the same length as the input sequence, where it replaces each element with the rest of the sequence's weighted average. Below, we look at different masking techniques that are used to zero out certain weights. By zeroing out the weights, the mask decides which entries can be attended to by the attention mechanism at a given output timestep. As highlighted in Figure 5, by using fully visible mask, the attention mechanism can attend to the entire input sequence when producing each entry of its output.

In causal mask, the attention mechanism can attend only to the previous tokens and is prohibited from attending to the input tokens from the future. That is, while producing the *i*th entry, causal masks prevent the attention mechanism from attending to all the entries occurring after the *i*th entry so that the model cannot see into the future. Prefix–causal mask is a combination of these two approaches, allowing the attention mechanism to use a fully visible mask on a portion of the input sequence (called the prefix) and a causal mask on the rest of the sequence.

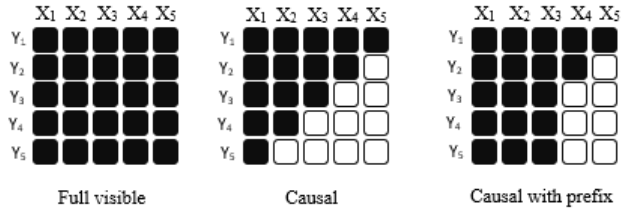

Figure 5. Different mask types in attention mechanism.

4. Pretraining—Strategies and Objectives

The pretraining process makes the model learn and capture language representation and (general or domain) knowledge, which are then used in the downstream NLU and NLG tasks. Such pretraining of the language model using neural networks has proven to be more effective in improving the performance on various NLP tasks. The pretraining process is usually unsupervised and based on some objective function that leverages the unlabeled data to provide the model with language understanding and generation capabilities. Most of the objectives can be formulated as input-to-target tasks, where the model is conditioned on the context represented by the input and is expected to generate the target as the output. The model is trained with the maximum likelihood to predict the target token. Once pretrained, the model is further finetuned on downstream tasks in a supervised manner. This pretrained approach leads to faster and better generalization than training the model from scratch. Below, we explore several objectives that have been successfully used during the pretraining process.

4.1. Objectives

4.1.1. Left-to-Right (LTR) Language Model Objective

In the LTR objective, the token can attend only to previous tokens, so this objective is unsuitable for applications requiring information from both directions, such as question answering and text summarizing. The LTR objective-based models do well on NLG tasks, but, because of the unidirectional attention mechanism, the model cannot fully capture the dependencies between the context words, which is required for good performance on NLU tasks.

4.1.2. Prefix Language Model Objective

In this prefix language model objective, the given text is split into input (prefix) and target sequence. The input is fed to the encoder with a fully visible mask, and the target is to be predicted by the decoder with a causal mask. In this objective, as the fully visible mask is used on the prefix portion of the input sequence, they do better on both NLG and NLU tasks. As causal masking is used in the decoder, to predict the target token 'i', the model depends upon the tokens appearing before 'i'. As the prefix is consumed bidirectionally, prefix-based language models offer more modeling power than the unidirectional encoding of inputs used in vanilla left-to-right-based language models.

4.1.3. Masked Language Model Objective

A masked language model (MLM) is a type of denoising objective that aims to predict the missing or corrupted tokens from the input. In MLM, 15% of the tokens are randomly masked/corrupted from the input, and the goal is to predict these masked words given the left and right context. It is observed in [25] that, as the corruption rate increased to 50%, the performance of the LLM on the benchmark tasks decreased. Out of the masked tokens, most of them are replaced with a masked token, while the others are replaced with random tokens. Regarding this objective, the encoder produces contextualized representations suited for understanding tasks but then, for the same reason, does not perform well for generation tasks.

In [25], three corruption strategies, 'Mass-style', 'Replace Corrupted Span', and 'Drop Corrupted Tokens', were investigated. 'Mass-style' works similar to MLM except it focuses on replacing 15% of tokens with mask tokens and excludes the random token swapping step. In 'Replace Corrupted Span', a unique or single mask token is used to replace the

consecutive span of corrupted tokens. Lastly, in 'Drop Corrupted Tokens', the task is to reconstruct the corrupted tokens that are dropped from the input sequence. REALM [30] uses the salient span technique to focus on problems that require world knowledge.

4.1.4. General Language Mode Objective

Ref. [61] proposed the general language model objective based on autoregressive blank infilling. This objective performs well for both NLU and NLG (conditional and unconditional) tasks. Following the idea of autoencoding, GLM randomly blanks out spans of continuous tokens from the input text and then, similar to autoregressive pretraining, trains the model to reconstruct the spans sequentially, demonstrating how, by varying the number and lengths of missing spans, the autotregressive blank filling objective can prepare the language model for both conditional and unconditional generation.

To support language understanding and generation, Ref. [52] uses two mask tokens. [MASK] was used to mask short blanks having lengths up to a certain portion or threshold of the input. When [MASK] is used, GLM-130B behaves similarly to BERT and T5. Meanwhile, [gMASK] was used for long blanks at the end of sentences having random lengths with prefix contexts provided. When [gMASK] is used, GLM-130B behaves similarly to PrefixLM.

4.1.5. Span Corruption Objective

When multiple consecutive tokens are masked or corrupted, they are referred to as a 'span'. A unique and single mask token is used to replace the entire span. In span corruption, the model refers to the uncorrupted tokens from the past and future to predict the corrupted target tokens (span). This objective was evaluated in T5 [25], where it was parameterized by number of spans to be corrupted and the percentage of tokens to be corrupted.

4.1.6. Deshuffle Objective

In this objective, a sequence of tokens are shuffled and fed as input, and the original (deshuffled) sequence is used as target.

4.1.7. Next-Sentence Prediction (NSP) Objective

Many language tasks, such as natural language inference (NLI) and question answering, require understanding of the relationship between sentences. The NSP objective is used to capture such relationships, where, given an input sentence, the goal is to predict the following sentence. The NSP task takes two sequences (S1 and S2) as input to predict whether S2 is the direct continuation of S1 or not. Table 2 details different objectives, datasets, and tokens and/or corpus sizes used during the pretraining of prominent LLM models.

Table 2. Pretraining details of LLMs.

Model	Architecture	Objectives	Pretraining Dataset	Tokens, Corpus Size
Transformer-base [24]	encoder–decoder			
Transformer-big [24]	encoder–decoder	MLM, NSP	WMT 2014	-
BERT-base [26]	Encoder-only			
BERT-large [26]	Encoder-only	MLM, NSP	BooksCorpus, English Wikipedia	137 B, -
GPT-1 [27]	Decoder-only	Causal/LTR-LM	BooksCorpus, 1B Word Benchmark	-
GPT-2 [28]	Decoder-only	Causal/LTR-LM	Reddit, WebText	-, 40 GB
GPT-3 [29]	Decoder-only	Causal/LTR-LM	Common Crawl, WebText, English-Wikipedia, Books1, Books2	300 B, 570 GB
T5 [25]	encoder–decoder	MLM, Span Correction	C4	(1T tokens) 34 B, 750 GB
REALM [30]	Retriever + Encoder	Salient Span Masking	English Wikipedia (2018)	-
Jurassic-1 [31]	Decoder-only	Causal/LTR-LM	Wikipedia, OWT, Books, C4, PileCC	300 B
mT5 [32]	encoder–decoder	MLM, Span Correction	mC4	-

Table 2. *Cont.*

Model	Architecture	Objectives	Pretraining Dataset	Tokens, Corpus Size
Pangu-Alpha [33]	Decoder + Query Layer	LM	Public datasets (e.g., BaiDuQA, CAIL2018, Sogou-CA, etc.), Common Crawl, encyclopedia, news and e-books	1.1 TB (80 TB raw)
CPM-2 [34]	encoder–decoder	MLM	encyclopedia, novels, QA, scientific literature, e-book, news, and reviews.	-, 2.3 TB Chinese data and 300 GB English Data
Yuan 1.0 [35]	Decoder-only	LM, PLM	Common Crawl, Public Datasets, Encyclopedia, Books	5 TB
HyperClova [36]	Decoder-only	LM	Blog, Cafe, News, Comments, KiN, Modu, WikiEn, WikiJp, Others	561 B
GLaM [37]	Sparse/MoE Decoder-only	LM	Web Pages, Wikipedia, Forums, Books, News, Conversations	1.6 T tokens,
ERNIE 3.0 [38]	Transformer-XL structure	UKTP	plain texts and a large-scale knowledge graph	375 billion, 4 TB
Gopher [39]	Decoder-only	LM	MassiveText (MassiveWeb, Books, C4, News, GitHub, Wikipedia)	300 B
Chinchilla [40]	-	-	MassiveText	1.4 T
AlphaCode [41]	encoder–decoder	MLM, LM	Github, CodeContests	967 B
CodeGEN [42]	decoder-only	LM	THEPILE, BIGQUERY, and BIGPYTHON	505.5 B
CodeGeeX [43]	decoder-only	LM	The Pile, CodeParrot Collected	850 B
FLAN [44]	Decoder-only	LM	web documents, dialog data, and Wikipedia	2.49 T tokens,
InstructGPT [45]	Decoder-only	LTR-LM	Common Crawl, WebText, English-Wikipedia, Books1, Books2, Prompt Dataset (SFT, RM, PPO)	300 B, 570 GB
LaMDA [46]	Decoder-Only	LM	public dialog data and web text	168 B (2.97 B documents, 1.12 B dialogs, and 13.39 B) 1.56T words, -
T0 [47]	encoder–decoder	MLM + LM	C4	1T tokens + 100 B
GPT NeoX 20 B [48]	Decoder-only	LM	The Pile	- 825 GB
OPT [49]	Decoder-only	-	BookCorpus, Stories, the Pile, and PushShift.io Reddit	180 B tokens
MINERVA [50]	Decoder-only + Parallel Layers	LM	technical content dataset (containing scientific and mathematical data), questions from MIT's OpenCourseWare, in addition to PaLM pretraining dataset	38.5 B tokens (math content),
AlexaTM 20 B [51]	seq2seq (encoder–decoder)	mix of denoising and Causal language modeling (CLM) tasks	Wikipedia and mC4 datasets	1 Trillion tokens, -
GLM-130 B [52]	bidirectional encoder and unidirectional decoder,	GLM, MIP (Multitask Instruction pretraining)		400 billion tokens,
XGLM [53]	decoder-only	Causal LM	CC100-XL	
PaLM [54]	Decoder-only + Parallel Layers	LM	Social media conversations, Filtered webpages, Wikipedia (multilingual), Books, Github, News (English)	780 B tokens,
Galactica [55]	decoder-only	-	papers, code, reference material, knowledge bases, filtered CommonCrawl, prompts, GSM8k, OneSmallStep, Khan Problems, Workout, Other	106 B
Pali [56]	encoder–decoder and Vision Transformers	mixture of 8 pretraining tasks	WebLI (10 B images and texts in over 100 languages), 29 billion image-OCR pairs	-
LLaMA [57]	transformer	LM	CommonCrawl, C4, Github, Wikipedia, Books, ArXiv, StackExchange	1.4 T tokens,
UL2 [58]	Enc–Dec, decoder-Only Prefix-LM	R, S, X denoising	C4	32 B tokens,
Pythia [59]	decoder-only	LM	the Pile	300 B tokens -

Table 2. *Cont.*

Model	Architecture	Objectives	Pretraining Dataset	Tokens, Corpus Size
WeLM [60]	-	-	Common Crawl, news, books, forums, academic writings.	-
BLOOM [22]	Causal Decoder-only	LM	ROOTS corpus (46 natural languages and 13 programming languages)	366 B, 1.61 TB
GLM [61]	bidirectional encoder and unidirectional decoder	GLM		
GPT-J	Mesh Transformer JAX	LM	PILE	402 B
YaLM			online texts, The Pile, books, other resources (in English, Russian)	1.7 TB
Alpaca				
Falcon	encoder-only	LM	RefinedWeb, Reddit	1T
(Xmer) XXXL [62]	encoder–decoder	MLM	C4	1T tokens
[63]	decoder-only (MoE)	LM	BookCorpus, English Wikipedia, CC-News, OpenWebText, CC-Stories, CC100	300 B
XLM-R [64]	encoder-only	Multilingual MLM	CommonCrawl (CC-100)	2.5 TB

4.2. Learning Strategies

4.2.1. Multitask Pretraining

In multitask learning (MTL), parameters are shared between multiple tasks during pretraining. This leads to better generalization and performance improvement of related tasks. MTL helps to improve performance on new domains by leveraging the knowledge and representation learned from related tasks during pretraining. MTL uses a single model to perform many downstream tasks simultaneously. However, unlike the adapter layers, MTL requires simultaneous access to the tasks during pretraining. The networks' lower MTL layers (and their weights) are shared among the tasks, using specialized higher layers based on the downstream tasks.

During 'multitask learning', datasets from different tasks are mixed and used. As experimented in T5 [25], multitask learning involves pretraining the model on multiple tasks simultaneously. Although multiple tasks were used during pretraining, the T5 model was finetuned separately on supervised downstream tasks. One crucial factor to consider in multitask learning is how much data the model should be trained on from each task.

There needs to be a proper balance where the model sees enough data to perform well regarding the task while not exposing it to more data such that it starts memorizing (overfitting) the dataset. Additionally, the proportion of data also depends upon factors such as dataset sizes, difficulty of learning the task, regularization, and task interference since performing better on one task might degrade the performance on another task. In T5 [25], as the same training objective was used for every task, only a single set of hyperparameters was required for effective finetuning on all downstream tasks.

In MTL, as the same model performs many different tasks, the language model becomes conditioned on the input and the task to be performed. Such task conditioning can be implemented at the architecture level. However, a recent technique from GPT-2 [28] suggests a simplified mechanism where tasks, inputs, and outputs can all be specified as a sequence of symbols. That is, to be architecture-independent, the input can be transformed to incorporate task-aware information as a context (added as task-prefix) to the input sequence. Also, as stated in T5, every text processing problem can be mapped to "text-to-text" format, where the input and output are both text. For instance, to translate an English sentence "I am good" to French, the prefix "translate English to French: I am good. Target:" will be used, where the model will then be asked to generate the remainder "je vais bien" of the sequence in an autoregressive manner. So, similar to a translation of a sequence of (translate to French, English sentence, French sentence), a reading comprehension example can likewise be written as a tuple consisting of (answer the question, document, question, answer). Using this framework, the same encoding and decoding procedure is used across

various tasks, without requiring any change to the model architecture. Therefore, the same model can be effectively applied for transfer and inference purposes on many different downstream tasks, allowing it to generalize and perform well on new and related domains.

As hypothesized in T0 [47], because of the implicit multitask learning, LLMs can attain reasonable zero-shot generalization on diverse tasks. For instance, during pretraining, some tasks would appear in explicit form with the task instructions, input, and output pairs. For example, there are websites containing FAQs and their answers, which act as supervised training data for the closed-book QA task. Such multitask supervision might play a crucial role in zero-shot generalization during pretraining. To test the hypothesis, T0 attempts to induce zero-shot generalization by explicit multitask learning, where it uses the T5 [25] model and finetunes it in a supervised manner on a dataset with a wide variety of tasks in natural language prompted format. Due to this approach, T0 was able to better generalize on held-out tasks without requiring data at a massive scale and became more robust to the prompt wording. WeLM [60] also reinforced generalization across tasks through explicit multitask learning, where the trained model was then tested on a set of held-out tasks.

4.2.2. Multilingual Pretraining

Researchers have investigated incorporating a multilingual corpus during pretraining to make models perform in multiple languages. For example, XLM-R [64] is a multilingual model pretrained on CommonCrawl100 corpus with text from around 100 languages. It obtained SOTA performance on cross-lingual tasks such as question answering, classification, and sequence labeling. XLM-R also demonstrated how pretraining the multilingual model at scale helps to improve performance across various cross-lingual transfer tasks. For low-resource languages, XLM models trained on CommonCrawl-100 performed better than those trained using Wikipedia.

Another example is mT5 [32], which uses multilingual corpus mC4 to pretrain the model. When dealing with multilingual models (especially in zero-shot settings), there is a chance of 'accidental translation', where the model might translate the prediction in the wrong language. For example, suppose the model proceeds through English-only finetuning. In that case, the probability of generating non-English tokens decreases, reaching a point where English becomes the most likely language to answer any question. Here, the model outputs English when provided a non-English test input because the model never observed a non-English target during finetuning. To address 'accidental translation', mT5 [32] mixed the unlabeled pretraining data during finetuning, dramatically alleviating this issue.

AlexaTM 20B [51] is the first seq2seq model trained using multilingual that can perform in-context learning and provide SOTA performance on multilingual tasks. When tested on the Flores-101 machine translation benchmark dataset, it outperformed existing supervised models almost across all language pairs using only the one-shot method. It also achieved a significant performance boost on machine translation tasks involving to-and-from low-resource languages, such as Telugu, Marathi, and Tamil. AlexaTM20B achieved SOTA performance on Paws-X, XWinograd, XNLI, and XCOPA multilingual tasks in a zero-shot setting. It also performed better on SuperGLUE and SqUADv3 datasets than GPT-3 under zero-shot setting. In the one-shot summarizing task, AlexaTM20B performed better than models that were much larger in scale than its size, such as 540B PaLM decoder model.

4.2.3. Mixture of Experts (MoE)-Based Pretraining

Pretraining LLMs requires significant computing power and resources. To address this issue, sparse experts were proposed, incurring substantially decreased training costs compared to dense models. The same parameters are reused and applied to all the inputs in a traditional static neural network. Instead, a mixture of experts (MoE)-based network enables dynamic selection of parameters for each incoming input and improves model capacity without incurring additional computation costs. In MoE, although a large number of weights are used during training, only relevant experts are needed to compute a small

subset of the computational graph at inference time. Additionally, in static networks, as the entire model becomes activated for every example, the training cost is increased (roughly quadratically) with the increase in model size and training examples [65]. Meanwhile, ST-MoE [66] demonstrated how a 269B sparse parameter model has comparable or similar computational cost to an encoder–decoder transformer model with only 32B parameters and still achieves SOTA performance across a variety of NLP tasks. However, in MoE, when the model size is scaled by increasing the number of sparsely gated experts, it can significantly enlarge the parameter size, requiring more storage memory (it can reach the order of hundreds of GBs).

In MoE, a trainable gating network determines which combination of sparse experts needs to be selected to process the given input. Ref. [65] introduced MoE and demonstrated how conditional computation using sparsely gated experts improved model capacity by 1000 times, with a minor loss in computational efficiency. This is helpful, especially for language modeling and machine translation tasks, where the model capacity is essential to assimilate or absorb large amounts of information from the corpora. Using MoE, Ref. [65] performed better on language modeling and machine translation tasks than prior studies.

Similarly, with MoE, GShard [67] was able to efficiently perform training and inference using conditional computation, where only a subnetwork is activated on a per-input basis. Additionally, the translation quality of GShard increased with model size, but, due to MoE, the wall time of training increased only sublinearly. GShard, pretrained on multilingual, when translating text from 100 languges to English, was able to achieve better translation quality compared to prior cases. Additionally, an annotation technique was used by GShard to annotate the tensors either for distribution or replication across a cluster of devices.

MoE-based models incur additional storage space. This might create difficulty in the model training and inference phase if GPUs capacity is exceeded. To address this issue, CPM2 [34] proposed the INFMOE framework. This framework uses a dynamically scheduled offloading strategy and enables MoE model inference on a single GPU. The parameters of experts from MoE layers are offloaded to CPU memory to enable the inference of the model on a single GPU.

As demonstrated in [63], for model training and inference, MoEs yield competitive zero- and few-shot performance (except full-shot finetuning) at a fraction of the computation. MoEs can match the dense model performance with four times less computing. Furthermore, the performance gap between MoE and dense models varies greatly across domains and tasks, indicating that MoE and dense models might generalize differently. GLaM [37] also used sparsely activated MoE architecture to achieve competitive few-shot task results compared to SOTA-dense models while being more computationally efficient. Although GLaM (1.2 T parameters) is seven times larger than GPT-3 in parameters, it activates a subnetwork of 96.6B (8% of 1.2 T) parameters, consumes only one-third of the energy used to train GPT-3, requires only half of the computation flops for inference, and achieves better overall zero-, one-, and few-shot performances across 29 NLP tasks.

Spare expert models have resulted in a pretraining speedup of 4–7 times while keeping the computational cost (FLOPs per token) constant. Although sparse expert models have many parameters, they reduce the carbon footprint by an order of magnitude. For example, they achieve the same level of one-shot performance as GPT-3 but use only one-third of the energy training cost. Although MoE requires additional storage space for parameters, the sparse language model is one of the promising alternatives to save energy costs.

The experts in the MoE layers are shared across many devices since the sheer size makes it infeasible to replicate them across all devices. Also, MoE sparse models do suffer from training instabilities worse than those encountered in traditional static densely activated models. Switch-transformer [68] addressed some of the issues observed in MoE models, such as complexity, communication costs, and training instability. Switch-transformer simplified the MoE routing algorithm and proposed an architecture that mitigates the instabilities in computational efficiency and with reduced communication.

4.2.4. Knowledge-Enhanced Pretraining

Commonly, plain text is used during pretraining, which lacks explicit linguistic and world knowledge representation. The plain text also lacks structured representation and does not have the explicit grounding to entities from the real world. Such representations fail to capture the entities and the facts among those entities. ERNIE [69] incorporated structured knowledge facts during pretraining using knowledge graphs to address this issue. Using knowledge graphs, ERNIE could exploit syntactic, knowledge, and lexical information, which helped it perform better on several knowledge-driven tasks. In KnowBert [70], multiple knowledge bases were used during pretraining to enhance the representations further. In relationship extraction, entity typing, and word sense disambiguation downstream tasks, KnowBert demonstrated improved perplexity and better ability to recall facts after it was integrated with WordNet and a subset of Wikipedia knowledge bases.

To learn commonsense knowledge, CALM [71] proposed generative and contrastive objectives and incrementally pretrained the model. As its parameters can capture concept-centric commonsense understanding and reasoning, it does not have to rely on external knowledge graphs. The results demonstrate how, despite being trained on a minimal dataset, CALM outperformed the T5-base model on all commonsense-related datasets. To accelerate the pretraining process, CPM2 [34] proposed a knowledge inheritance technique where it uses knowledge from existing pretrained models instead of training the models from scratch. Instead of self-supervised, WKLM [72] proposed a weakly supervised pretraining objective that helped it incorporate knowledge of real-world entities, where it achieved significant improvements in fact completion and two entity-related tasks.

KEPLER [73] jointly optimized the language modeling and knowledge embedding (KEs) objective. As a result, language representation and factual knowledge were better aligned to produce more effective text-enhanced KEs. Similarly, CoLAKE [74] used extended MLM objectives to learn contextualized representation for language and knowledge jointly. Instead of just using entity embeddings, CoLAKE also considers the knowledge context of those entities derived from large-scale knowledge bases. Using these knowledge contexts along with the language context information, a word–knowledge graph was constructed to deal with the heterogeneity of language and knowledge context. Experimental results demonstrated the effectiveness of CoLAKE on knowledge-required tasks after it was pretrained on the large-scale word–knowledge graph.

When injecting knowledge information, previous methods mainly updated the original parameters of the pretrained models. This works fine if only one knowledge base is to be injected. If multiple knowledge bases are injected, the history of previously injected knowledge gets erased. K-ADAPTER [75] overcame this issue by using a neural adapter for each kind of infused knowledge, where there is no information flow between adapters. Hence, they can be trained in a distributed way. K-ADAPTER used this framework that keeps the pretrained model's original parameters fixed, so the parameters that learned from the old knowledge base are not affected after injecting the new knowledge base. K-ADAPTER supports continual knowledge infusion development, and, as adapters are smaller, the model scales much more favorably. As a case study, after injecting K-ADAPTER with two kinds of knowledge, results on three knowledge-driven tasks brought further improvements.

Ref. [46] demonstrated how finetuning with annotated data and consulting external knowledge sources led to significant improvements, especially in the model's safety and factual grounding aspects. These responses, grounded on external knowledge, were first filtered (for safety) before ranking them regarding quality score. LaMDA demonstrated how this quality was improved as the model was scaled. However, to improve the safety and groundness of the model, LaMDA has to rely on an external retrieval system through API calls. ERNIE 3.0 [38] is trained on plain texts and large-scale knowledge graphs. It integrated autoencoder and autoregressive networks into a single unified framework. So, it was able to deal with NLG as well as NLU tasks in finetuning and zero/few-shot learning settings. Additionally, ERNIE 3.0 used prompt-tuning during finetuning to better exploit knowledge from the pretrained model.

4.2.5. Mixture of Denoisers (MoD)-Based Pretraining

Typically, the objectives used during pretraining differ in the context in which the model is conditioned. For example, span correction objectives use bidirectional context and are helpful for language understanding and fact completion tasks. In contrast, prefix-LM objectives use unidirectional context (previous tokens) and are helpful for more open-ended and generative tasks. To enable strong performance across all the different tasks, UL2 [58] proposed mixture of denoisers (MoD) objectives that uniformly combine several paradigms to achieve hybrid self-supervised objectives. UL2 distinguishes between these different denoiser modes during pretraining and adaptively switches modes while finetuning the downstream tasks using discrete prompting. This is achieved by introducing an additional paradigm token ([R], [S], or [X]) during pretraining so that the model can select a mode that is more appropriate for the task at hand. This helps bind or associate the downstream finetuning behavior with the specific mode used during pretraining. MoD consists of the following denoising objectives:

Extreme Denoising

It considers extreme span lengths to have a corruption rate of up to 50%. Therefore, given a small or moderate part of the input, the model is supposed to recover or predict a large chunk of the sequence. The pretraining objective is considered to be highly denoising if it has a long span (for example, equal to or greater than 12) or has a large corruption rate (for example, more significant or more than 30%). So, it covers scenarios with long spans and low corruption, long spans and high corruption, and short spans and high corruption, where it generates long targets based on relatively limited information from memory.

Sequential Denoising

This objective strictly follows sequence order, i.e., the prefix language modeling. The target tokens cannot attend to the future context tokens, but the prefix context does use bidirectional architecture.

Regular Denoising

This denoising approach has short spans, a range of two to five tokens, and a low corruption rate that masks up to 15% of the sequence. Because of the short span length, they are not fit for generating text but are preferred for acquiring knowledge and understanding tasks.

With the MoD approach, UL2 outperformed GPT-3 on the SuperGLUE benchmark in the zero-shot setting, and, in the one-shot setting, it tripled the performance of T5-XXL on the summarizing task. In the zero-shot setting, UL2-20B also outperformed T0 and T5 on the massive multitask language understanding (MMLU) benchmark and performed well with chain-of-thought processes using prompting and reasoning steps. UL2-20B, when tested with FLAN instruction tuning, achieved a competitive score compared to FLAN-PaLM 62B on MMLU and Big-Bench benchmarks. After using the MoD objective, U-PaLM [76] achieved the same performance as PaLM-540B but with only half of its computational budget.

4.2.6. Prompt Pretraining

Instead of using a generic dataset, Galactica [55] focused on using a highly curated high-quality scientific dataset for pretraining. Galactica also differs from existing LLMs in that it augments pretraining by including task prompts alongside the corpora, which helps it outperform existing models on a range of scientific tasks. Galactica outperformed GPT-3 on technical knowledge probe tasks, performed better than PaLM-540 B on MATH, and outperformed Chinchilla on the MMLU benchmark. Despite not having been trained on general corpora, Galactica performed better than BLOOM and OPT-175B on the Big-Bench benchmark. It also achieved state-of-the-art results on PubMedQA and MedMCQA benchmarks.

4.2.7. Information-Retrieval-Based Pretraining

Although LLMs implicitly store knowledge in the network parameters, it becomes difficult to determine which knowledge is stored at which location. REALM [30] addressed this issue by adding a discrete retrieval step called 'textual knowledge retriever' to the pretraining algorithm. This retriever is rewarded for retrieving documents with relevant information and penalized otherwise. REALM uses this retriever to retrieve the relevant documents and attend to only those retrieved documents to make predictions.

Pangu-Alpha [33] uses a query layer, which helps to explicitly induce the expected output. The query layer is stacked on top and resembles the transformer layer, except an additional embedding is fed as an input. This additional input represents the next position used as the query vector in the attention mechanism. Similarly, Falcon uses a multiquery attention mechanism, sharing keys and values across all heads. This does not influence pretraining significantly. However, it improves the scalability of inference.

5. Transfer Learning Strategies

Discriminatively trained models perform well if labeled data are available in abundance, but they do not perform adequately for tasks with scarce datasets as this limits their learning abilities. To address this issue, LLMs were first pretrained on large unlabeled datasets using the self-supervised approach, where the learning was then transferred discriminatively on specific tasks. As a result, transfer learning helps to leverage the capabilities of pretrained models and is advantageous, especially in data-scare settings. For example, GPT [27] used the generative language model objective for pretraining, followed by discriminative finetuning. Compared to pretraining, the transfer learning process is inexpensive and converges faster than training the model from scratch. Additionally, pretraining uses an unlabeled dataset and follows a self-supervised approach, whereas transfer learning follows a supervised technique using a labeled dataset particular to the downstream task. The pretraining dataset comes from a generic domain, whereas, during transfer learning, data come from specific distributions (supervised datasets specific to the desired task).

5.1. Finetuning

Transfer learning started with feature-based techniques, where pretrained embeddings such as Word2Vec were used on the custom downstream models. Once learned, the embeddings are not refined to the downstream tasks, making them task-dependent. In finetuning, after copying the weights of the pretrained network, they are finetuned to adapt to the peculiarities of the target task. In finetuning, as the parameters learned during pretraining are adjusted to a specific downstream task, it outperforms the feature-based transfer technique. Such finetuning enables the model to learn task-specific features and improve the downstream task performance. As a result, the finetuned embeddings adapt not only to the context but also to the downstream task in consideration. So, unlike feature- or representation-based transfer, finetuning does not require task-specific model architecture. Although the finetuning strategy yields strong performance on many benchmarks, it has some limitations, such as the need for a large amount of downstream task-specific datasets, which can lead to poor generalization for data from out of distribution and the possibility of spurious features. During finetuning, instead of including all the parameters, adapter layers and gradual unfreezing techniques were proposed, which considered only a subset of the parameters during finetuning.

5.2. Adapter Tuning

Feature and vanilla finetuning techniques could be more parameter-efficient since they require new network weights for every downstream task. So, these techniques require an entirely new model for every downstream task. To address this issue, Ref. [77] proposed a transfer with the adapter module in which a module is added between the layers of a pretrained network. In each block of the transformer, these adapter layers, which are

dense-RELU-dense blocks, are added after the feed-forward networks. Since their output dimensionality matches their input, no structural or parameter changes are required to insert adapter layers. During finetuning, most of the original model is kept fixed, and only the parameters from adapter layers get updated. In adapter tuning, task-specific layers are inserted, with only a few trainable parameters added per task. Also, a high degree of parameter sharing occurs as the original network is kept fixed.

Unlike the feature-based technique, which reads the inner layer parameters to form the embeddings, adapters write to the inner layers instead, enabling them to reconfigure network features. The main hyperparameter of this approach is the feed-forward network's inner dimensionality 'd' since it determines the number of new parameters that will be added to the model. This approach is a promising technique in the experiments conducted in [25]. Adapter tuning attains comparable performance with finetuning on NLU and NLG benchmarks by using only 2–4% task-specific parameters. Experiments from [77] demonstrated how BERT with adapters added only a few (3:6%) parameters per task to attain near SOTA on the GLUE benchmark.

5.3. Gradual Unfreezing

In gradual unfreezing, more and more of the model's parameters are finetuned over time. In this approach, at the start of finetuning, only the parameters of the final layer are updated first. Next, the parameters of the second-last layers are included in the finetuning. This process continues until the parameters of all the network layers are finetuned (updated). It is normally recommended to include an additional layer in finetuning, after each epoch of training. This approach was used in [25], where gradual unfreezing resulted in minor performance degradation across all the tasks.

5.4. Prefix Tuning

Finetuning, although it leverages the knowledge from pretrained models to perform downstream tasks, requires a separate copy of the entire model for each task as it modifies all the network parameters. To address this issue, prefix tuning [78] keeps the pretrained parameters frozen and optimizes only the task-specific vectors. These continuous task-specific vectors, called prefixes, are prepended to the input sequence so the subsequent tokens can attend to these vectors. Prefix tuning uses a small trainable module to train and optimize these small task-specific vectors associated with the prefix. The errors are backpropagated to prefix activations prepended to each layer during tuning. In prefix tuning for each task, only the prefix parameters are stored, making it a lightweight, modular, and space-efficient alternative. Despite learning 1000× fewer parameters than finetuning, prefix tuning [78] outperformed finetuning in low-data settings and maintained comparable performance in full-data settings. It also extrapolated better to the examples with topics that were unseen during training by learning only 0.1% of the parameters.

5.5. Prompt-Tuning

Although finetuning the pretrained language models has successfully improved the results of downstream tasks, one of its shortcomings is that there can be a significant gap between the objectives used in pretraining and those required by downstream tasks. For instance, downstream tasks require objective forms such as labeling (parts of speech tagging) or classification, whereas pretraining is usually formalized as a next-token prediction task. One of the reasons behind the prompt-tuning approach was to bridge this gap between pretraining and finetuning objectives and help in better adaption of knowledge from pretrained models to downstream tasks. In prompt-tuning, prompts are used to interact with LLMs, where a prompt is a user-provided input to which the model responds. Prompting is prepending extra information for the model to condition on during the generation of output. This extra information typically includes questions, instructions, and a few examples as tokens to the task input.

5.5.1. Prompt Engineering

Prompt engineering involves the process of carefully designing optimal prompts to obtain optimal results. Prompts need to be constructed to best elicit knowledge and maximize the prediction performance of the language model. The prompt-based approach is a promising alternative to finetuning since, as the scale of LLMs grows, learning via prompts becomes efficient and cost-effective. Additionally, unlike finetuning, where a separate model is required for each downstream task, a single model serves multiple downstream tasks in prompt-tuning. They also help the model generalize better to held-out tasks and cross-tasks by using multitask prompts.

As per [79], finetuning on downstream tasks for trillion-scale models results in poor transferability. Also, these models need to be larger to memorize the samples in finetuning quickly. To overcome these issues, the prompt-tuning or P-tuning approach [80] is used, which is a parameter-efficient tuning technique. For example, GPT3 [29] (which was not designed for finetuning), heavily relied on handcraft prompts to steer the model for downstream applications. Prompt-tuning came into play to scale this (manual) prompt engineering technique. Prompt-tuning can be categorized into discrete and continuous approaches.

Unlike finetuning, where a separate model is required for each downstream task, in prompt-tuning, a single model serves multiple different downstream tasks. In discrete prompt-tuning, as human efforts are involved in crafting the prompts, the process becomes time-consuming and fallible as human efforts are involved in crafting the prompts. It sometimes can be non-intuitive for many tasks (e.g., textual entailment). Additionally, improper construction of contexts leads to low model performance. To overcome these issues, a continuous or tunable prompt-tuning technique was proposed.

5.5.2. Continuous Prompt-Tuning

In continuous prompt-tuning, additional k tunable tokens are used per downstream task, which are prepended to the input text. These prompts are learned through backpropagation and are tunable or adjustable to incorporate signals from any number of labeled examples. Unlike finetuning, only the parameters of these inserted prompt tokens get updated in prompt-tuning. Hence, they are also called soft prompts. Ref. [80] demonstrated how their approach outperformed GPT-3's few-shot learning based on discrete text prompts by a large margin. They also demonstrated that prompt-tuning becomes more competitive with scale, where it matches the performance of finetuned models. For example, prompt-tuning of T5 matched the model's finetuning quality as the size increased while enabling the reuse of a single frozen model for all the tasks.

P-tuning uses a small trainable model that encodes the text prompt and generates task-specific tokens. These tokens are then appended to the prompt and passed to the LLM during finetuning. When the tuning process is complete, these tokens are stored in a lookup table and used during inference, replacing the smaller model. In this approach, the time required to tune a smaller model is much less. Ref. [79] utilized a P-tuning technique to automatically search prompts in the continuous space, which enabled the GPT-style model to perform better on NLU tasks. Unlike the discrete-prompt approach, in continuous prompt, as there are trainable embedding tensors, the prompt encoder can be optimized in a differentiable way. P-tuning helped to augment the pretrained model's NLU ability by automatically searching for better prompts in the continuous space. As demonstrated in [79], the P-tuning method improves GPTs and BERTs in both few-shot and fully supervised settings.

Additionally, as only the parameters of prompt tokens are stored, which are less than 0.01% of the total model parameters, the prompt-tuning approach saves a significant amount of storage space. For example, CPM-2 [34] used only 100 prompt tokens, where only 409.6 K trainable parameters were to be updated compared to the 11B parameters of finetuning. As demonstrated in CPM-2, except for the Sogou-Log task, CPM-2 with prompt-tuning achieved comparable performance to the finetuning approach. In prompt-tuning, as the number of parameters to be optimized is much smaller, the size required for

tensors (gradient and optimizer state) significantly decreased. As a result, prompt-tuning can save at most 50% GPU memory as compared to finetuning.

However, prompt engineering also has limitations, such as prompt-tuning taking many more steps to converge and hence more time. Additionally, only a small number of examples can be used, which limits the level of control. Also, as the examples are part of the prompt, it affects the token budget.

5.6. Multilingual Finetuning

Most language models are monolingual, using data in the English language only during pretraining. Such models, therefore, cannot be used to deal with tasks that are non-English-language-related. To overcome this issue, multilingual models were proposed to enable the processing of non-English languages. Such multilingual models can also be used for cross-lingual tasks like translation. However, models such as GPT-3 were potentially limited in dealing with cross-lingual tasks and generalization because most of these models had English-dominated training datasets.

XGLM [53] focused on using a multilingual dataset (comprising a diverse set of languages) for finetuning. As a result, XGLM achieved cross-lingual solid transfer, demonstrating SOTA few-shot learning performance on the FLORES-101 machine translation benchmark between many language pairs. When BloomZ [81] was finetuned with xP3, a multilingual task dataset of 46 languages, the model achieved better zero-shot task generalization (than the P3-trained baseline) on English and non-English tasks. Furthermore, when xP3mt, a machine-translated multilingual dataset of xP3, was used to finetune BloomZ on non-English prompts, the performance of held-out tasks with non-English human-written prompts significantly improved. In other words, as models generalize to tasks they had never intentionally seen, they learn the higher-level capabilities that are both task- and language-agnostic.

Typically, a cross-lingual dataset is used to make the model language-agnostic, and, to make it task-agnostic, a multitask dataset is required. Also, for large multilingual models, zero-shot performance tends to be significantly lower than finetuned performance. So, to improve the multilingual model's zero-shot task generalization, BloomZ [81] focused on cross-lingual and multitask finetuning. This enabled the model to be usable for low-resource language tasks without further finetuning.

5.7. Reinforcement Learning from Human Feedback (RLHF) Finetuning

Although the LMs can be prompted to generate responses to a range of NLP tasks, sometimes, these models might showcase unintended behavior by generating toxic responses or results that are not aligned with the user instructions. This happens because the objectives used to pretrain LLMs focus on predicting the next token, which might differ or misalign from human intention (user's query or instruction objective). To address this misalignment issue, Ref. [45] proposed reinforcement learning (RL) from human feedback to finetune GPT-3. In the RL-based approach, human labels are used to train a model of reward and then optimize that model. Using human feedback, it tries to align the model by the user's intention, which encompasses explicit and implicit (such as being truthful and not being toxic, harmful, or biased) intentions.

RLHF aims to make the model honest, helpful, and harmless. The RLHF approach uses human preferences as a reward signal to finetune the model. It was demonstrated how, despite having 100× fewer parameters, the outputs from the InstructGPT model with 1.3 B parameters were preferred over GPT-3 with 175 B parameters.

Using the RLHF approach, InstructGPT demonstrated improvement in toxicity and truthfulness over GPT-3 and generalized well to held-out instructions. Ref. [82] applied reinforcement learning (RL) to complex tasks defined only by human judgment, where only humans can tell whether a result is good or bad. In [82], the pretrained model was finetuned using reinforcement learning rather than supervised learning, where it demonstrated its results on summarizing and continuation tasks by applying reward learning to language generation. Ref. [83] recursively used the RL approach to produce novel summaries and

achieve SOTA results for book-length summarizing on the BookSum dataset. Similarly, using the reinforcement learning technique, Ref. [84] trained a model to predict the human-preferred summary and used it as a reward function to finetune the summarizing policy. It could outperform larger models finetuned using a supervised approach and human reference summaries and generalize well to new datasets.

5.8. Instruction Tuning

In instruction tuning, the model is finetuned on a collection of datasets where the NLP tasks are described using natural language instructions. Natural language instructions are added to the prompt to let the model know which task to perform for a given input. For instance, to ask the model to perform a sentiment analysis task on a given input, instructions such as 'Classify this review either as negative, positive, or neutral' can be provided in the prompt. Various factors determine the effectiveness of instruction tuning on LLMs, such as the prompt format used, objectives used during finetuning, diversity of tuning tasks, distribution of datasets, etc. Additionally, the zero-shot task generalization of LLMs performs poorly across tasks. To address this, multitask finetuning (MTF) has emerged and become one of the promising techniques to improve the performance of LLMs in zero-shot settings.

Creating instruction datasets for many tasks from scratch is a resource-intensive process. Instead, FLAN [44] expresses existing 62 NLP datasets in the instructional format. This transformed dataset with instructions is then used to finetune the model. For each dataset, 10 unique templates were created to describe the task in instructional format for that dataset. Based on the task type, the datasets were grouped into clusters, and then, to evaluate the performance on each task, the specific task cluster was held out while the remaining clusters were used during instruction tuning.

FLAN demonstrated how instruction tuning substantially improved the zero-shot performance on held-out tasks that were not part of the instruction tuning process and also helped the model generalize well on unseen tasks. FLAN outperformed GPT-3 (zero- and few-shot) on 20 of the 25 datasets used for evaluation. It was observed that the instruction tuning approach is more effective for tasks such as QA, NLI, and translation that can easily be verbalized as instructions. Instruction tuning is less effective for tasks where the instructions are redundant since they can be formulated simply as language modeling tasks, such as commonsense reasoning. FLAN also demonstrated how instruction tuning can hurt smaller models since their capacity is mostly exhausted in learning different instruction tasks.

Alpaca uses Meta's LLaMA model and finetunes it with 52 K instructions following demonstrations in a supervised manner. These instructions were generated using GPT3.5 (text-davinci-003), where 175 human-written instruction–output pairs from the self-instruct were used as a seed to generate more instructions. Tk-INSTRUCT [85] proposed a benchmark with instructions for 1616 nlp tasks, so such a benchmark dataset can be beneficial in studying multitask learning and cross-task generalization. This dataset, called 'SUPER-NATURAL-INSTRUCTIONS (SUP-NATINST)', is publicly available. It covers instructions in 55 different languages, and the 1616 nlp tasks can be categorized under 76 broad task types. For each task, it provides instructions comprising several examples with the desired output along with the definition that maps input text to task output. When evaluated on 119 unseen tasks (English and multilingual variants), TK-INSTRUCT outperformed InstructGPT by 9.9 ROUGE-L points, and mTK-INSTRUCT outperformed InstructGPT by 13.3 points on 35 non-English tasks.

OPT-IML [86], instruction-tuned-on OPT, conducted experiments by scaling the model size and benchmark datasets to see the effect of instruction tuning on performance. It also proposed a benchmark called 'OPT-IML Bench', consisting of 2000 NLP tasks. This benchmark can be used to measure three types of generalizations to tasks from held-out categories, held-out tasks from seen categories, and held-out instances from seen tasks. OPT-IML achieved all these generalization abilities at different scales and benchmarks (PromptSource, FLAN, Super-NaturalInstructions, and UnifiedSKG), having diverse tasks and input formats. OPT-IML was also highly competitive with finetuned models on each

specific benchmark. Furthermore, to improve the performance on reasoning tasks, it used 14 reasoning datasets during instruction tuning, where the output included a rationale (chain-of-thought process) before the answer. Similarly, there was experimentation by adding dialogues as auxiliary datasets to see if that could induce chatbot behavior in the model.

Ref. [87] experimented with instruction tuning regarding model size, number of tasks, and chain-of-thought datasets. It was observed that instruction finetuning scales well, and the model performance substantially improved with the increased size of models and number of finetuning tasks. Additionally, when nine CoT datasets were added to the instruction tuning dataset mixture, the model could perform better on evaluation reasoning tasks. This contradicts other work where instruction finetuning instead degraded CoT task performance. So, Ref. [87] demonstrated how CoT data improves performance reasoning tasks when jointly finetuned with an instruction dataset. After instruction tuning model classes such as T5, PaLM, and U-PaLM, Ref. [87] observed a significant boost in performance for different types of prompting setups (zero, few, and CoT) and benchmarks as compared to the original models (without instruction finetuning).

In self-instruct [88], the bootstrap technique is used to improve the model's instruction following capabilities. Here, the existing collection of instructions is leveraged to generate new and more broad-coverage instructions. Using a language model, self-instruct generates instructions along with input–output samples, filters invalid, low-quality, or repeated instructions, and uses the remaining valid ones to finetune the original model. Along with the instructions, the framework also creates input–output instances, which can be used to supervise the finetuning of instructions. When self-instruct was applied to GPT-3, it achieved a 33% performance gain on SUPER-NATURALINSTRUCTIONS over the original model, which was on par with the InstructGPT performance.

5.9. Code-Based Finetuning

Generating code is a translation task that maps a natural language problem statement to a solution or code in programming language. Recent LLMs are capable of completing programming tasks by generating code. Codex [89] uses the GPT model, which was finetuned on publicly available code from GitHub. It studied Python code-writing capabilities, focused on generating standalone Python functions from docstrings, and then evaluated the correctness of the generated code samples. It was able to solve 28.8% of the HumanEval dataset problems, while GPT-3 solved 0% and GPT-J solved 11.4%. It needs help with docstrings describing long operations chains and binding operations to variables.

To enable the model to solve complex problems and provide deeper reasoning, the AlphaCode [41] model was pretrained on a collection of open-source code from GitHub and then finetuned on a curated set called CodeContests of competitive programming problems. The pretraining dataset consisted of code from several popular programming languages. AlphaCode achieved a ranking of top 54.3% on average in simulated programming competitions with more than 5000 participants that were hosted on the Codeforces platform.

Furthermore, CodeGEN [42] introduced a multistep approach where a user can progressively communicate with the system to provide specifications. Such multiple-step specification eases the understanding of a model, leading to enhanced program synthesis. CodeGeeX [43] is a multilingual model trained on 23 programming languages. To evaluate multilingual models, it proposed a HumanEval-X benchmark where the solutions in C++, Java, JavaScript, and Go were hand-written. CodeGeeX was able to outperform multilingual code models of similar scale for translation on HumanEval-X as well as code generation tasks.

6. In-Context Learning

Finetuning is task-agnostic, but it uses a supervised approach during transfer learning and hence requires access to a large amount of labeled datasets for every downstream task. Furthermore, having such a task-specific dataset leads to finetuning the model on a very narrow distribution, which might potentially yield poor generalization on out-of-

distribution datasets. It might also be overly specific to the distribution, exploiting spurious correlations and features of the training data. The need for such labeled datasets limits the applicability of language models.

To overcome these limitations, in-context learning (ICL) was proposed in GPT-3 [29], where the language model uses in-context information for inference. The main benefits of ICL are the minimal need for task-specific data and the fact that it does not go through any parameter updates or architectural modifications. In ICL, a prompt feeds the model with input–label pair examples, avoiding the need for large labeled datasets. Unlike finetuning, ICL has no gradient updates, so the weights of the model parameters are not updated. In ICL, the abilities that are developed by LLMs during pretraining are applied to adapt to or recognize the task at inference time, enabling the model to easily switch between many tasks.

As experimented in GPT-3, the larger model with 175B parameters outperformed the smaller models by efficiently using in-context information. Based on the experiments conducted in GPT-3, ICL showed initial promise and improved out-of-domain generalization. However, the results are far inferior to those of the finetuning technique. ICL helps to analyze whether the model rapidly adapts to the tasks that are unlikely to be directly contained in the training set. In ICL, the model is conditioned on task instruction and a couple of task demonstrations as a context and is expected to complete the target instance of the task. As transformer-based models are conditioned by a bounded-length context (e.g., 2048 tokens in GPT-3), ICL cannot fully exploit data longer than the context window. Based on the number of demonstrations provided for inference in the context window, ICAL can be categorized as few-shot, one-shot, and zero-shot. We describe each of them below.

6.1. Few-Shot Learning

In few-shot learning, a few examples are provided in the prompt, which helps the model understand how to solve the given task question. In a few-shot setting, the number of demonstrations provided in the prompt typically ranges between 10 and 100, or it includes as many examples that can fit into the model's context window. Compared to the finetuning approach, in a few-shot setting, the number of task-specific examples required is drastically reduced, making it a viable alternative for tasks with smaller dataset sizes. In case the task has many edge cases or is fuzzily defined, having more examples in the prompt can help the model understand the task and predict the result more accurately.

It was shown in GPT-3 [29] how the model performance rapidly improved after a few examples, which, along with a task description, were provided as the context through the window. Similarly, it was demonstrated in Jurassic-1 [31] how classification task accuracy improved after adding more examples in the few-shot setting. Because of the type of tokenizer used in Jurassic-1, it could fit in more examples in the prompt, leading to significant performance gain.

However, it was demonstrated in some of the papers, such as [90], that the examples used in the few-shot setting, the sequence in which the examples were ordered, and the format of the prompt directly affected the accuracy. Ref. [90] demonstrated how this instability in few-shot learning stems from the language model's bias toward predicting specific answers. For example, the model can be biased towards answers placed towards the end of the prompt, those appearing frequently, or those that are familiar in the pretrained dataset. To address this instability, Ref. [90] first estimated the model's bias towards each answer. It then used calibration parameters that caused the prediction for the input to be uniform across answers. This calibration procedure improved GPT-3 and GPT-2's average accuracy by up to 30.0% on a diverse set of tasks and also reduced variance across different prompt choices.

Instead of randomly sampling few-shot examples, Ref. [91] investigated to find effective strategies that could select the in-context learning examples judiciously, which would help in better leveraging the model's capabilities in a few-shot setting. It proposed "KATE", a non-parametric selection approach, which retrieved in-context examples that were semantically similar to the test sample. This strategy helped to provide more relevant and informative inputs to the model, such as GPT-3, and unleashed the model's needed knowledge to solve the problem. GPT-3's performance using KATE was improved by a significant

margin as compared to the random sampling on several NLU and NLG tasks. In [92], the study compared how the model generalizes in few-shot finetuning and in-context learning settings. During the comparison, the model size and number of examples and parameters used in the experiment were controlled. The results demonstrated how the finetuned model generalized similarly to the ICL model regarding out-of-domain conditions and improved performance as the models became larger.

6.2. One-Shot Learning

This approach is similar to few-shot learning except only one example is provided as context in addition to the task description. The pretrained-model can view only one demonstration before making the prediction.

6.3. Zero-Shot Learning

In zero-shot learning, the model is prompted without any example. As there are no demonstrations, only the task instruction is fed as input to the model. Zero-shot learning is helpful when there is no or negligible task-specific dataset available. GPT [27] and GPT-2 GPT2 demonstrated how zero-shot acquires practical linguistic knowledge required for downstream tasks. In GPT-3, the performance of zero-shot setting on tasks such as reading comprehension, NLI, and QA was worse than that of few-shot performance. One of the possible justifications is that, because of the lack of examples, the model finds it challenging to predict correct results based on the prompts that were not similar to the format of pretrained data. Ref. [93] compared different architectures and pretraining objectives and their impact on zero-shot generalization. Experiments from [93] demonstrated that causal decoder-only models trained on an autoregressive language modeling objective using unsupervised pretraining exhibited the strongest zero-shot generalization.

6.4. Chain-of-Thought Learning

Despite the progress made by in-context learning, state-of-the-art models still struggle when dealing with reasoning tasks such as arithmetic reasoning problems, commonsense reasoning, and math word problems, which require solving intermediate steps in precise sequence. The chain-of-thought (CoT) approach is used to address this issue, where examples are provided with a series of intermediate reasoning steps to help the model develop the reasoning required to deduce the answer. In other words, CoT comprises the rationale required as part of the explanation that is used to solve and compute the answer to a complex problem. In [94], it was demonstrated how a CoT-based prompting technique helped to significantly improve LLM performance for complex reasoning tasks. When LLMs are prompted using the CoT technique, they demonstrate the intermediate reasoning steps involved in computing the final answer to unseen problems. CoT prompts indirectly help the model to access relevant knowledge (acquired during pretraining), which helps to improve the reasoning ability of the model.

Experiments have shown how CoT-based prompting improves reasoning-oriented tasks, such as symbolic, commonsense, and arithmetic-based tasks. For example, when PaLM-540B was prompted using eight CoT examples, it surpassed finetuned GPT-3 to achieve SOTA performance on the GSM8K benchmark with math word problems. Similarly, Minerva [50] used the PaLM model and further finetuned it on the technical and mathematical dataset. When Minerva was prompted with CoT examples that included step-by-step solutions, it generated a chain-of-thought answer and demarcated a final answer. Of two hundred undergraduate college-level problems used for evaluation, Minerva answered nearly a third of them from mathematics, science, and engineering domains requiring quantitative reasoning. PaLM [54] analyzed the effect of CoT prompting with model scaling and demonstrated how CoT-based few-shot matched or outperformed state-of-the-art finetuned models on various reasoning tasks.

In zero-shot chain of thought with no examples, CoT reasoning can explicitly be activated by using some trigger phrases, such as “let’s think step-by-step” or “Let’s think

about this logically", to prompt the model to generate explanations. OPT-IML [86] used 15 reasoning datasets and studied the effects of different proportions of reasoning data on different held-out task clusters. The default mechanism or approach used in CoT is greedy decoding, where the most common way of reasoning is selected to solve the problem. Ref. [95] proposed a self-consistency decoding alternative, where, instead of taking the greedy path, it explores different ways of solving a complex reasoning problem that lead to the unique correct answer. Ref. [95] demonstrated how adapting the self-consistency approach in CoT prompts improved performance on benchmarks of commonsense and arithmetic reasoning tasks across four large language models with varying scales. However, this alternative does incur increased computational cost.

As addressed in Galactica [55], some limitations are associated with the CoT process. The CoT process needs some few-shot examples to understand the step-by-step reasoning process, which takes up the context space. Also, as internet data are used for pretraining, such data may have only some of the necessary intermediate steps. Since some trivial, easy, and practiced steps are internally computed and memorized by humans, they may only write down some necessary details or steps as it would lead to long and tedious answers. As only principal steps are involved, this leads to missing data where internally computed steps are not written. As a result, more effort is required to review the datasets and explicitly inject missing steps. Table 3 lists the finetuning methods used in the prominent LLM models along with additional details, such as pretraining (PT) and finetuning (FT) batch sizes and epochs.

Table 3. Finetuning details of LLMs.

Model	PT, FT Batch-Size	Context Size	PT, FT Epochs	Activation, Optimizer	Finetuning Methods
Transformer-base [24]	-	-	100,000		
Transformer-big [24]	-	-	300,000	-, Adam	Feature-based
BERT-base [26]	256, 32	128, 512	40, 4		
BERT-large [26]	256, 32	128, 512	40, 4	GELU, Adam	FT
GPT-1 [27]	64	512	100	GELU, Adam	FT, zero-shot
GPT-2 [28]	512	1024	-	GELU, Adam	zero-shot
GPT-3 [29]	3.2M	2048	-	-	few-shot, one-shot, zero-shot
T5 [25]	128, 128	512	2^{19}, 2^{18} steps	RELU, AdaFactor	FT
REALM [30]	512, 1	-	200 k steps, 2 epochs	-	-
Jurassic-1 [31]	3.2 M tokens	2048	-	-	few-shot, zero-shot
mT5 [32]	-	-	-	GeGLU,	FT, zero-shot
Pangu-Alpha [33]	-	1024	130 K 260 K	GeLU	few-shot, one-shot, zero-shot
CPM-2 [34]	-	-	-	-	FT, PT
Yuan 1.0 [35]	-	-	-	-	few-shot, zero-shot
HyperClova [36]	1024,-	-	-	-, AdamW	few-shot, zero-shot, PT
GLaM [37]	-	1024	-	-, Adafactor	zero-, one-, and few-shot
ERNIE 3.0 [38]	6144	512	-	GeLU, Adam	FT, zero- and few-shot
Gopher [39]	-	2048	-	Adam	FT, few-shot, zero-shot
Chinchilla [40]	-	-	-	AdamW	FT, zero-shot
AlphaCode [41]	2048	-	205 K	-	finetuning
CodeGEN [42]	2 M	2048	-	-	zero-shot
CodeGeeX [43]	3072	-	-	FastGELU, Adam	finetuning
FLAN [44]	-, 8192	1024	30 K	-, Adafactor	Instruction Tuning, zero-shot
InstructGPT [45]	3.2 M	2048	-	-	RLHF
LaMDA [46]	-	-	-	gated-GELU,	FT
T0 [47]	-	-	-	RELU, AdaFactor	FT, zero-shot
GPT NeoX 20B [48]	3.15 M tokens	2048	150 K steps	-, AdamW with ZeRO	few-shot
OPT [49]	2 M tokens	2048	-	ReLU, AdamW	few-shot, zero-shot
MINERVA [50]	-	1024	399 K	SwiGLU, Adafactor	few-shot, chain-of-thought context
AlexaTM 20 B [51]	2 million tokens	-	-	Adam	Finetuning, few-shot, one-shot, zero-shot
GLM-130 B [52]	4224	2048	-	GeGLU,	zero-shot, few (5) shots
XGLM [53]					
PaLM [54]	512, 1024, 2048 (1, 2, 4 M tokens), -	2048	1 (255 k steps)	SwiGLU, Adafactor	few-shot, chain-of-thought, finetuning
Galactica [55]	2 M	2048	4 epochs	GeLU	zero-shot
Pali [56]	-	-		-	
LLaMA [57]	4 M tokens	-	-	SwiGLU, AdamW	zero-shot, few-shot, Instruction Tuning
UL2 [58]	128	512	500 K steps	SwiGLU, Adafactor	in-context learning, zero-shot, one-shot, finetuning, instruction tuning
Pythia [59]	1024	2048	1.5 Epochs	Adam	zero-shot
WeLM [60]	2048	2048	-	-	zero-shot, few-shot
BLOOM [22]	20,482,048	2048	-	GELU, -	zero-shot, few-shot, multitask-prompted (fine)-tuning
GLM [61]	1024		200 K Steps		FT
GPT-J		2048	383,500 steps	-	FT

Table 3. *Cont.*

Model	PT, FT Batch-Size	Context Size	PT, FT Epochs	Activation, Optimizer	Finetuning Methods
YaLM					
Alpaca		-		-	FT, IT (instruction Tuning)
Falcon	-	2048	-	-	-
(Xmer) XXXL [62]			2^{19}		FT
[63]		2048			FT, zero-shot, few-shot

7. Scalability

In recent years, transformer-based language models' capacity has increased rapidly, from a few million parameters to a trillion parameters. Each increase has improved the model's language learning abilities and downstream task performance. Recent research has demonstrated how the loss decreases as the model size increases and follows a smooth trend of improvement with scale. Recent work has demonstrated how scaling up the LLMs improves their abilities across various tasks. LLMs have demonstrated that scaling up language models significantly improves task-agnostic few-shot performance. Recent work has shown that scaling up produces better performance than more carefully engineered methods. If the LLMs are sufficiently pretrained on a large corpus, it can lead to significant performance improvements on diverse tasks. Over time, it has become evident through experiments that the performance of LLMs can steadily be improved by scaling the model size and training data and training the model longer (increasing the training steps).

As stated in [96], emergent abilities are not present in small models but start manifesting or resurfacing in larger models due to scaling. When the parameter size exceeds 100 billion, such emergent zero-shot and few-shot abilities of the model start resurfacing [96]. As stated in [48], such abilities manifest above a certain threshold, and such properties therefore cannot be examined or found in smaller models.

For instance, GPT-3 with 175B parameters performed better with fewer shots (32 labeled examples) than the fully supervised BERT-Large model on various benchmarks. Additionally, with the increase in size, the GPT model has been effective even in zero- and few-shot settings, sometimes matching the finetuning performance. The experiments in [29] demonstrated that, with an increase in model size, model performance improved steadily for zero-shot and rapidly for few-shot models. As their size increases, models tend to be more proficient and efficient at in-context learning. As demonstrated in [33], perplexity decreases with the increase in model capacity, training data, and computational resources. Below, we look at different ways to scale up the model and how it affects the model performance.

7.1. Model Width (Parameter Size)

Kaplan et al. [97] analyzed the effect of model size, computing power, and training data on the performance of language models. The key finding from [97] was that LM performance improves smoothly and predictably as model size, data, and computation are scaled up appropriately. Additionally, large models were more sample-efficient than smaller models as they reached the same level of performance with fewer data points and optimization steps. As per [97], most of the increase in computation should go towards increasing the model size. Also, a relatively small increase in data is needed to avoid reuse, where larger batch sizes can help to boost and increase parallelism. Additionally, larger batches and training for more steps become possible as more computing becomes available. T5 [25] conducted experiments that started with the baseline model having 220 M parameters and then scaled it up to a model with 11 B parameters. The experiments conducted in T5 showed how performance degraded as the data size shrank and improved with the increase in model size and training time.

7.2. Training Tokens and Data Size

Although Kaplan et al. [97] showed a power law relationship between the number of model parameters and its performance, they did not consider pretraining tokens or

corpus data size. Hoffmann et al. [40] also reached the same conclusion but recommended that large models should be trained for many more training tokens. Specifically, given a 10× increase in computational budget, Kaplan et al. [97] suggest that the model size should increase 5.5× while the number of training tokens should only increase 1.8×. Instead, Chinchilla [40] finds that, as the computation budget increases, model size and the number of training tokens (training data) should be scaled in approximately equal proportions. Although large models achieved better performances, Chinchilla on the other hand demonstrated how, for a given computation budget, the best performance is achieved by smaller models trained on larger or more data as compared to large models. For instance, although LLaMA had 13 B parameters, it outperformed GPT-3 with 175 B on most benchmarks despite being ten times smaller. Chinchilla helps to answer how, given a fixed computational budget, one should balance model size and the number of training tokens.

With one-fourth fewer parameters and four times more data than Gopher, Chinchilla could significantly outperform Gopher on a large range of downstream evaluation tasks. Not only does Chinchilla outperform its much larger counterpart, Gopher, but, because of its smaller model size, it uses less computing for finetuning and inference, which reduces finetuning and inference costs considerably and greatly facilitates downstream usage on smaller hardware. Although [40] it determines how to scale the model size and dataset for a given computation budget, it disregards the inference budget, which is crucial since the preferred model is the one that is fastest at inference and not at training. As per [57], a smaller model trained longer is cheaper at inference. For example, Falcon-40B requires 70 GB of GPU memory to make inferences, whereas Falcon-7B needs only 15 GB, making inference and finetuning accessible even on consumer hardware.

Additionally, although [40] recommended training a 10 B model on 200 B tokens, [57] demonstrated how the performance of a model with 7 B parameters continued to improve even after it was trained on 1 T tokens. Furthermore, unlike Chinchilla, PaLM, or GPT-3, LLaMA demonstrated how it can train models and achieve SOTA performance using publicly available datasets without relying on proprietary and inaccessible datasets. WeLM [60], a Chinese LM, demonstrated how, by carefully cleaning, balancing, and scaling up the training data size, WeLM outperformed models with similar or larger sizes. For instance, on zero-shot evaluations, it matched the performance of ERNIE 3.0 Titan, which is 25× larger.

7.3. Model Depth (Network Layers)

In LLMs, network width is captured by the parameter size (hidden representation dimension), whereas network depth is the number of self-attention layers. Previous studies have indicated that increasing the network depth is the same as increasing the network representation. However, recent studies, such as [98], confirm the contrary. For instance, deepening is not favorable over widening for smaller network sizes. That is, when the width of the deeper network is not large enough, it cannot use its excess layers efficiently. Meanwhile, the transition into depth efficiency is clearly demonstrated when the network width is increased. It was shown in [98] that the transition between depth-efficiency and depth-inefficiency regimes exponentially depended on the network's depth. From a certain network width onwards, increasing the network depth does help improve efficiency. However, if the depth is increased with the network width, then it leads to efficiency. So, first, the width of the network must be chosen appropriately to leverage the full extent of the power brought by the depth of the network. For a given parameter budget, there is an optimal depth. So, for the same parameter budget, the deeper network performs better.

As per the proposed theory in [98], the optimal depth for GPT3's 175 B parameters should have been 80 layers instead of 96. As per [98], Jurassic-1 [31] used 76 layers for the 178 B parameter model and found a significant gain in runtime performance. Using the same hardware configuration compared to GPT3, Jurassic-1 had 1.5% speedup per iteration and 7% and 23% gain in batch inference and text generation. Also, by shifting computation

resources from depth to width, more operations can be performed in parallel (width) rather than sequentially (depth).

Additionally, Ref. [97] mainly focused on the upstream (pretraining) loss, whereas [62] found that scaling laws differ in upstream and downstream environments. For instance, in addition to model size, model shape matters for downstream tasks, and it is recommended to increase the model's depth (DeepNarrow strategy) before uniform scaling of any other dimensions. This demonstrates that such redesigned models were able to achieve a similar performance to the T5-base model but with 50% fewer parameters and 40% faster training.

7.4. Architecture—Parallelism

Large models often require a great deal of storage space to store the parameters. For instance, storing 178 B parameters requires more than 350 GB of memory with half-precision. As stated in [33], as the model size grows beyond 10 B parameters, it becomes difficult to train the model. To store [33] model of 200 B parameters, 750 GB space is required. Additionally, as gradients and optimizer states are required for updating the parameters, the model demands more memory during its training.

As large GPUs available today have a memory of around 80 GB, additional space is required to store the optimizer's state and intermediate calculations used during backpropagation. As a result, training must be distributed across hundreds of nodes, each with multiple GPUs, which might result in a communication bottleneck. In order to use the nodes efficiently, different parallelization strategies highlighted in Figure 6 (such as data, model, and pipeline) are used to achieve higher end-to-end throughput. Below, we discuss each of these approaches.

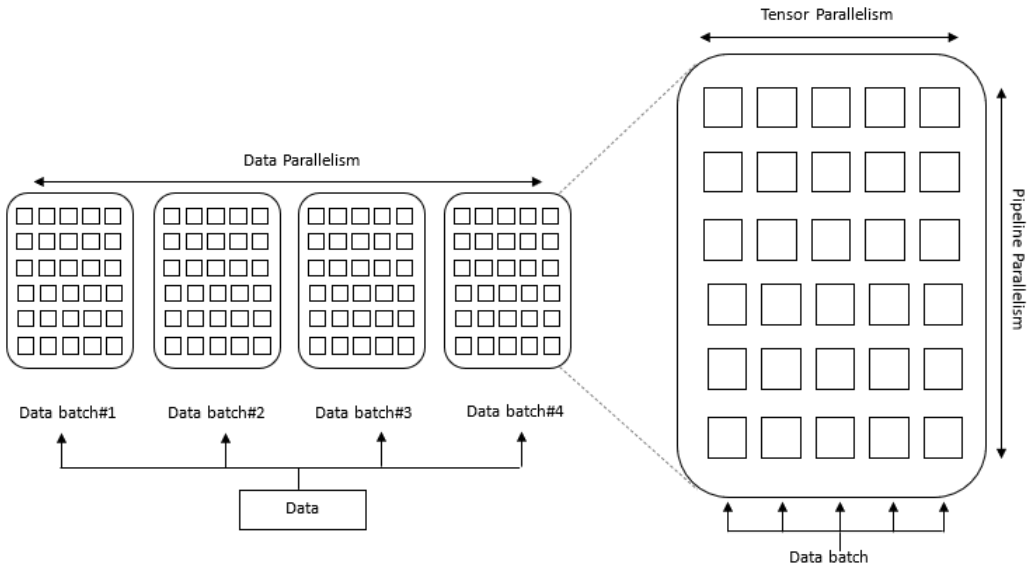

Figure 6. 3D parallelism.

7.4.1. Data Parallelism

In this data parallelism approach, the training batches are partitioned across the devices, followed by the synchronization of gradients from different devices before executing the optimizer step.

7.4.2. Tensor Parallelism (Op-Level Model Parallelism)

Each layer is partitioned across devices within a node. This approach reduces memory consumption by slicing the parameters and activating memory. However, to keep the distributed tensor layouts consistent, it does incur additional communication overheads.

7.4.3. Pipeline Parallelism

Pipeline parallelism splits the model layers among multiple nodes. Each node is one stage in the pipeline, which receives input from the previous stage and sends results to the next stage. Here, the layers are partitioned into stages. These states are placed on different devices. As each device holds only a subset of the total layers of the model, this approach is beneficial from a memory perspective since communications happen only at the boundaries of stages.

8. LLM Challenges

Language models can generate biased outputs of misinformation and be used maliciously. Large language models reproduce and might amplify existing biases in the training data, generating toxic or offensive content. During training, as the language models absorb biases and toxicity expressed in the text, they are prone to replicating them. This growing field aims to build benchmarks to concretely evaluate these harms, particularly around stereotypes, social bias, and toxicity. Making progress on these issues is challenging since well-intended intervention might have side effects on LM behavior. Below, we discuss the prominent benchmarks that are proposed to overcome these LLM challenges.

8.1. Toxic Content

Language models are capable of generating toxic language—including insults, hate speech, profanities, and threats. A model can generate a very large range of toxic content, making a thorough evaluation challenging. Language models are trained to reproduce their input distribution (and not to engage in conversation), so the trend is that toxicity increases with the model scale. Several recent works have considered the RealToxicityPrompts benchmark [99] as an indicator of how toxic their model is.

To measure toxicity, Ref. [99] proposed the RealToxicityPrompts benchmark. RealToxicityPromptsdataset is used to evaluate the tendency of LLM models to respond with toxic language. In LLaMA, toxicity increased with the size of the model. RealToxicityPrompts is quite a straightforward stress test: the user utters a toxic statement to see how the system responds. Some LLMs, for example, OPT, even when provided with a relatively innocuous prompt, have a high propensity to generate toxic language.

SaferDialogues [100] and Safety Bench [101] are two benchmarks that are used to test dialogue safety evaluations. SaferDialogues measures the ability of the model to recover from explicit safety failures, usually in the form of apologizing or recognizing its mistake. In contrast, Safety Bench Unit Tests measure how unsafe a model's response is and classify them as Safe, Realistic, Unsafe, and Adversarial.

8.2. Hallucination

LLMs are said to hallucinate when they generate information that is fake or incorrect. The hallucination can either be intrinsic or extrinsic. In intrinsic hallucination, the model generates information that contradicts the content of the source text. In contrast, the generated content cannot be contradicted or supported by the source text in extrinsic hallucination. There are various reasons why a model can hallucinate or generate fake information during inference. For instance, if the model misunderstands the information or facts provided in the source text, it can lead the model to hallucinate. So, to be truthful, the model should have reasoning ability to correctly understand the information from the source text. The other reason why LLMs can generate false information is when the provided contextual information conflicts with the parametric knowledge acquired during pretraining. Additionally, it is observed that models have parametric knowledge bias, where the model assigns more importance to the knowledge acquired during pretraining over the provided contextual information.

Also, teacher forcing is used during pretraining, where the decoder is conditioned on the ground truth prefix sequences to predict the next token. However, such a teacher forcing technique is missing during the inference, and such a discrepancy can also make

a model hallucinate. Several techniques have been proposed to detect hallucinations in LLMs, such as

1. sample not one but multiple outputs and check the information consistency between them to check which statements are factual and which are hallucinated;
2. validate the correctness of the model output by relying on and using external knowledge source;
3. check if the generated named entities or <subject, relation, object> tuples appear in the ground truth knowledge source or not, etc.

Benchmarks such as TruthfulQA [102] have been developed to measure the truthfulness of language models. This benchmark can evaluate the risks of a model to generate misinformation or false claims that mimic popular misconceptions and false beliefs. It was observed in [102] that, generally, the largest models were the least truthful, so scaling up the model size increased performance but was less promising in improving the model's truthfulness.

8.3. Biases

Gopher [39] studies gender and occupation bias via two different evaluations. First, it measures the probability of gendered words following different occupation contexts. Second, it evaluates the Winogender coreference resolution dataset [103], where similar coreference accuracy across different pronouns indicates less gender bias. To measure "Gender Word Probability" or how probable different gender words are in different occupation contexts, Gopher inputs an occupation prompt like "The occupation was a" into the model and computes a gender bias metric by comparing the probabilities of the prompt being followed by either male or female gendered terms.

CrowSPairs [104] is another crowdsourced benchmark dataset that allows measuring biases in nine categories: gender, religion, sexual orientation, socioeconomic status, race/color, disability, age, nationality, and physical appearance. Additionally, the StereoSet [105] dataset is used to measure stereotypical bias across four categories: profession, gender, religion, and race.

8.4. Cost and Carbon Footprints

As stated in CPM-2 [34], the cost of using pretrained language models increases with the growth of model sizes. The cost consists mainly of three parts.

1. Computational cost for pretraining: large language models require thousands of GPUs with several weeks of pretraining.
2. Storage cost for finetuned models: a large language model usually takes hundreds of gigabytes (GBs) to store, and as many model copies as the number of downstream tasks need to be stored.
3. Equipment cost for inference: it is expected to use multiple GPUs to infer a large language model.

So, as the model size increases, they become hard to use with limited computational resources and unaffordable for most researchers and institutions.

Furthermore, the pretraining phase of large language models consumes massive energy responsible for carbon dioxide emissions. The formulas used in LLaMA to estimate the Watt hour (Wh) and carbon emissions are listed in Equations (6) and (7), where 0.385 in Equation (7) is the US national average carbon intensity factor (0.385 kg $CO2eq$/KWh) and PUE represents power usage effectiveness.

$$Wh = (GPU - h) \times (GPU\ power\ consumption) \times (PUE) \tag{6}$$

$$tCO2eq = MWh \times 0.385 \tag{7}$$

As stated in LLaMA [57], carbon emission also depends on the data center's location used during pretraining of the network. For instance, BLOOM uses a grid that emits 0.057 kg $CO2eq$/KWh, leading to $27tCO2eq$, and OPT uses a grid that emits 0.231 kg

$CO2eq$/KWh, leading to $82tCO2eq$. As stated in [106], a couple of factors are involved in computing the electricity required to run an NLP model, such as algorithm, program, number of processors running the program, speed and power of those processors, a data center's efficiency in delivering power and cooling the processors, and the energy supply mix (renewable, gas, or coal). Cloud data centers can also be $1.4 - 2X$ more energy-efficient than typical data centers. A more detailed and granular formula stated in Equation (8) was presented in [106] that captures the carbon footprint of an NLP model:

$$\begin{aligned} Footprint = (&electrical\ energy_{\ train} + \\ = +\ &queries \times electrical\ energy_{\ inference}) \\ &\times CO2e_{\ datacenter}/KWh \end{aligned} \tag{8}$$

To decrease the footprint of training, an ML researcher should pick the DNN model, the processor, and the data center carefully. The above Equations (6) and (7) can be restated in terms of energy consumption and CO_2 emissions as Equations (9) and (10) below.

$$\begin{aligned} KWh = Hours\ to\ train \times Number\ of\ Processors \times \\ Average\ Power\ per\ Processor \times PUE \div 1000 \end{aligned} \tag{9}$$

$$tCO2e = KWh \times kg\ CO2e\ per\ KWh \div 1000 \tag{10}$$

To address the cost and carbon footprint problems, there is a need to improve the energy efficiency of algorithms, data centers, software, and hardware involved in implementing NLP models. Emphasis should be placed on reducing carbon footprint by building more efficient LLMs. For example, OPT [49] is comparable to GPT-3 and requires only 1/7th of the carbon footprint to develop.

Ref. [106] also recommends three suggestions that could eventually help to reduce the CO2e footprint:

1. report energy consumed and CO2e explicitly;
2. reward improvements in efficiency as well as traditional metrics at ML conferences;
3. to help everyone understand its cost, include the time and number of processors used during training.

As highlighted in [106], large but sparsely activated DNNs can consume <1/10th the energy of large, dense DNNs without sacrificing accuracy despite using as many or even more parameters.

8.5. Open-Source and Low-Resource Aspects

The costs of training LLMs are only affordable for well-resourced organizations. Furthermore, until recently, most LLMs were private and not publicly released. As a result, most of the research community is yet to be included in developing LLMs. Language-specific language models other than English are limited in availability. Very few non-English LMs, such as [33,36], are available in the market. So, there are many untapped non-English resources available on the internet that need to be explored. More work is required to accommodate low-resource and non-English languages regarding LLMs. Furthermore, the impact of increasing the proportion of multilingual data on multilingual and cross-lingual tasks needs to be explored.

9. Future Directions and Development Trends

LLMs have set the stage for a paradigm shift in developing future software applications. Also, LLMs have the potential to disrupt many well-established businesses. To address LLMs' full potential, in this section, we attempt to describe their future directions, possible development trends, and their unrealized utility. Although we enumerate these directions and trends under different facets to facilitate elucidation, there is a strong interconnectedness among the facets.

9.1. Interpretability and Explainability

An LLM's ability to explain its decisions and predictions is crucial to promote trust, accountability, and widespread acceptance. The current research targets methods that can explain the model's decisionmaking process and inner workings in a format understandable to humans. The approaches we discuss below originated in the machine learning domain. They need to evolve to serve the LLMs context.

Some architectures are inherently interpretable. For example, decision trees, rule-based models, or sparse models facilitate understanding a model's decision in a transparent and human-understandable format. More research in this area is critical for advancing LLM applications. Using the LLM's attention mechanism, we can highlight important parts of the input data that contributed to the model's predictions. Attention weights indicate the model's focus and thus serve as a means for interpretability.

Another approach involves extracting human-readable rules or generating post hoc explanations that explain model predictions in natural language or other more straightforward representations. Creating simpler proxy or surrogate models that approximate the behavior of complex original models is another approach to improving interpretability. LIME (local interpretable model-agnostic explanations) or SHAP (SHapley Additive exPlanations) are approaches for developing feature importance and attribution. This helps to attribute the model's predictions to specific input features. Salience maps and heatmaps (i.e., gradient-based visualization) also help to highlight essential regions in the input data that influence predictions. Another approach involves extracting human-readable rules or generating post hoc explanations. Investigating methods to provide certified or verified explanations guarantees the reliability of explanations.

Developing interactive tools that enable users to interact with the model at various levels of granularity can be used to provide user-centric explanations. This is akin to the *drill-down* and *roll-up* features of online analytical processing (OLAP) in data warehousing applications. Disclosing a model's capabilities, biases, and potential errors to users is a required step toward emphasizing the importance of ethical considerations and transparency. Lastly, educating users about model capabilities and limitations and providing guidance on interpreting model outputs is mandatory for advancing LLMs.

9.2. Fairness

Bias and fairness, if not adequately addressed, pose serious societal implications in the form of biased language generation and its impact on some segments of society. Basis can creep into LLMs from several sources discussed below. The first source of bias, *dataset bias*, stems from the datasets that were used to train the LLMs. If the datasets contain biases related to race, gender, religion, or socioeconomic status, the models inherit and amplify them.

Underrepresentation or misrepresentation of certain groups in the training data can lead to *representation bias* and biased language generation. The LLM developers should have checks and balances to ensure that all perspectives are adequately represented in the datasets. Otherwise, the model will produce inaccurate or skewed output for underrepresented groups. If the training data contain stereotypes, models amplify stereotyping and perpetuate prejudices. Fairness across demographics is a complex challenge but essential for advancing LLMs.

Centextual bias stems from the context in which the language models are used. This poses severe and negative implications in applications such as recommender systems, employee hiring and promotions, clustering, and sentiment analysis. The model evaluation metrics and benchmarks used in traditional machine learning are inadequate to capture bias in LLMs. Comprehensive evaluation methods are needed to consider various aspects of bias in LLMs. A multifaceted approach is required to address bias and fairness issues in LLMs. Approaches to data curation, model development, evaluation strategies, and ethical issues need to be reexamined for their suitability for LLMs. Mitigating biases in the datasets using debiasing approaches such as modifying loss functions, altering training data distributions, and adversarial training requires LLM-contextualized research.

9.3. Robustness and Adversarial Attacks

LLMs are susceptible to adversarial attacks. Small but carefully crafted perturbations can cause model misinterpretation. Addressing these issues is critical for ensuring the reliability and trustworthiness of LLMs. Ensuring consistent performance under perturbations requires eliminating susceptibility to adversarial manipulation. Mitigation approaches include input preprocessing and transformation, adversarial training, robust optimization techniques, adversarial example detection, defensive distillation, model ensembling, adaptive adversarial training and transferability analysis, adversarial attack-aware training data augmentation, certified robustness, explainable robustness mechanisms, and benchmarking and evaluation metrics.

In the input preprocessing and transformation approach, certain transformations are applied to the datasets to make the models robust to perturbations. For example, input denoising or transformation-based defenses modify inputs to remove adversarial perturbations. In adversarial training, the datasets are augmented with adversarial samples. This enhances a model's resilience to adversarial attacks. Robust optimization techniques, such as adversarial regularizations, modify the training objective functions to make the models more robust against adversarial perturbations.

Adversarial example detection involves methods to detect and flag adversarial examples during model inference. Techniques for this task include input reconstruction, uncertainty estimation, and anomaly detection. Defensive distillation and model ensembles combine predictions from multiple models to mitigate the impact of adversarial attacks. Also, ensembling diverse models reduces vulnerability to specific attack strategies. The adaptive adversarial training and transferability analysis approach employs adaptive adversarial training. Adversarial examples are dynamically generated during training to enhance a model's robustness. Analyzing the transferability of attacks across models provides insights into developing more universally robust defenses.

In adversarial attack-aware training data augmentation, the training data are enhanced with adversarial attack-aware data. Certified robustness methods provide formal guarantees on the model's robustness against certain kinds of adversarial attacks. This emerging research area offers provable bounds on the model's performance under attack. If the model design incorporates explainable robustness mechanisms, then examining how the model handles adversarial attacks is feasible. Lastly, the availability of benchmarks and evaluation metrics contextualized to adversarial attacks helps to compare the effectiveness of different models and techniques. The techniques mentioned above originally came from the traditional machine learning domain. Research is needed to adapt these to the LLMs context. Moreover, research is needed to develop new approaches to adversarial attacks given the unique characteristics of LLMs.

9.4. Multimodal LLMs

LLMs currently primarily deal with large amounts of text data. Research is underway to enhance LLMs with image data. However, integrating diverse data modalities, including text, images, graphics, audio, and videos seamlessly, is required to realize the full potential of LLMs. With the ubiquity of camera-equipped mobile devices, more and more images and videos are produced every day. Some estimate that about 3.7 million new videos are uploaded to YouTube daily. For LLMs to comprehensively understand the content in diverse media, generating content that includes all the relevant elements from diverse media is essential. This is a challenging task and requires groundbreaking research. The current research in this direction includes multimodal preprocessing and feature extraction, fine-grained multimodal representations, spatiotemporal understanding in videos, semantics and contextual understanding, multimodal fusion architectures, cross-modal pretraining and transfer learning, alignment and cross-modal correspondence, real-time multimodal inference, and multimodal pretraining datasets and benchmarks.

The greatest challenge for realizing multimodal LLMs is in developing effective preprocessing techniques and feature extraction methods specific to each modality. The next step

is to integrate the different modalities within the model architecture. Creating fine-grained multimodal representations involves capturing the complex relationships between diverse modalities. One approach to this is to learn joint/multimodal contextual embeddings. Spatiotemporal understanding in videos involves extracting temporal relationships, detecting motion patterns, and synthesizing spatial information. An LLM's ability to understand semantics and context across diverse modalities is essential for generating contextually relevant multimodal outputs.

New architectures for LLMs are required to integrate information from multiple modalities. These multimodal fusion architectures require integrating cross-modal embeddings with attention mechanisms. Advances in cross-modal pretraining are required for learning shared representations across modalities. Also, transfer learning from pretrained models is required for better performance on downstream multimodal tasks. Approaches for aligning information across modalities require new research investigations. For example, cross-modal alignment through attention or similarity measures is required to establish the correspondences between elements in different modalities. Some downstream applications require efficient processing of multimodal inputs. For this scenario, real-time multimodal inference is required. Lastly, the availability of curated large-scale multimodal datasets and associated benchmarks for evaluating multimodal models is essential to advance multimodal LLMs.

9.5. Energy Efficiency and Environmental Impact

Training LLMs requires tremendous computing power. Minimizing environmental impact through energy efficiency is a paramount concern in advancing LLMs. There are several facets to achieving energy efficiency, as detailed below. Developing energy-efficient algorithms for training LLMs is a coveted goal. Such algorithms will require faster convergence or fewer computational resources through adaptive learning rate schedules, low-precision training, and gradient checkpointing.

Another promising area of research is designing specialized hardware accelerators optimized for LLM training and inference. Such hardware optimization and accelerators will significantly contribute to efficient computation and thus reduce energy consumption. Related to optimized hardware accelerators is model architecture optimization. Topics to be researched in this direction include model structure optimization, reducing redundant parameters, and developing sparse models. Pruning and sparsity induction through identifying and eliminating redundant or less significant parameters contribute to creating leaner models. Transfer learning and few-shot learning methods reduce the need for extensive training of LLMs on new tasks or domains. Advances in this area can significantly reduce energy requirements via better model generalization with less training. Energy consumption can also be optimized by employing energy-aware training and inference strategies, which include adaptive precision tuning, dynamic pruning, and model scaling.

Quantization of model weights and compression schemes contributes to the reduced computational overhead of LLMs. For example, knowledge distillation is a technique that helps to decrease the model's memory and computational requirements. Research is needed in lifecycle assessment and environmental impact to inform researchers and provide guidelines and best practices for developing and using LLMs. Such research will document the environmental impact of LLMs by quantifying the carbon footprint and suggestions for footprint reduction. Data center efficiency is pivotal in developing LLMs and deploying downstream applications. Supporting data center efficiency initiatives, including renewable energy sources, is critical. Lastly, collaboration between academia, industry, and policymakers is needed to share best practices, application frameworks, and tools for energy-aware LLMs.

9.6. Different Languages and Domains

The current LLM research and development are primarily confined to the English language. According to Ethnologue, there are 7168 living languages in the world. A

language becomes endangered when its users begin to teach and speak a more dominant language to their children than their native language. Over 3045 languages are endangered today. LLMs can play a pivotal role in preserving and promoting all world languages. Low-resource languages need more curated and annotated datasets in machine-readable format to train LLMs. Also, some languages are spoken only without written counterparts. For such cases, speech-to-text transcription is required. To ensure linguistic inclusivity, researchers are investigating the following strategies.

Data augmentation and synthesis techniques are investigated to create synthetic data to enlarge the training datasets. Some techniques include back-translation, paraphrasing, and data generation through linguistic rules. Another approach to deal with low linguistic resources is to leverage transfer learning. This involves pretraining models on high-resource languages (e.g., English) and transferring knowledge to low-resource languages. As multilingual LLMs share model parameters across languages, this helps in improving performance for low-resource languages. Also, developing models capable of zero-shot or few-shot learning using high-resource languages enables them to perform tasks in low-resource languages with minimal or no annotated data. For example, methods such as meta-learning and cross-lingual transfer target this goal. However, the effectiveness of such methods remains to be seen.

Semi-supervised and self-supervised learning approaches can be leveraged for labeled and unlabeled data for model training in low-resource contexts. Unlabeled data can be effectively utilized using techniques such as self-training or pseudo-labeling. Another approach to help low-resource situations is to design language-specific architectures and models that are tailored to the linguistic characteristics of low-resource languages. Adapting LLMs to specific linguistic features and morphological structures improves their effectiveness. Community involvement in building datasets for low-resource languages through collaboration and crowdsourcing is vital. Resource sharing and knowledge transfer between linguistic communities in the form of datasets, linguistic tools, and methodologies will immensely help low-resource languages.

Once LLMs are developed for low-resource languages, they can aid in preserving and promoting them. For example, LLMs can help in documentation, translation, education, and cultural preservation. LLMs can be leveraged to document low-resource and endangered languages by analyzing written texts and transcribing spoken language. LLMs will also enable the creation of digital archives, cataloging historical texts, and documenting stories and folklore in native languages. More importantly, LLMs can be used to support indigenous communities by providing tools that assist in preserving their languages and traditions. These activities help to preserve linguistic heritage that might otherwise be lost.

LLMs can translate between high-resource and low-resource languages, making the information more accessible and fostering communication across linguistic barriers. Also, LLMs can be used to support language revitalization efforts by providing language learning resources and generating teaching materials. Furthermore, LLMs will aid in developing language-learning applications for low-resource and endangered languages. LLMs will provide language researchers with advanced tools and resources for linguistic analysis, corpus creation, and comparative studies on a scale that was infeasible before. Furthermore, LLMs will foster collaborative language preservation by facilitating collective work and communication across language barriers. LLMs will facilitate technology democratization by developing inclusive technologies to communicate with users in their native languages and cultural contexts.

9.7. Privacy-Preserving Models

The challenge for privacy-preserving models is ensuring user data privacy while guaranteeing model performance and utility. This requires a multipronged approach, as outlined below. Privacy-preserving techniques such as anonymization during data preprocessing help to protect sensitive information before using it for model training. Another approach is to perform computations directly on user devices to minimize data

transfer and centralization. This reduces the privacy risks associated with data transmission. Using trusted execution environments (TEEs) such as Intel SGX or ARM TrustZone secures computations within isolated environments, which protects user data from unauthorized access. Another way to preserve user privacy is by designing privacy-preserving metrics and evaluation methodologies.

Federated model training involves training models across decentralized devices or servers without exchanging raw data. Privacy is preserved by aggregating model updates while keeping the user data local. Differential privacy is an approach to privacy preservation where a noise or perturbation is added to the data before the training process. This prevents the extraction of sensitive information from individual data samples as the model does not memorize specific data points. Techniques such as *homomorphic encryption* allow computation on encrypted data without decrypting them. This approach preserves data privacy throughout the computation process. Protocols such as secure multiparty computation (MPC) enable multiple parties to compute a function while keeping their inputs private. This paves the way for collaborative model training without sharing raw data.

The model aggregation and ensemble approach aggregates predictions from multiple models without sharing individual user data. This approach enables leveraging the collective knowledge of models while preserving user privacy. The development of privacy-preserving metrics and evaluation methodologies guarantees that model evaluation processes do not compromise user privacy. Lastly, compliance with legal and ethical frameworks like GDPR protects users' privacy rights.

9.8. Continual Learning and Adaptability

For LLMs to have excellent utility, they must continually learn from new data, adapt to changing contexts, and retain previously learned knowledge. Approaches to accomplishing these goals require research investigations along multiple directions. First, the development of algorithms and methodologies for incremental learning is needed to enable models to learn new information without forgetting already learned information. Replay-based methods, regularization, and parameter isolation are some techniques that need further investigation.

LLMs with external memory components like *attentional interfaces* help to retain previously learned information. These are referred to as memory-augmented architectures. LLMs need a mechanism to prioritize new information while preserving old knowledge to realize continual learning. Using *adaptive learning rate schedules*, a model can dynamically adjust learning rates for different parts of the model or specific examples. Task-agnostic representations help LLMs learn more generalized features that transfer across different tasks. Learning task-agnostic representations helps in continual learning as models can adapt to new tasks without drastic retraining.

Regularization methods encourage model parameters to remain stable and selectively update them for new information, which aids in continual learning. For example, elastic weight consolidation (EWC) and synaptic intelligence help models to retain learned information. As noted earlier, meta-learning and few-shot learning approaches enable models to adapt quickly to new tasks or domains with minimal data. Finetuning the models on new data while leveraging pretrained representations helps in adaptation. Another approach to adaptation is through ensemble models, which combine learning paradigms such as episodic memory systems and continual learning techniques.

9.9. Ethical Use and Societal Impact

Several key strategies are required to address issues regarding the ethical use of LLMs. Ethical guidelines and frameworks are needed to guide LLMs' development, deployment, and operation. Language researchers, technologists, application developers, and policymakers need to come together to develop ethical guidelines and frameworks. More importantly, researchers and organizations should embrace the guidelines to ensure responsible development and deployment of LLM applications.

Responsible AII practices should integrate the principles of fairness, explainability, transparency, accountability, and privacy preservation into the development lifecycle of language models and downstream applications. LLMs have exacerbated the detection and mitigation of misinformation, harmful content, and hate speech. Content moderation strategies should be integral to operating LLMs and downstream applications. LLMs should be enhanced and continually monitored to avoid generating harmful content. Regular audits and impact assessments of LLMs should be conducted to identify biases, ensure ethical and regulatory compliance, and assess societal impacts.

9.10. Real-World Applications and Human–LLM Collaboration

Compared to developing and deploying traditional software applications, LLM downstream applications require additional considerations. Accurate identification and documentation of real-world use cases are critical since LLM models must be tailored through finetuning to address the use cases effectively. This requires a precise understanding of the goals, challenges, and specific requirements of application domains. The design of intuitive and user-friendly interfaces takes center stage to ensure seamless interaction between humans and LLM applications. User-centric design principles guarantee accessibility and ease of use for diverse users. Human-in-the-loop methodologies play a central role in designing LLM applications. The methodologies require human feedback to improve model performance and refine its outputs continually. Also, accessibility and inclusivity mechanisms via language support, assistive technologies, and diverse interaction modalities are critical to meeting diverse user needs.

10. Conclusions

This paper comprehensively studied different types of architecture, masking techniques, and phases that go into building language models. It explained in detail how the language models have transitioned from task-and-language-specific to task-and-language-agnostic. It also looked at LLMs through the lens of scalability and compared them based on parameters such as network depth, width, hardware, objectives, datasets, and corpus size used during pretraining. It elucidated different in-context, pretraining, and transfer learning strategies and their advantageous and disadvantageous applications or scenarios where they performed better. It also comprehensively analyzed different ways to scale and incorporate parallelism into the model to increase computational efficiency.

Furthermore, the article also sheds light on challenges encountered in LLMs, such as biases, toxic content, hallucination, privacy, cost and energy efficiency, adversarial attacks, and the social and environmental impact (in terms of carbon footprint). The article also lists possible future directions and development trends in the field of LLMs, which include interpretability, explainability, continual learning, adaptability, ethical use, fairness, robustness, and multimodal, multilingual, and multidomain aspects of LLMs. In future work, we plan to investigate the role that retrieval-augmented generation has to play in mitigating hallucination. Overall, the article empirically compared the existing trends and techniques and comprehensively analyzed where the field of LLMs currently stands.

Author Contributions: R.P. worked on Introduction, Pretraining, Transfer learning, In-Context Learning, Scalability and Conclusion sections, while V.G. worked on LLM Challenges, Development Trends, and Future Directions sections. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Harris, Z.S. Distributional structure. *Word* **1954**, *10*, 146–162. [CrossRef]
2. Brown, P.F.; Cocke, J.; Della Pietra, S.A.; Della Pietra, V.J.; Jelinek, F.; Lafferty, J.; Mercer, R.L.; Roossin, P.S. A statistical approach to machine translation. *Comput. Linguist.* **1990**, *16*, 79–85.
3. Salton, G.; Lesk, M.E. Computer evaluation of indexing and text processing. *J. ACM (JACM)* **1968**, *15*, 8–36. [CrossRef]
4. Jones, K.S. A statistical interpretation of term specificity and its application in retrieval. *J. Doc.* **1972**, *28*, 11–21. [CrossRef]
5. Salton, G.; Wong, A.; Yang, C.S. A vector space model for automatic indexing. *Commun. ACM* **1975**, *18*, 613–620. [CrossRef]
6. Tang, B.; Shepherd, M.; Milios, E.; Heywood, M.I. Comparing and combining dimension reduction techniques for efficient text clustering. In Proceeding of the SIAM International Workshop on Feature Selection for Data Mining, Newport Beach, CA, USA, 21 April 2005; pp. 17–26.
7. Hyvärinen, A.; Oja, E. Independent component analysis: Algorithms and applications. *Neural Netw.* **2000**, *13*, 411–430. [CrossRef]
8. Le, Q.; Mikolov, T. Distributed representations of sentences and documents. In Proceedings of the International Conference on Machine Learning, Montreal, QC, Canada, 8–13 December 2014; pp. 1188–1196.
9. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781.
10. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. *Adv. Neural Inf. Process. Syst.* **2013**, *26*. . [CrossRef]
11. Pennington, J.; Socher, R.; Manning, C.D. Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1532–1543.
12. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguist.* **2017**, *5*, 135–146. [CrossRef]
13. Vilnis, L.; McCallum, A. Word representations via gaussian embedding. *arXiv* **2014**, arXiv:1412.6623.
14. Athiwaratkun, B.; Wilson, A.G. Multimodal word distributions. *arXiv* **2017**, arXiv:1704.08424.
15. Melamud, O.; Goldberger, J.; Dagan, I. context2vec: Learning generic context embedding with bidirectional lstm. In Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, Berlin, Germany, 11–12 August 2016; pp. 51–61.
16. McCann, B.; Bradbury, J.; Xiong, C.; Socher, R. Learned in translation: Contextualized word vectors. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.
17. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep Contextualized Word Representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, New Orleans, LA, USA, 1–6 June 2018; Volume 1, pp. 2227–2237.
18. Howard, J.; Ruder, S. Universal language model fine-tuning for text classification. *arXiv* **2018**, arXiv:1801.06146.
19. Dong, L.; Yang, N.; Wang, W.; Wei, F.; Liu, X.; Wang, Y.; Gao, J.; Zhou, M.; Hon, H.W. Unified language model pre-training for natural language understanding and generation. *Adv. Neural Inf. Process. Syst.* **2019**, *32*.
20. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078.
21. Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to sequence learning with neural networks. *Adv. Neural Inf. Process. Syst.* **2014**, *27*.
22. Scao, T.L.; Fan, A.; Akiki, C.; Pavlick, E.; Ilić, S.; Hesslow, D.; Castagné, R.; Luccioni, A.S.; Yvon, F.; Gallé, M.; et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv* **2022**, arXiv:2211.05100.
23. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, arXiv:1409.0473.
24. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*.
25. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* **2020**, *21*, 5485–5551.
26. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
27. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. 2018. Available online: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf (accessed on 1 February 2024).
28. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
29. Brown. T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
30. Guu, K.; Lee, K.; Tung, Z.; Pasupat, P.; Chang, M. Retrieval augmented language model pre-training. In Proceedings of the International Conference on Machine Learning, Online, 9–11 September 2020; pp. 3929–3938.
31. Lieber, O.; Sharir, O.; Lenz, B.; Shoham, Y. Jurassic-1: Technical details and evaluation. *White Pap. AI21 Labs* **2021**, *1*, 9.
32. Xue, L.; Constant, N.; Roberts, A.; Kale, M.; Al-Rfou, R.; Siddhant, A.; Barua, A.; Raffel, C. mT5: A massively multilingual pre-trained text-to-text transformer. *arXiv* **2020**, arXiv:2010.11934.
33. Zeng, W.; Ren, X.; Su, T.; Wang, H.; Liao, Y.; Wang, Z.; Jiang, X.; Yang, Z.; Wang, K.; Zhang, X.; et al. Pangu-α: Large-scale autoregressive pretrained Chinese language models with auto-parallel computation. *arXiv* **2021**, arXiv:2104.12369.

34. Zhang, Z.; Gu, Y.; Han, X.; Chen, S.; Xiao, C.; Sun, Z.; Yao, Y.; Qi, F.; Guan, J.; Ke, P.; et al. Cpm-2: Large-scale cost-effective pre-trained language models. *AI Open* **2021**, *2*, 216–224. [CrossRef]
35. Wu, S.; Zhao, X.; Yu, T.; Zhang, R.; Shen, C.; Liu, H.; Li, F.; Zhu, H.; Luo, J.; Xu, L.; et al. Yuan 1.0: Large-scale pre-trained language model in zero-shot and few-shot learning. *arXiv* **2021**, arXiv:2110.04725.
36. Kim, B.; Kim, H.; Lee, S.W.; Lee, G.; Kwak, D.; Jeon, D.H.; Park, S.; Kim, S.; Kim, S.; Seo, D.; et al. What changes can large-scale language models bring? intensive study on hyperclova: Billions-scale korean generative pretrained transformers. *arXiv* **2021**, arXiv:2109.04650.
37. Du, N.; Huang, Y.; Dai, A.M.; Tong, S.; Lepikhin, D.; Xu, Y.; Krikun, M.; Zhou, Y.; Yu, A.W.; Firat, O.; et al. Glam: Efficient scaling of language models with mixture-of-experts. In Proceedings of the International Conference on Machine Learning, Baltimore, MD, USA, 17–23 July 2022; pp. 5547–5569.
38. Sun, Y.; Wang, S.; Feng, S.; Ding, S.; Pang, C.; Shang, J.; Liu, J.; Chen, X.; Zhao, Y.; Lu, Y.; et al. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv* **2021**, arXiv:2107.02137.
39. Rae, J.W.; Borgeaud, S.; Cai, T.; Millican, K.; Hoffmann, J.; Song, F.; Aslanides, J.; Henderson, S.; Ring, R.; Young, S.; et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv* **2021**, arXiv:2112.11446.
40. Hoffmann, J.; Borgeaud, S.; Mensch, A.; Buchatskaya, E.; Cai, T.; Rutherford, E.; Casas, D.D.L.; Hendricks, L.A.; Welbl, J.; Clark, A.; et al. Training compute-optimal large language models. *arXiv* **2022**, arXiv:2203.15556.
41. Li, Y.; Choi, D.; Chung, J.; Kushman, N.; Schrittwieser, J.; Leblond, R.; Eccles, T.; Keeling, J.; Gimeno, F.; Dal Lago, A.; et al. Competition-level code generation with alphacode. *Science* **2022**, *378*, 1092–1097. [CrossRef]
42. Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; Xiong, C. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv* **2022**, arXiv:2203.13474.
43. Zheng, Q.; Xia, X.; Zou, X.; Dong, Y.; Wang, S.; Xue, Y.; Wang, Z.; Shen, L.; Wang, A.; Li, Y.; et al. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv* **2023**, arXiv:2303.17568.
44. Wei, J.; Bosma, M.; Zhao, V.Y.; Guu, K.; Yu, A.W.; Lester, B.; Du, N.; Dai, A.M.; Le, Q.V. Finetuned language models are zero-shot learners. *arXiv* **2021**, arXiv:2109.01652.
45. Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. Training language models to follow instructions with human feedback. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 27730–27744.
46. Thoppilan, R.; De Freitas, D.; Hall, J.; Shazeer, N.; Kulshreshtha, A.; Cheng, H.T.; Jin, A.; Bos, T.; Baker, L.; Du, Y.; et al. Lamda: Language models for dialog applications. *arXiv* **2022**, arXiv:2201.08239.
47. Sanh, V.; Webson, A.; Raffel, C.; Bach, S.H.; Sutawika, L.; Alyafeai, Z.; Chaffin, A.; Stiegler, A.; Scao, T.L.; Raja, A.; et al. Multitask prompted training enables zero-shot task generalization. *arXiv* **2021**, arXiv:2110.08207.
48. Black, S.; Biderman, S.; Hallahan, E.; Anthony, Q.; Gao, L.; Golding, L.; He, H.; Leahy, C.; McDonell, K.; Phang, J.; et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv* **2022**, arXiv:2204.06745.
49. Zhang, S.; Roller, S.; Goyal, N.; Artetxe, M.; Chen, M.; Chen, S.; Dewan, C.; Diab, M.; Li, X.; Lin, X.V.; et al. Opt: Open pre-trained transformer language models. *arXiv* **2022**, arXiv:2205.01068.
50. Lewkowycz, A.; Andreassen, A.; Dohan, D.; Dyer, E.; Michalewski, H.; Ramasesh, V.; Slone, A.; Anil, C.; Schlag, I.; Gutman-Solo, T.; et al. Solving quantitative reasoning problems with language models. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 3843–3857.
51. Soltan, S.; Ananthakrishnan, S.; FitzGerald, J.; Gupta, R.; Hamza, W.; Khan, H.; Peris, C.; Rawls, S.; Rosenbaum, A.; Rumshisky, A.; et al. Alexatm 20b: Few-shot learning using a large-scale multilingual seq2seq model. *arXiv* **2022**, arXiv:2208.01448.
52. Zeng, A.; Liu, X.; Du, Z.; Wang, Z.; Lai, H.; Ding, M.; Yang, Z.; Xu, Y.; Zheng, W.; Xia, X.; et al. Glm-130b: An open bilingual pre-trained model. *arXiv* **2022**, arXiv:2210.02414.
53. Lin, X.V.; Mihaylov, T.; Artetxe, M.; Wang, T.; Chen, S.; Simig, D.; Ott, M.; Goyal, N.; Bhosale, S.; Du, J.; et al. Few-shot Learning with Multilingual Generative Language Models. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates, 7–11 December 2022; pp. 9019–9052.
54. Chowdhery, A.; Narang, S.; Devlin, J.; Bosma, M.; Mishra, G.; Roberts, A.; Barham, P.; Chung, H.W.; Sutton, C.; Gehrmann, S.; et al. Palm: Scaling language modeling with pathways. *arXiv* **2022**, arXiv:2204.02311.
55. Taylor, R.; Kardas, M.; Cucurull, G.; Scialom, T.; Hartshorn, A.; Saravia, E.; Poulton, A.; Kerkez, V.; Stojnic, R. Galactica: A large language model for science. *arXiv* **2022**, arXiv:2211.09085.
56. Chen, X.; Wang, X.; Changpinyo, S.; Piergiovanni, A.J.; Padlewski, P.; Salz, D.; Goodman, S.; Grycner, A.; Mustafa, B.; Beyer, L.; et al. Pali: A jointly-scaled multilingual language-image model. *arXiv* **2022**, arXiv:2209.06794.
57. Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. Llama: Open and efficient foundation language models. *arXiv* **2023**, arXiv:2302.13971.
58. Tay, Y.; Dehghani, M.; Tran, V.Q.; Garcia, X.; Wei, J.; Wang, X.; Chung, H.W.; Bahri, D.; Schuster, T.; Zheng, S.; et al. Ul2: Unifying language learning paradigms. In Proceedings of the Eleventh International Conference on Learning Representations, Virtual Event, 25–29 April 2022.
59. Biderman, S.; Schoelkopf, H.; Anthony, Q.G.; Bradley, H.; O'Brien, K.; Hallahan, E.; Khan, M.A.; Purohit, S.; Prashanth, U.S.; Raff, E.; et al. Pythia: A suite for analyzing large language models across training and scaling. In Proceedings of the International Conference on Machine Learning, Nashville, TN, USA, 10–12 July 2023; pp. 2397–2430.
60. Su, H.; Zhou, X.; Yu, H.; Chen, Y.; Zhu, Z.; Yu, Y.; Zhou, J. Welm: A well-read pre-trained language model for chinese. *arXiv* **2022**, arXiv:2209.10372.

61. Du, Z.; Qian, Y.; Liu, X.; Ding, M.; Qiu, J.; Yang, Z.; Tang, J. Glm: General language model pretraining with autoregressive blank infilling. *arXiv* **2021**, arXiv:2103.10360.
62. Tay, Y.; Dehghani, M.; Rao, J.; Fedus, W.; Abnar, S.; Chung, H.W.; Narang, S.; Yogatama, D.; Vaswani, A.; Metzler, D. Scale efficiently: Insights from pre-training and fine-tuning transformers. *arXiv* **2021**, arXiv:2109.10686.
63. Artetxe, M.; Bhosale, S.; Goyal, N.; Mihaylov, T.; Ott, M.; Shleifer, S.; Lin, X.V.; Du, J.; Iyer, S.; Pasunuru, R.; et al. Efficient large scale language modeling with mixtures of experts. *arXiv* **2021**, arXiv:2112.10684.
64. Conneau, A.; Khandelwal, K.; Goyal, N.; Chaudhary, V.; Wenzek, G.; Guzmán, F.; Grave, E.; Ott, M.; Zettlemoyer, L.; Stoyanov, V. Unsupervised cross-lingual representation learning at scale. *arXiv* **2019**, arXiv:1911.02116.
65. Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv* **2017**, arXiv:1701.06538.
66. Zoph, B.; Bello, I.; Kumar, S.; Du, N.; Huang, Y.; Dean, J.; Shazeer, N.; Fedus, W. St-moe: Designing stable and transferable sparse expert models. *arXiv* **2022**, arXiv:2202.08906.
67. Lepikhin, D.; Lee, H.; Xu, Y.; Chen, D.; Firat, O.; Huang, Y.; Krikun, M.; Shazeer, N.; Chen, Z. GShard: Scaling giant models with conditional computation and automatic sharding. *arXiv* **2020**, arXiv:2006.16668.
68. Fedus, W.; Zoph, B.; Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.* **2022**, *23*, 5232–5270.
69. Zhang, Z.; Han, X.; Liu, Z.; Jiang, X.; Sun, M.; Liu, Q. ERNIE: Enhanced language representation with informative entities. *arXiv* **2019**, arXiv:1905.07129.
70. Peters, M.E.; Neumann, M.; Logan, R.L., IV; Schwartz, R.; Joshi, V.; Singh, S.; Smith, N.A. Knowledge enhanced contextual word representations. *arXiv* **2019**, arXiv:1909.04164.
71. Zhou, W.; Lee, D.H.; Selvam, R.K.; Lee, S.; Lin, B.Y.; Ren, X. Pre-training text-to-text transformers for concept-centric common sense. *arXiv* **2020**, arXiv:2011.07956.
72. Xiong, W.; Du, J.; Wang, W.Y.; Stoyanov, V. Pretrained encyclopedia: Weakly supervised knowledge-pretrained language model. *arXiv* **2019**, arXiv:1912.09637.
73. Wang, X.; Gao, T.; Zhu, Z.; Zhang, Z.; Liu, Z.; Li, J.; Tang, J. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Trans. Assoc. Comput. Linguist.* **2021**, *9*, 176–194. [CrossRef]
74. Sun, T.; Shao, Y.; Qiu, X.; Guo, Q.; Hu, Y.; Huang, X.; Zhang, Z. Colake: Contextualized language and knowledge embedding. *arXiv* **2020**, arXiv:2010.00309.
75. Wang, R.; Tang, D.; Duan, N.; Wei, Z.; Huang, X.; Cao, G.; Jiang, D.; Zhou, M. K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv* **2020**, arXiv:2002.01808.
76. Tay, Y.; Wei, J.; Chung, H.W.; Tran, V.Q.; So, D.R.; Shakeri, S.; Garcia, X.; Zheng, H.S.; Rao, J.; Chowdhery, A.; et al. Transcending scaling laws with 0.1% extra compute. *arXiv* **2022**, arXiv:2210.11399.
77. Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; Gelly, S. Parameter-efficient transfer learning for NLP. In Proceedings of the International Conference on Machine Learning, Vancouver, BC, Canada, 13 December 2019; pp. 2790–2799.
78. Li, X.L.; Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv* **2021**, arXiv:2101.00190.
79. Liu, X.; Zheng, Y.; Du, Z.; Ding, M.; Qian, Y.; Yang, Z.; Tang, J. GPT understands, too. *AI Open* **2023**, *in press*. [CrossRef]
80. Lester, B.; Al-Rfou, R.; Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv* **2021**, arXiv:2104.08691.
81. Muennighoff, N.; Wang, T.; Sutawika, L.; Roberts, A.; Biderman, S.; Scao, T.L.; Bari, M.S.; Shen, S.; Yong, Z.X.; Schoelkopf, H.; et al. Crosslingual generalization through multitask finetuning. *arXiv* **2022**, arXiv:2211.01786.
82. Ziegler, D.M.; Stiennon, N.; Wu, J.; Brown, T.B.; Radford, A.; Amodei, D.; Christiano, P.; Irving, G. Fine-tuning language models from human preferences. *arXiv* **2019**, arXiv:1909.08593.
83. Wu, J.; Ouyang, L.; Ziegler, D.M.; Stiennon, N.; Lowe, R.; Leike, J.; Christiano, P. Recursively summarizing books with human feedback. *arXiv* **2021**, arXiv:2109.10862.
84. Stiennon, N.; Ouyang, L.; Wu, J.; Ziegler, D.; Lowe, R.; Voss, C.; Radford, A.; Amodei, D.; Christiano, P.F. Learning to summarize with human feedback. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 3008–3021.
85. Wang, Y.; Mishra, S.; Alipoormolabashi, P.; Kordi, Y.; Mirzaei, A.; Arunkumar, A.; Ashok, A.; Dhanasekaran, A.S.; Naik, A.; Stap, D.; et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv* **2022**, arXiv:2204.07705.
86. Iyer, S.; Lin, X.V.; Pasunuru, R.; Mihaylov, T.; Simig, D.; Yu, P.; Shuster, K.; Wang, T.; Liu, Q.; Koura, P.S.; et al. Opt-iml: Scaling language model instruction meta learning through the lens of generalization. *arXiv* **2022**, arXiv:2212.12017.
87. Chung, H.W.; Hou, L.; Longpre, S.; Zoph, B.; Tay, Y.; Fedus, W.; Li, E.; Wang, X.; Dehghani, M.; Brahma, S.; et al. Scaling instruction-finetuned language models. *arXiv* **2022**, arXiv:2210.11416.
88. Wang, Y.; Kordi, Y.; Mishra, S.; Liu, A.; Smith, N.A.; Khashabi, D.; Hajishirzi, H. Self-instruct: Aligning language model with self generated instructions. *arXiv* **2022**, arXiv:2212.10560.
89. Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H.P.D.O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. Evaluating large language models trained on code. *arXiv* **2021**, arXiv:2107.03374.
90. Zhao, Z.; Wallace, E.; Feng, S.; Klein, D.; Singh, S. Calibrate before use: Improving few-shot performance of language models. In Proceedings of the International Conference on Machine Learning, Virtual, 18–24 July 2021; pp. 12697–12706.

91. Liu, J.; Shen, D.; Zhang, Y.; Dolan, B.; Carin, L.; Chen, W. What Makes Good In-Context Examples for GPT-3? *arXiv* **2021**, arXiv:2101.06804.
92. Mosbach, M.; Pimentel, T.; Ravfogel, S.; Klakow, D.; Elazar, Y. Few-shot Fine-tuning vs. In-context Learning: A Fair Comparison and Evaluation. *arXiv* **2023**, arXiv:2305.16938.
93. Wang, T.; Roberts, A.; Hesslow, D.; Le Scao, T.; Chung, H.W.; Beltagy, I.; Launay, J.; Raffel, C. What language model architecture and pretraining objective works best for zero-shot generalization? In Proceedings of the International Conference on Machine Learning, PMLR, Online, 28–30 March 2022; pp. 22964–22984.
94. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q.V.; Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 24824–24837.
95. Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv* **2022**, arXiv:2203.11171.
96. Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B.; Borgeaud, S.; Yogatama, D.; Bosma, M.; Zhou, D.; Metzler, D.; et al. Emergent abilities of large language models. *arXiv* **2022**, arXiv:2206.07682.
97. Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T.B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; Amodei, D. Scaling laws for neural language models. *arXiv* **2020**, arXiv:2001.08361.
98. Levine, Y.; Wies, N.; Sharir, O.; Bata, H.; Shashua, A. Limits to depth efficiencies of self-attention. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 22640–22651.
99. Gehman, S.; Gururangan, S.; Sap, M.; Choi, Y.; Smith, N.A. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv* **2020**, arXiv:2009.11462.
100. Ung, M.; Xu, J.; Boureau, Y.L. Saferdialogues: Taking feedback gracefully after conversational safety failures. *arXiv* **2021**, arXiv:2110.07518.
101. Dinan, E.; Abercrombie, G.; Bergman, A.S.; Spruit, S.; Hovy, D.; Boureau, Y.L.; Rieser, V. Anticipating safety issues in e2e conversational ai: Framework and tooling. *arXiv* **2021**, arXiv:2107.03451.
102. Lin, S.; Hilton, J.; Evans, O. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv* **2021**, arXiv:2109.07958.
103. Rudinger, R.; Naradowsky, J.; Leonard, B.; Van Durme, B. Gender bias in coreference resolution. *arXiv* **2018**, arXiv:1804.09301.
104. Nangia, N.; Vania, C.; Bhalerao, R.; Bowman, S.R. CrowS-pairs: A challenge dataset for measuring social biases in masked language models. *arXiv* **2020**, arXiv:2010.00133.
105. Nadeem, M.; Bethke, A.; Reddy, S. StereoSet: Measuring stereotypical bias in pretrained language models. *arXiv* **2020**, arXiv:2004.09456.
106. Patterson, D.; Gonzalez, J.; Le, Q.; Liang, C.; Munguia, L.M.; Rothchild, D.; So, D.; Texier, M.; Dean, J. Carbon emissions and large neural network training. *arXiv* **2021**, arXiv:2104.10350.