Review

Not peer-reviewed version

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

Rajvardhan Patil * and And Venkat Gudivada

Posted Date: 6 February 2024

doi: 10.20944/preprints202402.0357.v1

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

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; patilr@gvsu.edu
[2] Computer Science Department, East Carolina University; 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 pre-training objectives, benchmarks, and transfer learning methods used in LLMs. It also examines different *fine-tuning* and *In-Context learning* techniques used in downstream tasks. It also 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 lens 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 stand.

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

1. Introduction

1.1. Background

Most feature-engineering methods before GPT relied on manually curated labeled data, which 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 of using unlabeled data in self-supervised manner to extract and leverage linguistic information emerged as 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 [10]. Word embeddings like Word2Vec ([11], [12]), Glove [13], and FastText[14] were generated from the unlabeled corpora using self-supervised approach. They improved performance across a variety of NLP tasks.

 doi:10.20944/preprints202402.0357.v1

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 ([8], [9]) representations. In a continuous vector space, shallow feed-forward networks were used to generate the word embeddings or word vectors. As the 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) [12] and skip-gram [12] during training. Unlike statistical approaches, in this approach, the network automatically discovers the embeddings. The task of explict 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, although 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. been 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 its 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, [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, 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 multiple layers (DNN) were used to generate context-sensitive and more robust (deep) representations that were transferred or applied to task-specific architectures. To overcome the task-specific architecture, Transformer based Pretrained Language Models (PLMs) came into play. Recent work using Transformers has focused on a task-independent approach, where transfer and fine-tuning of the self-attention block is sufficient. These transformer-based language models are flexible and task-agnostic. They can be fine-tuned on different downstream tasks without requiring architecture modifications. They have also led to significant improvement boosts, especially in capturing long-range dependencies.

 doi:10.20944/preprints202402.0357.v1

As shown in Figure 1, the phases in these transformed 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 factor and challenges of LLMs.

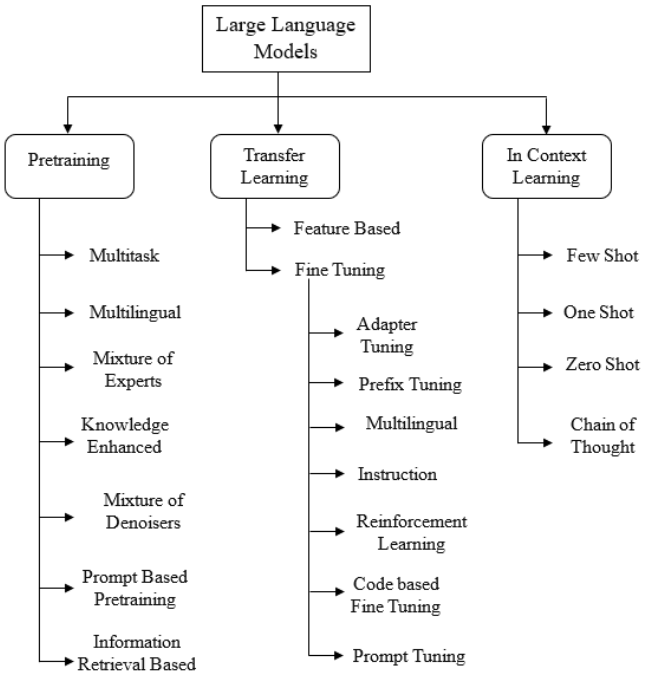

Figure 1. Large Language Model Phases

The outline of this survey paper is as follows: In Section 2, we look at the Language Model denition 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 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 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 fine-tuned 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. Each downstream task has separate fine-tuned models but are initialized with the same pre-trained parameters. This approach helps in learning a universal representation of language, which required 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 the 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.

 doi:10.20944/preprints202402.0357.v1

As stated in Bloom [55], 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: word, subword, character or byte, etc. Normally in the pretraining phase of Laguage Models, next word prediction objective is used, which is conditioned on the previous tokens as context. So for a given input or source sequence $S = (s_1, s_2, ..., s_n)$, the model predicts the joint probability of the output or target sequence $T = (t_1, t_2, ..., t_n)$, shown in equation 1.

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

This approach is referred to as autoregressive language modeling and can be seen as iteratively predicting the probability of the next token, as shown in equation 2.

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

Here, to deal with different downstream tasks (question answering, translation, summariztion 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 fine-tuning 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 similar mechanism of pretraining and fine tuning as 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, attention layer helps the decoder understand which inputs are more important. This enables the decoder focus on the right place or context during the prediction of each output tokens. 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 Query and the 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 inidicates that it is relatively more important to query than others with less scores. To get the probabilities of the match between keys and queries, the scores are ran through softmax to fit a distribution between 0 and 1. These probabilities act as an indexing-mechanism into the Value (V) vector. So these probabilities are further multiplied with the value vectors (V), which result in the alignment vectors. Equation 4 represents this computation from 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, teacher forcing technique is used, which leads to faster convergence and higher accuracy. In teacher forcing, instead of the model's output from previous

 doi:10.20944/preprints202402.0357.v1

timestep, the ground-truth (correct answer) is fed as an input at each time step. To enable this teacher forcing, the pre-attention decoder takes the target tokens and shifts them one place to the right.

2.3. MultiHead Attention

Instead of recurrent layers, Transformer differ from sequence to sequence by using multi-head attention layers, and hence they don't suffer from vanishing gradients problems that are related to the length of the sequences. Figure 2 highlights the Multihead attention mechanism in Transformers. In multihead attention mechanism, a set of parallel self-attention layers are added, which are called heads.

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

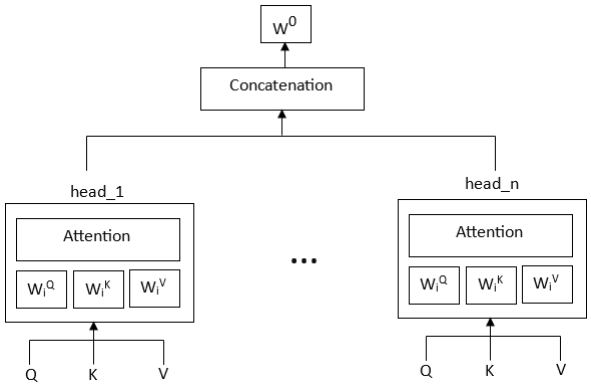

Figure 2. Multihead Attention Mechanism

As shown in the Figure 2, the output of these heads are further concatenated to produce a single output. This multi-head 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. Multi-head attention mechanism executes the attention of the scaled dot-product in parallel. Multi-headed 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 [22], using an RNN encoder-decoder architecture for NMT tasks, the fixed-length encoded vector generated by encoder became a performance bottleneck. Performance of such an RNN-based encoder–decoder model deteriorated rapidly as the length of an input sentence increased. It also gave 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, [22] proposed *attention mechanism* in the decoder, which automatically soft-searched 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.

 doi:10.20944/preprints202402.0357.v1

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 [22] 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 [22] improved significantly, it used bidirectional RNN as an encoder. RNN, being sequential, prohibits parallelization, leading to more computational time. [23] proposed transformer architecture that relied solely on the attention mechanism, altogether eliminating RNN, CNN components. Transformers handled long-term dependencies way 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 auto-regressive, it considers the previously generated token as an additional input to generate the next target token. As show 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 auto-regressive property.

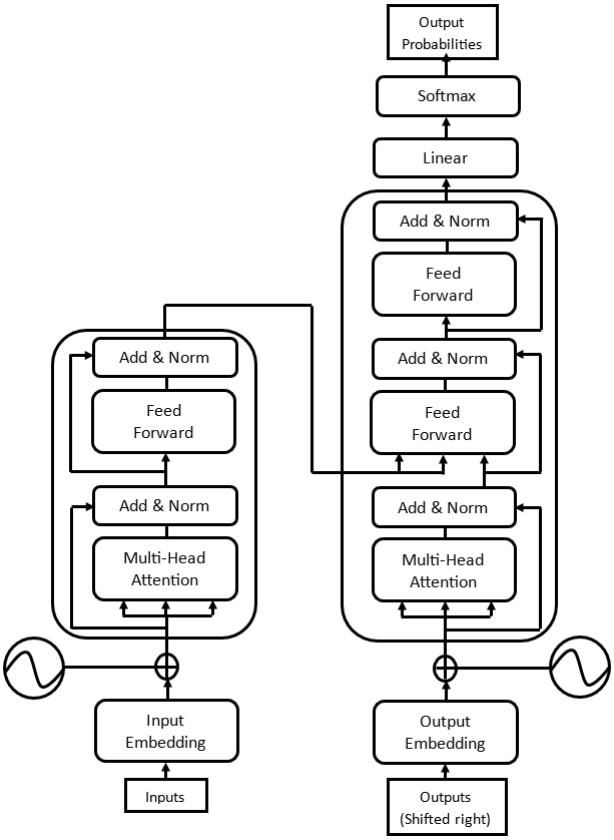

Figure 3. Transformer Encoder Decoder Architecture

 doi:10.20944/preprints202402.0357.v1

3. Transformer

After its inception, the 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

The example of Encoder-Decoder architecture is the Transformer model proposed in [23]. Its encoder and decoder blocks are stacked with multiple layers. As show in Figure 3, Transformer encoder layer consists of two consecutive layers, namely a self-attention layer followed by a position-wise feed-forward layer. Decoder is similar to encoder, except it adds third cross-attention layer, which attends over encoder output.

Encoder-decoder models adopt bidirectional attention for the encoder, unidirectional attention for the decoder, and cross attention mechanism between them. Cross-attention in the decoder has access only to the fully processed encoder output, and is responisble for connecting input tokens to target tokens. The encoder-decoder-based models are pre-trained for seq2seq tasks. They can also be pretrained on conditional generation tasks, where the output is generated in regards to the given input, for example in summarization, question answer, and translation tasks. T5 [26] uses encoder-decoder architecture. As stated in T5, using encoder-decoder structure helped achieve good performance over classification as well as generative tasks.

Although, Encoder-decoder models end up having twice as much 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 set 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 do 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 [25], 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 summarization.

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 auto-regressive. 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 representation 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 [24] was one of the first models which 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 summarization.

 doi:10.20944/preprints202402.0357.v1

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 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 the 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 grey lines represent causal masking (unidirectional attention).

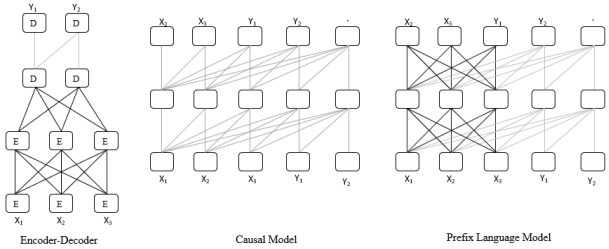

Figure 4. Transformer Models

As shown in Figure 4, in the encoder-decoder architecture, fully visible masking is used in the encoder, and causal mask 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.

3.5. Mask Types

Self-attention is the variant of the attention mechanism proposed in [22]. It generates an output sequence with the same length as the input sequence, replacing 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 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.

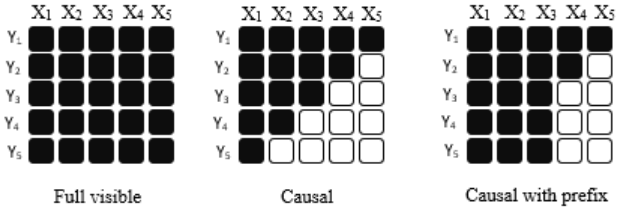

Figure 5. Mask Types

 doi:10.20944/preprints202402.0357.v1

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. The 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.

Table 1. Architecture Details of LLMs

Model	Param-Size	Layers	d-model	Atte-ntion heads	Hardware
Transformer-base [23]	-	6E, 6D	512	8	8 NVIDIA P100 GPUs
Transformer-big [23]	-	12E, 12D	1024	16	8 NVIDIA P100 GPUs
BERT-base [25]	110M	12E	768	12	4 Cloud TPUs
BERT-large [25]	340M	24E	1024	16	16 Cloud TPUs (64 TPU chips)
GPT-1 [24]	117M	12D	768	12	-
GPT-2 [28]	117M to 1.5B	24D to 48D	1600	48	-
GPT-3 [29]	175B	96	12288	96	V 100 GPUs (285K CPU cores, 10K GPUs)
T5 [26]	220M-11B	(12E, 12D)	-	-	1024 TPU v3
REALM [27]	330M	-	-	-	64 Google Cloud TPUs, 12GB GPU
Jurassic-1 [30]	178B	76	13824	96	
mT5 [32]	13B	-	-	-	-
Pangu-Alpha [33]	207B	64	16384	128	2048 Ascend 910 AI processors
CPM-2 [34]	198B	24	4096	64	-
Yuan 1.0 [35]	245B	-	-	-	-
HyperClova [36]	82B	64	10240	80	128 DGX servers with 1024 A100 GPUs
GLaM [47]	1.2T (96.6)	64 MoE	8,192	128	1024 Cloud TPU-V4 chips (Single System)
ERNIE 3.0 [91]	10B	48, 12	4096, 768	64, 12	384 NVDIA v100 GPU cards
Gopher [78]	280B	80	16384	128	4 DCN-connected TPU v3 Pods (each with 1024 TPU v3 chips)
Chinchilla [79]	70B	80	8192	64	-
AlphaCode [64]	41.1B	8E, 56D	6144	48, 16	-
CodeGEN [65]	16.1B	34	256	24	-
CodeGeeX [66]	13B	39	5120	40	1,536 Ascend 910 AI Processors
FLAN [37]	137B	-	-	-	TPUv3 with 128 cores
InstructGPT [38]	175B	96	12288	96	V 100 GPUs
LaMDA [39]	137B	64	8192	128	1024 TPU-v3 chips
T0 [43]	11B	12	-	-	-
GPT NeoX 20B [44]	20B	44	6144	64	12 AS-4124GO-NART servers (each with 8 NVIDIA A100-SXM4-40GB GPUs)
OPT [45]	175B	96	12288	96	992 80GB A100 GPUs
MINERVA [52]	540.35B	118	18432	48	-
AlexaTM 20B [48]	20B (19.75B)	46E, 32D	4096	32	128 A100 GPUs
GLM-130B [41]	130B	70	12288	96	96 NVIDIA DGX-A100 (8x40G)
XGLM [59]	7.5B	32	4096		
PaLM [54]	540.35B	118	18432	48	6144 TPU v4 chips (2 Pods)
Galactica [63]	120B	96	10,240	80	128 NVIDIA A100 80GB nodes
Pali [62]	16.9 (17)B	-	-	-	-
LLaMA [46]	65B	80	8192	64	2048 A100 GPU (80GB RAM)
UL2 [56]	20B	32E, 32D	4096	16	64 to 128 TPUv4 chips
Pythia [68]	12B	36	5120	40	-
WeLM [53]	10B	32	5120	40	128 A100-SXM4-40GB GPUs
BLOOM [55]	176B	70	14336	112	48 nodes having 8 NVIDIA A100 80GB GPUs (384 GPUs)
GLM [40]	515M	30	1152	18	64 V100 GPUs
GPT-J	6B	28	4096	16	TPU v3-256 pod
YaLM	100B				800 A100
Alpaca	7B				8 80GB A100s
Falcon	40B	-	-	-	-
(Xmer) XXXL [82]	30B	28	1280	256	64 TPU-v3 chips
[77]	1.1T	32	4096	512 (experts)	
XLM-R [100]	550M	24	1024	16	

4. Pretraining - Strategies & Objectives

The pretraining process makes the model learn and capture language representation and (general or domain) knowledge, which is then used in the downstream NLU and NLG tasks. Such a 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 an input-to-target task. 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

 doi:10.20944/preprints202402.0357.v1

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 summarization. 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 the 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 LMs offer more modeling power than the unidirectional encoding of inputs used in vanilla (LTR) 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 [26], that as the corrpution rate is 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. In 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 [26], three corruption strategies, 'Mass-style,' 'Replace Corrupted Span', and 'Drop Corrupted Tokens', were investigated. The'Mass-style' works similar to the MLM, except it focuses on replacing 15% of tokens with mask tokens, and excludes the random token swapping step. In the 'Replace Corrupted Span', a unique or single mask token is used to replace the consecutive span of corrupted tokens. Lastly, in the 'Drop Corrupted Tokens', the task is to reconstruct the corrupted tokens that are dropped from the input sequence. REALM [27] uses Salient span technique to focus on problems that require world knowledge.

4.1.4. General Language Mode Objective

[40] 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 auto-encoding, 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. It demonstrated how 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, [41] 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]

 doi:10.20944/preprints202402.0357.v1

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 the 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 leverages all uncorrupted tokens from the past and future as inputs for predicting the corrupted span (targets). This objective was evaluated in T5 [26], 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, 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, S2) as input and predicts whether S2 is the direct continuation of S1. Table 2 details different objectives, datasets, and tokens and/or corpus size used during the pretraining of prominent LLM models.

4.2. Learning Strategies

4.2.1. Multi-Task 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 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 at once simultaneously. However, unlike the adapter-layers, MTL requires simultaneous access to the tasks during pretraining. The networks' lower MTL layers (and their weights) get 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 [26], the multitask learning approach involves pretraining the model on multiple tasks simultaneously. Although multiple tasks were used during pretraining, the T5 model was fine-tuned 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 on the task and not expose it to more data 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 [26], as the same training objective was used for every task, only a single set of hyperparameters were required for effective fine-tuning on all downstream tasks.

 doi:10.20944/preprints202402.0357.v1

Table 2. Pretraining Details of LLMs

Model	Archit-ecture	Objectives	Pretraining-Dataset	Tokens, Corpus Size
Transformer-base [23]	Encoder-Decoder			
Transformer-big [23]	Encoder-Decoder	MLM, NSP	WMT 2014	-
BERT-base [25]	Encoder-only			
BERT-large [25]	Encoder-only	MLM, NSP	BooksCorpus, English Wikipedia	137B, -
GPT-1 [24]	Decoder-only	Causal/LTR-LM	BooksCorpus, 1B Word Benchmark	-
GPT-2 [28]	Decoder-only	Causal/LTR-LM	Reddit, WebText	-, 40GB
GPT-3 [29]	Decoder-only	Causal/LTR-LM	Common Crawl, WebText, English-Wikipedia, Books1, Books2	300B, 570GB
T5 [26]	Encoder-Decoder	MLM, Span Correction	C4	(1T tokens) 34B, 750GB
REALM [27]	Retriever + Encoder	Salient Span Masking	English Wikipedia (2018)	-
Jurassic-1 [30]	Decoder-only	Causal/LTR-LM	Wikipedia, OWT, Books, C4, PileCC	300B,
mT5 [32]	Encoder-Decoder	MLM, Span Correction	mC4	-
Pangu-Alpha [33]	Decoder + Query Layer	LM	Public datasets (e.g., BaiDuQA, CAIL2018, Sogou-CA, etc.) , Common Crawl, encyclopedia, news and e-books	1.1TB (80TB raw)
CPM-2 [34]	Encoder-Decoder	MLM	encyclopedia, novels, QA, scientific literature, e-book, news, and reviews.	-, 2.3TB Chinese data and 300GB English Data
Yuan 1.0 [35]	Decoder-only	LM, PLM	Common Crawl, Public Datasets, Encyclopedia, Books	, 5TB
HyperClova [36]	Decoder-only	LM	Blog, Cafe, News, Comments, KiN, Modu, WikiEn, WikiJp, Others	561B
GLaM [47]	Sparse/MoE Decoder-only	LM	Web Pages, Wikipedia, Forums, Books, News, Conversations	1.6T tokens,
ERNIE 3.0 [91]	Transformer-XL structure	UKTP	plain texts and a large-scale knowledge graph	375 billion, 4TB
Gopher [78]	Decoder-only	LM	MassiveText (MassiveWeb, Books, C4, News, GitHub, Wikipedia)	300B
Chinchilla [79]	-	-	MassiveText	1.4T
AlphaCode [64]	encoder-decoder	MLM, LM	Github, CodeContests	967B
CodeGEN [65]	decoder only	LM	THEPILE, BIGQUERY, and BIGPYTHON	505.5B
CodeGeeX [66]	decoder only	LM	The Pile, CodeParrot Collected	850B
FLAN [37]	Decoder-only	LM	web documents, dialog data, and Wikipedia	2.49T tokens,
InstructGPT [38]	Decoder-only	LTR-LM	Common Crawl, WebText, English-Wikipedia, Books1, Books2, Prompt Dataset (SFT, RM, PPO)	300B, 570GB
LaMDA [39]	Decoder Only	LM	public dialog data and web text	168B (2.97B documents, 1.12B dialogs, and 13.39B) 1.56T words, -
T0 [43]	Encoder-Decoder	MLM + LM	C4	1T tokens + 100B
GPT NeoX 20B [44]	Decoder-only	LM	The Pile	- 825 GB
OPT [45]	Decoder-only	-	BookCorpus, Stories, the Pile, and PushShift.io Reddit	180B tokens
MINERVA [52]	Decoder-only + Parallel Layers	LM	technical content dataset (containing scientific and mathematical data) , questions from MIT's OpenCourseWare, in addition to PaLM pretraining dataset	38.5B tokens (math content),
AlexaTM 20B [48]	seq2seq (Encoder-Decoder)	mix of denoising and Causal Language Modeling (CLM) tasks	Wikipedia and mC4 datasets	1 Trillion tokens, -
GLM-130B [41]	bidirectional encoder & unidirectional decoder,	GLM, MIP (Multi-Task Instruction Pre-Training)		400 billion tokens,
XGLM [59]	decoder-only	Causal LM	CC100-XL	
PaLM [54]	Decoder-only + Parallel Layers	LM	Social media conversations, Filtered webpages, Wikipedia (multilingual), Books, Github, News (English)	780B tokens,
Galactica [63]	decoder-only	-	papers, code, reference material, knowledge bases, filtered CommonCrawl, prompts, GSM8k, OneSmallStep, Khan Problems, Workout, Other	106B
Pali [62]	encoder-decoder and Vision Transformers	mixture of 8 pretraining tasks	WebLI (10B images and texts in over 100 languages), 29 billion image-OCR pairs	-
LLaMA [46]	transformer	LM	CommonCrawl, C4, Github, Wikipedia, Books, ArXiv, StackExchange	1.4T tokens,
UL2 [56]	Enc-Dec, decoder-Only Prefix LM	R, S, X denoising	C4	32B tokens,
Pythia [68]	decoder-only	LM	the Pile	300B tokens -
WeLM [53]	-	-	Common Crawl, news, books, forums, academic writings.	-
BLOOM [55]	Causal Decoder-only	LM	ROOTS corpus (46 natural languages and 13 programming languages)	366B, 1.61TB
GLM [40]	bidirectional encoder & unidirectional decoder	GLM		
GPT-J	Mesh Transformer JAX	LM	PILE	402B
YaLM			online texts, The Pile, books, other resources (in English, Russian)	, 1.7TB
Alpaca				
Falcon	encoder only	LM	RefinedWeb, Reddit	1T
(Xmer) XXXL [82]	encoder-decoder	MLM	C4	1T tokens
[77]	decoder-only (MoE)	LM	BookCorpus, English Wikipedia, CC-News, OpenWebText, CC-Stories, CC100	300B
XLM-R [100]	encoder only	Multilingual MLM	CommonCrawl (CC-100)	, 2.5TB

In MTL, as the same model performs many different tasks, the language model gets conditioned on the input and the task to be performed. Such Task conditioning can be implemented at the architecture level. But 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

 doi:10.20944/preprints202402.0357.v1

sentence, French sentence), a reading comprehension example can be likewise written as a sequence 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 [43], 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 T5[26] model and fine-tunes 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 massive scale, and became more robust to the prompt wording. WeLM [53] 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 [100] is a multilingual model pre-trained on CommonCrawl100 corpus having 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 improve performance across various cross-lingual transfer tasks. For low-resource languages, XLM models trained on CommonCrawl-100 did better than the ones 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 fine-tuning. 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 given a non-English test input because the model never observed a non-English target during fine-tuning. To address 'accidental translation', mT5 [32] mixed the unlabeled pretraining data during fine-tuning, dramatically alleviating this issue.

AlexaTM 20B [48] is 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 one-shot. 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 did better on SuperGLUE and SqUADv3 datasets than GPT-3 under zero-shot setting. In the one-shot summarization task, AlexaTM20B did 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 require significant amounts of computing power and resources. To address this issue, sparse experts were proposed, incurring substantially less training cost 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

 doi:10.20944/preprints202402.0357.v1

for each incoming input and improves model capacity without incurring additional computation costs. In MoE, although a huge 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 gets activated for every example, training cost is increased (roughly quadratically) with the increase in model size and training examples [73]. Whereas, ST-MoE [75], 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 (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. [73] 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, [73] did better on language modeling and machine translation tasks than prior studies.

Similarly, with MoE, GShard [74] was able to efficiently perform training and inference using conditional computation, where only a sub-network gets 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 sub-linearly. GShard, been pretrained on multilingual, when translating text from 100 languges to English, it was able to achieve better translation quality compared to prior art. 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 space storage. This might create difficulty in the model training and inference phase if GPUs capacity is exceeded. To address this issue, CPM2 [34] proposed 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 [77], for model training and inference, MoEs yield competitive zero and few-shot performance (except full-shot fine tuning) at a fraction of the computation. MoEs can match the dense model performance with 4 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 [47] 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 (1.2T parameters) is seven times larger than GPT-3 in parameters, it activates a subnetwork of 96.6B (8% of 1.2T) 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 model has many parameters, they reduce the carbon footprint by an order of magnitude. For example, it achieves the same level of one-shot performance as GPT-3 but uses only 1/3 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 [76] addressed some of the issues observed in MoE models, such as complexity, communication costs, and training instability. Switch-Transformer simplified the MoE routing

 doi:10.20944/preprints202402.0357.v1

algorithm and proposed an architecture that mitigates the instabilities in computationally efficient 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 [84] 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 [85], 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 [87] proposed generative and contrastive objectives and incrementally pretrained the model. As its parameters can capture the concept-centric commonsense understanding and reasoning, it does not have to rely on external knowledge graphs. The results demonstrated 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 knowledge inheritance technique, where it uses knowledge from existing pretrained models, instead of training the models from scratch. Instead of self-supervised, WKLM [86] 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 [88] 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 [89] 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 the knowledge information, previous methods mainly updated the original parameters of the pre-trained 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 [90] 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 pre-trained model's original parameters fixed, so the parameters 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.

[39] demonstrated how fine-tuning 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 on quality score. LaMDA demonstrated how this quality was improved as the model got 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 [91] is trained on plain texts and large-scale knowledge graphs. It integrated auto-encoder and auto-regressive networks into a single unified framework. So, it was able to deal with NLG as well as NLU tasks in fine-tuning and zero/few-shot learning settings.

 doi:10.20944/preprints202402.0357.v1

Additionally, ERNIE 3.0 used prompt-tuning during fine-tuning to better exploit knowledge from the pre-trained 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 on. 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 [56] proposed a 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 fine-tuning 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 fine-tuning behavior with the specific mode used during pre-training. 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 the large chunk of the sequence. The pretraining objective is considered to be highly denoising if it has a long span (for example, equal 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 2 to 5 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 summarization 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 a chain of thought processes using prompting and reasoning steps. UL2-20B, when experimented with FLAN instruction tuning, achieved a competitive score to FLAN-PaLM 62B on MMLU and Big-Bench benchmarks. After using the MoD objective, U-PaLM [60] 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 [63] 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 -540B on MATH, and outperformed Chinchilla on the MMLU

 doi:10.20944/preprints202402.0357.v1

benchmark. Despite not been trained on general corpora, Galactica did 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 [27] 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 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 multi-query 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 is available in abundance, but they don't perform adequately for tasks with scarce dataset as it limits their learning abilities. To address this issue, LLMs were first pre-trained 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 leverage the capabilities of pre-trained models and is advantageous, especially in data-scare settings. For example, GPT [24] used generative language model objective for pretraining, followed by discriminative fine-tuning. 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 comes from specific distributions (supervised datasets specific to the desired task).

5.1. Fine Tuning

Transfer learning started with feature-based techniques, where pre-trained 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 pre-trained 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 Fine-tuned 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 parameters during finetuning.

 doi:10.20944/preprints202402.0357.v1

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, [92] proposed a transfer with the Adapter module, in which a module gets added between layers of a pre-trained network. In each block of the transformer, these adapter layers, which are dense-RELU-dense blocks, get 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 get 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 get added to the model. This approach is a promising technique in the experiments conducted in [26]. Adapter tuning attains comparable performance with finetuning on NLU and NLG benchmarks by using only 2-4% task-specific parameters. Experiments from [92] demonstrated how BERT with adapters added only a few (3:6%) parameters per task to attain near SOTA on 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 fine-tuning, only the parameters of the final layer are updated first. Next, the parameters of the second-last layers are included in the fine-tuning. This process continues, until the parameters of all network layers get fine-tuned (updated). It is normally recommended to included an additional layer in fine-tuning, after each epoch of training. This approach was used in [26], where gradual-unfreezing caused minor degradation in performance across all tasks. In [26], it was found that the basic approach of updating all of a pre-trained model's parameters during fine-tuning outperformed methods that are designed to update fewer parameters, although updating all parameters is most expensive.

5.4. Prefix Tuning

Fine-tuning, although it leverages the knowledge from pre-trained 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 [94] keeps the pre-trained 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 get stored, making it a lightweight, modular, and space-efficient alternative. Despite learning 1000x fewer parameters than fine-tuning, Prefix Tuning [94] outperformed fine-tuning in low data settings and maintained comparable performance in full data settings. It also extrapolated better to the examples having topics that were unseen during training by learning only 0.1% of the parameters.

5.5. Prompt Tuning

Although fine-tuning the pre-trained 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

 doi:10.20944/preprints202402.0357.v1

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 fine-tuning 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 to. 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

The motivation behind "prompt engineering" is that not all prompts lead to the same accuracy. Thus, one should tune the prompt's format and examples to achieve the best possible performance. 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 fine-tuning since, as the scale of LLMs grow, learning via prompts becomes efficient and cost-effective. Additionally, unlike fine-tuning, 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 [31], fine-tuning on downstream tasks for trillion-scale models results in poor transferability. Also, these models need to be bigger to memorize the samples in fine-tuning quickly. To overcome these issues, the Prompt-tuning or P-tuning approach [112] is used, which is a parameter-efficient tuning technique. For example, GPT3 [29] (which was not designed for fine-tuning), 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 fine-tuning where separate model is required for each downstream task, in prompt tuning single model serves multiple different downstream tasks. In discrete prompt tuing, 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). Also, models are susceptible to this context, so improperly constructed contexts cause artificially low 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 tasks, 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 fine-tuning, only the parameters of these inserted prompt tokens get updated in prompt tuning. Hence, they are also called as soft-prompts. [96] 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 fine-tuned models. For example, prompt tuning of T5 matched the model's fine-tuning quality as size increased while enabling the reuse of a single frozen model for all 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 fine-tuning. 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. [31] 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

 doi:10.20944/preprints202402.0357.v1

approach, in continuous-prompt, as there are trainable embedding tensors, the prompt encoder can be optimized in a differentiable way. P-tuning helped augment the pre-trained model's NLU ability by automatically searching for better prompts in the continuous space. As demonstrated in [31], the P-tuning method improves GPTs and BERTs in both few-shot and fully-supervised settings.

Additionally, as the parameters of only 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 409.6K trainable parameters were to be updated compared to the 11B parameters of fine-tuning. As demonstrated in CPM-2, except for the Sogou-Log task, CPM-2 with prompt-tuning achieved comparable performance to the fine-tuning approach. The total size required for gradient tensors and optimizer state tensors also significantly decreases since, in prompt tuning, the number of parameters needed to be optimized is also much smaller. As a result, Prompt tuning can save at most 50% GPU memory as compared to fine-tuning. However, prompt tuning takes many more steps to converge, hence more time.

[36] demonstrated how p-tuning with only 4K examples provided comparable results to RoBERTwhich was fine-tuned on 150K data. P-tuning was able to significantly enhance the robustness of HyperCLOVA as well as the accuracy. Bloom [55] used Multitask prompted fine-tuning where it was fine-tuned on a training mixture composed of a large set of different tasks specified through natural language prompts. T0 and Bloom demonstrated how language models fine-tuned on a multitask mixture of prompted datasets have strong zero-shot task generalization abilities. MemPrompt [72] is a memory-enhanced GPT-3 that allows users to interact and improve the model without retraining. It pairs GPT-3 with a growing memory of recorded cases where the model misunderstood the user's intents, along with user feedback for clarification. Such a memory allows the system to produce enhanced prompts for any new query based on the user feedback for error correction in similar cases in the past.

PTR [95] proposed prompt tuning with rules for many-class text classification, which apply logic rules to construct (task-specific) prompts with several sub-prompts. This enables PTR to encode prior knowledge about tasks and classes into prompt tuning. This introduction of sub-prompts can further alleviate the difficulty of designing templates and sets of label words. AutoPrompt [93] creates a prompt by combining the original task inputs with a collection of trigger tokens according to a template. The same set of trigger tokens is used for all inputs and is learned using a variant of the gradient-based search. AutoPrompt searches for a sequence of discrete trigger words and concatenates it with each input to elicit sentiment or factual knowledge from a masked LM. AutoPrompt elicited more accurate factual knowledge from MLMs than the manually created prompts on the LAMA benchmark. These results demonstrate that automatically generated prompts are a viable parameter-free alternative to existing probing methods since prompting does not introduce large amounts of additional parameters. In contrast with AutoPrompt, the Prefix-Tuning method optimizes continuous prefixes, which are more expressive, and focuses on language generation tasks.

However, prompt engineering also has limitations, such as: 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 [59] focused on using the multilingual dataset (comprising a diverse set of languages) for fine-tuning. As a result, XGLM achieved cross-lingual solid transfer, demonstrating SOTA few-shot

 doi:10.20944/preprints202402.0357.v1

learning performance on FLORES-101 machine translation benchmark between many language pairs. When BloomZ [58] was fine-tuned with xP3, a multilingual task dataset of 46 languages, the model achieved better zero-shot task generalization (than P3-trained baseline) on English and non-English tasks. Furthermore, when xP3mt, a machine-translated multilingual dataset of xP3, was used to fine-tune BloomZ on non-English prompts, the performance of held-out tasks with non-English human-written prompts significantly improved. In other words, models could zero-shot generalization to tasks in languages they had never intentionally seen. So, the models learn 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 multilingual large models, zero-shot performance tends to be significantly lower than fine-tuned performance. So, to improve the multilingual model's zero-shot task generalization BloomZ [58] focused on crosslingual and multitask fine-tuning. This enabled the model to be usable for low-resource language tasks without further fine-tuning.

5.7. Reinforcement Learning from Human Feedback (RLHF) Fine Tuning

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 pre-train 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, [38] proposed Reinforcement Learning (RL) from human feedback to fine-tune 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 fine-tune the model. It was demonstrated how, despite having 100x fewer parameters, the outputs from InstructGPT model having 1.3B parameters were preferred over GPT-3 with 175B parameters. The RLHF process consist mainly of three steps:

1. In the first step, supervised fine-tuning is used, where the dataset consisting of prompts along with their desired output behavior is given as input.
2. Another dataset of comparisons between model outputs is collected, where for a given input, labelers identify which output they would prefer using labels. This comparison data is then used to train a Reward Model to predict human-preferred output (which model output the labelers prefer).
3. The policy generates an output for which the reward model generates a reward. This reward is then used to update (maximize) the policy's Proximal Policy Optimization (PPO) algorithm.

Using the RLHF approach, InstructGPT demonstrated improvement in toxicity and truthfulness over GPT-3 and generalized well to held-out instructions. [69] 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 [69], the pre-trained model is fine-tuned using reinforcement learning rather than supervised learning, where it demonstrated its results on summarization and continuation tasks by applying reward learning to language generation. [70] recursively used the RL approach to produce novel summaries and achieve SOTA results for book-length summarization on the BookSum dataset. Similarly, using the reinforcement learning technique, [71] trained a model to predict the human-preferred summary and used it as a reward function to fine-tune the summarization policy. It could outperform larger models fine-tuned using a supervised approach and human reference summaries and generalize well to new datasets.

 doi:10.20944/preprints202402.0357.v1

5.8. Instruction Tuning

In instruction tunning, the model is fine-tuned 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 the LLMs, such as the prompt format used, objectives used during fine-tuning, diversity of tuning tasks, distribution of datasets, etc. Additionally, the zero-shot task generalization of LLMs does poorly across tasks. To address this, Multitask fine-tuning (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 [37] expresses existing 62 NLP datasets in the instructional format. This transformed dataset with instructions is then used to fine-tune 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 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-shots) on 20 of 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. The 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 fine-tunes it with 52K 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 [42] 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 made 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 instruction comprising several examples with desired output along with the definition that maps input text to task output. When evaluated on 119 unseen tasks (English and multilingual variant), 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 [57], 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 fine-tuned 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, it experimented by adding dialogues as auxiliary datasets to see if that could induce chatbot behavior in the model.

[61] experimented with instruction tuning with model size, number of tasks, and chain-of-thought datasets. It was observed that instruction fine-tuning scales well, and model performance substantially

 doi:10.20944/preprints202402.0357.v1

improved with the increased size of models and number of fine-tuning 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 [61] demonstrated how CoT data improves performance reasoning tasks when jointly fine-tuned with instruction dataset. After instruction tuning model classes such as T5, PaLM, and U-PaLM, [61] observed a significant boost in performance for different types of prompting setups (zero, few, CoT), and benchmarks as compared to the original models (without instruction fine-tuning).

In Self-Instruct [51], 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 fine-tune the original model. Along with the instructions, the framework also creates input-output instances, which can be used to supervise the fine-tuning of instructions. When self-instruct was applied to GPT-3, it achieved 33% performance gain on SUPER-NATURALINSTRUCTIONS over the original model, which was on par with InstructGPT performance.

5.9. Code based Fine Tuning

Code generation problem can be viewed as a sequence-to-sequence translation task, where given a problem description X in natural language, produce a corresponding solution Y in a programming language. Recent LLMs models have demonstrated an impressive ability to generate code and can now complete simple programming tasks. Codex [49] uses the GPT model, which was fine-tuned 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 solves 11.4%. It needs help with docstrings describing long operations chains and binding operations to variables.

However, these models still need to improve when evaluated on more complex, unseen problems that require problem-solving skills beyond simply translating instructions into code. For example, competitive programming problems that require an understanding of algorithms and complex natural language remain highly challenging. Creating code that solves a specific task requires searching in a vast structured space of programs with a sparse reward signal. To address this gap and to enable deeper reasoning, the AlphaCode [64] model was pre-trained on a collection of open-source code from GitHub and then fine-tuned on a curated set called CodeContests of competitive programming problems. The pre-training stage helps the model learn good code representations and generate code fluently, while the fine-tuning stage helps the model adapt to the target competitive programming domain. The pre-training dataset consisted of code from several languages such as C++, C#, Go, Java, JavaScript, Lua, PHP, Python, Ruby, Rust, Scala, and TypeScript. In simulated programming competitions hosted on the Codeforces platform, AlphaCode achieved, on average, a ranking of the top 54.3% in competitions with more than 5,000 participants. During fine-tuning, it used the natural language problem description for the encoder and the program solution for the decoder. It was also found that using a shallow encoder and a deep decoder significantly improved training efficiency without hurting the problem-solving rate.

CodeGEN [65] introduced a multi-turn program synthesis approach. A user communicates with the synthesis system by progressively providing specifications in natural language while receiving responses from the system in the form of synthesized subprograms, such that the user and the system complete the program in multiple steps. Such multiple step specification eases the understanding of a model, leading to enhanced program synthesis. CodeGEN demonstrated how the same intent provided to CODEGEN in a multiturn fashion significantly improves program synthesis over that provided as a

 doi:10.20944/preprints202402.0357.v1

single turn. CodeGEN [65] proposed an open benchmark called Multi-Turn Programming Benchmark (MTPB), comprising 115 diverse problem sets that are factorized into multi-turn prompts. MTPB is used to measure the models' capacity for multi-turn program synthesis. To solve a problem in this benchmark, a model needs to synthesize a program in multiple steps with a user who specifies the intent in turn in natural language.

CodeGeeX [66] is a multilingual model trained on 23 programming languages. It proposed a HumanEval-X benchmark for evaluating multilingual models by hand-writing the solutions in C++, Java, JavaScript, and Go. CodeGeeX was able to outperform multilingual code models of similar scale for both the tasks of code generation and translation on HumanEval-X.

6. In-Context Learning

Fine-tuning is task-agnostic, but it uses a supervised approach during transfer learning and hence requires access to a large amount of labeled dataset for every downstream task. Furthermore, having such a task-specific dataset, leads to fine-tuning the model on a very narrow distribution, which might potentially yield poor generalization on out-of-distribution dataset. 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 fine-tuning, 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 has shown initial promises and improved out-of-domain generalization. However, the results are far inferior to those of the fine-tuning technique. ICL helps analyze whether the model rapidly adapts to the tasks that are unlikely to be directly contained in the training set. In ICL, the model gets 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 into a few-shot, one-shot, and zero-shot. We describe each of them below.

6.1. Few-Shot learning

In a 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 to 100, or with as many examples that can fit into the model's context window. Compared to the fine-tuning 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 if 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, along with task description, were provided as the context through the window. Similarly, it was demonstrated in Jurassic-1 [30] 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.

 doi:10.20944/preprints202402.0357.v1

However, it was demonstrated in some of the papers, such as [81], that the examples used in the few-shot, the sequence in which the examples were ordered, and the format of the prompt directly affected the accuracy. [81] 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, or those appearing frequently, or to those familiar in the pre-trained dataset. To address this instability, [81] 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, [99] 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 which were semantically similar to the test sample. This strategy helped give 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 [97], the study compared how the model generalizes in a few-shot fine-tuning and in-context learning setting. During the comparison, the model size and, number of examples and parameters used in the experiment were controlled. The results demonstrated how the fine-tuned model generalized similarly to the ICL model to out-of-domain and improved performance as models became larger.

6.2. One-Shot learning

This approach is similar to the few-shot learning, except only one example is given as context in addition to the task description. The pre-trained-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 [24] 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 pre-trained data. [98] compared different architectures and pretraining objectives and their impact on zero-shot generalization. Experiments from [98] 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 [101], it was demonstrated how CoT-based prompting technique helped significantly improve LLMs' performance for complex reasoning tasks. When the LLMs are prompted using CoT technique, they demonstrate the intermediate reasoning steps involved in computing the final answer to unseen problems. CoT prompts indirectly help the

 doi:10.20944/preprints202402.0357.v1

model to access relevant knowledge (acquired during pretraining), which helps 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 8 CoT examples, it surpassed fine-tuned GPT-3 to achieve SOTA performance on the GSM8K benchmark having math word problems. Similarly, Minerva [52] used PaLM model, and further fine-tuned 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 fine-tuned 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 [57] used 15 reasoning dataset 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 of greedy decoding, where the most common way of reasoning is selected to solve the problem. [102] proposed a self-consistency decoding alternative, where instead of taking the greedy path, it explores different ways of solving a complex reasoning problem that leads to the unique correct answer. [102] demonstrated how by 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 more computational cost.

As addressed in Galactica [63], 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 is 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 enlists fine-tuning methods used in the prominent LLMmodels along with additional details, such as Pretraining (PT) and Fine Tuning (FT) batch sizes and epochs.

 doi:10.20944/preprints202402.0357.v1

Table 3. Fine Tuning Details of LLMs

Model	PT, FT batch-size	Conte-xt Size	PT, FT Epochs	Activation, Optimizer	Fine Tuning Methods
Transformer-base [23]	-	-	100,000		
Transformer-big [23]	-	-	300,000	- , Adam	Feature Based
BERT-base [25]	256, 32	128, 512	40, 4		
BERT-large [25]	256, 32	128, 512	40, 4	GELU, Adam	FT
GPT-1 [24]	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, zero-shot
T5 [26]	128, 128	512	2^{19}, 2^{18} steps	RELU, AdaFactor	FT
REALM [27]	512, 1	-	200k steps, 2 epochs	-	-
Jurassic-1 [30]	3.2M tokens	2048	-	-	few-shot, zero-shot
mT5 [32]	-	-	-	GeGLU,	FT, zero-shot
Pangu-Alpha [33]	-	1024	130K 260K	GeLU	few-shot, one, 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 [47]	-	1024	-	-, Adafactor	zero, one and few-shot
ERNIE 3.0 [91]	6144	512	-	GeLU, Adam	FT, zero and few-shot
Gopher [78]	-	2048	-	Adam	FT, few-shot, zero-shot
Chinchilla [79]	-	-	-	AdamW	FT, zero-shot
AlphaCode [64]	2048	-	205K	-	fine tuning
CodeGEN [65]	2M	2048	-	-	zero-shot
CodeGeeX [66]	3072	-	-	FastGELU, Adam	fine tuning
FLAN [37]	-, 8192	1024	, 30K	-, Adafactor	Instruction Tuning, zero-shot
InstructGPT [38]	3.2M	2048	-	-	RLHF
LaMDA [39]	-	-	-	gated-GELU,	FT
T0 [43]	-	-	-	RELU, AdaFactor	FT, zero-shot
GPT NeoX 20B [44]	3.15M tokens	2048	150K steps	-, AdamW with ZeRO	few-shot
OPT [45]	2M tokens	2048	-	ReLU, AdamW	few-shot, zero-shot
MINERVA [52]	-	1024	399K	SwiGLU, Adafactor	few-shot, chain-of-thought context
AlexaTM 20B [48]	2 million tokens	-	-	, Adam	Fine Tuning, few-shot, one-shot, zero-shot
GLM-130B [41]	4224	2048	-	GeGLU,	zero-shot, few (5) shots
XGLM [59]					
PaLM [54]	512, 1024, 2048 (1, 2, 4M tokens) , -	2048	1 (255k steps)	SwiGLU, Adafactor	few-shot, Chain of Thought, finetuning
Galactica [63]	2M	2048	4 epochs	GeLU	zero-shot
Pali [62]	-	-		-	
LLaMA [46]	4M tokens	-	-	SwiGLU, AdamW	zero-shot, few-shot, Instruction Tuning
UL2 [56]	128	512	500K steps	SwiGLU, Adafactor	in-context learning, zero-shot, one-shot, fine-tuning, instruction tuning
Pythia [68]	1024	2048	1.5 Epochs	Adam	zero-shot
WeLM [53]	2048	2048	-	-	zero-shot, few-shot
BLOOM [55]	20,482,048	2048	-	GELU, -	zero-shot, few-shot, multitask prompted (fine) tuning
GLM [40]	1024		200K Steps		FT
GPT-J		2048	383,500 steps	-	FT
YaLM					
Alpaca		-		-	FT, IT (instruction Tuning)
Falcon	-	2048	-	-	-
(Xmer) XXXL [82]			2^{19}		FT
[77]		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 pre-trained 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 [83], new behaviors that arise due to scaling language models have been increasingly referred to as emergent abilities, which are the abilities that are not present in smaller models but are present (resurface/are discovered) in larger models. In other words, quantitative changes in a system result in qualitative changes in behavior. Large language models with over 100 billion parameters have presented attractive scaling laws where emergent zero-shot and few-shot capabilities suddenly arouse [83]. As stated in [44], many of the most exciting capabilities of LLMs only emerge above a certain number of parameters, and they have many properties that cannot be studied 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 the increase in model size,

 doi:10.20944/preprints202402.0357.v1

model performance improved steadily for zero-shot and rapidly for few-shot. As their size increase, models tend to be more proficient and efficient at in-context learning. As highlighted in [29], the gap between zero, on, and few-shot performance often grows with model capacity, suggesting that larger models are more proficient meta-learners. As demonstrated in [33], the perplexity decreases with the increase in model capacity, training data, and computational resources. As experimented in [28], when a large language model is trained on a sufficiently large and diverse dataset, it can perform well across many domains and datasets.

There are various ways to scale, including using a bigger model, training the model for more steps, and training the model on more data, which we explore below. We also look at how the scaling up of the model and data size has affected the performance of models.

7.1. Model Width (Parameter Size)

Kaplan et al. [80] analyzed the effect of the size of neural models, the computing power used to train them, and the data available for this training process on the performance of the language models. It observed precise power-law scalings for performance as a function of training time, context length, dataset size, model size, and compute budget. The key finding from [80] was that LM performance improves smoothly and predictably as model size, data, and compute 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 [80], for optimally compute-efficient training, most of the increase in compute 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 boost and increase parallelism. Additionally, larger batches and training for more steps become possible as more computing becomes available.

T5 [26] conducted experiments that started with the baseline model having 220M parameters and then scaled it up to a model having 11B parameters. Experiments conducted in T5 showed how performance degraded as the data size shrank. Also, increasing the training time and model size consistently improved the baseline and resulted in an additional bump in performance.

7.2. Training Tokens & Data Size

Although Kaplan et al. [80] showed a power law relationship between the number of model parameters and its performance, it did not consider pretraining tokens or corpus data size. Hoffmann et al. [79] also reached the same conclusion, but it recommended that large models should be trained for many more training tokens. Specifically, given a 10x increase in computational budget, Kaplan et al. [80] suggest that the model size should increase 5.5x while the number of training tokens should only increase 1.8x. Instead, Chinchilla [79] 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, especially in few-shot settings, these efforts were based on the assumption that more parameters would lead to better performance. However, recent work from [79] shows that, for a given compute budget, the best performances are not achieved by the largest models but by smaller models trained on more data. For instance, although LLaMA had 13B parameters, it outperformed GPT-3 with 175B on most benchmarks despite being 10x smaller. Chinchilla helps answer how, given a fixed computational budget, one should trade-off model size and the number of training tokens. It states that the model size and the number of training tokens should be scaled equally: for every doubling of the model size, the number of training tokens should also be doubled.

With 1/4 less parameters and 4x 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 fine-tuning and inference, which reduces fine-tuning and inference costs considerably and greatly facilitates downstream usage on smaller hardware. Although [79] it determines how to scale the

 doi:10.20944/preprints202402.0357.v1

model size and the dataset for a given compute 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. For example, Falcon-40B requires 70GB of GPU memory to make inferences, whereas Falcon-7B needs only 15GB, making inference and fine-tuning accessible even on consumer hardware.

Additionally, as per [46], although it may be cheaper to train a large model to reach a certain level of accuracy, a smaller model trained longer will be cheaper at inference. For instance, although [79] recommended training a 10B model on 200B tokens, [46] demonstrated that the performance of a 7B model continues to improve even after 1T 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 [53], a Chinese LM, demonstrated how, by carefully cleaning, balancing, and scaling up the training data size, WeLM could significantly outperform existing models with similar or larger sizes. For instance, on zero-shot evaluations, it matched the performance of Ernie 3.0 Titan which is 25x 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 [111], 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. Whereas, the transition into depth efficiency was clearly demonstrated when the network width is increased. It was shown in [111], 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 [111], the optimal depth for GPT3's 175B parameters should have been 80 layers instead of 96. As per [111], Jurassic-1 [30] used 76 layers for the 178B 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, 7% and 23% gain in batch inference, and text generation. Also, by shifting compute resources from depth to width, more operations can be performed in parallel (width) rather than sequentially (depth).

Additionally, [80] did present a comprehensive study of the scaling behavior of Transformer language models, but it mainly focused on the upstream (pretraining) loss. Whereas [82] found that scaling laws differ in upstream and downstream setups. [82] focuses on transfer learning and shows that, aside from only the model size, model shape matters for downstream fine-tuning and that scaling protocols operate differently at different compute regions. [82] demonstrated how their redesigned models achieve similar downstream fine-tuning quality while having 50% fewer parameters and training 40% faster than the widely adopted T5-base model. [82] recommends the DeepNarrow strategy, where the model's depth is preferentially increased before considering any other forms of uniform scaling across other dimensions.

7.4. Architecture - Parallelism

Large models often require a lot of storage space to store the parameters. For instance, storing 178B parameters requires more than 350GB of memory with half-precision. As stated in [33], as the model size grows beyond 10B parameters, it becomes difficult to train the model. To store [33] model of 200B parameters, 750GB space is required. Training such a model demands several times more

 doi:10.20944/preprints202402.0357.v1

memory than just storing the parameters since gradients and optimizer states are also essential for updating the parameters.

As large GPUs available today have a memory of around 80GB, 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, such as data, model, and pipeline, are used to acquire large end-to-end throughput (keeping high resource utilization on a cluster of processors).

Figure 6 demonstrates different types of parallelism techniques. Each parallelism dimension trades computation (or communication) overheads for memory (or throughput) benefits. To acquire maximum end-to-end throughput, a balanced composition point should be found along these dimensions. The problem becomes more challenging when considering the heterogeneous bandwidths in a cluster of devices. Below we discuss each of these approaches.

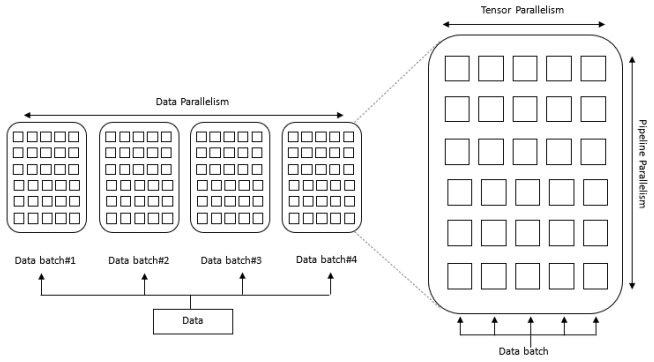

Figure 6. 3D Parallelism

7.4.1. Data parallelism

In this approach, the training batches are partitioned across the devices, followed by the synchronization of gradients from different devices before taking 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, it does introduce communication overheads to keep the distributed tensor layouts consistent between successive operators.

7.4.3. Pipeline parallelism

Pipeline parallelism splits the layers of LMs 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 total number of layers are partitioned into stages. These states are placed on different devices. This approach is beneficial from a memory perspective since each device holds only a subset of the total layers of the model, and the communications happen only at the boundaries of stages.

7.5. Miscellaneous

7.5.1. Training Steps

Over time, it has become evident through experiments that the performance of LLMs can steadily be improved by training the model longer (increasing the training steps). In [26], it was observed that training a smaller model on more data was often outperformed by training a larger model for fewer

 doi:10.20944/preprints202402.0357.v1

steps. Increasing the size of T5 yields consistent improvement but comes at a significant computational cost from Base to 11B. In contrast, with the help of the 'textual knowledge retriever' that REALM uses, it outperformed the largest T5-11B model while being 30 times smaller. It is also important to note that T5 accesses additional reading comprehension data from SQuAD during its pre-training (100,000+ examples). Access to such data could also benefit REALM, but it was not used in our experiments. Primer [50] proposed a new architecture that has a smaller training cost as compared to other transformer variants. Its improvements were attributed mainly to squaring ReLU activations and adding a depthwise convolution layer after each Q, K, and V projection in self-attention. As a result, the Primer needs much less compute time to reach a target one-shot performance. For instance, Primer uses 1/3 of the training compute to achieve the same one-shot performance as Transformer.

7.5.2. Checkpoints

Parameter checkpoints are created while pretraining the model to reduce memory requirements and speed up pre-training. These checkpoints allow researchers to quickly boost and attain strong performance on many tasks without needing to perform the expensive pretraining themselves. For example, the checkpoints released by [26] were used to achieve SOTA results on many benchmarks.

7.5.3. Ensembling

It is common practice to train and evaluate models using an ensemble of models, as it helps to use additional computation. [26] demonstrated how an ensemble of models provides substantially better results than a single model, which provides an orthogonal means of leveraging additional computation. It was observed in [26] that ensembling models that were fine-tuned from the same base pre-trained model performed worse than pre-training and fine-tuning all models completely separately, though fine-tune-only ensembling still substantially outperformed a single model.

8. LLM Challenges

Language models can generate biased outputs of misinformation and be used maliciously. Large Language Models carry potential risks such as outputting offensive language, propagating social biases, and leaking private information. Large language models reproduce and might amplify existing biases in the training data, generating toxic or offensive content. During training, as the language model absorbs biases and toxicity expressed in the text, they are prone to replicating them. This occurs because of the significant presence of unmoderated social media discussions in the pre-training dataset, providing additional inductive bias to aid in such classification tasks. 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. Biases

Human biases and undesired social stereotypes exist in large pre-trained language models. One such bias is the gender and occupation bias. The Winogender benchmark [108] is a coreference resolution task and measures gender bias in English occupation nouns such as "nurse" and "engineer." Winogender tests whether a model can correctly determine if a pronoun refers to different occupation words. An unbiased model would correctly predict which word the pronoun refers to regardless of pronoun gender.

WinoGender is made of Winograd schema, and biases are evaluated by determining if the gender of the pronoun impacts a model coreference resolution performance. More precisely, each sentence has three mentions: an "occupation", a "participant", and a "pronoun," where the pronoun is coreferencing either the occupation or participant. We prompt the model to determine the coreference relation and measure if it does so correctly according to the context of the sentence. The goal is to reveal if the

 doi:10.20944/preprints202402.0357.v1

model has captured societal biases associated with occupations. For example, doctors are more likely to associate with the pronoun 'he', while nurses are more likely to associate with the pronoun 'she'. GLaM was able to close the performance gap between stereotypical and anti-stereotypical examples on the WinoGender benchmark, suggesting that large, sparsely activated models may rely less on superficial statistical correlations.

Gopher [78] 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, 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. It was observed in PaLM that accuracy on the Winogender coreference task improves with model scale, and PaLM 540B sets a new state-of-the-art result in 1-shot and few-shot settings. Secondly, co-occurrence analysis performed on race/religion/gender prompt continuation demonstrates the model's potential to affirm stereotypes falsely.

There have been attempts such as ETHOS dataset, which helps measure LLMs' ability to identify whether certain English statements are racist or sexist or neither. Furthermore, CrowSPairs is a crowdsourced benchmark aiming to measure intrasentence level biases in 9 categories: gender, religion, race/color, sexual orientation, age, nationality, disability, physical appearance, and socioeconomic status. Each example consists of a pair of sentences representing a stereotype, or anti-stereotype, regarding a particular group to measure model preference towards stereotypical expressions. Higher scores indicate higher bias exhibited by a model. CrowSPairs[109] dataset allows measuring biases in 9 categories: gender, religion, race/color, sexual orientation, age, nationality, disability, physical appearance, and socioeconomic status. Each example is composed of a stereotype and an anti-stereotype. Additionally, StereoSet [110] is another dataset used to measure stereotypical bias across four categories: profession, gender, religion, and race. In addition to intrasentence measurement (similar to CrowSPairs), StereoSet includes measurement at the intersentence level to test a model's ability to incorporate additional context. To account for a potential trade-off between bias detection and language modeling capability, StereoSet includes two metrics: Language Modeling Score (LMS) and Stereotype Score (SS), which are then combined to form the Idealized Context Association Test score (ICAT).

8.2. 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 [105] as an indicator of how toxic their model is.

To measure toxicity, [105] proposed the RealToxicityPrompts benchmark. RealToxicityPromptsdataset is used to evaluate the tendency of LLM models to respond with toxic language. RealToxicityPrompts consists of about 100k prompts the model must complete; then, a toxicity score is automatically evaluated. 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, have a high propensity to generate toxic language and reinforce harmful stereotypes, even when provided with a relatively innocuous prompt, and adversarial prompts are trivial to find.

SaferDialogues [106], and Safety Bench [107] are two benchmarks that are used to test dialogue safety evaluations. SaferDialogues measures the ability of the model to recover from explicit safety

 doi:10.20944/preprints202402.0357.v1

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, stratified across four levels of topic sensitivity: Safe, Realistic, Unsafe, and Adversarial.

8.3. 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 given 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 gives 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 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 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 [103] 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 [103] that generally, the largest models were the least truthful, and so scaling up the model size increased performance but was less promising in improving the model's truthfulness.

8.4. Cost & Carbon Footprints

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

1. Computational cost for pre-training: a super large model requires several weeks of pre-training with thousands of GPUs.
2. Storage cost for fine-tuned 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 pre-training 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 equation 6 and 7, where 0.385 in equation 7 is the US national average carbon intensity factor $(0.385 kgCO2eq/KWh)$ and PUE represents Power Usage Effectiveness.

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

 doi:10.20944/preprints202402.0357.v1

$$tCo2eq = MWh \times 0.385 \tag{7}$$

As stated in LLaMA [46], carbon emission also depends on the data center's location used during pre-training of the network. For instance, BLOOM uses a grid that emits $0.057kgCO2eq/KWh$, leading to $27tCO2eq$, and OPT uses a grid that emits $0.231kgCO2eq/KWh$, leading to $82tCO2eq$. As stated in [104], 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, 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 [104] 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 datacenter carefully. The above equations 6 and 7 can be restated in terms of energy consumption and CO2 emission 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 given to reducing carbon footprint by building more efficient LLMs. For example, OPT [45] is comparable to GPT-3, and requires only 1/7th of the carbon footprint to develop.

[104] also recommends three suggestions that could eventually help reduce CO2e footprint:

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

As highlighted in [104], 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 & Low Resource

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] and [36], are available in the market. So, there are a lot of 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 into the LLMs. Furthermore, the impact of increasing the proportion of multilingual data on multilingual and cross-lingual tasks needs to be explored.

Strategies aimed at mitigating the low-resource challenge is to acquire more language data. Some of these attempts have focused on collecting human translations, while others have leveraged

 doi:10.20944/preprints202402.0357.v1

large-scale data mining and monolingual data pipelines to consolidate data found across the web. The latter techniques are often plagued by noise and biases, making it difficult to validate the quality of the created datasets. Finally, they also require high-quality evaluation datasets or benchmarks to test the models. NLLB [67] has attempted and strived to understand the low-resource translation problem from the perspective of native speakers and studies how to create training data to move low-resource languages towards high-resource automatically. It proposed Flores-200 many-to-many benchmark dataset that doubled the language coverage of a previous effort known as Flores-101. Flores-200 is created using professional human translators who translate the FLORES source dataset into the target language, where a separate group of reviewers perform quality assessments and provide translation feedback to the translators.

9. Future Directions & 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 unrealized utility of LLMs. Though we enumerate these directions and trends under different facets to facilitate elucidation, there is a strong interconnectedness among the facets.

9.1. Interpretability & Explainability

An LLM's ability to explain its decisions and predictions is crucial to promote trust, accountability, and widespread acceptance. Current research targets methods that can explain the model's decision-making 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.

 doi:10.20944/preprints202402.0357.v1

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 contains 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 multi-faceted 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 the 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 & 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 pre-processing 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, benchmarking and evaluation metrics.

In the input pre-processing 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. 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 is 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,

 doi:10.20944/preprints202402.0357.v1

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 relevant elements from diverse media is essential. This is a challenging task and requires groundbreaking research. 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 is required for learning shared representations across modalities. Also, transfer learning from pre-trained 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 & Environmental Impact

Training LLMs require 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 a 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 computations and thus reduce energy consumption. Related to the optimized hardware accelerators is the model architecture optimization. Topics to be researched in this direction include model structure optimization, reducing redundant parameters, and developing sparse models. Pruning

 doi:10.20944/preprints202402.0357.v1

and sparsity induction through identifying and eliminating redundant or less significant parameters contributes 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 contribute to reduced computational overhead of LLMs. For example, knowledge distillation is a technique that helps decrease the model's memory and computational requirements. Research is needed in lifecycle assessment and environmental impact to inform the 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 & Domains

Current LLM research and development is primarily confined to the English language. According to Ethnologue, there are 7,168 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 3,045 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 pre-training 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 row-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,

 doi:10.20944/preprints202402.0357.v1

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 to privacy-preserving models is ensuring user data privacy while guaranteeing model performance and utility. This requires a multi-pronged approach, as outlined below. Privacy-preserving techniques such as anonymization during data pre-processing help 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 it. This approach preserves data privacy throughout the computation process. Protocols such as Secure Multi-party 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.

Model aggregation and ensemble approach aggregate 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 & 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 the *attentional interfaces* help 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

 doi:10.20944/preprints202402.0357.v1

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 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. Fine-tuning the models on new data while leveraging pre-trained 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 & Societal Impact

Several key strategies are required to address issues in 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 & 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 is critical since LLM models must be tailored through fine-tuning to address the use cases effectively. This requires a precise understanding of 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 pre-training. It elucidated different in-context, pre-training, 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 make them compute efficient. Finally, it also sheds light on future directions and challenges encountered in LLMs. Overall, the article empirically

 doi:10.20944/preprints202402.0357.v1

compared existing trends and techniques and comprehensively analyzed where the field of LLMs currently stands.

References

1. Harris, Z. S.Distributional structure, *Word*, 10 (2-3), pp. 146-162, 1954.
2. Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J., Mercer, R. L., and Roossin, P. S. *A statistical approach to machine translation*,Computational linguistics, 16(2), pp. 79-85, 1990.
3. Salton, G., and Lesk, M. E. *Computer evaluation of indexing and text processing*,Journal of the ACM (JACM), 15(1), pp. 8-36, 1968.
4. Jones, K. S. *A statistical interpretation of term specificity and its application in retrieval*,Journal of documentation, 1972.
5. Salton, G., Wong, A., and Yang, C. S. *A vector space model for automatic indexing*,Communications of the ACM, 18(11), pp. 613-620, 1975.
6. Tang, B., Shepherd, M., Milios, E., and Heywood, M. I. *Comparing and combining dimension reduction techniques for efficient text clustering*,In Proceeding of SIAM international workshop on feature selection for data mining, pp. 17-26, 2005.
7. Hyvärinen, A., and Oja, E. *Independent component analysis: algorithms and applications*,Neural networks, 13(4-5), pp. 411-430, 2000.
8. Vilnis, L., and McCallum, A. *Word representations via gaussian embedding*,arXiv preprint arXiv:1412.6623, 2014.
9. Athiwaratkun, B.,and Wilson, A. G. *Multimodal word distributions*,arXiv preprint arXiv:1704.08424, 2017.
10. Le, Q., and Mikolov, T. *Distributed representations of sentences and documents*,In International conference on machine learning, pp. 1188-1196, PMLR, 2014.
11. Mikolov, T., Chen, K., Corrado, G., and Dean, J. *Efficient estimation of word representations in vector space*,arXiv preprint arXiv:1301.3781, 2013.
12. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. *Distributed representations of words and phrases and their compositionality*,Advances in neural information processing systems, 26, 2013.
13. Pennington, J., Socher, R., and Manning, C. D. *Glove: Global vectors for word representation*,In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532-1543, 2014.
14. Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. *Enriching word vectors with subword information*,Transactions of the association for computational linguistics, 5, 135-146, 2017.
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, pp. 51-61, 2016.
16. McCann, B., Bradbury, J., Xiong, C., and Socher, R. *Learned in translation: Contextualized word vectors*,Advances in neural information processing systems, 30, 2017.
17. Peters, M. E., Neumann M., Iyyer M., Gardner M., Clark C., Lee K., and 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, Vol 1, pages 2227–2237, 2018.
18. Howard, J., & Ruder, S. *Universal language model fine-tuning for text classification*,arXiv preprint arXiv:1801.06146, 2018.
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*,Advances in Neural Information Processing Systems, 32, 2019.
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 preprint arXiv:1406.1078, 2014.
21. Sutskever, I., Vinyals, O., & Le, Q. V. *Sequence to sequence learning with neural networks*,Advances in neural information processing systems, 27, 2014.
22. Bahdanau, D., Cho, K., & Bengio, Y. *Neural machine translation by jointly learning to align and translate*,arXiv preprint arXiv:1409.0473, 2014.
23. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. & Polosukhin, I. *Attention is all you need*,Advances in neural information processing systems, 30, 2017.

24. Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. *Improving language understanding by generative pre-training*,URL https://s3-us-west-2. amazonaws. com/openaiassets/research-covers/languageunsupervised/language understanding paper.pdf, 2018.
25. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. *Bert: Pre-training of deep bidirectional transformers for language understanding*,arXiv preprint arXiv:1810.04805, 2018.
26. 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*,The Journal of Machine Learning Research, 21(1), pp. 5485-5551, 2020.
27. Guu, K., Lee, K., Tung, Z., Pasupat, P., & Chang, M. *Retrieval augmented language model pre-training*,In International conference on machine learning, pp. 3929-3938, 2020, November, PMLR.
28. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. *Language models are unsupervised multitask learners*,OpenAI blog, 1(8), 9, 2019.
29. Brown. T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. & Agarwal, S. *Language models are few-shot learners*,Advances in neural information processing systems, 33, pp.1877-1901, 2020.
30. Lieber, O., Sharir, O., Lenz, B., & Shoham, Y. *Jurassic-1: Technical details and evaluation*,White Paper. AI21 Labs, 1, 2021.
31. Liu, X., Zheng, Y., Du, Z., Ding, M., Qian, Y., Yang, Z. & Tang, J., *GPT understands, too*,AI Open, 2023.
32. Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A. and Raffel, C., *mT5: A massively multilingual pre-trained text-to-text transformer*,arxiv preprint arXiv:2010.11934, 2020.
33. Zeng, W., Ren, X., Su, T., Wang, H., Liao, Y., Wang, Z., ... & Tian, Y. *Pangu-α: Large-scale autoregressive pretrained Chinese language models with auto-parallel computation.* ,arxiv preprint arXiv:2104.12369, 2021.
34. Zhang, Z., Gu, Y., Han, X., Chen, S., Xiao, C., Sun, Z., Yao, Y., Qi, F., Guan, J., Ke, P. and Cai, Y., *Cpm-2: Large-scale cost-effective pre-trained language models*,AI Open, 2, pp.216-224, 2021.
35. Wu, S., Zhao, X., Yu, T., Zhang, R., Shen, C., Liu, H., Li, F., Zhu, H., Luo, J., Xu, L. and Zhang, X., *Yuan 1.0: Large-scale pre-trained language model in zero-shot and few-shot learning*,arxiv preprint arXiv:2110.04725, 2021.
36. Kim, B., Kim, H., Lee, S.W., Lee, G., Kwak, D., Jeon, D.H., Park, S., Kim, S., Kim, S., Seo, D. and Lee, H., *What changes can large-scale language models bring? intensive study on hyperclova: Billions-scale korean generative pretrained transformers*,arxiv preprint arXiv:2109.04650, 2021.
37. Wei, J., Bosma, M., Zhao, V.Y., Guu, K., Yu, A.W., Lester, B., Du, N., Dai, A.M. and Le, Q.V., *Finetuned language models are zero-shot learners*,arxiv preprint arXiv:2109.01652, 2021.
38. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A. and Schulman, J., *Training language models to follow instructions with human feedback*,Advances in Neural Information Processing Systems, 35, pp.27730-27744, 2022.
39. Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.T., Jin, A., Bos, T., Baker, L., Du, Y. and Li, Y., *Lamda: Language models for dialog applications*,arxiv preprint arXiv:2201.08239, 2022.
40. Du, Z., Qian, Y., Liu, X., Ding, M., Qiu, J., Yang, Z. and Tang, J., *Glm: General language model pretraining with autoregressive blank infilling*,arXiv preprint arXiv:2103.10360, 2021.
41. Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X. and Tam, W.L., *Glm-130b: An open bilingual pre-trained model*,arxiv preprint arXiv:2210.02414, 2022.
42. Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Arunkumar, A., Ashok, A., Dhanasekaran, A.S., Naik, A., Stap, D. and Pathak, E., *Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks*,arxiv preprint arXiv:2204.07705, 2022.
43. Sanh, V., Webson, A., Raffel, C., Bach, S.H., Sutawika, L., Alyafeai, Z., Chaffin, A., Stiegler, A., Scao, T.L., Raja, A. and Dey, M., *Multitask prompted training enables zero-shot task generalization*,arxiv preprint arXiv:2110.08207, 2021.
44. Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J. and Pieler, M., *Gpt-neox-20b: An open-source autoregressive language model*,arxiv preprint arXiv:2204.06745, 2022.
45. Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X.V. and Mihaylov, T., *Opt: Open pre-trained transformer language models*,URL https://arxiv. org/abs/2205.01068, 2022.

46. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F. and Rodriguez, A., *Llama: Open and efficient foundation language models*,arxiv preprint arXiv:2302.13971, 2023.
47. Du, N., Huang, Y., Dai, A.M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A.W., Firat, O. and Zoph, B., *Glam: Efficient scaling of language models with mixture-of-experts*,In International Conference on Machine Learning, pp. 5547-5569, PMLR, June, 2022.
48. Soltan, S., Ananthakrishnan, S., FitzGerald, J., Gupta, R., Hamza, W., Khan, H., Peris, C., Rawls, S., Rosenbaum, A., Rumshisky, A. and Prakash, C.S., *Alexatm 20b: Few-shot learning using a large-scale multilingual seq2seq model*,arxiv preprint arXiv:2208.01448, 2022.
49. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.D.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G. and Ray, A., *Evaluating large language models trained on code*,arxiv preprint arXiv:2107.03374, 2021.
50. So, D. R., Mańke, W., Liu, H., Dai, Z., Shazeer, N., & Le, Q. V. *Primer: Searching for efficient transformers for language modeling*,arxiv preprint arXiv:2109.08668, 2021.
51. Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N.A., Khashabi, D. and Hajishirzi, H., *Self-instruct: Aligning language model with self generated instructions*,arXiv preprint arXiv:2212.10560, 2022.
52. Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T. and Wu, Y., *Solving quantitative reasoning problems with language models*,Advances in Neural Information Processing Systems, 35, pp.3843-3857, 2022.
53. 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 preprint arXiv:2209.10372, 2022.
54. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H.W., Sutton, C., Gehrmann, S. and Schuh, P., *Palm: Scaling language modeling with pathways*,arxiv preprint arXiv:2204.02311, 2022.
55. Scao, T.L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A.S., Yvon, F., Gallé, M. and Tow, J., *Bloom: A 176b-parameter open-access multilingual language model*,arxiv preprint arXiv:2211.05100, 2022.
56. Tay, Y., Dehghani, M., Tran, V.Q., Garcia, X., Wei, J., Wang, X., Chung, H.W., Bahri, D., Schuster, T., Zheng, S. and Zhou, D., *Ul2: Unifying language learning paradigms*,In The Eleventh International Conference on Learning Representations September, 2022.
57. Iyer, S., Lin, X.V., Pasunuru, R., Mihaylov, T., Simig, D., Yu, P., Shuster, K., Wang, T., Liu, Q., Koura, P.S. and Li, X., *Opt-iml: Scaling language model instruction meta learning through the lens of generalization*,arxiv preprint arXiv:2212.12017, 2022.
58. Muennighoff, N., Wang, T., Sutawika, L., Roberts, A., Biderman, S., Scao, T.L., Bari, M.S., Shen, S., Yong, Z.X., Schoelkopf, H. and Tang, X., *Crosslingual generalization through multitask finetuning*,arxiv preprint arXiv:2211.01786, 2022.
59. Lin, X.V., Mihaylov, T., Artetxe, M., Wang, T., Chen, S., Simig, D., Ott, M., Goyal, N., Bhosale, S., Du, J. and Pasunuru, R., *Few-shot Learning with Multilingual Generative Language Models*,In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, (pp. 9019-9052), 2022.
60. Tay, Y., Wei, J., Chung, H.W., Tran, V.Q., So, D.R., Shakeri, S., Garcia, X., Zheng, H.S., Rao, J., Chowdhery, A. and Zhou, D., *Transcending scaling laws with 0.1% extra compute*,arxiv preprint arXiv:2210.11399, 2022.
61. Chung, H.W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, E., Wang, X., Dehghani, M., Brahma, S. and Webson, A., *Scaling instruction-finetuned language models*,arxiv preprint arXiv:2210.11416, 2022.
62. Chen, X., Wang, X., Changpinyo, S., Piergiovanni, A.J., Padlewski, P., Salz, D., Goodman, S., Grycner, A., Mustafa, B., Beyer, L. and Kolesnikov, A., *Pali: A jointly-scaled multilingual language-image model*,URL https://arxiv. org/abs/2209.06794, 2022.
63. Taylor, R., Kardas, M., Cucurull, G., Scialom, T., Hartshorn, A., Saravia, E., Poulton, A., Kerkez, V. and Stojnic, R., *Galactica: A large language model for science*,arxiv preprint arXiv:2211.09085, 2022.
64. Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A. and Hubert, T., *Competition-level code generation with alphacode*,Science, 378(6624), pp.1092-1097, 2022.
65. Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S. and Xiong, C., *Codegen: An open large language model for code with multi-turn program synthesis*,arxiv preprint arXiv:2203.13474, 2022.

66. Zheng, Q., Xia, X., Zou, X., Dong, Y., Wang, S., Xue, Y., Wang, Z., Shen, L., Wang, A., Li, Y. and Su, T., *Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x*,arxiv preprint arXiv:2303.17568, 2023.
67. Costa-jussà, M.R., Cross, J., Çelebi, O., Elbayad, M., Heafield, K., Heffernan, K., Kalbassi, E., Lam, J., Licht, D., Maillard, J. and Sun, A., *No language left behind: Scaling human-centered machine translation*,arxiv preprint arXiv:2207.04672, 2022.
68. Biderman, S., Schoelkopf, H., Anthony, Q.G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M.A., Purohit, S., Prashanth, U.S., Raff, E. and Skowron, A., *Pythia: A suite for analyzing large language models across training and scaling*,In International Conference on Machine Learning, pp. 2397-2430, July, 2023, PMLR.
69. Ziegler, D.M., Stiennon, N., Wu, J., Brown, T.B., Radford, A., Amodei, D., Christiano, P. and Irving, G., *Fine-tuning language models from human preferences*,arXiv preprint arXiv:1909.08593, 2019.
70. Wu, J., Ouyang, L., Ziegler, D.M., Stiennon, N., Lowe, R., Leike, J. and Christiano, P., *Recursively summarizing books with human feedback*,arXiv preprint arXiv:2109.10862, 2021.
71. Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Voss, C., Radford, A., Amodei, D. and Christiano, P.F., *Learning to summarize with human feedback*,Advances in Neural Information Processing Systems, 33, pp.3008-3021, 2020.
72. Madaan, A., Tandon, N., Clark, P. and Yang, Y., *Memory-assisted prompt editing to improve gpt-3 after deployment*,arXiv preprint arXiv:2201.06009, 2022.
73. Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G. and Dean, J., *Outrageously large neural networks: The sparsely-gated mixture-of-experts layer*,arXiv preprint arXiv:1701.06538, 2017.
74. Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N. and Chen, Z., *GShard: Scaling giant models with conditional computation and automatic sharding*,arxiv preprint arXiv:2006.16668,2020.
75. Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N. and Fedus, W., *St-moe: Designing stable and transferable sparse expert models*,arXiv preprint arXiv:2202.08906, 2022.
76. Fedus, W., Zoph, B. and Shazeer, N., *Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity*,The Journal of Machine Learning Research, 23(1), pp.5232-5270, 2022.
77. Artetxe, M., Bhosale, S., Goyal, N., Mihaylov, T., Ott, M., Shleifer, S., Lin, X.V., Du, J., Iyer, S., Pasunuru, R. and Anantharaman, G., *Efficient large scale language modeling with mixtures of experts*,arXiv preprint arXiv:2112.10684, 2021.
78. Rae, J.W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S. and Rutherford, E., *Scaling language models: Methods, analysis & insights from training gopher*,arxiv preprint arXiv:2112.11446, 2021.
79. Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D.D.L., Hendricks, L.A., Welbl, J., Clark, A. and Hennigan, T., *Training compute-optimal large language models*,arXiv preprint arXiv:2203.15556, 2022.
80. Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. and Amodei, D., *Scaling laws for neural language models*,arXiv preprint arXiv:2001.08361, 2020.
81. Zhao, Z., Wallace, E., Feng, S., Klein, D. and Singh, S., *Calibrate before use: Improving few-shot performance of language models*,In International Conference on Machine Learning, pp. 12697-12706. PMLR, 2021.
82. Tay, Y., Dehghani, M., Rao, J., Fedus, W., Abnar, S., Chung, H.W., Narang, S., Yogatama, D., Vaswani, A. and Metzler, D., *Scale efficiently: Insights from pre-training and fine-tuning transformers*,arXiv preprint arXiv:2109.10686, 2021.
83. Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D. and Chi, E.H., *Emergent abilities of large language models*,arXiv preprint arXiv:2206.07682, 2022.
84. Zhang, Z., Han, X., Liu, Z., Jiang, X., Sun, M. and Liu, Q., *ERNIE: Enhanced language representation with informative entities*,arXiv preprint arXiv:1905.07129, 2019.
85. Peters, M.E., Neumann, M., Logan IV, R.L., Schwartz, R., Joshi, V., Singh, S. and Smith, N.A., *Knowledge enhanced contextual word representations*,arXiv preprint arXiv:1909.04164, 2019.
86. Xiong, W., Du, J., Wang, W.Y. and Stoyanov, V., *Pretrained encyclopedia: Weakly supervised knowledge-pretrained language model*,arXiv preprint arXiv:1912.09637, 2019.
87. Zhou, W., Lee, D.H., Selvam, R.K., Lee, S., Lin, B.Y. and Ren, X., *Pre-training text-to-text transformers for concept-centric common sense*,arXiv preprint arXiv:2011.07956, 2020.

88. Wang, X., Gao, T., Zhu, Z., Zhang, Z., Liu, Z., Li, J. and Tang, J., *KEPLER: A unified model for knowledge embedding and pre-trained language representation*,Transactions of the Association for Computational Linguistics, 9, pp.176-194, 2021.
89. Sun, T., Shao, Y., Qiu, X., Guo, Q., Hu, Y., Huang, X. and Zhang, Z., *Colake: Contextualized language and knowledge embedding*,arXiv preprint arXiv:2010.00309, 2020.
90. Wang, R., Tang, D., Duan, N., Wei, Z., Huang, X., Cao, G., Jiang, D. and Zhou, M., *K-adapter: Infusing knowledge into pre-trained models with adapters*,arXiv preprint arXiv:2002.01808, 2020.
91. Sun, Y., Wang, S., Feng, S., Ding, S., Pang, C., Shang, J., Liu, J., Chen, X., Zhao, Y., Lu, Y. and Liu, W., *Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation*,arxiv preprint arXiv:2107.02137, 2021.
92. Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M. and Gelly, S., *Parameter-efficient transfer learning for NLP*,In International Conference on Machine Learning, pp. 2790-2799, PMLR, 2019.
93. Shin, T., Razeghi, Y., Logan IV, R.L., Wallace, E. and Singh, S., *Autoprompt: Eliciting knowledge from language models with automatically generated prompts*,arXiv preprint arXiv:2010.15980, 2020.
94. Li, X.L. and Liang, P., *Prefix-tuning: Optimizing continuous prompts for generation*,arXiv preprint arXiv:2101.00190, 2021.
95. Han, X., Zhao, W., Ding, N., Liu, Z. and Sun, M., *Ptr: Prompt tuning with rules for text classification*,AI Open, 3, pp.182-192, 2022.
96. Lester, B., Al-Rfou, R. and Constant, N., *The power of scale for parameter-efficient prompt tuning*,arXiv preprint arXiv:2104.08691, 2021.
97. Mosbach, M., Pimentel, T., Ravfogel, S., Klakow, D. and Elazar, Y., *Few-shot Fine-tuning vs. In-context Learning: A Fair Comparison and Evaluation*,arXiv preprint arXiv:2305.16938, 2023.
98. Wang, T., Roberts, A., Hesslow, D., Le Scao, T., Chung, H.W., Beltagy, I., Launay, J. and Raffel, C., *What language model architecture and pretraining objective works best for zero-shot generalization?*,In International Conference on Machine Learning, pp. 22964-22984. PMLR, 2022.
99. Liu, J., Shen, D., Zhang, Y., Dolan, B., Carin, L. and Chen, W., *What Makes Good In-Context Examples for GPT-3?*,arXiv preprint arXiv:2101.06804, 2021.
100. Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L. and Stoyanov, V., *Unsupervised cross-lingual representation learning at scale*,arXiv preprint arXiv:1911.02116, 2019.
101. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V. and Zhou, D., *Chain-of-thought prompting elicits reasoning in large language models*,Advances in Neural Information Processing Systems, 35, pp.24824-24837, 2022.
102. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A. and Zhou, D., *Self-consistency improves chain of thought reasoning in language models*,arXiv preprint arXiv:2203.11171, 2022.
103. Lin, S., Hilton, J. and Evans, O., *Truthfulqa: Measuring how models mimic human falsehoods*,arXiv preprint arXiv:2109.07958, 2021.
104. Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.M., Rothchild, D., So, D., Texier, M. and Dean, J., *Carbon emissions and large neural network training*,arXiv preprint arXiv:2104.10350, 2021.
105. Gehman, S., Gururangan, S., Sap, M., Choi, Y. and Smith, N.A., *Realtoxicityprompts: Evaluating neural toxic degeneration in language models*,arXiv preprint arXiv:2009.11462, 2020.
106. Ung, M., Xu, J. and Boureau, Y.L., *Saferdialogues: Taking feedback gracefully after conversational safety failures*,arXiv preprint arXiv:2110.07518, 2021.
107. Dinan, E., Abercrombie, G., Bergman, A.S., Spruit, S., Hovy, D., Boureau, Y.L. and Rieser, V., *Anticipating safety issues in e2e conversational ai: Framework and tooling*,arXiv preprint arXiv:2107.03451, 2021.
108. Rudinger, R., Naradowsky, J., Leonard, B. and Van Durme, B., *Gender bias in coreference resolution*,arXiv preprint arXiv:1804.09301, 2018.
109. Nangia, N., Vania, C., Bhalerao, R. and Bowman, S.R., *CrowS-pairs: A challenge dataset for measuring social biases in masked language models*,arXiv preprint arXiv:2010.00133, 2020.
110. Nadeem, M., Bethke, A. and Reddy, S., *StereoSet: Measuring stereotypical bias in pretrained language models*,arXiv preprint arXiv:2004.09456, 2020.

 doi:10.20944/preprints202402.0357.v1

111. Levine, Y., Wies, N., Sharir, O., Bata, H. and Shashua, A., *Limits to depth efficiencies of self-attention*,Advances in Neural Information Processing Systems, 33, pp.22640-22651, 2020.
112. Lester, B., Al-Rfou, R. and Constant, N., *The power of scale for parameter-efficient prompt tuning*,arXiv preprint arXiv:2104.08691, 2021.