Large Language Models: A Survey

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu
Richard Socher, Xavier Amatriain, Jianfeng Gao

***Abstract*—Large Language Models (LLMs) have drawn a lot of attention due to their strong performance on a wide range of natural language tasks, since the release of ChatGPT in November 2022. LLMs' ability of general-purpose language understanding and generation is acquired by training billions of model's parameters on massive amounts of text data, as predicted by scaling laws [1], [2]. The research area of LLMs, while very recent, is evolving rapidly in many different ways. In this paper, we review some of the most prominent LLMs, including three popular LLM families (GPT, LLaMA, PaLM), and discuss their characteristics, contributions and limitations. We also give an overview of techniques developed to build, and augment LLMs. We then survey popular datasets prepared for LLM training, fine-tuning, and evaluation, review widely used LLM evaluation metrics, and compare the performance of several popular LLMs on a set of representative benchmarks. Finally, we conclude the paper by discussing open challenges and future research directions.**

I. Introduction

Language modeling is a long-standing research topic, dating back to the 1950s with Shannon's application of information theory to human language, where he measured how well simple n-gram language models predict or compress natural language text [3]. Since then, statistical language modeling became fundamental to many natural language understanding and generation tasks, ranging from speech recognition, machine translation, to information retrieval [4], [5], [6].

The recent advances on transformer-based large language models (LLMs), pretrained on Web-scale text corpora, significantly extended the capabilities of language models (LLMs). For example, OpenAI's ChatGPT and GPT-4 can be used not only for natural language processing, but also as general task solvers to power Microsoft's Co-Pilot systems, for instance, can follow human instructions of complex new tasks performing multi-step reasoning when needed. LLMs are thus becoming the basic building block for the development of general-purpose AI agents or artificial general intelligence (AGI).

As the field of LLMs is moving fast, with new findings, models and techniques being published in a matter of months or weeks [7], [8], [9], [10], [11], AI researchers and practitioners often find it challenging to figure out the best recipes to build LLM-powered AI systems for their tasks. This paper gives a timely survey of the recent advances on LLMs. We hope this survey will prove a valuable and accessible resource for students, researchers and developers.

LLMs are large-scale, pre-trained, statistical language models based on neural networks. The recent success of LLMs is an accumulation of decades of research and development of language models, which can be categorized into four waves that have different starting points and velocity: statistical language models, neural language models, pre-trained language models and LLMs.

Statistical language models (SLMs) view text as a sequence of words, and estimate the probability of text as the product of their word probabilities. The dominating form of SLMs are Markov chain models known as the n-gram models, which compute the probability of a word conditioned on its immediate proceeding $n-1$ words. Since word probabilities are estimated using word and n-gram counts collected from text corpora, the model needs to deal with data sparsity (i.e., assigning zero probabilities to unseen words or n-grams) by using *smoothing*, where some probability mass of the model is reserved for unseen n-grams [12]. N-gram models are widely used in many NLP systems. However, these models are incomplete in that they cannot fully capture the diversity and variability of natural language due to data sparsity.

Early neural language models (NLMs) [13], [14], [15], [16] deal with data sparsity by mapping words to low-dimensional continuous vectors (embedding vectors) and predict the next word based on the aggregation of the embedding vectors of its proceeding words using neural networks. The embedding vectors learned by NLMs define a hidden space where the semantic similarity between vectors can be readily computed as their distance. This opens the door to computing semantic similarity of any two inputs regardless their forms (e.g., queries vs. documents in Web search [17], [18], sentences in different languages in machine translation [19], [20]) or modalities (e.g., image and text in image captioning [21], [22]). Early NLMs are task-specific models, in that they are trained on task-specific data and their learned hidden space is task-specific.

Pre-trained language models (PLMs), unlike early NLMs, are task-agnostic. This generality also extends to the learned hidden embedding space. The training and inference of PLMs follows the *pre-training and fine-tuning* paradigm, where language models with recurrent neural networks [23] or transformers [24], [25], [26] are pre-trained on Web-scale unlabeled text corpora for general tasks such as word prediction, and then finetuned to specific tasks using small amounts of (labeled) task-specific data. Recent surveys on PLMs include [8], [27], [28].

Large language models (LLMs) mainly refer to transformer-based neural language models [1] that contain tens to hundreds of billions of parameters, which are pre-trained on massive text data, such as PaLM [31], LLaMA [32], and GPT-4 [33], as summarized in Table III. Compared

[1]Recently, several very promising non-transformer LLMs have been proposed, such as the LLMs based on structured state space models [29], [30]. See Section VII for more details.

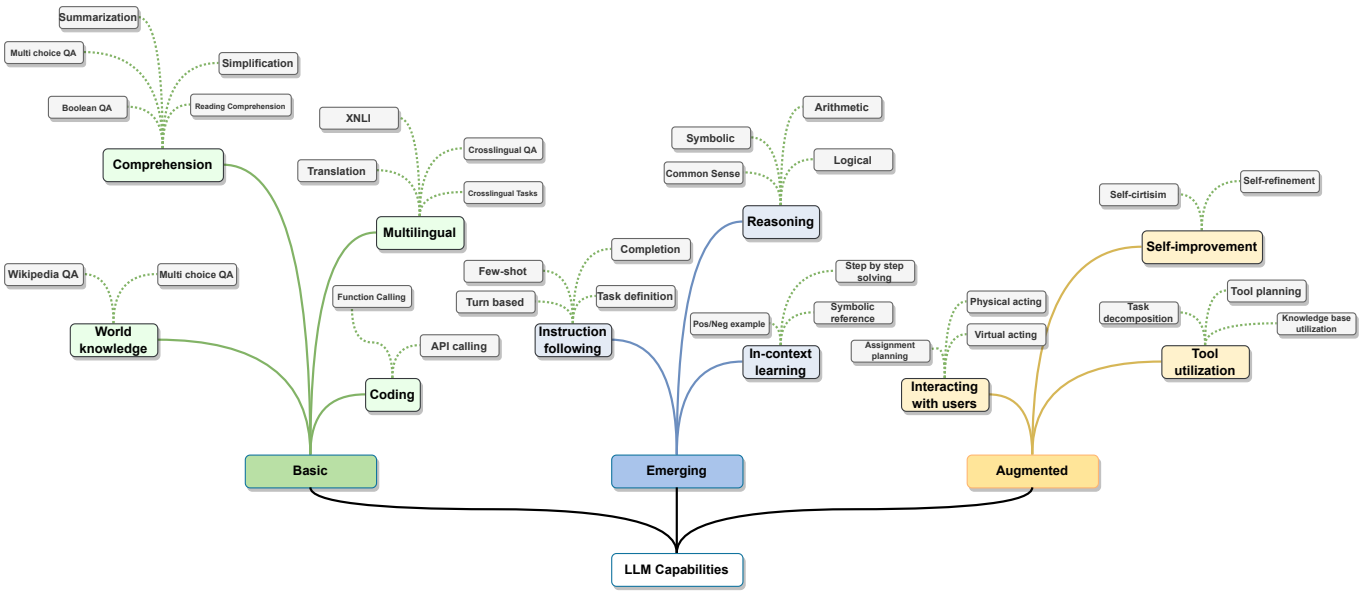


Fig. 1: LLM Capabilities.

to PLMs, LLMs are not only much larger in model size, but also exhibit stronger language understanding and generation abilities, and more importantly, emergent abilities that are not present in smaller-scale language models. As illustrated in Fig. 1, these emergent abilities include (1) in-context learning, where LLMs learn a new task from a small set of examples presented in the prompt at inference time, (2) instruction following, where LLMs, after instruction tuning, can follow the instructions for new types of tasks without using explicit examples, and (3) multi-step reasoning, where LLMs can solve a complex task by breaking down that task into intermediate reasoning steps as demonstrated in the chain-of-thought prompt [34]. LLMs can also be augmented by using external knowledge and tools [35], [36] so that they can effectively interact with users and environment [37], and continually improve itself using feedback data collected through interactions (e.g. via reinforcement learning with human feedback (RLHF)).

Through advanced usage and augmentation techniques, LLMs can be deployed as so-called AI agents: artificial entities that sense their environment, make decisions, and take actions. Previous research has focused on developing agents for specific tasks and domains. The emergent abilities demonstrated by LLMs make it possible to build general-purpose AI agents based on LLMs. While LLMs are trained to produce responses in static settings, AI agents need to take actions to interact with dynamic environment. Therefore, LLM-based agents often need to augment LLMs to e.g., obtain updated information from external knowledge bases, verify whether a system action produces the expected result, and cope with when things do not go as expected, etc. We will discuss in detail LLM-based agents in Section IV.

In the rest of this paper, Section II presents an overview of state of the art of LLMs, focusing on three LLM families (GPT, LLaMA and PaLM) and other representative models. Section III discusses how LLMs are built. Section IV discusses how LLMs are used, and augmented for real-world applications Sections V and VI review popular datasets and benchmarks for evaluating LLMs, and summarize the reported LLM evaluation results. Finally, Section VII concludes the paper by summarizing the challenges and future research directions.

II. Large Language Models

In this section we start with a review of early pre-trained neural language models as they are the base of LLMs, and then focus our discussion on three families of LLMs: GPT, LlaMA, and PaLM. Table I provides an overview of some of these models and their characteristics.

A. Early Pre-trained Neural Language Models

Language modeling using neural networks was pioneered by [38], [39], [40]. Bengio et al. [13] developed one of the first neural language models (NLMs) that are comparable to n-gram models. Then, [14] successfully applied NLMs to machine translation. The release of RNNLM (an open source NLM toolkit) by Mikolov [41], [42] helped significantly popularize NLMs. Afterwards, NLMs based on recurrent neural networks (RNNs) and their variants, such as long short-term memory (LSTM) [19] and gated recurrent unit (GRU) [20], were widely used for many natural language applications including machine translation, text generation and text classification [43].

Then, the invention of the Transformer architecture [44] marks another milestone in the development of NLMs. By applying self-attention to compute in parallel for every word in a sentence or document an "attention score" to model the influence each word has on another, Transformers allow for much more parallelization than RNNs, which makes it possible to efficiently pre-train very big language models on large amounts of data on GPUs. These pre-trained language models (PLMs) can be fine-tuned for many downstream tasks.

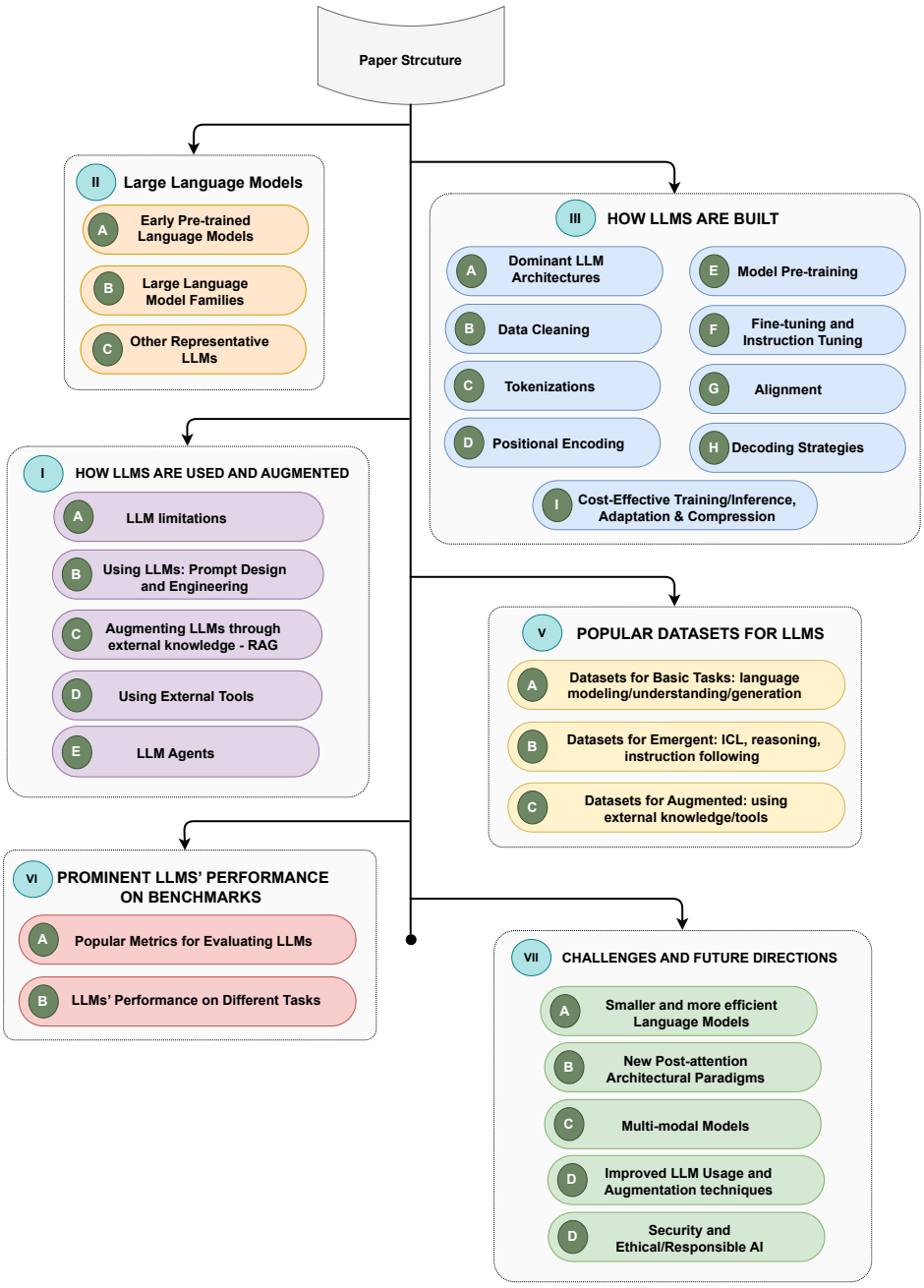


Fig. 2: The paper structure.

We group early popular Transformer-based PLMs, based on their neural architectures, into three main categories: encoder-only, decoder-only, and encoder-decoder models. Comprehensive surveys of early PLMs are provided in [43], [28].

1) Encoder-only PLMs: As the name suggests, the encoder-only models only consist of an encoder network. These models are originally developed for language understanding tasks, such as text classification, where the models need to predict a class label for an input text. Representative encoder-only models include BERT and its variants, e.g., RoBERTa, ALBERT, DeBERTa, XLM, XLNet, UNILM, as to be described below.

BERT (Birectional Encoder Representations from Transformers) [24] is one of the most widely used encoder-only language models. BERT consists of three modules: (1) an embedding module that converts input text into a sequence of embedding vectors, (2) a stack of Transformer encoders that converts embedding vectors into contextual representation vectors, and (3) a fully connected layer that converts the representation vectors (at the final layer) to one-hot vectors. BERT is pre-trained uses two objectives: masked language modeling (MLM) and next sentence prediction. The pre-trained BERT model can be fine-tuned by adding a classifier layer for many language understanding tasks, ranging from text

TABLE I: High-level Overview of Popular Language Models

Type	Model Name	#Parameters	Release	Base Models	Open Source	#Tokens	Training dataset
Encoder-Only	BERT	110M, 340M	2018	-	✓	137B	BooksCorpus, English Wikipedia
	RoBERTa	355M	2019	-	✓	2.2T	BooksCorpus, English Wikipedia, CC-NEWS, STORIES (a subset of Common Crawl), Reddit
	ALBERT	12M, 18M, 60M, 235M	2019	-	✓	137B	BooksCorpus, English Wikipedia
	DeBERTa	-	2020	-	✓	-	BooksCorpus, English Wikipedia, STORIES, Reddit content
	XLNet	110M, 340M	2019	-	✓	32.89B	BooksCorpus, English Wikipedia, Giga5, Common Crawl, ClueWeb 2012-B
Decoder-only	GPT-1	120M	2018	-	✓	1.3B	BooksCorpus
	GPT-2	1.5B	2019	-	✓	10B	Reddit outbound
Encoder-Decoder	T5 (Base)	223M	2019	-	✓	156B	Common Crawl
	MT5 (Base)	300M	2020	-	✓	-	New Common Crawl-based dataset in 101 languages (m Common Crawl)
	BART (Base)	139M	2019	-	✓	-	Corrupting text
GPT Family	GPT-3	125M, 350M, 760M, 1.3B, 2.7B, 6.7B, 13B, 175B	2020		×	300B	Common Crawl (filtered), WebText2, Books1, Books2, Wikipedia
	CODEX	12B	2021	GPT	✓	-	Public GitHub software repositories
	WebGPT	760M, 13B, 175B	2021	GPT-3	×	-	ELI5
	GPT-4	1.76T	2023	-	×	13T	-
LLaMA Family	LLaMA1	7B, 13B, 33B, 65B	2023	-	✓	1T, 1.4T	Online sources
	LLaMA2	7B, 13B, 34B, 70B	2023	-	✓	2T	Online sources
	Alpaca	7B	2023	LLaMA1	✓	-	GPT-3.5
	Vicuna-13B	13B	2023	LLaMA1	✓	-	GPT-3.5
	Koala	13B	2023	LLaMA	✓	-	Dialogue data
	Mistral-7B	7.3B	2023		✓	-	-
	Code Llama	34	2023	LLaMA2	✓	500B	Publicly available code
	LongLLaMA	3B, 7B	2023	OpenLLaMA	✓	1T	-
	LLaMA-Pro-8B	8.3B	2024	LLaMA2-7B	✓	80B	Code and math corpora
	TinyLlama-1.1B	1.1B	2024	LLaMA1.1B	✓	3T	SlimPajama, Starcoderdata
PaLM Family	PaLM	8B, 62B, 540B	2022	-	×	780B	Web documents, books, Wikipedia, conversations, GitHub code
	U-PaLM	8B, 62B, 540B	2022	-	×	1.3B	Web documents, books, Wikipedia, conversations, GitHub code
	PaLM-2	340B	2023	-	✓	3.6T	Web documents, books, code, mathematics, conversational data
	Med-PaLM	540B	2022	PaLM	×	780B	HealthSearchQA, MedicationQA, LiveQA
	Med-PaLM 2	-	2023	PaLM 2	×	-	MedQA, MedMCQA, HealthSearchQA, LiveQA, MedicationQA
Other Popular LLMs	FLAN	137B	2021	LaMDA-PT	✓	-	Web documents, code, dialog data, Wikipedia
	Gopher	280B	2021	-	×	300B	MassiveText
	ERNIE 4.0	10B	2023	-	×	4TB	Chinese text
	Retro	7.5B	2021	-	×	600B	MassiveText
	LaMDA	137B	2022	-	×	168B	public dialog data and web documents
	ChinChilla	70B	2022	-	×	1.4T	MassiveText
	Galactia-120B	120B	2022	-		450B	
	CodeGen	16.1B	2022	-	✓	-	THE PILE, BIGQUERY, BIGPYTHON
	BLOOM	176B	2022	-	✓	366B	ROOTS
	Zephyr	7.24B	2023	Mistral-7B	✓	800B	Synthetic data
	Grok-0	33B	2023	-	×	-	Online source
	ORCA-2	13B	2023	LLaMA2	-	2001B	-
	StartCoder	15.5B	2023	-	✓	35B	GitHub
	MPT	7B	2023	-	✓	1T	RedPajama, m Common Crawl, S2ORC, Common Crawl
	Mixtral-8x7B	46.7B	2023	-	✓	-	Instruction dataset
	Falcon 180B	180B	2023	-	✓	3.5T	RefinedWeb
	Gemini	1.8B, 3.25B	2023		✓	-	Web documents, books, and code, image data, audio data, video data
	DeepSeek-Coder	1.3B, 6.7B, 33B	2024	-	✓	2T	GitHub's Markdown and StackExchange
	DocLLM	1B,7B	2024	-	×	2T	IIT-CDIP Test Collection 1.0, DocBank

classification, question answering to language inference. A high-level overview of BERT framework is shown in Fig 3. As BERT significantly improved state of the art on a wide range of language understanding tasks when it was published, the AI community was inspired to develop many similar encoder-only language models based on BERT.

RoBERTa [25] significantly improves the robustness of BERT using a set of model design choices and training strategies, such as modifying a few key hyperparameters, removing the next-sentence pre-training objective and training with much larger mini-batches and learning rates. ALBERT [45] uses two parameter-reduction techniques to lower memory consumption and increase the training speed of BERT: (1) splitting the embedding matrix into two smaller matrices, and (2) using repeating layers split among groups. DeBERTa (Decoding-enhanced BERT with disentangled attention) [26] improves the BERT and RoBERTa models using two novel techniques. The first is the disentangled attention mechanism, where each word is represented using two vectors that encode its content and position, respectively, and the attention weights among words

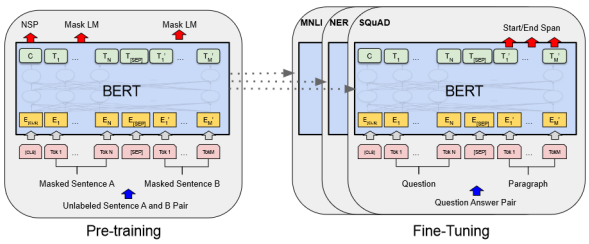


Fig. 3: Overall pre-training and fine-tuning procedures for BERT. Courtesy of [24]

are computed using disentangled matrices on their contents and relative positions, respectively. Second, an enhanced mask decoder is used to incorporate absolute positions in the decoding layer to predict the masked tokens in model pre-training. In addition, a novel virtual adversarial training method is used for fine-tuning to improve models' generalization. ELECTRA [46] uses a new pre-training task, known as replaced token detection (RTD), which is empirically proven to be more sample-efficient than MLM. Instead of masking the input, RTD corrupts it by replacing some tokens with plausible alternatives sampled from a small generator network. Then, instead of training a model that predicts the original identities of the corrupted tokens, a discriminative model is trained to predict whether a token in the corrupted input was replaced by a generated sample or not. RTD is more sample-efficient than MLM because the former is defined over all input tokens rather than just the small subset being masked out, as illustrated in Fig 4.

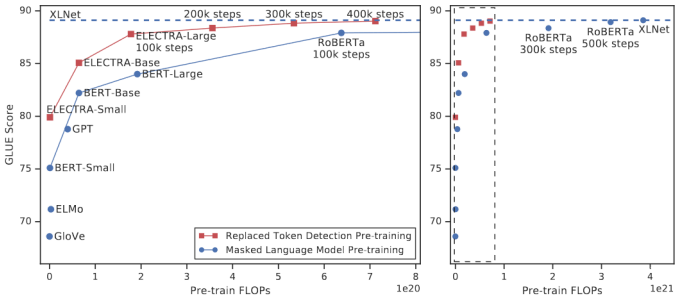


Fig. 4: A comparison between replaced token detection and masked language modeling. Courtesy of [46].

XLMs [47] extended BERT to cross-lingual language models using two methods: (1) a unsupervised method that only relies on monolingual data, and (2) a supervised method that leverages parallel data with a new cross-lingual language model objective, as illustrated in Fig 5. XLMs had obtained state-of-the-art results on cross-lingual classification, unsupervised and supervised machine translation, at the time they were proposed.

There are also encoder-only language models that leverage the advantages of auto-regressive (decoder) models for model training and inference. Two examples are XLNet and UNILM. XLNet [48] is based on Transformer-XL, pre-trained using a generalized autoregressive method that enables learning bidirectional contexts by maximizing the expected likelihood over

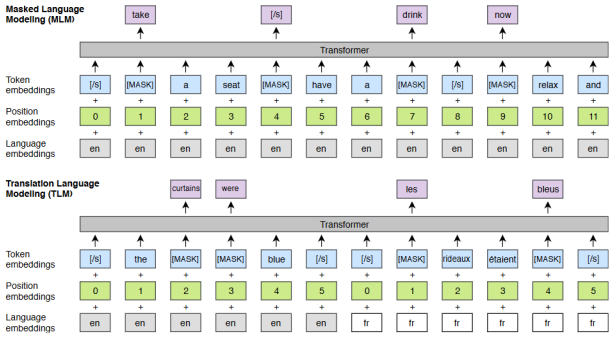


Fig. 5: Cross-lingual language model pretraining. The MLM objective is similar to BERT, but with continuous streams of text as opposed to sentence pairs. The TLM objective extends MLM to pairs of parallel sentences. To predict a masked English word, the model can attend to both the English sentence and its French translation, and is encouraged to align English and French representations. Courtesy of [47].

all permutations of the factorization order. UNILM (UNIfied pre-trained Language Model) [49] is pre-trained using three types of language modeling tasks: unidirectional, bidirectional, and sequence-to-sequence prediction. This is achieved by employing a shared Transformer network and utilizing specific self-attention masks to control what context the prediction is conditioned on, as illustrated in Fig 6. The pre-trained model can be fine-tuned for both natural language understanding and generation tasks.

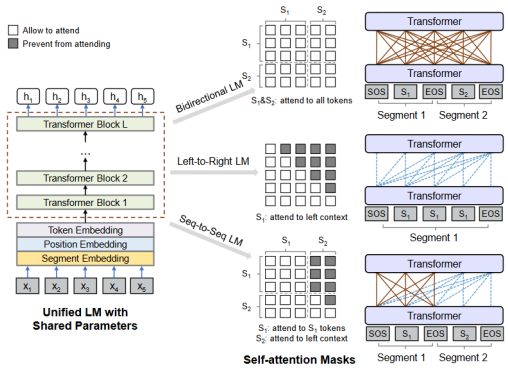


Fig. 6: Overview of unified LM pre-training. The model parameters are shared across the LM objectives (i.e., bidirectional LM, unidirectional LM, and sequence-to-sequence LM). Courtesy of [49].

2) Decoder-only PLMs: Two of the most widely used decoder-only PLMs are GPT-1 and GPT-2, developed by OpenAI. These models lay the foundation to more powerful LLMs subsequently, i.e., GPT-3 and GPT-4.

GPT-1 [50] demonstrates for the first time that good performance over a wide range of natural language tasks can be obtained by Generative Pre-Training (GPT) of a decoder-only Transformer model on a diverse corpus of unlabeled text in a self-supervised learning fashion (i.e., next word/token predic-

tion), followed by discriminative fine-tuning on each specific downstream task (with much fewer samples), as illustrated in Fig 7. GPT-1 paves the way for subsequent GPT models, with each version improving upon the architecture and achieving better performance on various language tasks.

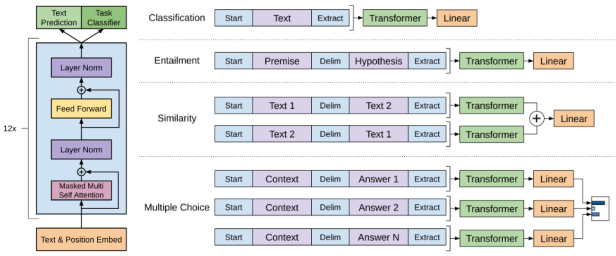


Fig. 7: High-level overview of GPT pretraining, and fine-tuning steps. Courtesy of OpenAI.

GPT-2 [51] shows that language models are able to learn to perform specific natural language tasks without any explicit supervision when trained on a large WebText dataset consisting of millions of webpages. The GPT-2 model follows the model designs of GPT-1 with a few modifications: Layer normalization is moved to the input of each sub-block, additional layer normalization is added after the final self-attention block, initialization is modified to account for the accumulation on the residual path and scaling the weights of residual layers, vocabulary size is expanded to 50,25, and context size is increased from 512 to 1024 tokens.

3) Encoder-Decoder PLMs: In [52], Raffle et al. shows that almost all NLP tasks can be cast as a sequence-to-sequence generation task. Thus, an encoder-decoder language model, by design, is a unified model in that it can perform all natural language understanding and generation tasks. Representative encoder-decoder PLMs we will review below are T5, mT5, MASS, and BART.

T5 [52] is a Text-to-Text Transfer Transformer (T5) model, where transfer learning is effectively exploited for NLP via an introduction of a unified framework in which all NLP tasks are cast as a text-to-text generation task. mT5 [53] is a multilingual variant of T5, which is pre-trained on a new Common Crawl-based dataset consisting of texts in 101 languages.

MASS (MAsked Sequence to Sequence pre-training) [54] adopts the encoder-decoder framework to reconstruct a sentence fragment given the remaining part of the sentence. The encoder takes a sentence with randomly masked fragment (several consecutive tokens) as input, and the decoder predicts the masked fragment. In this way, MASS jointly trains the encoder and decoder for language embedding and generation, respectively.

BART [55] uses a standard sequence-to-sequence translation model architecture. It is pre-trained by corrupting text with an arbitrary noising function, and then learning to reconstruct the original text.

B. Large Language Model Families

Large language models (LLMs) mainly refer to transformer-based PLMs that contain tens to hundreds of billions of parameters. Compared to PLMs reviewed above, LLMs are not only much larger in model size, but also exhibit stronger language understanding and generation and emergent abilities that are not present in smaller-scale models. In what follows, we review three LLM families: GPT, LLaMA, and PaLM, as illustrated in Fig 8.

*1) **The GPT Family**:* Generative Pre-trained Transformers (GPT) are a family of decoder-only Transformer-based language models, developed by OpenAI. This family consists of GPT-1, GPT-2, GPT-3, InstrucGPT, ChatGPT, GPT-4, CODEX, and WebGPT. Although early GPT models, such as GPT-1 and GPT-2, are open-source, recent models, such as GPT-3 and GPT-4, are close-source and can only be accessed via APIs. GPT-1 and GPT-2 models have been discussed in the early PLM subsection. We start with GPT-3 below.

GPT-3 [56] is a pre-trained autoregressive language model with 175 billion parameters. GPT-3 is widely considered as the first LLM in that it not only is much larger than previous PLMs, but also for the first time demonstrates emergent abilities that are not observed in previous smaller PLMs. GPT-3 shows the emergent ability of in-context learning, which means GPT-3 can be applied to any downstream tasks without any gradient updates or fine-tuning, with tasks and few-shot demonstrations specified purely via text interaction with the model. GPT-3 achieved strong performance on many NLP tasks, including translation, question-answering, and the cloze tasks, as well as several ones that require on-the-fly reasoning or domain adaptation, such as unscrambling words, using a novel word in a sentence, 3-digit arithmetic. Fig 9 plots the performance of GPT-3 as a function of the number of examples in in-context prompts.

CODEX [57], released by OpenAI in March 2023, is a general-purpose programming model that can parse natural language and generate code in response. CODEX is a descendant of GPT-3, fine-tuned for programming applications on code corpora collected from GitHub. CODEX powers Microsoft's GitHub Copilot.

WebGPT [58] is another descendant of GPT-3, fine-tuned to answer open-ended questions using a text-based web browser, facilitating users to search and navigate the web. Specifically, WebGPT is trained in three steps. The first is for WebGPT to learn to mimic human browsing behaviors using human demonstration data. Then, a reward function is learned to predict human preferences. Finally, WebGPT is refined to optimize the reward function via reinforcement learning and rejection sampling.

To enable LLMs to follow expected human instructions, InstructGPT [59] is proposed to align language models with user intent on a wide range of tasks by fine-tuning with human feedback. Starting with a set of labeler-written prompts and prompts submitted through the OpenAI API, a dataset of labeler demonstrations of the desired model behavior is collected. Then GPT-3 is fine-tuned on this dataset. Then, a dataset of human-ranked model outputs is collected to further fine-tune the model using reinforcement learning. The method is known Reinforcement Learning from Human Feedback

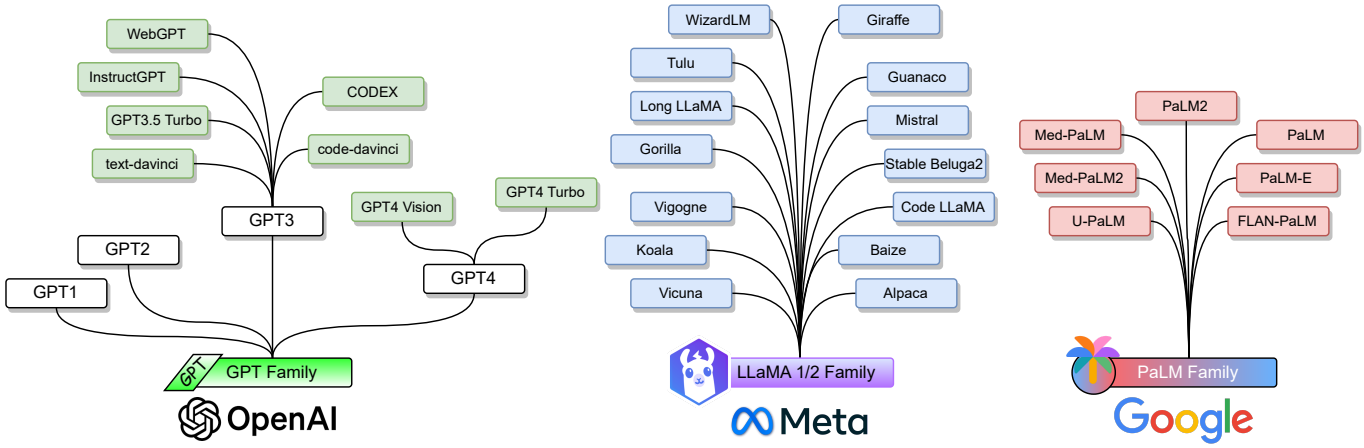


Fig. 8: Popular LLM Families.

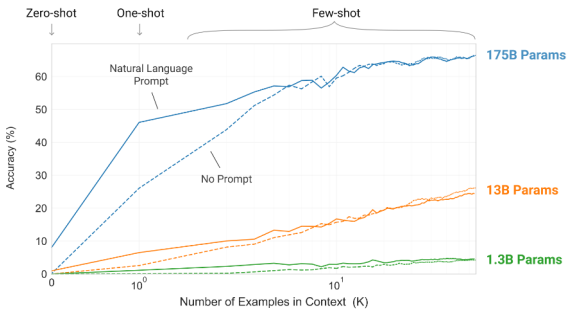


Fig. 9: GPT-3 shows that larger models make increasingly efficient use of in-context information. It shows in-context learning performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description. Courtesy of [56].

(RLHF), as shown in 10. The resultant InstructGPT models have shown improvements in truthfulness and reductions in toxic output generation while having minimal performance regressions on public NLP datasets.

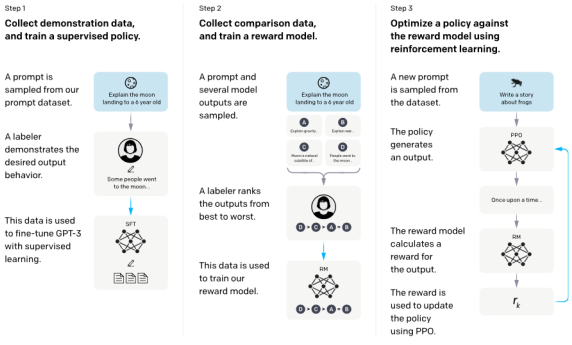


Fig. 10: The high-level overview of RLHF. Courtesy of [59].

The most important milestone of LLM development is the launch of ChatGPT (Chat Generative Pre-trained Transformer) [60] on November 30, 2022. ChatGPT is chatbot that enables users to steer a conversation to complete a wide range of tasks such as question answering, information seeking, text summarization, and more. ChatGPT is powered by GPT-3.5 (and later by GPT-4), a sibling model to InstructGPT, which is trained to follow an instruction in a prompt and provide a detailed response.

GPT-4 [33] is the latest and most powerful LLM in the GPT family. Launched in March, 2023, GPT-4 is a multi-modal LLM in that it can take image and text as inputs and produce text outputs. While still less capable than humans in some of the most challenging real-world scenarios, GPT-4 exhibits human-level performance on various professional and academic benchmarks, including passing a simulated bar exam with a score around the top 10% of test takers, as shown in Fig 11. Like early GPT models, GPT-4 was first pre-trained to predict next tokens on large text corpora, and then fine-tuned with RLHF to align model behaviors with human-desired ones.

*2) **The LLaMA Family**:* LLaMA is a collection of foundation language models, released by Meta. Unlike GPT models, LLaMA models are open-source, i.e., model weights are released to the research community under a noncommercial license. Thus, the LLaMA family grows rapidly as these models are widely used by many research groups to develop better open-source LLMs to compete the closed-source ones or to develop task-specific LLMs for mission-critical applications.

The first set of LLaMA models [32] was released in February 2023, ranging from 7B to 65B parameters. These models are pre-trained on trillions of tokens, collected from publicly available datasets. LLaMA uses the transformer architecture of GPT-3, with a few minor architectural modifications, including (1) using a SwiGLU activation function instead of ReLU, (2) using rotary positional embeddings instead of absolute positional embedding, and (3) using root-mean-squared layer-normalization instead of standard layer-normalization. The open-source LLaMA-13B model outperforms the proprietary GPT-3 (175B) model on most benchmarks, making it a good baseline for LLM research.

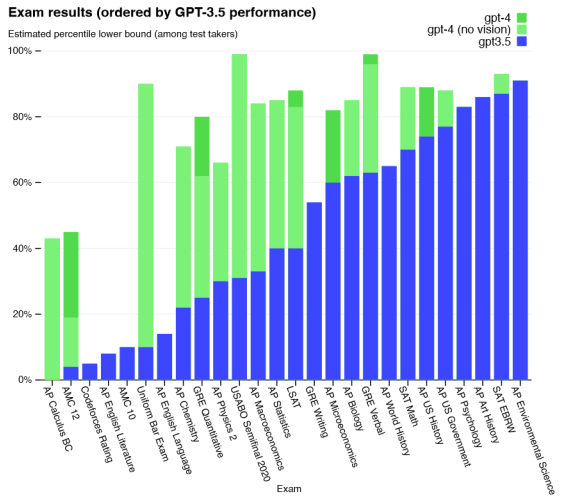


Fig. 11: GPT-4 performance on academic and professional exams, compared with GPT 3.5. Courtesy of [33].

In July 2023, Meta, in partnership with Microsoft, released the LLaMA-2 collection [61], which include both foundation language models and Chat models finetuned for dialog, known as LLaMA-2 Chat. The LLaMA-2 Chat models were reported to outperform other open-source models on many public benchmarks. Fig 12 shows the training process of LLaMA-2 Chat. The process begins with pre-training LLaMA-2 using publicly available online data. Then, an initial version of LLaMA-2 Chat is built via supervised fine-tuning. Subsequently, the model is iteratively refined using RLHF, rejection sampling and proximal policy optimization. In the RLHF stage, the accumulation of human feedback for revising the reward model is crucial to prevent the reward model from being changed too much, which could hurt the stability of LLaMA model training.

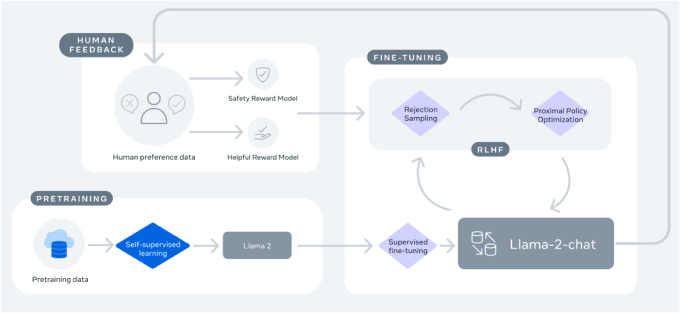


Fig. 12: Training of LLaMA-2 Chat. Courtesy of [61].

Alpaca [62] is fine-tuned from the LLaMA-7B model using 52K instruction-following demonstrations generated in the style of self-instruct using GPT-3.5 (text-davinci-003). Alpaca is very cost-effective for training, especially for academic research. On the self-instruct evaluation set, Alpaca performs similarly to GPT-3.5, despite that Alpaca is much smaller.

The Vicuna team has developed a 13B chat model, Vicuna-13B, by fine-tuning LLaMA on user-shared conversations collected from ShareGPT. Preliminary evaluation using GPT-4 as a evaluator shows that Vicuna-13B achieves more than 90% quality of OpenAI's ChatGPT, and Google's Bard while outperforming other models like LLaMA and Stanford Alpaca in more than 90% of cases. 13 shows the relative response quality of Vicuna and a few other well-known models by GPT-4. Another advantage of Vicuna-13B is its relative limited computational demand for model training. The training cost of Vicuna-13B is merely $300.

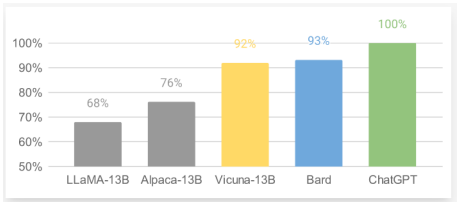


Fig. 13: Relative Response Quality of Vicuna and a few other well-known models by GPT-4. Courtesy of Vicuna Team.

Like Alpaca and Vicuna, the Guanaco models [63] are also finetuned LLaMA models using instruction-following data. But the finetuning is done very efficiently using QLoRA such that finetuning a 65B parameter model can be done on a single 48GB GPU. QLoRA back-propagates gradients through a frozen, 4-bit quantized pre-trained language model into Low Rank Adapters (LoRA). The best Guanaco model outperforms all previously released models on the Vicuna benchmark, reaching 99.3% of the performance level of ChatGPT while only requiring 24 hours of fine-tuning on a single GPU.

Koala [64] is yet another instruction-following language model built on LLaMA, but with a specific focus on interaction data that include user inputs and responses generated by highly capable closed-source chat models such as ChatGPT. The Koala-13B model performs competitively with state-of-the-art chat models according to human evaluation based on real-world user prompts.

Mistral-7B [65] is a 7B-parameter language model engineered for superior performance and efficiency. Mistral-7B outperforms the best open-source 13B model (LLaMA-2-13B) across all evaluated benchmarks, and the best open-source 34B model (LLaMA-34B) in reasoning, mathematics, and code generation. This model leverages grouped-query attention for faster inference, coupled with sliding window attention to effectively handle sequences of arbitrary length with a reduced inference cost.

The LLaMA family is growing rapidly, as more instruction-following models have been built on LLaMA or LLaMA-2, including Code LLaMA [66], Gorilla [67], Giraffe [68], Vigogne [69], Tulu 65B [70], Long LLaMA [71], and Stable Beluga2 [72], just to name a few.

3) ***The PaLM Family****:* The PaLM (Pathways Language Model) family are developed by Google. The first PaLM model [31] was announced in April 2022 and remained private until March 2023. It is a 540B parameter transformer-based LLM. The model is pre-trained on a high-quality text corpus consisting of 780 billion tokens that comprise a wide range of natural language tasks and use cases. PaLM is pre-trained

on 6144 TPU v4 chips using the Pathways system, which enables highly efficient training across multiple TPU Pods. PaLM demonstrates continued benefits of scaling by achieving state-of-the-art few-shot learning results on hundreds of language understanding and generation benchmarks. PaLM-540B outperforms not only state-of-the-art fine-tuned models on a suite of multi-step reasoning tasks, but also on par with humans on the recently released BIG-bench benchmark.

The U-PaLM models of 8B, 62B, and 540B scales are continually trained on PaLM with UL2R, a method of continue training LLMs on a few steps with UL2's mixture-of-denoiser objective [73]. An approximately 2x computational savings rate is reported.

U-PaLM is later instruction-finetuned as Flan-PaLM [74]. Compared to other instruction finetuning work mentioned above, Flan-PaLM's finetuning is performed using a much larger number of tasks, larger model sizes, and chain-of-thought data. As a result, Flan-PaLM substantially outperforms previous instruction-following models. For instance, Flan-PaLM-540B, which is instruction-finetuned on 1.8K tasks, outperforms PaLM-540B by a large margin (+9.4% on average). The finetuning data comprises 473 datasets, 146 task categories, and 1,836 total tasks, as illustrated in Fig 14.

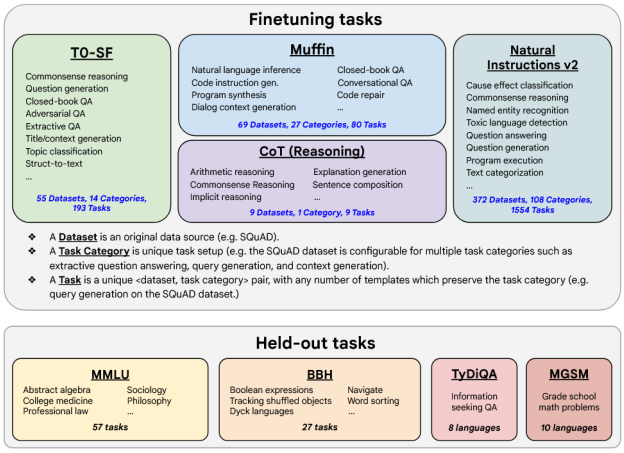


Fig. 14: Flan-PaLM finetuning consist of 473 datasets in above task categories. Courtesy of [74].

PaLM-2 [75] is a more compute-efficient LLM with better multilingual and reasoning capabilities, compared to its predecessor PaLM. PaLM-2 is trained using a mixture of objectives. Through extensive evaluations on English, multilingual, and reasoning tasks, PaLM-2 significantly improves the model performance on downstream tasks across different model sizes, while simultaneously exhibiting faster and more efficient inference than PaLM.

Med-PaLM [76] is a domain-specific PaLM, and is designed to provide high-quality answers to medical questions. Med-PaLM is finetuned on PaLM using instruction prompt tuning, a parameter-efficient method for aligning LLMs to new domains using a few exemplars. Med-PaLM obtains very encouraging results on many healthcare tasks, although it is still inferior to human clinicians. Med-PaLM 2 improves Med-PaLM via med-domain finetuning and ensemble prompting [77]. Med-PaLM 2 scored up to 86.5% on the MedQA dataset (i.e., a benchmark combining six existing open question answering datasets spanning professional medical exams, research, and consumer queries), improving upon Med-PaLM by over 19% and setting a new state-of-the-art.

C. Other Representative LLMs

In addition to the models discussed in the previous subsections, there are other popular LLMs which do not belong to those three model families, yet they have achieved great performance and have pushed the LLMs field forward. We briefly describe these LLMs in this subsection.

FLAN: In [78], Wei et al. explored a simple method for improving the zero-shot learning abilities of language models. They showed that instruction tuning language models on a collection of datasets described via instructions substantially improves zero-shot performance on unseen tasks. They take a 137B parameter pretrained language model and instruction tune it on over 60 NLP datasets verbalized via natural language instruction templates. They call this instruction-tuned model FLAN. Fig 15 provides a comparison of instruction tuning with pretrain–finetune and prompting.

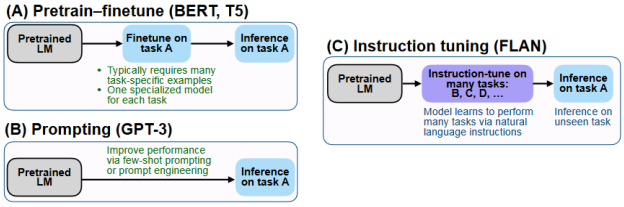


Fig. 15: comparison of instruction tuning with pretrain–finetune and prompting. Courtesy of [78].

Gopher: In [79], Rae et al. presented an analysis of Transformer-based language model performance across a wide range of model scales — from models with tens of millions of parameters up to a 280 billion parameter model called Gopher. These models were evaluated on 152 diverse tasks, achieving state-of-the-art performance across the majority. The number of layers, the key/value size, and other hyper-parameters of different model sizes are shown in Fig 16.

Model	Layers	Number Heads	Key/Value Size	d_{model}	Max LR	Batch Size
44M	8	16	32	512	6×10^{-4}	0.25M
117M	12	12	64	768	6×10^{-4}	0.25M
417M	12	12	128	1,536	2×10^{-4}	0.25M
1.4B	24	16	128	2,048	2×10^{-4}	0.25M
7.1B	32	32	128	4,096	1.2×10^{-4}	2M
Gopher 280B	80	128	128	16,384	4×10^{-5}	3M → 6M

Fig. 16: Model architecture details of Gopher with different number of parameters. Courtesy of [78].

T0: In [80], Sanh et al. developed T0, a system for easily mapping any natural language tasks into a human-readable prompted form. They converted a large set of supervised datasets, each with multiple prompts with diverse wording.

These prompted datasets allow for benchmarking the ability of a model to perform completely held-out tasks. Then, a T0 encoder-decoder model is developed to consume textual inputs and produces target responses. The model is trained on a multitask mixture of NLP datasets partitioned into different tasks.

ERNIE 3.0: In [81], Sun et al. proposed a unified framework named ERNIE 3.0 for pre-training large-scale knowledge enhanced models. It fuses auto-regressive network and auto-encoding network, so that the trained model can be easily tailored for both natural language understanding and generation tasks using zero-shot learning, few-shot learning or fine-tuning. They have trained ERNIE 3.0 with 10 billion parameters on a 4TB corpus consisting of plain texts and a large-scale knowledge graph. Fig 17 illustrates the model architecture of Ernie 3.0.

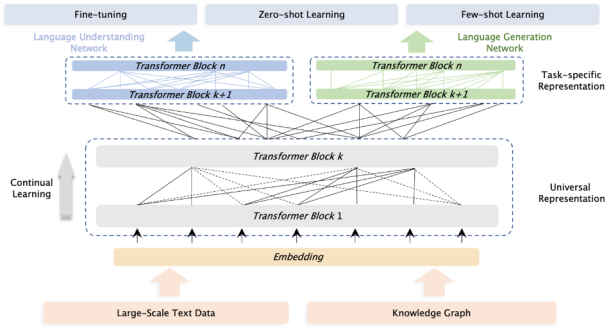


Fig. 17: High-level model architecture of ERNIE 3.0. Courtesy of [81].

RETRO: In [82], Borgeaud et al. enhanced auto-regressive language models by conditioning on document chunks retrieved from a large corpus, based on local similarity with preceding tokens. Using a 2-trillion-token database, the Retrieval-Enhanced Transformer (Retro) obtains comparable performance to GPT-3 and Jurassic-1 [83] on the Pile, despite using 25% fewer parameters. As shown in Fig 18, Retro combines a frozen Bert retriever, a differentiable encoder and a chunked cross-attention mechanism to predict tokens based on an order of magnitude more data than what is typically consumed during training.

GLaM: In [84], Du et al. proposed a family of LLMs named GLaM (Generalist Language Model), which use a sparsely activated mixture-of-experts architecture to scale the model capacity while also incurring substantially less training cost compared to dense variants. The largest GLaM has 1.2 trillion parameters, which is approximately 7x larger than GPT-3. It consumes only 1/3 of the energy used to train GPT-3 and requires half of the computation flops for inference, while still achieving better overall zero, one and few-shot performance across 29 NLP tasks. Fig 19 shows the high-level architecture of GLAM.

LaMDA: In [85], Thoppilan et al. presented LaMDA, a family of Transformer-based neural language models specialized for dialog, which have up to 137B parameters and are pre-trained on 1.56T words of public dialog data and web text.

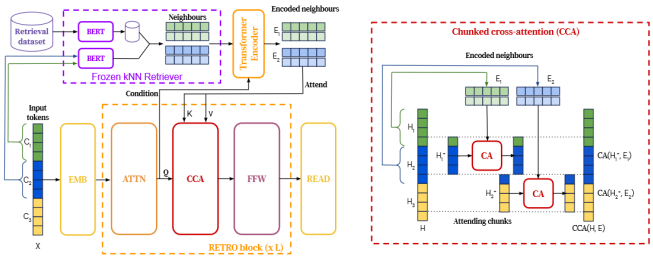


Fig. 18: Retro architecture. Left: simplified version where a sequence of length n = 12 is split into l = 3 chunks of size m = 4. For each chunk, we retrieve k = 2 neighbours of r = 5 tokens each. The retrieval pathway is shown on top. Right: Details of the interactions in the CCA operator. Causality is maintained as neighbours of the first chunk only affect the last token of the first chunk and tokens from the second chunk. Courtesy of [82].

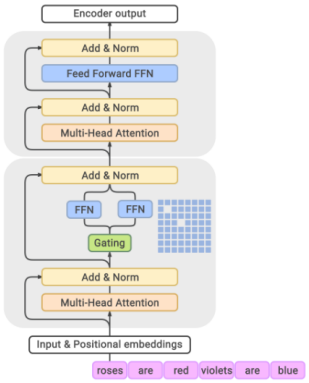


Fig. 19: GLaM model architecture. Each MoE layer (the bottom block) is interleaved with a Transformer layer (the upper block). Courtesy of [84].

They showed that fine-tuning with annotated data and enabling the model to consult external knowledge sources can lead to significant improvements towards the two key challenges of safety and factual grounding.

OPT: In [86], Zhang et al. presented Open Pre-trained Transformers (OPT), a suite of decoder-only pre-trained transformers ranging from 125M to 175B parameters, which they share with researchers. The OPT models' parameters are shown in 20

Model	#L	#H	d_{model}	LR	Batch
125M	12	12	768	$6.0e{-4}$	0.5M
350M	24	16	1024	$3.0e{-4}$	0.5M
1.3B	24	32	2048	$2.0e{-4}$	1M
2.7B	32	32	2560	$1.6e{-4}$	1M
6.7B	32	32	4096	$1.2e{-4}$	2M
13B	40	40	5120	$1.0e{-4}$	4M
30B	48	56	7168	$1.0e{-4}$	4M
66B	64	72	9216	$0.8e{-4}$	2M
175B	96	96	12288	$1.2e{-4}$	2M

Fig. 20: Different OPT Models' architecture details. Courtesy of [86].

Chinchilla: In [2], Hoffmann et al. investigated the optimal model size and number of tokens for training a transformer language model under a given compute budget. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, they found that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. They tested this hypothesis by training a predicted compute-optimal model, Chinchilla, that uses the same compute budget as Gopher but with 70B parameters and 4% more more data.

Galactica: In [87], Taylor et al. introduced Galactica, a large language model that can store, combine and reason about scientific knowledge. They trained on a large scientific corpus of papers, reference material, knowledge bases and many other sources. Galactica performed well on reasoning, outperforming Chinchilla on mathematical MMLU by 41.3% to 35.7%, and PaLM 540B on MATH with a score of 20.4% versus 8.8%.

CodeGen: In [88], Nijkamp et al. trained and released a family of large language models up to 16.1B parameters, called CODEGEN, on natural language and programming language data, and open sourced the training library JAX-FORMER. They showed the utility of the trained model by demonstrating that it is competitive with the previous state-of-the-art on zero-shot Python code generation on HumanEval. They further investigated the multi-step paradigm for program synthesis, where a single program is factorized into multiple prompts specifying sub-problems. They also constructed an open benchmark, Multi-Turn Programming Benchmark (MTPB), consisting of 115 diverse problem sets that are factorized into multi-turn prompts.

AlexaTM: In [89], Soltan et al. demonstrated that multilingual large-scale sequence-to-sequence (seq2seq) models, pre-trained on a mixture of denoising and Causal Language Modeling (CLM) tasks, are more efficient few-shot learners than decoder-only models on various task. They trained a 20 billion parameter multilingual seq2seq model called Alexa Teacher Model (AlexaTM 20B) and showed that it achieves state-of-the-art (SOTA) performance on 1-shot summarization tasks, outperforming a much larger 540B PaLM decoder model. AlexaTM consist of 46 encoder layers, 32 decoder layers, 32 attention heads, and $d_{model} = 4096$.

Sparrow: In [90], Glaese et al. presented Sparrow, an information-seeking dialogue agent trained to be more helpful, correct, and harmless compared to prompted language model baselines. They used reinforcement learning from human feedback to train their models with two new additions to help human raters judge agent behaviour. The high-level pipeline of Sparrow model is shown in Fig 21.

Minerva: In [91], Lewkowycz et al. introduced Minerva, a large language model pretrained on general natural language data and further trained on technical content, to tackle previous LLM struggle with quantitative reasoning (such as solving mathematics, science, and engineering problems).

MoD: In [92], Tay et al. presented a generalized and unified perspective for self-supervision in NLP and show how different pre-training objectives can be cast as one another and how interpolating between different objectives can be

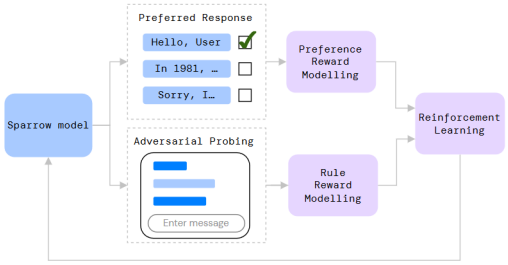


Fig. 21: Sparrow pipeline relies on human participation to continually expand a training set. Courtesy of [90].

effective. They proposed Mixture-of-Denoisers (MoD), a pre-training objective that combines diverse pre-training paradigms together. This framework is known as Unifying Language Learning (UL2). An overview of UL2 pretraining paradigm is shown in Fig 21.

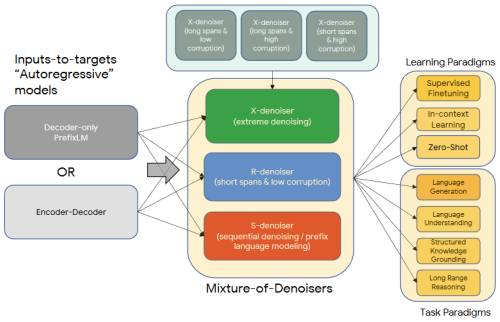


Fig. 22: An overview of UL2 pretraining paradigm. Courtesy of [92].

BLOOM: In [93], Scao et al. presented BLOOM, a 176B-parameter open-access language model designed and built thanks to a collaboration of hundreds of researchers. BLOOM is a decoder-only Transformer language model trained on the ROOTS corpus, a dataset comprising hundreds of sources in 46 natural and 13 programming languages (59 in total). An overview of BLOOM architecture is shown in Fig 23.

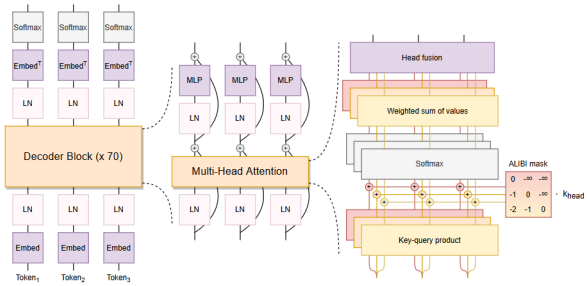


Fig. 23: An overview of BLOOM architecture. Courtesy of [93].

GLM: In [94], Zeng et al. introduced GLM-130B, a

bilingual (English and Chinese) pre-trained language model with 130 billion parameters. It was an attempt to open-source a 100B-scale model at least as good as GPT-3 (davinci) and unveil how models of such a scale can be successfully pre-trained.

Pythia: In [95], Biderman et al. introduced Pythia, a suite of 16 LLMs all trained on public data seen in the exact same order and ranging in size from 70M to 12B parameters. We provide public access to 154 checkpoints for each one of the 16 models, alongside tools to download and reconstruct their exact training dataloaders for further study.

Orca: In [96], Mukherjee et al. develop Orca, a 13-billion parameter model that learns to imitate the reasoning process of large foundation models. Orca learns from rich signals from GPT-4 including explanation traces; step-by-step thought processes; and other complex instructions, guided by teacher assistance from ChatGPT.

StarCoder: In [97], Li et al. introduced StarCoder and StarCoderBase. They are 15.5B parameter models with 8K context length, infilling capabilities and fast large-batch inference enabled by multi-query attention. StarCoderBase is trained on one trillion tokens sourced from The Stack, a large collection of permissively licensed GitHub repositories with inspection tools and an opt-out process. They fine-tuned StarCoderBase on 35B Python tokens, resulting in the creation of StarCoder. They performed the most comprehensive evaluation of Code LLMs to date and showed that StarCoderBase outperforms every open Code LLM that supports multiple programming languages and matches or outperforms the OpenAI code-cushman-001 model.

KOSMOS: In [98], Huang et al. introduced KOSMOS-1, a Multimodal Large Language Model (MLLM) that can perceive general modalities, learn in context (i.e., few-shot), and follow instructions (i.e. zero-shot). Specifically, they trained KOSMOS-1 from scratch on web-scale multi-modal corpora, including arbitrarily interleaved text and images, image-caption pairs, and text data. Experimental results show that KOSMOS-1 achieves impressive performance on (i) language understanding, generation, and even OCR-free NLP (directly fed with document images), (ii) perception-language tasks, including multimodal dialogue, image captioning, visual question answering, and (iii) vision tasks, such as image recognition with descriptions (specifying classification via text instructions).

Gemini: In [99], Gemini team introduced a new family of multimodal models, that exhibit promising capabilities across image, audio, video, and text understanding. Gemini family includes three versions: Ultra for highly-complex tasks, Pro for enhanced performance and deployability at scale, and Nano for on-device applications. Gemini architecture is built on top of Transformer decoders, and is trained to support 32k context length (via using efficient attention mechanisms).

Some of the other popular LLM frameworks (or techniques used for efficient developments of LLMs) includes Inner-Monologue [100], Megatron-Turing NLG [101], LongFormer [102], OPT-IML [103], MeTaLM [104], Dromedary [105], Palmyra [106], Camel [107], Yalm [108], MPT [109], ORCA-2 [110], Gorilla [67], PAL [111], Claude [112], CodeGen 2 [113], Zephyr [114], Grok [115], Qwen [116], Mamba [30], Mixtral-8x7B [117], DocLLM [118], DeepSeek-Coder [119], FuseLLM-7B [120], TinyLlama-1.1B [121], LLaMA-Pro-8B [122].

Fig 24 provides an overview of some of the most representative LLM frameworks, and the relevant works that have contributed to the success of LLMs and helped to push the limits of LLMs.

III. How LLMs Are Built

In this section, we first review the popular architectures used for LLMs, and then discuss data and modeling techniques ranging from data preparation, tokenization, to pre-training, instruction tuning, and alignment.

Once the model architecture is chosen, the major steps involved in training an LLM includes: data preparation (collection, cleaning, deduping, etc.), tokenization, model pre-training (in a self-supervised learning fashion), instruction tuning, and alignment. We will explain each of them in a separate subsection below. These steps are also illustrated in Fig 25.

A. Dominant LLM Architectures

The most widely used LLM architectures are encoder-only, decoder-only, and encoder-decoder. Most of them are based on Transformer (as the building block). Therefore we also review the Transformer architecture here.

1) ***Transformer****:* in a ground-breaking work [44], Vaswani et al. proposed the Transformer framework, which was originally designed for effective parallel computing using GPUs. The heart of Transformer is the (self-)attention mechanism, which can capture long-term contextual information much more effectively using GPUs than the recurrence and convolution mechanisms. Fig 26 provides a high-level overview of transformer work. In this section we provide an overview of the main elements and variants, see [44], [123] for more details.

The Transformer language model architecture, originally proposed for machine translation, consists of an encoder and a decoder. The encoder is composed of a stack of N = 6 identical Transformer layers. Each layer has two sub-layers. The first one is a multi-head self-attention layer, and the other one is a simple position-wise fully connected feed-forward network. The decoder is composed of a stack of 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder has a third sub-layer, which performs multi-head attention over the output of the encoder stack. The attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. Instead of performing a single attention function with d_{model} dimensional keys, values and queries, it is found to be beneficial to linearly project the queries, keys and values h with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. Positional encoding is incorporated to fuse information about the relative or absolute position of the tokens in the sequence.

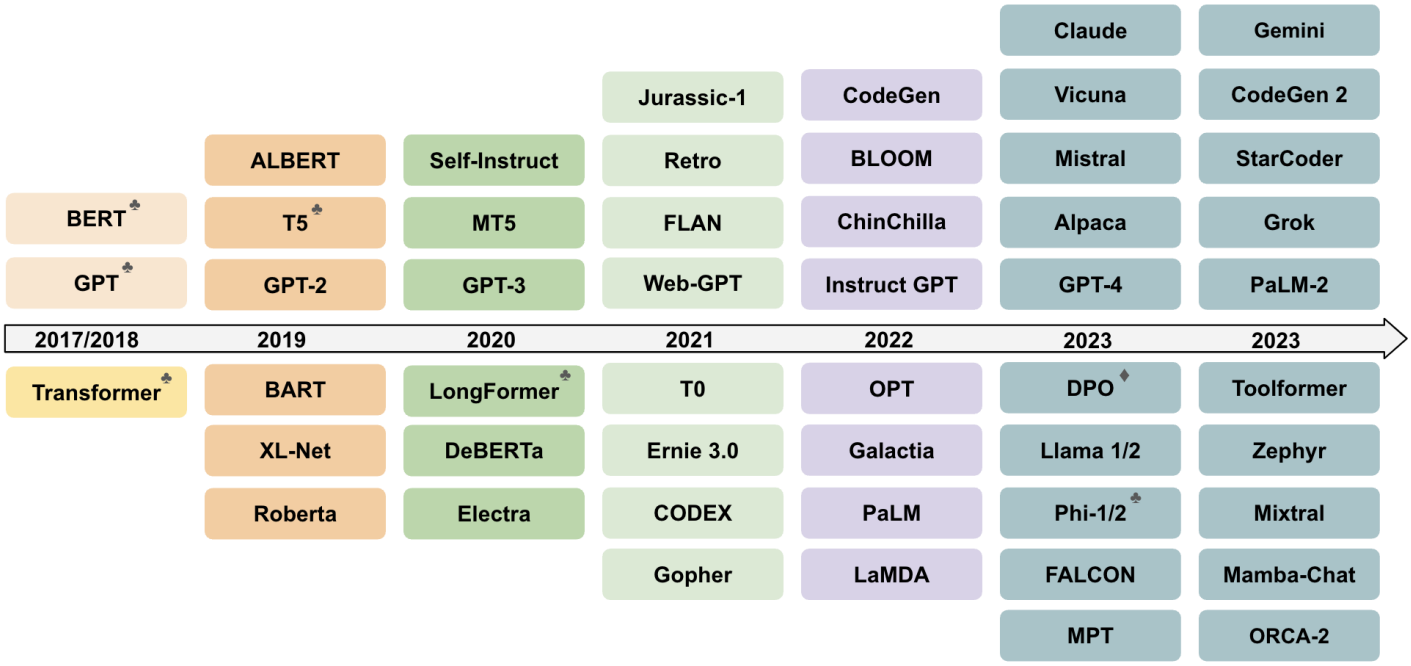


Fig. 24: Timeline of some of the most representative LLM frameworks (so far). In addition to large language models with our #parameters threshold, we included a few representative works, which pushed the limits of language models, and paved the way for their success (e.g. vanilla Transformer, BERT, GPT-1), as well as some small language models. ♣ shows entities that serve not only as models but also as approaches. ♦ shows only approaches.

*2) **Encoder-Only**:* For this family, at each stage, the attention layers can access all the words in the initial sentence. The pre-training of these models usually consist of somehow corrupting a given sentence (for instance, by masking random words in it) and tasking the model with finding or reconstructing the initial sentence. Encoder models are great for tasks requiring an understanding of the full sequence, such as sentence classification, named entity recognition, and extractive question answering. One prominent encoder only model is BERT (Bidirectional Encoder Representations from Transformers), proposed in [24].

*3) **Decoder-Only**:* For these models, at each stage, for any word, the attention layers can only access the words positioned before that in the sentence. These models are also sometimes called auto-regressive models. The pretraining of these models is usually formulated as predicting the next word (or token) in the sequence. The decoder-only models are best suited for tasks involving text generation. GPT models are prominent example of this model category.

*4) **Encoder-Decoder**:* These models use both encoder and decoder, and are sometimes called sequence-to-sequence models. At each stage, the attention layers of the encoder can access all the words in the initial sentence, whereas the attention layers of the decoder only accesses the words positioned before a given word in the input. These models are usually pretrained using the objectives of encoder or decoder models, but usually involve something a bit more complex. For instance, some models are pretrained by replacing random spans of text (that can contain several words) with a single mask special word, and the objective is then to predict the text that this mask word replaces. Encoder-decoder models are best suited for tasks about generating new sentences conditioned on a given input, such as summarization, translation, or generative question answering.

B. Data Cleaning

Data quality is crucial to the performance of language models trained on them. Data cleaning techniques such as filtering, deduplication, are shown to have a big impact on the model performance.

As an example, in **Falcon40B** [124], Penedo et al. showed that properly filtered and deduplicated web data alone can lead to powerful models; even significantly outperforming models from the state-of-the-art trained on The Pile. Despite extensive filtering, they were able to obtain five trillion tokens from CommonCrawl. They also released an extract of 600 billion tokens from our REFINEDWEB dataset, and 1.3/7.5B parameters language models trained on it. 27 shows the Refinement process of CommonCrawl data by this work.

1) Data Filtering: Data filtering aims to enhance the quality of training data and the effectiveness of the trained LLMs. Common data filtering techniques include:

Removing Noise: refers to eliminating irrelevant or noisy data that might impact the model's ability to generalize well. As an example, one can think of removing false information from the training data, to lower the chance of model generating false responses. Two mainstream approaches for quality filtering includes: classifier-based, and heuristic-based frameworks.

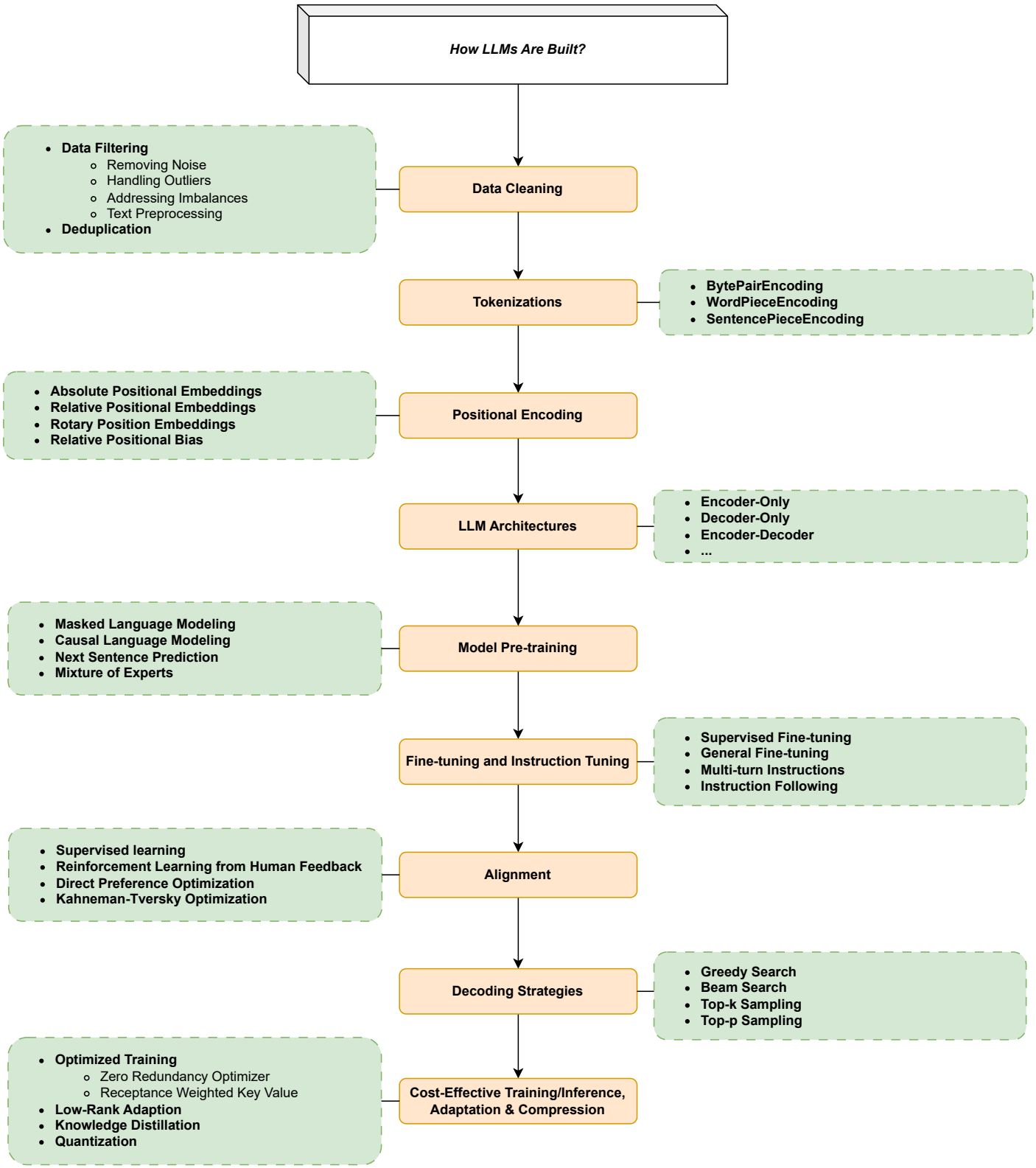


Fig. 25: This figure shows different components of LLMs.

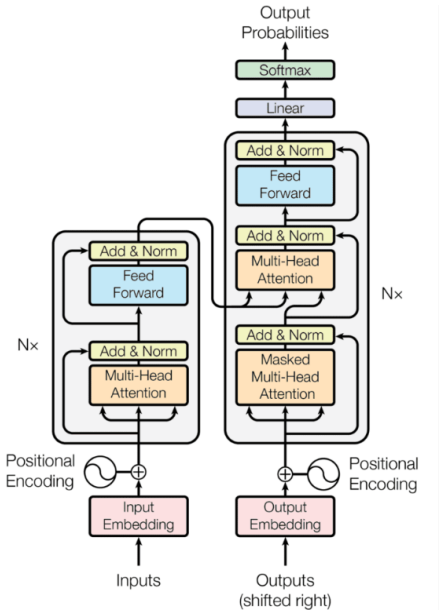


Fig. 26: High-level overview of transformer work. Courtesy of [44].

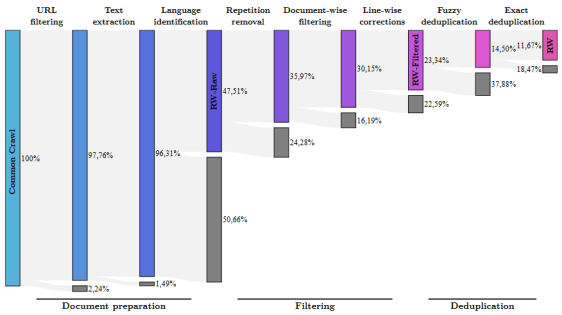


Fig. 27: Subsequent stages of Macrodata Refinement remove nearly 90% of the documents originally in CommonCrawl. Courtesy of [124].

Handling Outliers: Identifying and handling outliers or anomalies in the data to prevent them from disproportionately influencing the model.

Addressing Imbalances: Balancing the distribution of classes or categories in the dataset to avoid biases and ensure fair representation. This is specially useful for responsible model training and evaluation.

Text Preprocessing: Cleaning and standardizing text data by removing stop words, punctuation, or other elements that may not contribute significantly to the model's learning.

Dealing with Ambiguities: Resolving or excluding ambiguous or contradictory data that might confuse the model during training. This can help the model to provide more definite and reliable answers.

2) Deduplication: De-duplication refers to the process of removing duplicate instances or repeated occurrences of the same data in a dataset. Duplicate data points can introduce biases in the model training process and reduce the diversity, as the model may learn from the same examples multiple times, potentially leading to overfitting on those particular instances. Some works [125] have shown that de-duplication improves models' ability to generalize to new, unseen data.

The de-duplication process is particularly important when dealing with large datasets, as duplicates can unintentionally inflate the importance of certain patterns or characteristics. This is especially relevant in NLP tasks, where diverse and representative training data is crucial for building robust language models.

The specific de-duplication method can vary based on the nature of the data and the requirements of the particular language model being trained. It may involve comparing entire data points or specific features to identify and eliminate duplicates. At the document level, existing works mainly rely on the overlap ratio of high-level features (e.g. n-grams overlap) between documents to detect duplicate samples.

C. Tokenizations

Tokenization referes to the process of converting a sequence of text into smaller parts, known as tokens. While the simplest tokenization tool simply chops text into tokens based on white space, most tokenization tools rely on a word dictionary. However, out-of-vocabulary (OOV) is a problem in this case because the tokenizer only knows words in its dictionary. To increase the coverage of dictionaries, popular tokenizers used for LLMs are based on sub-words, which can be combined to form a large number of words, including the words unseen in training data or words in different languages. In what follows, we describe three popular tokenizers.

1) ***BytePairEncoding****: BytePairEncoding* is originally a type of data compression algorithm that uses frequent patterns at byte level to compress the data. By definition, this algorithm mainly tries to keep the frequent words in their original form and break down ones that are not common. This simple paradigm keeps the vocabulary not very large, but also good enough to represent common words at the same time. Also morphological forms of the frequent words can be represented very well if suffix or prefix is also commonly presented in the training data of the algorithm.

2) ***WordPieceEncoding****:* This algorithm is mainly used for very well-known models such as BERT and Electra. At the beginning of training, the algorithm takes all the alphabet from the training data to make sure that nothing will be left as UNK or *unknown* from the training dataset. This case happens when the model is given an input that can not be tokenized by the tokenizer. It mostly happens in cases where some characters are not tokenizable by it. Similar to BytePairEncoding, it tries to maximize the likelihood of putting all the tokens in vocabulary based on their frequency.

3) ***SentencePieceEncoding****:* Although both tokenizers described before are strong and have many advantages compared to white-space tokenization, they still take assumption of words being always separated by white-space as granted. This assumption is not always true, in fact in some languages, words can be corrupted by many noisy elements such as unwanted spaces or even invented words. SentencePieceEncoding tries to address this issue.

D. *Positional Encoding*

1) ***Absolute Positional Embeddings****:* (APE) [44] has been used in the original Transformer model to preserve the information of sequence order. Therefore, the positional information of words is added to the input embeddings at the bottom of both the encoder and decoder stacks. There are various options for positional encodings, either learned or fixed. In the vanilla Transformer, sine and cosine functions are employed for this purpose. The main drawback of using APE in Transformers is the restriction to a certain number of tokens. Additionally, APE fails to account for the relative distances between tokens.

2) ***Relative Positional Embeddings****:* (RPE) [126] involves extending self-attention to take into account the pairwise links between input elements. RPE is added to the model at two levels: first as an additional component to the keys, and subsequently as a sub-component of the values matrix. This approach looks at the input as a fully-connected graph with labels and directed edges. In the case of linear sequences, edges can capture information about the relative position differences between input elements. A clipping distance, represented as k $2 \leq k \leq n-4$, specifies the maximum limit on relative locations. This allows the model to make reasonable predictions for sequence lengths that are not part of the training data.

3) ***Rotary Position Embeddings****:* Rotary Positional Embedding (RoPE) [127] tackles problems with existing approaches. Learned absolute positional encodings can lack generalizability and meaningfulness, particularly when sentences are short. Moreover, current methods like T5's positional embedding face challenges with constructing a full attention matrix between positions. RoPE uses a rotation matrix to encode the absolute position of words and simultaneously includes explicit relative position details in self-attention. RoPE brings useful features like flexibility with sentence lengths, a decrease in word dependency as relative distances increase, and the ability to improve linear self-attention with relative position encoding. GPT-NeoX-20B, PaLM, CODEGEN, and LLaMA are among models that take advantage of RoPE in their architectures.

4) ***Relative Positional Bias****:* The concept behind this type of positional embedding is to facilitate extrapolation during inference for sequences longer than those encountered in training. In [128] Press et al. proposed Attention with Linear Biases (ALiBi). Instead of simply adding positional embeddings to word embeddings, they introduced a bias to the attention scores of query-key pairs, imposing a penalty proportional to their distance. In the BLOOM model, ALiBi is leveraged.

E. *Model Pre-training*

Pre-training is the very first step in large language model training pipeline, and it helps LLMs to acquire fundamental language understanding capabilities, which can be useful in a wide range of language related tasks. During pre-training, the LLM is trained on a massive amount of (usually) unlabeled texts, usually in a self-supervised manner. There are different approaches used for pre-training like next sentence prediction [24], two most common ones include, next token prediction (autoregressive language modeling), and masked language modeling.

In **Autoregressive Language Modeling** framework, given a sequence of n tokens x_1, ..., x_n, the model tries to predict next token x_{n+1} (and sometimes next sequence of tokens) in an auto-regressive fashion. One popular loss function in this case is the log-likelihood of predicted tokens as shown in Eq 2

$$\mathscr{L}_{ALM}(x) = \sum_{i=1}^{N} p(x_{i+n}|x_i, ..., x_{i+n-1}) \tag{1}$$

Given the auto-regressive nature of this framework, the decoder-only models are naturally better suited to learn how to accomplish these task.

In **Masked Language Modeling**, some words are masked in a sequence and the model is trained to predict the masked words based on the surrounding context. Sometimes people refer to this approach as denoising autoencoding, too. If we denote the masked/corrupted samples in the sequence x, as $\tilde{x}$, then the training objective of this approach can be written as:

$$\mathscr{L}_{MLM}(x) = \sum_{i=1}^{N} p(\tilde{x}|x \backslash \tilde{x}) \tag{2}$$

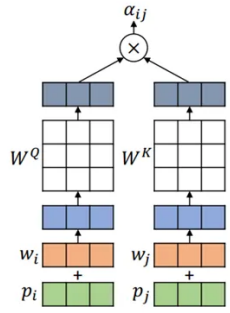
And more recently, **Mixture of Experts (MoE)** [130], [131] have become very popular in LLM space too. MoEs enable models to be pre-trained with much less compute, which means one can dramatically scale up the model or dataset size with the same compute budget as a dense model. MoE consists of two main elements: **Sparse MoE layers**, which are used instead of dense feed-forward network (FFN) layers, and have a certain number of "experts" (e.g. 8), in which each expert is a neural network. In practice, the experts are FFNs, but they can also be more complex networks. **A gate network or router**, that determines which tokens are sent to which expert. It is worth noting that, one can send a token to more than one expert. How to route a token to an expert is one of the big decisions when working with MoEs - the router is composed of learned parameters and is pretrained at the same time as the rest of the network. Fig 29 provides an illustration of a Switch Transformer encoder block, which are used in MoE.

F. *Fine-tuning and Instruction Tuning*

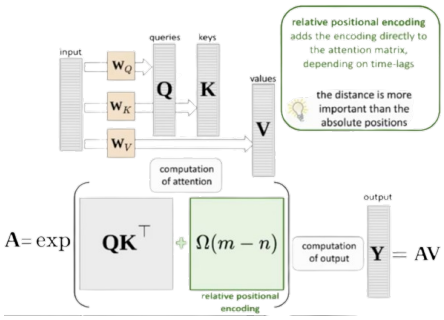
Early language models such as BERT trained using self-supervision as explained in section III-E were not able to perform specific tasks. In order for the foundation model to be useful it needed to be fine-tuned to a specific task with labeled data (so-called supervised fine-tuning or SFT for short). For example, in the original BERT paper [24], the model was fine-tuned to 11 different tasks. While more recent LLMs no longer require fine-tuning to be used, they can still benefit from task or data-specific fine-tuning. For example, OpenAI reports that the much smaller GPT-3.5 Turbo model can outperform GPT-4 when fine-tuned with task specific data [2].

Fine-tuning does not need to be performed to a single task though, and there are different approaches to multi-task fine-tuning (see e.g. Mahabi et al. [132]). Fine-tuning to one or more tasks is known to improve results and reduce the complexity of prompt engineering, and it can serve as an

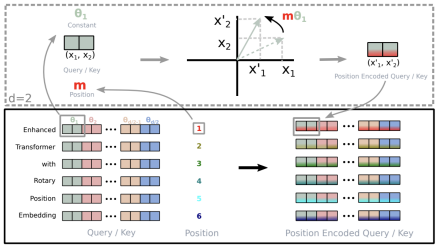
[2]https://platform.openai.com/docs/guides/fine-tuning



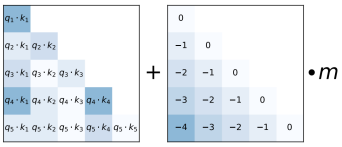
(a) Absolute Positional Embeddings [129]



(b) Relative Positional Embeddings



(c) Rotary Positional Embedding [127]



(d) Relative Positional Bias [128]

Fig. 28: Various positional encodings are employed in LLMs.

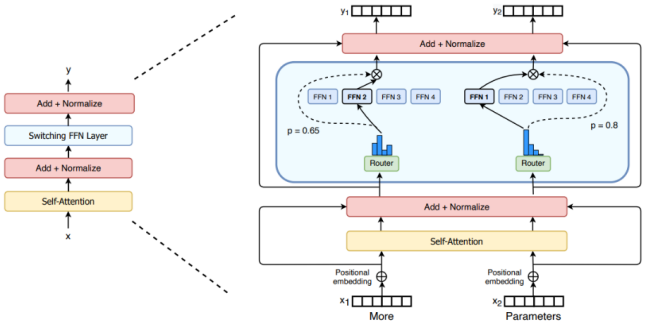


Fig. 29: : Illustration of a Switch Transformer encoder block. They replaced the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). . Courtesy of [131].

alternative to retrieval augmented generation. Furthermore, there are other reasons why it might be advisable to fine-tune. For example, one might want to fine-tune to expose the model to new or proprietary data that it has not been exposed to during pre-training.

An important reason to fine-tune LLMs is to align the responses to the expectations humans will have when providing instructions through prompts. This is the so-called **instruction tuning** [133]. We dive into the details of how to design and engineer prompts in section IV-B, but in the context of instruction tuning, it is important to understand that the instruction is a prompt that specifies the task that the LLM should accomplish. Instruction tuning datasets such as Natural Instructions [134] include not only the task definition but other components such as positive/negative examples or things to avoid.

The specific approach and instruction datasets used to instruction-tune an LLM varies, but, generally speaking, instruction tuned models outperform their original foundation models they are based on. For example, InstructGPT [59] outperforms GPT-3 on most benchmarks. The same is true for Alpaca [62] when compared to LLaMA.

Self-Instruct [135], proposed by Wang et al. is also a popular approach along this line, in which they introduced a framework for improving the instruction-following capabilities of pre-trained language models by bootstrapping their own generations. Their pipeline generates instructions, input, and output samples from a language model, then filters invalid or similar ones before using them to fine tune the original model.

G. Alignment

AI Alignment is the process of steering AI systems towards human goals, preferences, and principles. LLMs, pre-trained for word prediction, often exhibit unintended behaviors. For example, they might generate contents that are toxic, harmful, misleading and biased.

Instruction tuning, discussed above, gets LLMs a step closer to being aligned. However, in many cases, it is important to include further steps to improve the alignment of the model and avoid unintended behaviors [3]. We review the most popular

[3]According to very recent research by Ethayarajh et al. [136], further alignment besides SFT mainly improves models of at least 7B parameters. For smaller models, SFT is sufficient.

approaches to alignment in this subsection.

RLHF (reinforcement learning from human feedback) and **RLAIF** (reinforcement learning from AI feedback) are two popular approaches. RLHF uses a reward model to learn alignment from human feedback. This reward model, after being tuned, is able to rate different outputs and score them according to their alignment preferences given by humans. The reward model gives feedback to the original LLM and this feedback is used to tune the LLM further [137]. Reinforcement learning from AI feedback on the other hand, directly connects a pretrained and well-aligned model to the LLM and helps it to learn from larger and more aligned models [138].

In another recent work (known as **DPO**) [139], Rafailov et al. discussed that RLHF is a complex and often unstable procedure, and tried to address this with a new approach. They leveraged a mapping between reward functions and optimal policies to show that this constrained reward maximization problem can be optimized exactly with a single stage of policy training, essentially solving a classification problem on the human preference data. The resulting algorithm, which they called Direct Preference Optimization (DPO), is stable, performant, and computationally lightweight, eliminating the need for fitting a reward model, sampling from the LM during fine-tuning, or performing significant hyperparameter tuning. They observed that fine-tuning with DPO exceeds RLHF's ability to control sentiment of generations and improves response quality in summarization. Fig 30 shows the high-level comparison between DPO vs RLHF.

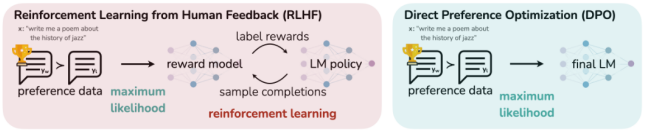


Fig. 30: DPO optimizes for human preferences while avoiding reinforcement learning. Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, without an explicit reward function or RL. Courtesy of [139].

Even more recently Ethayarajh et al. proposed a new alignment approach called the Kahneman-Tversky Optimization (KTO) [136]. Unlike existing state-of-the-art approaches, KTO does not require paired preference data (x, y_w, y_l), and it only needs (x,y) and knowledge of whether y is desirable or undesirable. KTO-aligned models are shown to be good or better than DPO-aligned models at scales from 1B to 30B, despite not using paired preferences. KTO is also far easier to use in the real world than preference optimization methods, as the kind of data it needs is far more abundant. As an example, every retail company has a lot of customer interaction data and whether that interaction was successful (e.g., purchase made) or unsuccessful (e.g., no purchase made). However, They have little to no counterfactual data (i.e., what would have made an unsuccessful customer interaction y_l into a successful one y_w). Fig 31 shows a high-level comparison between KTO and other alignment approaches discussed above.

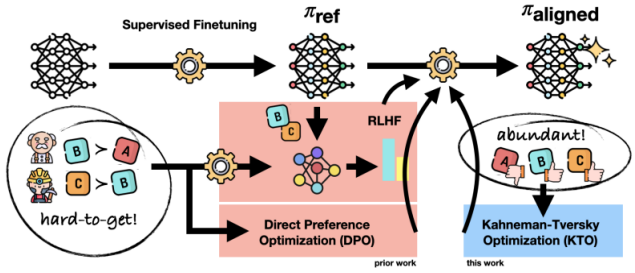


Fig. 31: LLM alignment involves supervised finetuning followed by optimizing a human-centered loss (HALO). However, the paired preferences that existing approaches need are hard-to-obtain. In contrast, KTO uses a far more abundant kind of data, making it much easier to use in the real world. Courtesy of [136].

H. Decoding Strategies

Decoding refers to the process of text generation using pretrained LLMs. Given an input prompt, the tokenizer translates each token in the input text into a corresponding token ID. Then, the language model uses these token IDs as input and predicts the next most likely token (or a sequence of tokens). Finally, the model generates logits, which are converted to probabilities using a softmax function. Different decoding strategies have been proposed. Some of the most popular ones are greedy search, beam search, as well as different sample techniques such as top-K, top-P (Nucleus sampling).

1) ***Greedy Search****:* Greedy search takes the most probable token at each step as the next token in the sequence, discarding all other potential options. As you can imagine, this is a simple approach and can loose a lot of temporal consistency and coherency. It only considers the most probable token at each step, without considering the overall effect on the sequence. This property makes it fast, but it also means that it can miss out on better sequences that might have appeared with slightly less probable next tokens.

2) ***Beam Search****:* Unlike greedy search that only considers the next most probable token, beam search takes into account the **N** most likely tokens, where **N** denotes the number of beams. This procedure is repeated until a predefined maximum sequence length is reached or an end-of-sequence token appears. At this point, the sequence of tokens (AKA "beam") with the highest overall score is chosen as the output. For example for beam size of 2 and maximum length of 5, the beam search needs to keep track of $2^5 = 32$ possible sequences. So it is more computationally intensive than greedy search.

3) ***Top-k Sampling****:* Top-k sampling is a technique that uses the probability distribution generated by the language model to select a token randomly from the k most likely options.

Suppose we have 6 tokens (A, B, C, D, E, F) and k=2, and P(A)= 30%, and P(B)= 20%, P(C)= P(D)= P(E)= P(F)=

12.5%. In top-k sampling, tokens C, D, E, F are disregarded, and the model outputs A 60% of the time, and B, 40% of the time. This approach ensures that we prioritize the most probable tokens while introducing an element of randomness in the selection process.

The randomness is usually introduced via the concept of temperature. The temperature T is a parameter that ranges from 0 to 1, which affects the probabilities generated by the softmax function, making the most likely tokens more influential. In practice, it simply consists of dividing the input logits by temperature value:

$$softmax(x_i) = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}} \tag{3}$$

A low temperature setting significantly alters the probability distribution (and is commonly used in text generation to control the level of "creativity" in the generated output), while a large temperature prioritizes the tokens with higher probabilities. Top-k is a creative way of sampling, and can be used along with beam search. The sequence chosen by top-k sampling may not be the sequence with highest probability in beam search. But it's important to remember that highest scores do not always lead to more realistic or meaningful sequences.

*4) **Top-p Sampling**:* Top-p sampling, also known as Nucleus sampling, takes a slightly different approach from top-k sampling. Instead of selecting the top k most probable tokens, nucleus sampling chooses a cutoff value p such that the sum of the probabilities of the selected tokens exceeds p. This forms a "nucleus" of tokens from which to randomly choose the next token. In other words, in top-p sampling the language model examines the most probable tokens in descending order and keeps adding them to the list until the sum of probabilities surpasses the threshold p. As you can imagine, this could be better specially for scenarios in which top-k tokens do not have a large probability mass. Unlike top-k sampling, the number of tokens included in the nucleus sampling is not fixed. This variability often results in a more diverse and creative output, making nucleus sampling popular for text generation related tasks.

I. Cost-Effective Training/Inference/Adaptation/Compression

In this part, we review some of the popular approaches used for more cost-friendly (and compute-friendly) training and usage of LLMs.

*1) **Optimized Training**:* There are many frameworks developed for optimized training of LLMs, here we introduce some of the prominent ones.

ZeRO: In [140], Rajbhandari et al. developed a novel solution, Zero Redundancy Optimizer (ZeRO), to optimize memory, vastly improving training speed of LLMs while increasing the model size that can be efficiently trained. ZeRO eliminates memory redundancies in data- and model-parallel training while retaining low communication volume and high computational granularity, allowing one to scale the model size proportional to the number of devices with sustained high efficiency.

RWKV: In [141], Peng et al. proposed a novel model architecture, Receptance Weighted Key Value (RWKV), that combines the efficient parallelizable training of Transformers with the efficient inference of RNNs. Their approach leverages a linear attention mechanism and allows them to formulate the model as either a Transformer or an RNN, which parallelizes computations during training and maintains constant computational and memory complexity during inference, leading to the first non-transformer architecture to be scaled to tens of billions of parameters. RWKV architecture is shown in Fig 32. The Time Complexity comparison of RWKV with different

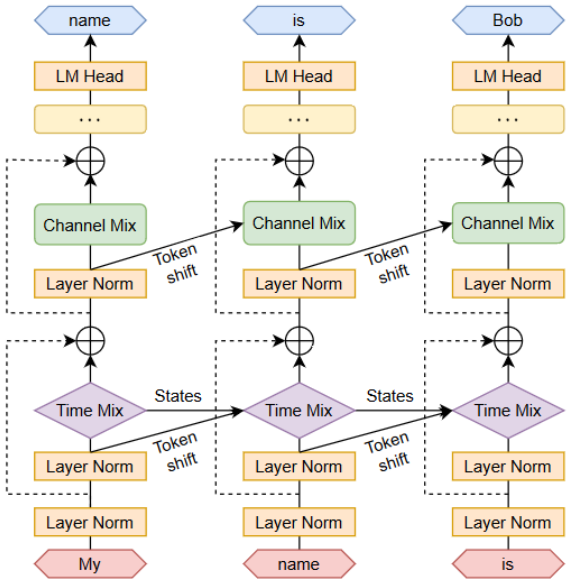


Fig. 32: RWKV architecture. Courtesy of [141].

Transformers are provided in Fig 33.

Model	Time	Space
Transformer	$O(T^2 d)$	$O(T^2 + Td)$
Reformer	$O(T \log T d)$	$O(T \log T + Td)$
Linear Transformers	$O(Td^2)$	$O(Td + d^2)$
Performer	$O(Td^2 \log d)$	$O(Td \log d + d^2 \log d)$
AFT-full	$O(T^2 d)$	$O(Td)$
MEGA	$O(cTd)$	$O(cTd)$
RWKV (ours)	$O(\mathbf{Td})$	$O(\mathbf{d})$

Fig. 33: Time Complexity comparison of RWKV with different Transformers. Here T denotes the sequence length, d the feature dimension, and c is MEGA's chunk size of quadratic attention. Courtesy of [141].

*2) **Low-Rank Adaption (LoRA)**:* Low-Rank Adaptation is a popular and lightweight training technique that significantly reduces the number of trainable parameters, and is based on a crucial insight that the difference between the fine-tuned weights for a specialized task and the initial pre-trained weights often exhibits "low intrinsic rank" - meaning that it can be approximated well by a low rank matrix [142].

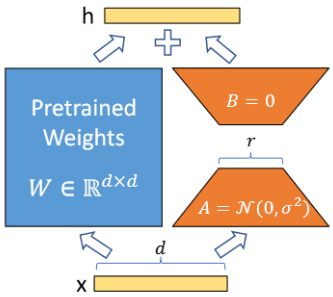


Fig. 34: An illustration of LoRA reparametrizan. Only A and B trained during this process. Courtesy of [142].

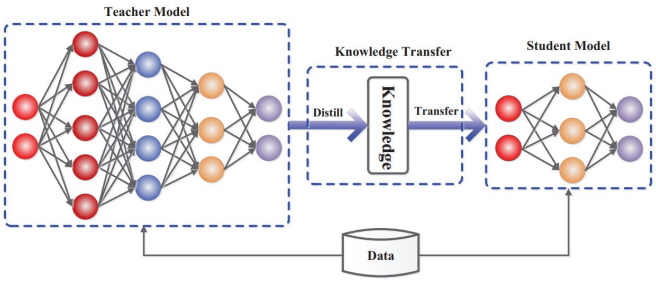


Fig. 35: A generic knowledge distillation framework with student and teacher (Courtesy of [144]).

Training with LoRA is much faster, memory-efficient, and produces smaller model weights (a few hundred MBs), that are easier to store and share. One property of low-rank matrices is that they can be represented as the product of two smaller matrices. This realization leads to the hypothesis that this delta between fine-tuned weights and initial pre-trained weights can be represented as the matrix product of two much smaller matrices. By focusing on updating these two smaller matrices rather than the entire original weight matrix, computational efficiency can be substantially improved.

Specifically, for a pre-trained weight matrix $W_0 \in R^{d \times k}$, LoRA constrains its update by representing the latter with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in R^{d \times r}$, $A \in R^{r \times k}$, and the rank $r \ll min(d, k)$. During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. It is worth mentioning that both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0 x$, their modified forward pass yields: $h = W_0 x + \Delta W x = W_0 x + BAx$. Usually a random Gaussian initialization is used for A, and zero initialization for B, so $\Delta W = BA$ is zero at the beginning of training. They then scale $\Delta W x$ by αr, where α is a constant in r. This reparametrization is illustrated in Figure 34

It is worth mentioning that LoRA can be applied to any a subset of weight matrices in a neural network to reduce the number of trainable parameters. In the Transformer architecture, there are four weight matrices in the self-attention module (W_q , W_k, W_v , W_o), and two in the MLP module. Most of the time, LoRA is focused on adapting the attention weights only for downstream tasks, and freezes the MLP modules, so they are not trained in downstream tasks both for simplicity and parameter-efficiency.

3) ***Knowledge Distillation****:* Knowledge distillation is the process of learning from a larger model [143]. Earlier days of best-performing models release have proven that this approach is very useful even if it is used in an API distillation approach. It is also referred to as an approach to distill the knowledge of not a single model but in fact multiple models into a smaller one. Creating smaller models by this approach yields smaller model sizes that can be used even on edge devices. Knowledge distillation as shown in Fig 35, illustrates a general setup of this training scheme.

Knowledge can be transferred by different forms of learning: response distillation, feature distillation, and API distillation. Response distillation is concerned only with the outputs of the teacher model and tries to teach the student model how to exactly or at least similarly perform (in the sense of prediction) as the teacher. Feature distillation not only uses the last layer but also intermediate layers as well to create a better inner representation for the student model. This helps the smaller model to have a similar representation as the teacher model.

API distillation is the process of using an API (typically from an LLM provider such as OpenAI) to train smaller models. In the case of LLMs, it is used to train the model from the direct output of the larger model which makes it very similar to response distillation. Many concerns are raised by this type of distillation because in cases where the model itself is not openly available, a (usually) paid API is exposed for end users. On the other hand, while users pay for each call, how to use the predictions is limited, for example, OpenAI prohibits usage of its API to create LLMs that later will be used to compete with it. The main value in such case is training data.

4) ***Quantization****:* deep learning in its core, is a set of mathematical functions applied to matrices, with a specific precision for model weights. Reducing the precision of the weights can be used to reduce the size of the model and also make it faster. As an example, Float-32 operations compared to Int-8 operations are slower. This process, which is called quantization, can be applied in different phases. Main approaches for model quantization can be categorized as: post training quantization and quantization-aware training. Post-training quantization is concerned with quantized trained models in two well-known methods: dynamic and static. Dynamic post-training quantization computes the range of quantization on the runtime and is slower compared to static. Quantization-aware training adds quantization criteria into training, and a quantized model is trained and optimized during training process. This approach ensures that the end model will have good performance and also does not need to be quantized after training.

IV. How LLMs Are Used and Augmented

Once the LLMs are trained, we can use them to generate desired outputs for a variety of tasks. LLMs can be used directly through basic prompting. However, in order to exploit their full potential or to address some of the shortcomings,

we need to augment the models through some external means. In this section we first provide a brief overview of the main shortcoming of LLMs, with a deeper look at the issue of hallucination. We then describe how prompting and some augmentation approaches can not only address those limitations but also be used to augment the capabilities of LLMs going as far as turning an LLM into a full-blown AI agent with the ability to interface with the external world.

A. LLM limitations

It is important to remember that LLMs are trained to predict a token. While fine-tuning and alignment improves their performance and adds different dimensions to their abilities, there are still some important limitations that come up, particularly if they are used naively. Some of them include the following:

- They don't have state/memory. LLMs on their own cannot remember even what was sent to them in the previous prompt. That is an important limitation for many of the uses cases that require some form of state.
- They are stochastic/probabilistic. If you send the same prompt to an LLM several times, you are likely to get different responses. While there are parameters, and in particular the temperature, to limit the variability in the response, this is an inherent property of their training that can create issues.
- They have stale information and, on their own, don't have access to external data. An LLM on its own does not even know about the current time or day and does not have access to any information that was not present in its training set.
- They are generally very large. This means that many costly GPU machines are needed for training and serving. In some cases, largest models have poor SLAs, particularly in terms of latency.
- They hallucinate. LLMs do not have a notion of "truth" and they have usually been trained on a mix of good and bad content. They can produce very plausible but untruthful answers.

While the previous limitations can all become important for some applications, it is worth for us to dive a bit into the last one, hallucinations, since it has gathered a lot of interest over the past few months and it has also sparked many of the prompt approaches and LLM augmentation methods we will later describe.

Hallucination: In the realm of Large Language Models (LLMs), the phenomenon of "hallucinations" has garnered significant attention. Defined in the literature, notably in the "Survey of Hallucination in Natural Language Generation" paper [145], hallucination in an LLM is characterized as "the generation of content that is nonsensical or unfaithful to the provided source." This terminology, although rooted in psychological parlance, has been appropriated within the field of artificial intelligence.

Hallucinations in LLMs can be broadly categorized into two types:

1) **Intrinsic Hallucinations**: These directly conflict with the source material, introducing factual inaccuracies or logical inconsistencies.
2) **Extrinsic Hallucinations**: These, while not contradicting, are unverifiable against the source, encompassing speculative or unconfirmable elements.

The definition of 'source' in LLM contexts varies with the task. In dialogue-based tasks, it refers to 'world knowledge', whereas in text summarization, it pertains to the input text itself. This distinction plays a crucial role in evaluating and interpreting hallucinations. The impact of hallucinations is also highly context-dependent. For instance, in creative endeavors like poem writing, hallucinations might be deemed acceptable or even beneficial.

LLMs, trained on diverse datasets including the internet, books, and Wikipedia, generate text based on probabilistic models without an inherent understanding of truth or falsity. Recent advancements like instruct tuning and Reinforcement Learning from Human Feedback (RLHF) have attempted to steer LLMs towards more factual outputs, but the fundamental probabilistic nature and its inherent limitations remain. A recent study, "Sources of Hallucination by Large Language Models on Inference Tasks" [146], highlights two key aspects contributing to hallucinations in LLMs: the veracity prior and the relative frequency heuristic, underscoring the complexities inherent in LLM training and output generation.

Effective automated measurement of hallucinations in LLMs requires a combination of statistical and model-based metrics.

Statistical Metrics:

- Metrics like ROUGE [147] and BLEU [148] are common for assessing text similarity, focusing on intrinsic hallucinations.
- Advanced metrics such as PARENT [149], PARENT-T [150], and Knowledge F1 [151] are utilized when structured knowledge sources are available. These metrics, while effective, have limitations in capturing syntactic and semantic nuances.

Model-Based Metrics:

- **IE-Based Metrics**: Utilize Information Extraction models to simplify knowledge into relational tuples, then compare these with the source.
- **QA-Based Metrics**: Assess the overlap between generated content and the source through a question-answering framework (see [152]).
- **NLI-Based Metrics**: Use Natural Language Inference datasets to evaluate the truthfulness of a generated hypothesis based on a given premise (see [153]).
- **Faithfulness Classification Metrics**: Offer a refined assessment by creating task-specific datasets for a nuanced evaluation (see [154]).

Despite advances in automated metrics, human judgment remains a vital piece. It typically involves two methodologies:

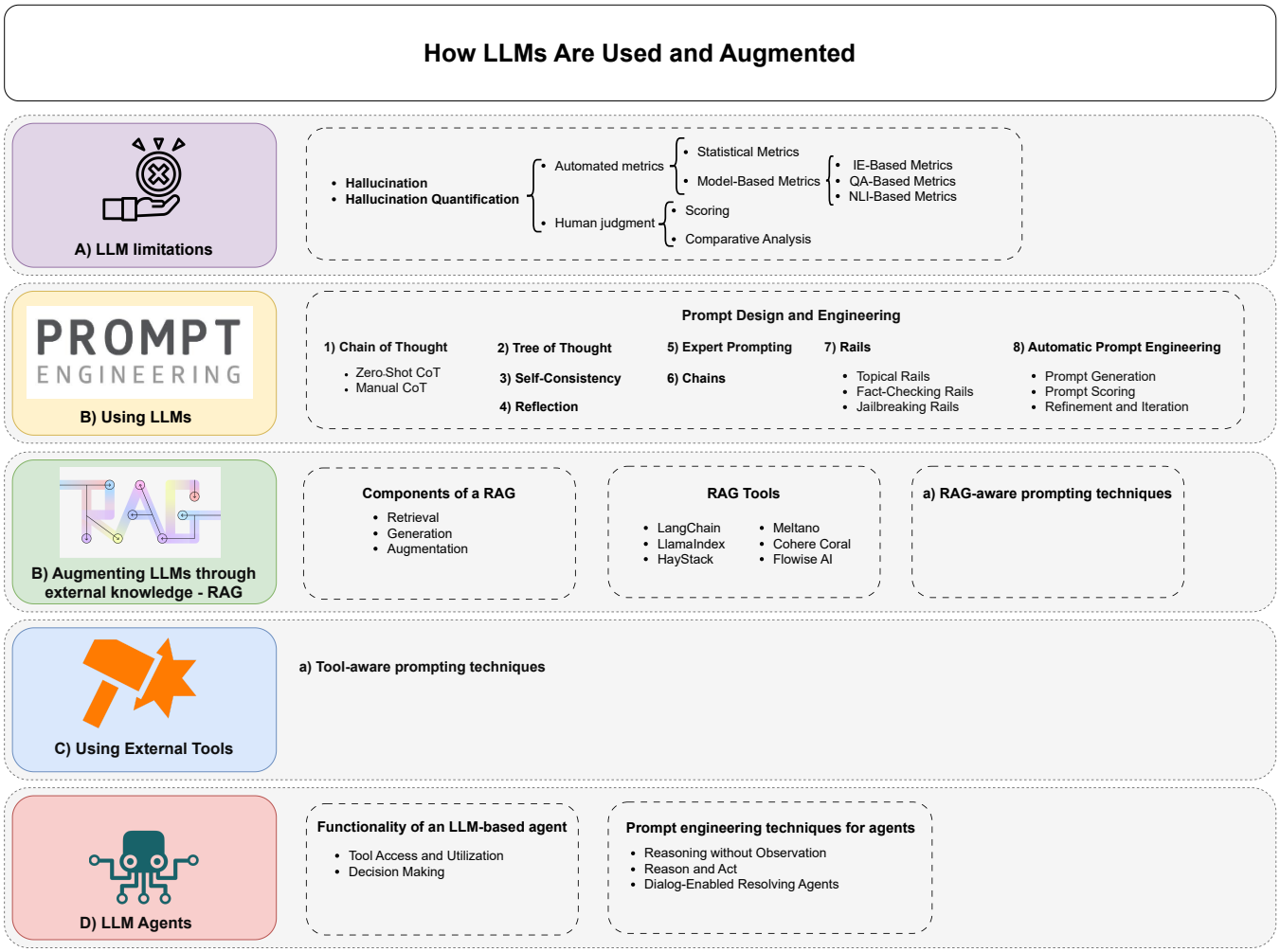


Fig. 36: How LLMs Are Used and Augmented.

1) **Scoring**: Human evaluators rate the level of hallucination within a predefined scale.
2) **Comparative Analysis**: Evaluators compare generated content against baseline or ground-truth references, adding an essential layer of subjective assessment.

FactScore [155] is a recent example of a metric that can be used both for human and model-based evaluation. The metric breaks an LLM generation into "atomic facts". The final score is computed as the sum of the accuracy of each atomic fact, giving each of them equal weight. Accuracy is a binary number that simply states whether the atomic fact is supported by the source. The authors implement different automation strategies that use LLMs to estimate this metric.

Finally, mitigating hallucinations in LLMs is a multifaceted challenge, requiring tailored strategies to suit various applications. Those include:

- Product Design and User Interaction Strategies such as use case design, structuring the input/output, or providing mechanisms for user feedback.
- Data Management and Continuous Improvement. Maintaining and analyzing a tracking set of hallucinations is essential for ongoing model improvement.
- Prompt Engineering and Metaprompt Design. Many of the advanced prompt techniques described in IV-B such as Retrieval Augmented Generation directly address hallucination risks.
- Model Selection and Configuration for Hallucination Mitigation. For exemple, larger models with lower temperature settings usually perform better. Also, techniques such as RLHF or domain-sepcific fine-tuning can mitigate hallucination risks.

B. Using LLMs: ***Prompt Design and Engineering***

A prompt in generative AI models is the textual input provided by users to guide the model's output. This could range from simple questions to detailed descriptions or specific tasks. Prompts generally consist of instructions, questions, input data, and examples. In practice, to elicit a desired response from an AI model, a prompt must contain either instructions or questions, with other elements being optional. Advanced prompts involve more complex structures, such as

”chain of thought” prompting, where the model is guided to follow a logical reasoning process to arrive at an answer.

Prompt engineering is a rapidly evolving discipline that shapes the interactions and outputs of LLMs and other generative AI models. The essence of prompt engineering lies in crafting the optimal prompt to achieve a specific goal with a generative model. This process is not only about instructing the model but also involves some understanding of the model's capabilities and limitations, and the context within which it operates.

Prompt engineering transcends the mere construction of prompts; it requires a blend of domain knowledge, understanding of the AI model, and a methodical approach to tailor prompts for different contexts. This might involve creating templates that can be programmatically modified based on a given dataset or context. For example, generating personalized responses based on user data might use a template that is dynamically filled with relevant user information.

Furthermore, prompt engineering is an iterative and exploratory process, akin to traditional machine learning practices such as model evaluation or hyperparameter tuning. The rapid growth of this field suggests its potential to revolutionize certain aspects of machine learning, moving beyond traditional methods like feature or architecture engineering. On the other hand, traditional engineering practices such as version control and regression testing need to be adapted to this new paradigm just like they were adapted to other machine learning approaches [156].

In the following paragraphs we detail some of the most interesting and popular prompt engineering approaches.

1) Chain of Thought (CoT): The Chain of Thought (CoT) technique, initially described in the paper “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”[34] by Google researchers, represents a pivotal advancement in prompt engineering for Large Language Models (LLMs). This approach hinges on the understanding that LLMs, while proficient in token prediction, are not inherently designed for explicit reasoning. CoT addresses this by guiding the model through essential reasoning steps.

CoT is based on making the implicit reasoning process of LLMs explicit. By outlining the steps required for reasoning, the model is directed closer to a logical and reasoned output, especially in scenarios demanding more than simple information retrieval or pattern recognition.

CoT prompting manifests in two primary forms:

1) **Zero-Shot CoT:** This form involves instructing the LLM to “think step by step”, prompting it to deconstruct the problem and articulate each stage of reasoning.
2) **Manual CoT:** A more complex variant, it requires providing step-by-step reasoning examples as templates for the model. While yielding more effective results, it poses challenges in scalability and maintenance.

Manual CoT is more effective than zero-shot. However, the effectiveness of this example-based CoT depends on the choice of diverse examples, and constructing prompts with such examples of step by step reasoning by hand is hard and error prone. That is where automatic CoT [157] comes into play.

2) Tree of Thought (ToT): The Tree of Thought (ToT) [158] prompting technique is inspired by the concept of considering various alternative solutions or thought processes before converging on the most plausible one. ToT is based on the idea of branching out into multiple ”thought trees” where each branch represents a different line of reasoning. This method allows the LLM to explore various possibilities and hypotheses, much like human cognitive processes where multiple scenarios are considered before determining the most likely one.

A critical aspect of ToT is the evaluation of these reasoning paths. As the LLM generates different branches of thought, each is assessed for its validity and relevance to the query. This process involves real-time analysis and comparison of the branches, leading to a selection of the most coherent and logical outcome.

ToT is particularly useful in complex problem-solving scenarios where a single line of reasoning might not suffice. It allows LLMs to mimic a more human-like problem-solving approach, considering a range of possibilities before arriving at a conclusion. This technique enhances the model's ability to handle ambiguity, complexity, and nuanced tasks, making it a valuable tool in advanced AI applications.

3) Self-Consistency: Self-Consistency [159] utilizes an ensemble-based method, where the LLM is prompted to generate multiple responses to the same query. The consistency among these responses serves as an indicator of their accuracy and reliability.

The Self-Consistency approach is grounded in the principle that if an LLM generates multiple, similar responses to the same prompt, it is more likely that the response is accurate. This method involves asking the LLM to tackle a query multiple times, each time analyzing the response for consistency. This technique is especially useful in scenarios where factual accuracy and precision are paramount.

The consistency of responses can be measured using various methods. One common approach is to analyze the overlap in the content of the responses. Other methods may include comparing the semantic similarity of responses or employing more sophisticated techniques like BERT-scores or n-gram overlaps. These measures help in quantifying the level of agreement among the responses generated by the LLM.

Self-Consistency has significant applications in fields where the veracity of information is critical. It is particularly relevant in scenarios like fact-checking, where ensuring the accuracy of information provided by AI models is essential. By employing this technique, prompt engineers can enhance the trustworthiness of LLMs, making them more reliable for tasks that require high levels of factual accuracy.

4) Reflection: Reflection [160] involves prompting LLMs to assess and potentially revise their own outputs based on reasoning about the correctness and coherence of their responses. The concept of Reflection centers on the ability of LLMs to engage in a form of self-evaluation. After generating

an initial response, the model is prompted to reflect on its own output, considering factors like factual accuracy, logical consistency, and relevance. This introspective process can lead to the generation of revised or improved responses.

A key aspect of Reflection is the LLM's capacity for self-editing. By evaluating its initial response, the model can identify potential errors or areas of improvement. This iterative process of generation, reflection, and revision enables the LLM to refine its output, enhancing the overall quality and reliability of its responses.

5) Expert Prompting: Expert Prompting [161] enhances the capabilities of Large Language Models (LLMs) by simulating the responses of experts in various fields. This method involves prompting the LLMs to assume the role of an expert and respond accordingly, providing high-quality, informed answers. A key strategy within Expert Prompting is the multi-expert approach. The LLM is prompted to consider responses from multiple expert perspectives, which are then synthesized to form a comprehensive and well-rounded answer. This technique not only enhances the depth of the response but also incorporates a range of viewpoints, reflecting a more holistic understanding of the subject matter.

6) Chains: Chains refer to the method of linking multiple components in a sequence to handle complex tasks with Large Language Models (LLMs). This approach involves creating a series of interconnected steps or processes, each contributing to the final outcome. The concept of Chains is based on the idea of constructing a workflow where different stages or components are sequentially arranged. Each component in a Chain performs a specific function, and the output of one serves as the input for the next. This end-to-end arrangement allows for more complex and nuanced processing, as each stage can be tailored to handle a specific aspect of the task. Chains can vary in complexity and structure, depending on the requirements. In "PromptChainer: Chaining Large Language Model Prompts through Visual Programming" [162], the authors not only describe the main challenges in designing chains, but also describe a visual tool to support those tasks.

7) Rails: Rails in advanced prompt engineering refer to a method of guiding and controlling the output of Large Language Models (LLMs) through predefined rules or templates. This approach is designed to ensure that the model's responses adhere to certain standards or criteria, enhancing the relevance, safety, and accuracy of the output. The concept of Rails involves setting up a framework or a set of guidelines that the LLM must follow while generating responses. These guidelines are typically defined using a modeling language or templates known as Canonical Forms, which standardize the way natural language sentences are structured and delivered.

Rails can be designed for various purposes, depending on the specific needs of the application:

- **Topical Rails:** Ensure that the LLM sticks to a particular topic or domain.
- **Fact-Checking Rails:** Aimed at minimizing the generation of false or misleading information.
- **Jailbreaking Rails:** Prevent the LLM from generating responses that attempt to bypass its own operational constraints or guidelines.

8) Automatic Prompt Engineering (APE): Automatic Prompt Engineering (APE) [163] focuses on automating the process of prompt creation for Large Language Models (LLMs). APE seeks to streamline and optimize the prompt design process, leveraging the capabilities of LLMs themselves to generate and evaluate prompts. APE involves using LLMs in a self-referential manner where the model is employed to generate, score, and refine prompts. This recursive use of LLMs enables the creation of high-quality prompts that are more likely to elicit the desired response or outcome.

The methodology of APE can be broken down into several key steps:

- **Prompt Generation:** The LLM generates a range of potential prompts based on a given task or objective.
- **Prompt Scoring:** Each generated prompt is then evaluated for its effectiveness, often using criteria like clarity, specificity, and likelihood of eliciting the desired response.
- **Refinement and Iteration:** Based on these evaluations, prompts can be refined and iterated upon, further enhancing their quality and effectiveness.

C. *Augmenting LLMs through external knowledge - RAG*

One of the main limitations of pre-trained LLMs is their lack of up-to-date knowledge or access to private or use-case-specific information. This is where retrieval augmented generation (RAG) comes into the picture [164]. RAG, illustrated in figure 37, involves extracting a query from the input prompt and using that query to retrieve relevant information from an external knowledge source (e.g. a search engine or a knowledge graph, see figure 38). The relevant information is then added to the original prompt and fed to the LLM in order for the model to generate the final response. A RAG system includes three important components: Retrieval, Generation, Augmentation [165].

a) RAG-aware prompting techniques: Because of the importance of RAG to build advanced LLM systems, several RAG-aware prompting techniques have been developed recently. One such technique is Forward-looking Active Retrieval Augmented Generation (FLARE)

Forward-looking Active Retrieval Augmented Generation (FLARE) [168] enhances the capabilities of Large Language Models (LLMs) by iteratively combining prediction and information retrieval. FLARE represents an evolution in the use of retrieval-augmented generation, aimed at improving the accuracy and relevance of LLM responses.

FLARE involves an iterative process where the LLM actively predicts upcoming content and uses these predictions as queries to retrieve relevant information. This method contrasts with traditional retrieval-augmented models that typically retrieve information once and then proceed with generation. In FLARE, this process is dynamic and ongoing throughout the generation phase. In FLARE, each sentence or segment generated by the LLM is evaluated for confidence. If the confidence level is below a certain threshold, the model uses the generated content as a query to retrieve relevant information, which is then used to regenerate or refine the sentence. This iterative

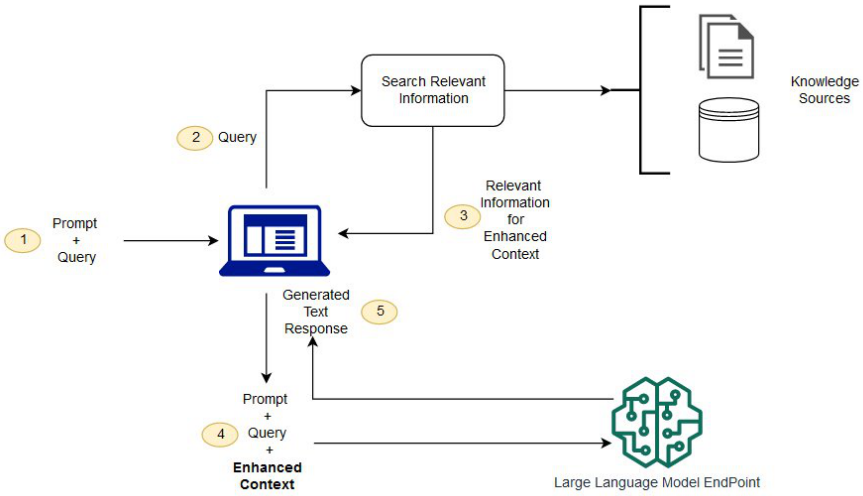


Fig. 37: An example of synthesizing RAG with LLMs for question answering application [166].

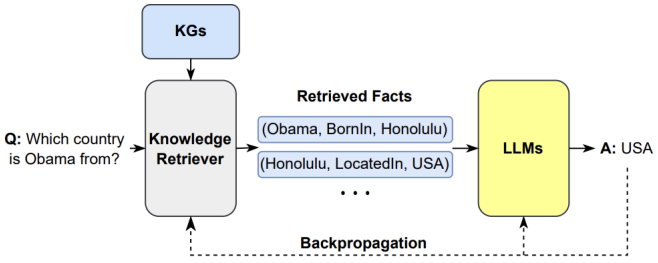


Fig. 38: This is one example of synthesizing the KG as a retriever with LLMs [167].

process ensures that each part of the response is informed by the most relevant and current information available.

For more details on RAG framework and its relevant works, we refer the readers to this survey of retrieval augmented generations [165].

D. *Using External Tools*

Retrieving information from an external knowledge source as described above is only one of the potential ways to augment an LLM. More generally, an LLM can access any number of external tools (e.g. an API to a service) to augment its functionality. In that regards, RAG can be seen as a specific instance of the broader category of the so called ”tools”.

Tools in this context are external functions or services that LLMs can utilize. These tools extend the range of tasks an LLM can perform, from basic information retrieval to complex interactions with external databases or APIs.

In the paper ”Toolformer: Language Models Can Teach Themselves to Use Tools” [169], the authors go beyond simple tool usage by training an LLM to decide what tool to use when, and even what parameters the API needs. Tools include two different search engines, or a calculator. In the following examples, the LLM decides to call an external Q&A tool, a calculator, and a Wikipedia Search Engine More recently, researchers at Berkeley have trained a new LLM called Gorilla [67] that beats GPT-4 at the use of APIs, a specific but quite general tool.

a) Tool-aware prompting techniques: Similarly to what was described with RAG, several tool-aware prompting approaches have been developed to make usage of tools more scalable. A popular technique is the so called Automatic Multi-step Reasoning and Tool-use (ART).

Automatic Multi-step Reasoning and Tool-use (ART) [170] is a prompt engineering technique that combines automated chain of thought prompting with the use of external tools. ART represents a convergence of multiple prompt engineering strategies, enhancing the ability of Large Language Models (LLMs) to handle complex tasks that require both reasoning and interaction with external data sources or tools.

ART involves a systematic approach where, given a task and input, the system first identifies similar tasks from a task library. These tasks are then used as examples in the prompt, guiding the LLM on how to approach and execute the current task. This method is particularly effective when tasks require a combination of internal reasoning and external data processing or retrieval.

E. *LLM Agents*

The idea of AI agents has been well-explored in the history of AI. An agent is typically an autonomous entity that can perceive the environment using its sensors, make a judgment based on the state it currently is, and accordingly act based on the actions that are available to it.

In the context of LLMs, an agent refers to a system based on a specialized instantiation of an (augmented) LLM that is capable of performing specific tasks autonomously. These agents are designed to interact with users and environment to make decisions based on the input and the intended goal of the interaction. Agents are based on LLMs equipped with the

ability to access and use tools, and to make decisions based on the given input. They are designed to handle tasks that require a degree of autonomy and decision-making, typically beyond simple response generation.

The functionalities of a generic LLM-based agent include:

- Tool Access and Utilization: Agents have the capability to access external tools and services, and to utilize these resources effectively to accomplish tasks.
- Decision Making: They can make decisions based on the input, context, and the tools available to them, often employing complex reasoning processes.

As an example, an LLM that has access to a function (or an API) such as weather API, can answer any question related to the weather of the specific place. In other words, it can use APIs to solve problems. Furthermore, if that LLM has access to an API that allows to make purchases, a purchasing agent can be built to not only have capabilities to read information from the external world, but also act on it [171].

Fig. 40 shows another example of LLM-based agents for conversational information seeking [36], where an LLM is augmented with a set of plug-and-play modules, including a *working memory* that tracks the dialog state, a *policy* that makes an execution plan for the task and selects next system action, an *action executor* that performs an action selected by the policy (consolidating evidence from external knowledge, or prompting the LLM to generate responses), and a *utility* that accesses the alignment of the LLM's responses with user expectations or specific business requirements, and generate feedback to improve agent performance.

For more details on LLM-based AI agents see recent survey [172], [173], [174].

a) Prompt engineering techniques for agents: Like RAG and Tools, prompt engineering techniques that specifically address the needs of LLM-based agents have been developed. Three such examples are Reasoning without Observation (ReWOO), Reason and Act (ReAct), and Dialog-Enabled Resolving Agents (DERA).

Reasoning without Observation (ReWOO) [175] aims to decouple reasoning from direct observations. ReWOO operates by enabling LLMs to formulate comprehensive reasoning plans or meta-plans without immediate reliance on external data or tools. This approach allows the agent to create a structured framework for reasoning that can be executed once the necessary data or observations are available. In ReWOO, the LLM initially develops a plan (a series of steps) that outlines how to approach and solve a given problem. This meta-planning phase is crucial as it sets the stage for the agent to process information once it becomes available. The execution phase then involves integrating actual data or observations into the pre-specified plan, leading to coherent and contextually relevant responses. ReWOO offers significant advantages in terms of token efficiency and robustness to tool failure. It enables LLMs to handle tasks where immediate access to external data is not available, relying instead on a well-structured reasoning framework. This method is particularly advantageous in scenarios where data retrieval is costly, slow, or uncertain, allowing the LLM-based agent to maintain a high level of performance and reliability.

Reason and Act (ReAct)[176] prompts LLMs to generate not only verbal reasoning but also actionable steps, thus enhancing the model's dynamic problem-solving capabilities. ReAct is grounded in the principle of integrating reasoning with action. In this approach, the LLM is prompted to alternate between generating reasoning traces (explanations) and taking actions (steps or commands) in an interleaved manner. This approach allows the model to dynamically reason about a problem, and propose and take concrete actions simultaneously.

Dialog-Enabled Resolving Agents (DERA) [177] are specialized AI agents that can engage in dialogue, resolve queries, and make decisions based on interactive exchanges. DERA is developed based on the idea of utilizing multiple agents within a dialog context, each with specific roles and functions. These agents can include Researchers, who gather and analyze information, and Deciders, who make final judgments based on the information provided. This division of roles allows for a well-organized and efficient approach to problem-solving and decision-making. DERA is particularly advantageous in scenarios requiring complex decision-making and problem-solving, such as those in medical diagnostics or customer service. The collaborative and interactive nature of DERA agents allows them to handle intricate queries with a level of depth and nuance that single-agent systems might struggle with. Moreover, this approach aligns well with human decision-making processes, making AI reasoning more relatable and trustworthy.

V. Popular Datasets for LLMs

Large language models exhibit promising accomplishments, but the main question that arises is how effectively they function and how their performance can be assessed in specific tasks or applications.

The evaluation of LLMs poses particular challenges due to the evolving landscape of their applications. The original intent behind developing LLMs was to boost the performance of NLP tasks such as translation, summarization, question-answering, and so on [178]. However, it is evident today that these models are finding utility across diverse domains including code generation and finance. Moreover, the evaluation of LLMs encompasses several critical considerations such as fairness and bias, fact-checking, and reasoning. In this section, we outline the commonly used benchmarks for assessing LLMs. These benchmarks are categorized based on training or evaluating the LLM Capabilities.

A. Datasets for Basic Tasks: language modeling/understanding/generation

This section provides an overview of the benchmarks and datasets suited to evaluate the basic abilities of LLMs.

- **Natural Questions** [179] is a QA dataset that consists of real anonymized, aggregated queries submitted to the Google search engine as questions. An annotator is presented with a question along with a Wikipedia page from the top 5 search results, and annotates a long answer (typically a paragraph) and a short answer

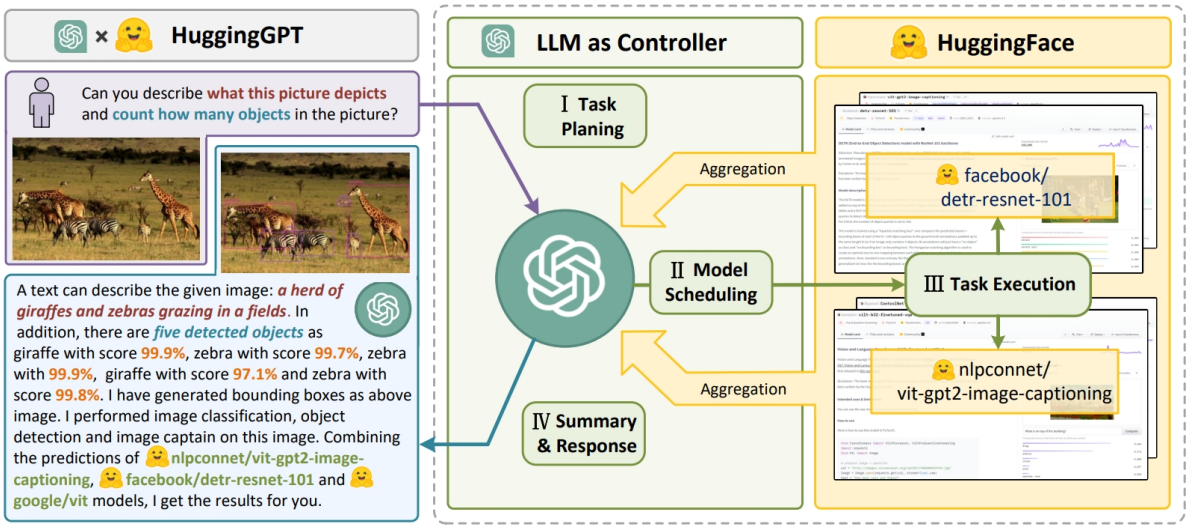


Fig. 39: HuggingGPT: An agent-based approach to use tools and planning [image courtesy of [171]]

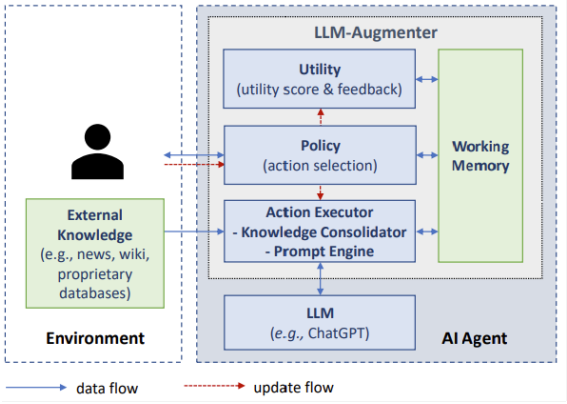


Fig. 40: A LLM-based agent for conversational information seeking. Courtesy of [36].

(one or more entities) if present on the page, or marks null if no long/short answer is present.

- **MMLU** [180] is intended to evaluate the knowledge gained in zero-shot and few-shot scenarios. That means that MMLU assesses both the general knowledge and problem-solving ability of a model. It covers 57 subjects in STEM, humanities, social sciences, and other areas. The benchmark varies in complexity, ranging from elementary to advanced professional. It is worth mentioning that the main contribution of this dataset is for multi-task language understanding, question answering, and arithmetic reasoning.
- **MBPP** [181] stands for "Mostly Basic Python Problems" and provides a benchmark for evaluating the performance of models designed for code generation. The benchmark encompasses 974 short Python programs including a wide range of topics, including fundamental programming concepts and standard library usage, and more. Each challenge comprises a task description, a code solution, and three automated test cases.
- **HumanEval** [182] is a dataset for code generation task. This dataset consists of 164 hand-crafted programming challenges. Each challenge is accompanied by a function signature, docstring, code body, and multiple unit tests. The main intuition behind developing this dataset is to guarantee the exclusion of its contents from training datasets for code generation models.
- **APPS** [183] is designed for code generation task focusing on the Python programming language. The APPS dataset contains a collection of $232,444$ Python programs. Each program in the dataset has an average of 18 lines of Python code. Additionally, APPS offers access to a repository of $10,000$ unique programming exercises, each with text-based problem descriptions. The final aspect to highlight is that the it includes test cases.
- **WikiSQL** [184] is crafted for code generation task and it has 87,726 carefully labeled pairs of SQL queries and corresponding natural language questions from Wikipedia tables. The SQL queries comprise three subsets: test sets ($17,284$ examples), development ($9,145$ examples), and training ($61,297$ examples).
- **TriviaQA** [185] is designed for QA task. This dataset comprises more than $650,000$ question-answer-evidence triples. There are $95,000$ question-answer pairs in this dataset, each authored by trivia enthusiasts and supported by an average of six independently sourced evidence documents. These documents are automatically acquired from Wikipedia or broader web search results. The dataset is categorized into two segments, including those with authentic answers from Wikipedia and web domains, and verified sets embody the accurately answered questions along with their associated documents from both Wikipedia and online.

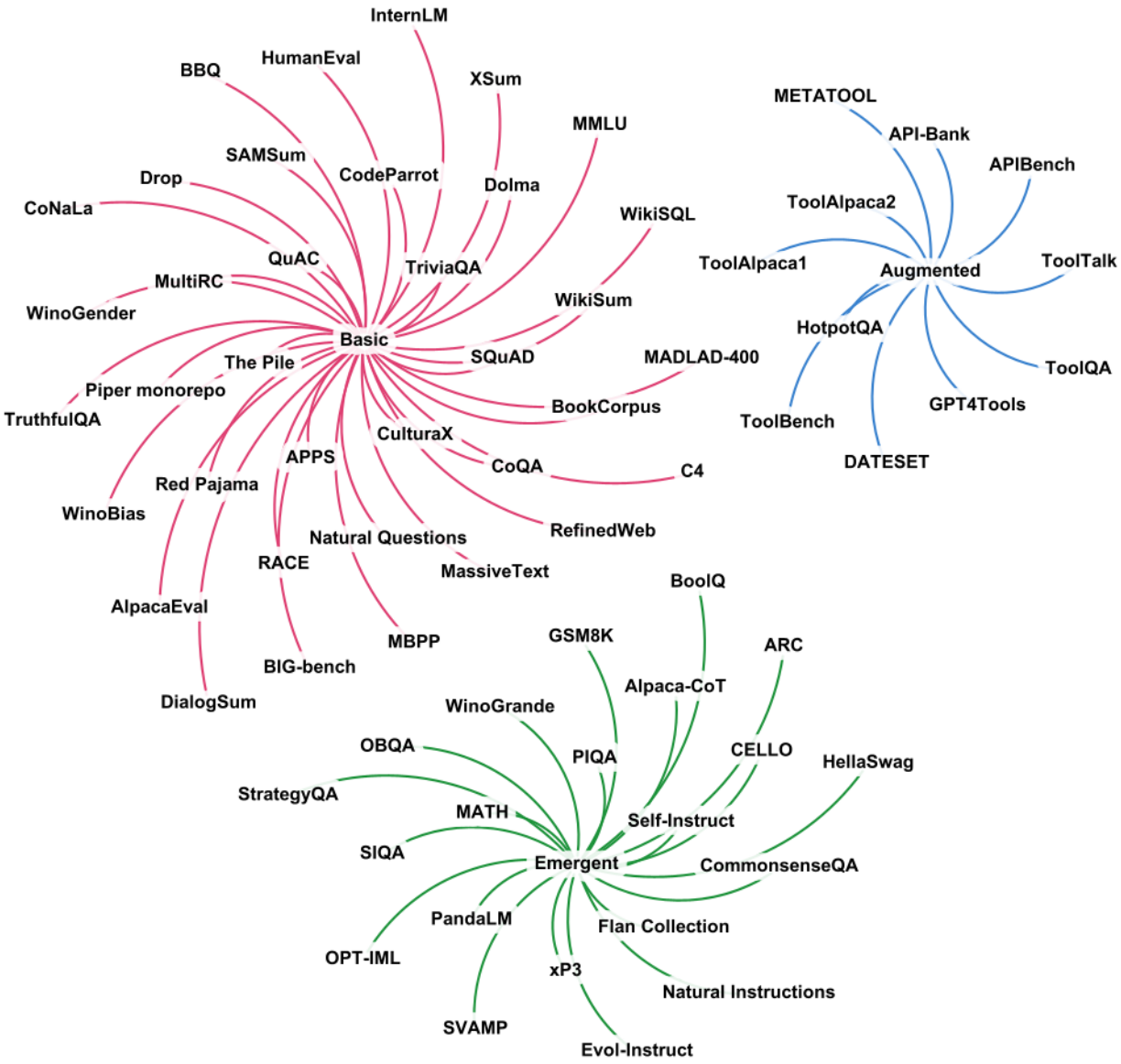


Fig. 41: Dataset applications.

- **RACE** [186] suits for reading comprehension task. This dataset is based on English tests completed by Chinese students from middle school and high school, aged 12 to 18, and it contains roughly $28,000$ texts and $100,000$ questions rigorously prepared by human specialists, primarily English instructors. This dataset contains a wide range of subjects that were purposefully chosen to assess students' comprehension and reasoning abilities. This dataset is available in three subgroups: RACE-M, RACE-H, and RACE. RACE-M refers to the middle school examinations, whereas RACE-H denotes the high school tests. Finally, RACE is the synthesis of RACE-M and RACE-H.

- **SQuAD** [187] stands for "Stanford Question Answering Dataset" and is a crowdsourced reading comprehension dataset based on Wikipedia articles. It has approximately $100,000$ question-answer pairs connected to more than 500 articles. The answers to these questions are typically text fragments or spans taken from the corresponding reading passages. The questions may be unanswerable in some cases. The dataset is divided into three sets: an 80% training set, a 10% development set, and a 10% hidden test set.

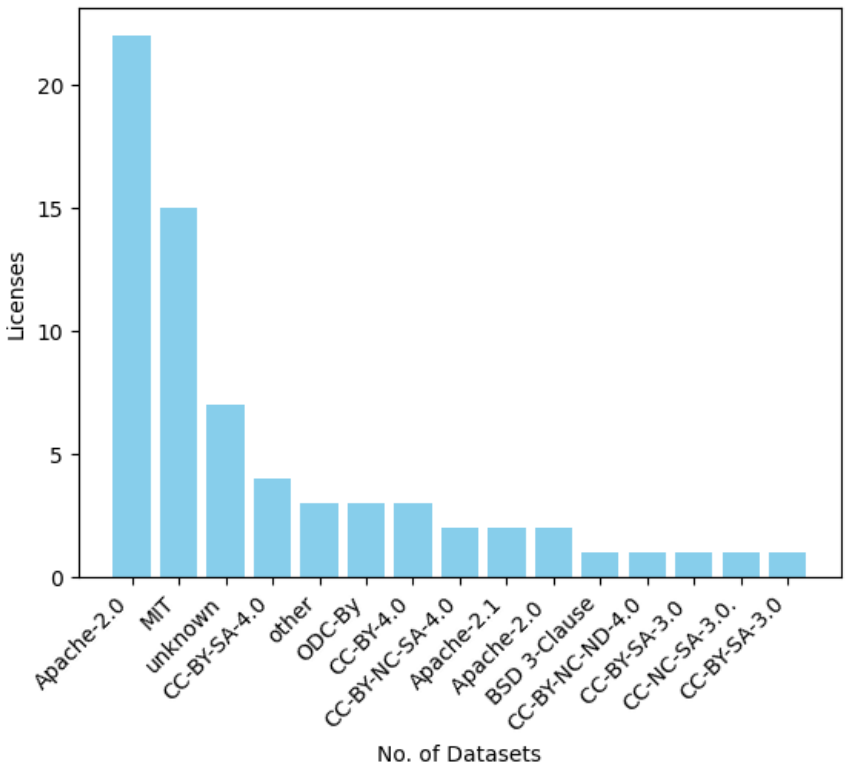


Fig. 42: Datasets licensed under different licenses.

- **BoolQ** [188] is a yes/no question-answering dataset where the goal is reading comprehension task. BoolQ includes $15,942$ examples. Each example is a triplet that includes a question, a relevant paragraph, and the solution. Although the main intuition behind this dataset is for reading comprehension, it can be used for reasoning, natural language inference, and question-answering tasks.
- **MultiRC** [189] is another dataset that fits reading comprehension task. MultiRC contains brief paragraphs as well as multi-sentence questions that can be answered using the information in the paragraph. The paragraphs in this dataset come from a variety of sources, including news, fiction, historical texts, Wikipedia articles, discussions on society and law, elementary school science textbooks, and 9/11 reports. Each question has many response choices, with one or more of them being correct. Answering the questions requires reasoning across several sentences. MultiRC dataset encompasses around $6,000$ multi-sentence questions gathered from over 800 paragraphs. On average, each question offers about two valid answer alternatives out of a total of five.

B. Datasets for Emergent: ICL, reasoning (CoT), instruction following

This section centers on the benchmarks and datasets employed to evaluate the emergent abilities of LLMs.

- **GSM8K** [190] is designed to evaluate the model's ability for multi-step mathematical reasoning. GSM8K includes 8.5K linguistically diverse grade school math word problems written by humans. The dataset is split into two sets: a training set with $7.5K$ problems, and a test set with 1K problems. These problems need 2 to 8 steps to be solved. Solutions mainly are a series of elementary calculations using basic arithmetic operations.
- **MATH** [191] enables to assess how well models can solve math problems. MATH dataset hast $12,500$ problems from high school math competitions. Each problem in the dataset has a step-by-step solution and a final answer enclosed in a box. The problems cover a wide range of topics and have different levels of complexity. There are seven subjects in total. Furthermore, the difficulty of each problem is rated based on the AoPS standards on a scale from $'1'$ to $'5'$. A $'1'$ shows the easiest problems in a subject, while $'5'$ represents the most difficult. In terms of formatting, all problems and solutions are presented using LATEX and the Asymptote vector graphics language.
- **HellaSwag** [192] is designed to assess commonsense reasoning in LLMs. This benchmark includes $70,000$ multiple-choice questions. Each question is derived from one of two domains: ActivityNet or WikiHow, and presents four answer choices regarding what might happen in the following situation. The correct answer provides an actual statement describing the

upcoming event, but the three wrong answers are created to confuse machines.

- **AI2 Reasoning Challenge (ARC)** [193] is used for commonsense reasoning. This benchmark encompasses $7,787$ science examination questions. These questions are in English, and most of them are set up in a multiple-choice format. The questions have been divided into two groups: a Challenge Set with $2,590$ difficult questions and an Easy Set with 5,197 questions. Each collection has also been pre-divided into Train, Development, and Test subsets.

- **PIQA** [194] is intended to evaluate the language representations on their knowledge of physical commonsense. In this dataset, the focus is on everyday situations with a preference for uncommon solutions. The central task is a multiple-choice question answering, where a question (q) is provided along with two potential solutions $(s1, s2)$. Then, the best solution is chosen by whether a model or a human. For each question, only one of the solutions is the correct answer.

- **SIQA** [195] provides a framework for evaluating models' ability for commonsense reasoning about social situations. SIQA dataset has $38,000$ multiple-choice questions designed to assess emotional and social intelligence in everyday circumstances. This dataset covers a wide variety of social scenarios. In SIQA, the potential answers is a mixture of human-selected responses and machine-generated ones that have been filtered through adversarial processes.

- **OpenBookQA (OBQA)** [196] is a new kind of question-answering dataset where answering its questions requires additional common and commonsense knowledge not contained in the book and rich text comprehension. This dataset includes around 6,000 multiple-choice questions. Each question is linked to one core fact, as well as an additional collection of over 6000 facts. The questions were developed using a multi-stage crowdsourcing and expert filtering procedure. OpenBookQA questions are difficult because they need multi-hop reasoning with limited background.

- **TruthfulQA** [197] is designed specifically to evaluate the truthfulness of language models in generating answers to questions. This dataset includes 817 questions, written by authors, from 38 different categories, including health, law, finance, and politics. These questions are purposefully designed to challenge human responders, as they may contain common misunderstandings that lead to incorrect answers.

- **OPT-IML Bench** [103] is a comprehensive benchmark for Instruction Meta-Learning. It covers 2000 NLP tasks from 8 existing benchmarks. The OPT-IML Bench consists of a training set with 17.9 M examples, a dev set with 145K samples, and a test set with 321K samples.

C. Datasets for Augmented: using external knowledge/tools

This section focuses on datasets designed for the augmented abilities of LLMs.

- **HotpotQA** [198] is designed to cover a diverse and explainable question-answering dataset that necessitates multi-hop reasoning. This dataset is derived from the English Wikipedia. It consists of roughly $113,000$ questions. Each question in the dataset comes with two paragraphs, called gold paragraphs, from two Wikipedia articles. Also, there is a list of sentences in those paragraphs that crowdworkers have picked as important for answering the question.

- **ToolQA** [199] is a question answering benchmark to evaluate LLMs' ability to use external tools for answering questions.

- **GPT4Tools** serves as an instructional dataset, generated by instructing advanced teachers (such as ChatGPT), with instructions conditioned on visual content and tool descriptions. This process results in the generation of instructions related to the use of tools. There are three versions of this dataset. The first version comprises 71,000 instruction-following data points utilized to fine-tune the GPT4Tools model. The next version consists of manually cleaned instruction data used for validation, covering instructions related to the tools from the first version. The last version is cleaned instruction data used for testing and includes instructions related to some tools that are not present in the first version.

VI. Prominent LLMs' Performance on Benchmarks

In this section we first provide an overview of some of popular metrics used for evaluating the performance of LLMs under different scenarios. We then look at the performance of prominent large language models on some of the popular datasets and benchmarks.

A. Popular Metrics for Evaluating LLMs

Evaluating the performance of generative language models depends on the underlying task they are going to be used for. Tasks that are mostly about selecting a choice out of given ones (such as sentiment analysis), can be seen as simple as classification and their performance can be evaluated using classification metrics. Metrics such as accuracy, precision, recall, F1, etc are applicable in this case. It is also important to note that the answers generated by the model for specific tasks such as multi-choice question answering are always either True or False. If the answer is not in a set of options, it can be seen as False as well.

However, some tasks that are purely open-ended text generation cannot be evaluated in the same way as for categorization. Different metrics are required for the specific purpose of the evaluation. Code generation is a very different case in open-ended generative evaluations. The generated code must pass the test suite but on the other hand, it is also important to understand if a model is capable of generating different

TABLE II: LLM Datasets Overview.

Benchmark Name	Evaluation Metric	Leaderboard	Source	paperswithcode
HumanEval	PASS@k	Link	Link	Link
MBPP	PASS@k, Accuracy	-	Link	Link
APPS	PASS@k, Accuracy	-	Link	Link
WikiSQL	Accuracy	-	Link	Link
CoNaLa	BLEU		Link	Link
CodeParrot	PASS@k	-	Link	-
HellaSwag	Accuracy	Link	Link	Link
AI2 Reasoning Challenge (ARC)	Accuracy	Link	Link	Link
BoolQ	Accuracy	-	Link	Link
MultiRC	F1-score, Accuracy	-	Link	Link
CNN/Daily Mail [200]	Accuracy	-	Link	-
SQuAD	F1-score, EM	Link	Link	Link
RACE	Accuracy	-	Link	Link
CNN/Daily Mail [201]	ROUGE	-	Link	Link
Drop	F1-score, EM	Link	Link	Link
QuAC	F1-score, HEQ-Q, HEQ-D	Link	Link	Link
TriviaQA	EM, F1-score, Accuracy	Link	Link	Link
Natural Questions	EM, F1-score, Accuracy	Link	Link	Link
StrategyQA	Accuracy, Recall@10, SARI	Link	Link	Link
CoQA	F1-score	Link	Link	Link
XSum	ROUGE	-	Link	Link
SAMSum	ROUGE	-	-	Link
WikiSum	ROUGE	-	Link	-
DialogSum	ROUGE	-	Link	Link
TruthfulQA	MC1 , MC2, % true, % info, BLEURT	Link	Link	Link
MMLU	Accuracy	Link	Link	Link
GSM8K	Accuracy	Link	Link	Link
PIQA	Accuracy	Link	Link	Link
SIQA	Accuracy	Link	Link	Link
OpenBookQA (OBQA)	Accuracy	Link	Link	Link
HotpotQA	EM, F1-score, Joint EM, Joint F1-score,	Link	Link	Link
MATH	Accuracy	-	Link	Link
CommonsenseQA	Accuracy	Link	Link	Link
Natural Instructions	ROUGE-L, Human	Link	Link	Link
BIG-bench	Accuracy, Average	-	Link	Link
ToolTalk	Success rate, Precision, Recall, Incorrect action rate, Percent of failing error types	-	Link	Link
MetaTool	Accuracy, Precision, Recall, F1-score	-	Link	Link
GPT4Tools	Successful Rate of Thought, Successful Rate of Action, Successful Rate of Arguments, Success Rate	-	Link	Link
API-Bank	Correctness, ROUGE, Error(API Hallucination, Has Exception, Invalid Input Parameters, False API Call Format, API Call, Miss Input Parameters)	-	Link	Link
Alpaca-CoT	-	-	Link	Link

solutions as a code, what is the probability of selecting the correct one among them. Pass@k is a very good metric in this case. It works in this manner that given a problem, different solutions as code are generated. They are tested for correctness using different functionality tests. Afterward, from generated n solutions, and the respective c number of them being correct equation 4 provides the final value.

$$\text{pass@}k := \mathop{\mathbb{E}}_{\text{Problems}} \left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right] \tag{4}$$

Exact match (EM) is another metric that is mostly concerned with exact matches from (pre-defined) answers. It counts a prediction as correct if it exactly matches one of more than one desired reference text token by token. In some cases, it can be the same as accuracy and the equation 5 shows the mathematical definition. Here M is total number of correct answers and N is the total number of questions [202].

$$EM = \frac{M}{N} \tag{5}$$

Human equivalence score (HEQ) on the other hand, is an alternative to F1 score [203]. HEQ-Q represents the precision of individual questions, wherein an answer is deemed correct if the model's F1 score surpasses the average human F1 score. Likewise, HEQ-D denotes the precision of each dialogue; it is deemed accurate when all questions within the dialogue meet the criteria of HEQ [182].

Evaluation of other generative tasks such as machine translation are based on metrics such as Rouge and BLEU. These scores work well when there is a reference text as ground truth (such as translation) and a hypothesis that is generated by the generative model, in our case the LLM. These scores are mostly used for cases where the goal is to detect the similarity of the answer and ground truth in a computation manner. In a computation manner, it meant that nothing more than N-Grams would be used. However, metrics such as BERT-Score are also good for these cases but they are also heavily

TABLE III: LLM categories and respective definitions.

Classification	Category	Description
Size	Small	Number of parameters $\leq$ 1B
	Medium	1B $<$ Number of parameters $\leq$ 10B
	Large	10B $<$ Number of parameters $\leq$ 100B
	Very Large	100B $<$ Number of parameters
Type	Foundation model	Pretrained language model
	Instruction model	Pretrained and instruction fine-tuned language model
	Chat model	Pretrained, instruction fine-tuned, and chat fine-tuned language model
Origin	Original model	An original model released with either Foundation, Instruction, or Chat model
	Tuned model	Fine-tuned version of an original model
Availability	Publicly available	Model and weights are available due to request to without request
	Publicly unavailable	Model and weights are not publicly available

TABLE IV: Different LLM categorization.

Model	Size	#Params (B)	Type	Availability	Origin
Davinci-002	Very Large	175	Instruction	Unavailable	Tuned
Davinci-003	Very Large	175	Instruction	Unavailable	Tuned
GPT 3.5-turbo	Large	20	Chat	Unavailable	Tuned
Falcon 7B	Medium	7	Foundation	Public	Original
Alpaca	Large	13	Chat	Public	Tuned
Pythia 7B	Medium	7	Foundation	Public	Original
Pythia 12B	Large	12	Foundation	Public	Original
LLAMA 7B	Medium	7	Chat	Public	Original
LLAMA 2 7B	Medium	7	Chat	Public	Tuned
LLAMA 2 7B	Medium	7	Foundation	Public	Original
Vicuna 13B	Large	13	Foundation	Public	Tuned
Vicuna 7B	Medium	7	Foundation	Public	Tuned
Claude	Large	93	Chat	Unavailable	Original
Claude 2	Very Large	137	Chat	Unavailable	Original

erroneous because another model is used to judge. Still, even today, evaluating purely generated content is very hard and no completely fitting metric is not found, metrics are either looking for simplistic features such as N-Gram, SkipGram, etc, or they are models with unknown accuracy and preciseness [204].

Generative evaluation metrics are also another type of evaluation metric for LLMs that use another LLM for evaluating the answer. However, depending on the task itself, evaluation can be possible in this way or not. Another dependency that makes generative evaluation error-prone is reliance on the prompt itself. RAGAS is one of the good examples that incorporate the usage of generative evaluation.

Various benchmarks and leaderboards have been proposed to address the most challenging question in the world of large language models: Which one is better? However not a simple answer can address this question. The answer depends on various aspects of large language models. Section V shows the categorical presentation of different tasks and the most important datasets in each category. We will follow the same categorization and provide a comparison based on each category. After providing comparison for each category, we will provide a broad overview of aggregated performance by averaging the reported performance metric on different tasks.

Evaluating different LLMs can be seen also from different perspectives. For example, a LLM with a drastically fewer number of parameters is not completely comparable to one with a larger number of parameters. From this perspective, we will categorize LLMs in four categories as well: **small** (less than or equal to 1 billion parameters), **medium** (between 1 and 10 billion), **large** (between 10 and 100 billion), and **very large** (more than 100 billion). Another classification for the LLMs we use is their primary use case. We consider each LLM to be either: **Foundation** model (pretrained language model with no instruction fine-tuning and chat fine-tuning), **Instruction** model (pretrained language model with only instruction fine-tuning), and **Chat** model (pretrained language model with instruction and chat fine-tuning). Apart from all the categorization described, another category is required to distinguish between original models and tuned ones. **Original** models are those that have been released as a foundation model or a fine-tuned one. **Tuned** models are those that grasped the original model and tuned it with different datasets or even different training approaches. It is also good to note that original models are usually foundation models that have been fine-tuned on specific datasets or even different approaches. Availability of the model weights regardless of the license is another category in our classification. Models that have their weights publicly available (even through request) are noted as **Public** models while others are noted as **Private**. Table III shows all of these definitions and abbreviations used in the rest of the article. Figure 43 illustrate these visually.

According to the provided categorizations, we can categorize and label each notable LLM as shown in table IV. As can be seen from this table, models categorized as very large are also unavailable as well.

B. LLMs' Performance on Different Tasks

Commonsense reasoning is one of the important capabilities each model can obtain. This capability denotes the ability of the model to use prior knowledge in combination with reasoning skills. In the case of HellaSwag for example, finding the continuation of text is challenging because the given text contains a partial part of the story while the given choices as continuation are tricky to select, and without having prior

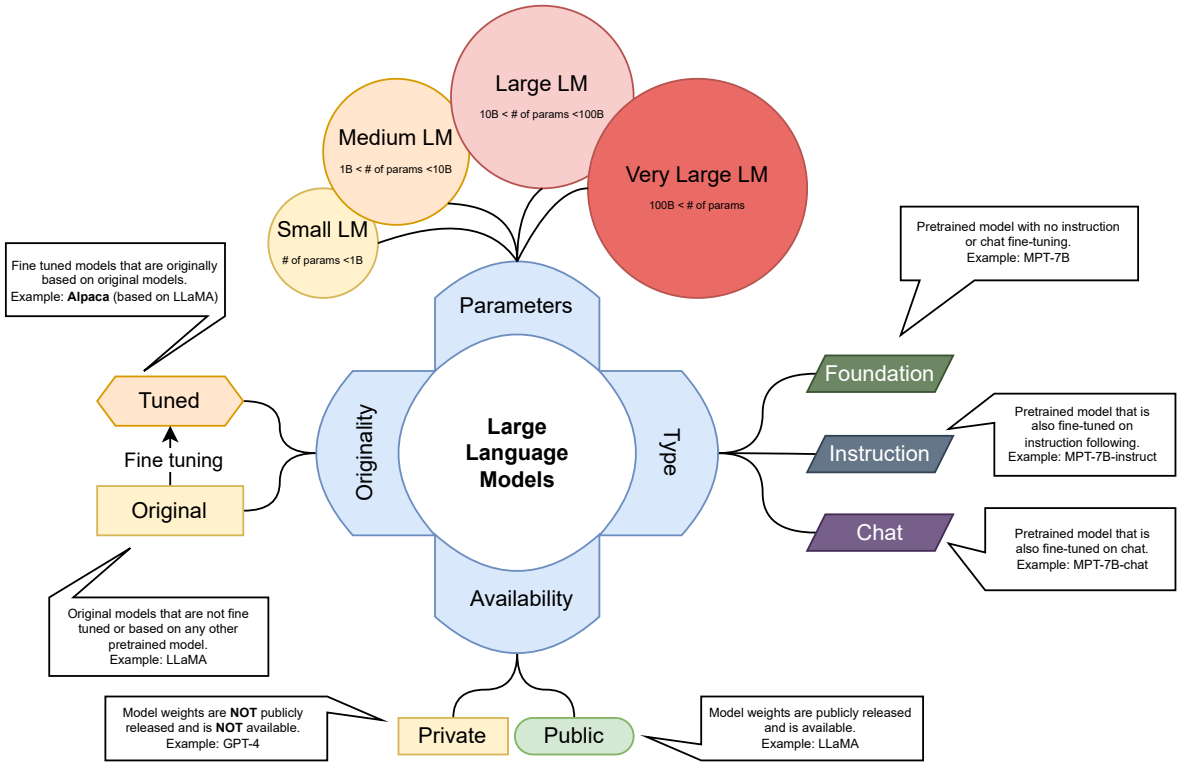


Fig. 43: LLM categorizations.

knowledge about the world it is not possible. This specific kind of reasoning deserves high attention because it is related to utilizing previous knowledge with open text-described scenes or facts. As can be seen from table V not just Unavailable models but also Public ones can achieve good results on various tests.

TABLE V: Commonsense reasoning comparison.

Model	OBQA	HellaSwag
Davinci-003	**51**	83.4
Falcon 7B	44.4	76.3
Alpaca	43.4	73.9
Pythia 7B	37.2	64
Pythia 12B	43.2	68.1
LLAMA 7B	42.4	73
Dolly 6B	41.2	67.6
Dolly 12B	40.4	71
Alpaca 7B	43.4	73.9
Alpaca Lora 7B	42.6	74
GPT-J 6.7B	38.2	66.2
LLama 7B	42.4	73
LLama 13B	42.2	76.2
Pythia 6.7B	37.2	64
Pythia 12B	38	67.3
StableLM Tuned	33.4	53.6
Koala 13B	42.8	72.6
Mosaic mpt-7B	42.6	76.3
LLAMA 2 70B	-	87.33
LLAMA 65B	-	86.09
Falcon 40B	-	85.3
Falcon 180B	-	88.86
MPT Instruct 30B	-	84.31
MPT Instruct 7B	-	77.91
Yi 6B	-	76.42
Yi 34B	-	85.69
GPT-4	-	**95.3**
Gemini Ultra	-	87.8

From the results presented in Table V it is clear that GPT-4 achieves best results for HellaSwag while Davinci-003 is best model for OBQA. It is also good to note that results for OBQA are not reported for all of the models and possibly davinci-003 is not the best model achieving highest results on OBQA.

Not all models report their performance on all datasets, and because of that, the number of models for which performance is reported in different tables varies.

TABLE VI: Symbolic reasoning comparison.

Model	Cobjects	Penguins
GPT-NeoX	26	33.56
OPT 66B	31.2	28.08
Bloomberg GPT	34.8	37.67
BLOOM 176B	36.8	40.41
PaLM 540B	38	44.5
Gopher-280B	49.2	40.6
Chinchilla-70B	59.7	48.7
PaLM 2	61.2	65.8

World knowledge is mostly about general knowledge questions, for example, in Wikifact dataset questions such as ”Who is the author of a specific well-known book” can be found and references are also provided. Table VII shows the results.

TABLE VII: World knowledge comparison.

Model	TriviaQA	NaturalQ	WebQ	ARC
BLOOM	-	-	-	32.9
BLOOM 176B	-	-	-	50.85
Bloomberg GPT	-	-	-	48.63
Chinchilla	-	35.5	-	-
Codex + REPLUG	76.8	44.7	-	-
GAL 120B	-	-	-	67.9
GLaM 62B/64E	75.8	32.5	15.5	50.3
Gopher	-	28.2	-	-
GPT-3 175B	71.2	29.9	41.5	85.2
GPT-4	-	-	-	96.4
GPT-NeoX	-	-	-	45.39
LLaMA 13B	-	-	-	52.7
LLaMA 2 70B	85	33	-	-
LLaMA 33B	-	24.9	-	57.8
LLaMA 65B	72.6	39.9	-	-
LLaMA 7B	-	-	-	47.6
Mistral 7B	69.9	28.8	-	55.5
Neo-6B	-	13.7	-	-
OPT	-	-	-	31.1
OPT 66B	-	-	-	44.54
OPT-175B	-	-	-	43.94
OPT-175B	-	-	-	25.6
PaLM 2-L	86.1	37.5	28.2	95.1
PaLM 2-M	81.7	32	26.9	64.9
PaLM 2-S	75.2	25.3	21.8	59.6
PaLM-540B	81.4	39.6	43.5	87.1
phi-1.5-web 1.3B	-	-	-	44.9
SparseGPT	-	-	-	38.99
SparseGPT	-	-	-	39.85
SparseGPT	-	-	-	41.3

For some specific use-case models, it is highly demanded to have coding and code-generation capability. Table VIII shows the results of different models on coding capability.

TABLE VIII: Coding capability comparison.

Model	HumanEval
Gemini Ultra	74.4
Gemini Pro	67.7
GPT-4	67
WizardCoder 15B	57.3
phi-1 1.3B	50.6
Code Llama	48.8
GPT-3.5	48.1
OctoCoder	46.2
phi-1-small	45
PaLM 2-S	37.6
InstructCodeT5+ 16B	35
Mistral 7B	30.5
LLaMA 2	29.9
phi-1-base	29
Codex-12B	28.81
PaLM 540B	26.2
CodeT5+ 2B	24.2
LLaMA 65B	23.7
LLaMA 33B	21.7
PaLM 62B	15.9
LLaMA 13B	15.8
LaMDA 137B	14
MIM-350M	13.7
LLaMA 7B	10.5
PaLM 8B	3.6

Arithmetic reasoning is another challenging reasoning capability to achieve. GSM8K for example contains grade school mathematical questions with respect to their answers. Table IX provides an insight for different model comparisons.

TABLE IX: Arithmetic reasoning comparison.

Model	GSM8k	MATH
Gemini Ultra	94.4	53.2
GPT-4	87.1	42.5
Gemini Pro	86.5	32.6
ToRA 70B	84.3	49.7
MathCoder-L-70B	83.9	-
MetaMath 70B	82.3	26
MuggleMATH 70B	82.3	-
MathCoder-CL-34B	81.7	45.2
ToRA-Code 34B	80.7	50.8
MetaMath-Mistral-7B	77.7	-
Arithmo2-Mistral-7B	76.4	-
ToRA-Code 13B	75.8	48.1
Arithmo-Mistral-7B	74.7	-
MathCoder-CL-13B	74.1	35.9
MuggleMATH 13B	74	-
CodeT5+	73.8	-
KwaiYiiMath 13B	73.3	-
ToRA-Code 7B	72.6	44.6
MathCoder-L-13B	72.6	29.9
MetaMath 13B	71	22.5
LLaMA 65B	69.7	10.6
MuggleMATH 7B	68.4	-
MathCoder-CL-7B	67.8	23.3
MetaMath 7B	66.4	19.4
RFT 70B	64.8	-
MathCoder-L-7B	64.2	-
Orca 2-13B	59.14	-
U-PaLM	58.5	-
PaLM-540B	58.1	8.8
LLaMA 2 70B	56.8	-
RFT 13B	55.3	-
LLaMA 33B	53.1	7.1
Mistral 7B	52.2	13.1
RFT 7B	51.2	-
LLaMA 65B	50.9	20.5
Orca 2-7B	47.23	-
Text-davinci-002	40.7	19.1
LLaMA 33B	35.6	3.9
GPT-Neo-2.7B	19.5	-
LLaMA 7B	18.1	2.9
PaLM 540B	17.9	8.8
LLaMA 13B	17.8	3.9
LLaMA 7B	11	2.9
GPT-Neo-125M	7.5	-
PaLM 8B	4.1	1.5
GPT-2	-	5.4
GPT-3 175B	-	5.2
PaLM 62B	-	4.4
GPT-3-13B	-	3
LLaMA 7B	11	2.9
PaLM 8B	-	1.5

Large language models in some cases are hallucinating answers simply because they are next-token prediction machines. Hallucination is one of the important factors in measuring how much a large language model is trustworthy and reliable. Measuring hallucination on the other hand is also not easy as it seems because each fact can be written in different styles and even the smallest changes in writing make it hard to detect. It is fair to assume if any particular LLM is more capable to detect hallucination of false information in text, it is also more trustworthy. HaluEval is one of the datasets that aims to measure hallucination in this field [205]. Evaluation can also be performed by another model judging the response with regard to the actual answer [206]. Table X shows the evaluation of different models based on these datasets.

VII. Challenges and Future Directions

As we have seen in the previous sections, large language models have achieved impressive results in the past 1-2 years.

TABLE X: Hallucination evaluation

Model	HHEM	HaluEval QA	HaluEval Dialogue	HaluEval Sum.	HaluEval General
GPT 4	97	-	-	-	-
GPT 4 Turbo	97	-	-	-	-
GPT 3.5 Turbo	96.5	62.59	72.4	58.53	79.44
Davinci002	-	60.05	60.81	47.77	80.42
Davinci003	-	49.65	68.37	48.07	80.4
GPT-3	-	49.21	50.02	51.23	72.72
Google Gemini Pro	95.2	-	-	-	-
Llama 2 70B	94.9	-	-	-	-
Llama 2 7B	94.4	49.6	43.99	49.55	20.46
Llama 2 13B	94.1	-	-	-	-
Cohere-Chat	92.5	-	-	-	-
Cohere	91.5	-	-	-	-
Claude 2	91.5	69.78	64.73	57.75	75
Claude 1		67.6	64.83	53.76	73.88
Microsoft Phi 2	91.5	-	-	-	-
Google Palm 2 (beta)	91.4	-	-	-	-
Mixtral 8x7B	90.7	-	-	-	-
Amazon Titan Express	90.6	-	-	-	-
Mistral 7B	90.6	-	-	-	-
Google Palm 2 Chat (beta)	90	-	-	-	-
Google Palm 2	87.9	-	-	-	-
Google Palm 2 Chat	72.8	-	-	-	-
ChatGLM	-	47.93	44.41	48.57	30.92
Falcon	-	39.66	29.08	42.71	18.98
Vicuna	-	60.34	46.35	45.62	19.48
Alpaca	-	6.68	17.55	20.63	9.54

At the same time this is still a new and extremely active research area where the pace of innovation is increasing rather than slowing down. As in any other evolving area though, there are still numerous challenges ahead. Here we briefly mention some of the challenges and main active areas which are known so far. It is worth noting that LLM challenges are discussed in details in a work by Kaddour et al. [207].

A. Smaller and more efficient Language Models

This is a survey on *large* language models, and there has been an initial push towards "larger is better" that has clearly been rewarded with ever larger models like GPT-4 getting better accuracy and performance in benchmarks. However, those large models are costly and inefficient in several dimensions (e.g. high latency). In response to all of this, there is a current research trend to come up with Small Language Models (SLMs) as a cost-effective alternative to LLMs, particularly when used on specific tasks that might not require the full generality of larger models. Prominent works in this direction include Phi-1 [208], Phi-1.5 [209], and Phi-2 from Microsoft.

More generally, we should expect many research efforts in this area of how to train smaller and more efficient models. Techniques such as parameter-efficient fine-tuning (PEFT), teacher/student, and other forms of distillation – see section III-I – will continue to be used to build a smaller model out of larger ones.

B. New Post-attention Architectural Paradigms

Transformer blocks have been a crucial and constant part of most of current LLM frameworks, and it's a big question mark how much longer this architecture will be in vogue, and what will be the next big architectural break-through in the field of deep learning (and NLP). Since AlexNet in 2012, we have seen many architectures go in and out of fashion, including LSTM, GRU, seq2seq, but Transformers have been the dominant approach since its inception. As described earlier, attention is the main mechanism driving transformers. More recently, there has been promising research in alternative approaches that are being labelled as post-attention.

An important class of such class of post-attention models are the so called State Space Models (SSMs). While the notion of State Space Models has a long history in machine learning, it should be noted that in the context of language models, SSM is usually used in reference to the newer Structure State Space Model architecture or S4 for short (see Gu et al. [29]). Some recent models in this category are Mamba [30], Hyena [210], and Striped Hyena [211].

While all of those models are very competitive in terms of performance in leaderboards and efficiency, they also address an important challenge in more traditional attention-based architectures: *the lack of support for larger context windows*.

Having a good answer to many prompts requires context. For example, the response to "Recommend some good movies for me" requires a lot of context about "me" as well as what movies are available and which ones I have not watched. Context length is especially important for RAG, where large portions of text might be retrieved and injected into the prompt for generation (see section IV-C.

The longer the context length, the more tokens we can squeeze into the context. The more information the model has access to, the better its response will be. But on the other hand, with very long context, it would be hard for the model to remember everything and efficiently process all the information. Attention-based models are highly inefficient for longer contexts and that is why we should expect more research in different mechanisms that enable processing longer contexts and generally come up with more efficient architectures.

That being said, new architectures might not only propose

alternatives for the attention mechanism but rather rethink the whole Transformer architecture. As an early example of this, Monarch Mixer [212] proposes a new architecture that uses the same sub-quadratic primitive that achieves high hardware efficiency on GPUs – Monarch matrices – along both sequence length and model dimension.

On the other end of the spectrum, it is worth mentioning that there are some attention-compatible architectural mechanisms that have been recently gaining steam and proving their value in creating better and more powerful LLMs. Probably the best example of such mechanism is Mixture of Experts (MoE). MoEs have been around in machine learning for years, even before the Deep Learning Era [213], but they have been gaining popularity since then, and particularly in the context of Transformer models and LLMs.

In LLMs, MoEs allow to train an extremely large model than is then only partially instantiated during inference when some of the experts are turned off wherever the gating/weighting function has a low weight assigned to them. As an example, the GLaM model has 1.2 trillion parameters, but during inference only 2 out of the 64 experts are used [84].

MoEs are nowadays an important component of the so-called frontier LLMs (i.e. the most advanced and capable models). GPT-4 itself is rumored to be based on a MoE architecture, and some of the best performing LLMs such as Mixtral [117], are basically an MoE version of pre-existing LLMs.

Finally, it is important to note that MoEs can be used as a component of any architecture regardless of whether it is based on attention or not. In fact, MoEs have also been applied to SSM-based LLMs like Mamba citepioro2024moemamba. We should continue to see MoE-driven improvements in the future regardless of the underlying architecture.

C. Multi-modal Models

Future LLMs are expected to be multi-modal and handle a variety of data types, such as text, images, and videos, audio, in a unified manner. This opens up possibilities for more diverse applications in fields like question answering, content generation, creative arts, and healthcare, robotics, and beyond. There are already several prominent multi-modal LLMs out there, including: LLAVA [214], LLAVA-Plus [215], GPT-4 [33], Qwen-vl [116], Next-GPT [216], but the trend is expected to be continued. Evaluation of these models also is a new research topic, especially conversational generative vision models [217]. Multi-modal LLMs can unlock huge potentials in a variety of tasks, and there has already been a descent progress in this direction, which needs a dedicated paper to discuss all its details.

D. Improved LLM Usage and Augmentation techniques

As we described in sectionIV, many of the shortcomings and limitations of LLMs such as *hallucination* can be addressed through advanced prompt engineering, use of tools, or other augmentation techniques. We should expect not only continued, but accelerated research in this area. It is worth mentioning that, in the specific case of software engineering, some works ([218]) tried to automatically eliminate this issue from the overall software engineering workflow

LLM-based systems are already starting to replace machine learning systems that were until recently using other approaches. As a clear example of this, LLMs are now being deployed to better understand people preference and interests, and provide more personalized interactions, whether in customer service, content recommendation, or other applications. This involves better understanding of user preferences, and analyzing their past interactions and using them as the context. We will continue to see research in the application and usage of LLMs for not only *personalization and recommendations*, but many other application areas using other machine learning techniques.

Finally, another important area of research we expect to gather increased attention is that of *LLM-based agents and multi-agent systems* [172], [173], [174]. The development of LLM systems with access to external tools and decision-making capabilities is both exciting and challenging. We will see continued research and progress in this important area that some argue could lead to Artificial General Intelligence (AGI).

E. Security and Ethical/Responsible AI

Ensuring the robustness and security of LLMs against adversarial attacks and other vulnerabilities is a critical area of research [219]. As LLMs are increasingly deployed in real-world applications, they need to be protected from potential threats, to prevent them being used to manipulate people or spread mis-information.

Addressing ethical concerns and biases in LLMs is another active area of research. Efforts are being made to ensure that LLMs are fair, unbiased, and capable of handling sensitive information responsibly. As LLMs are being used more and more by a large number of people on a daily basis, making sure they are unbiased and behave responsibly is crucial.

VIII. Conclusion

This paper present a survey of LLMs developed in the past few years. We first provide an overview of early pre-trained language models (e.g., as BERT), then review three popular LLM families (GPT, LLaMA, PaLM), and other representative LLMs. We then survey methods and techniques of building, augmenting, and using LLMs. We review popular LLM datasets and benchmarks, and compare performance of a set of prominent models on public benchmarks. Finally, we present open challenges and future research directions.

References

[1] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[2] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[3] C. E. Shannon, "Prediction and entropy of printed english," *Bell system technical journal*, vol. 30, no. 1, pp. 50–64, 1951.

[4] F. Jelinek, *Statistical methods for speech recognition*. MIT press, 1998.

[5] C. Manning and H. Schutze, *Foundations of statistical natural language processing*. MIT press, 1999.

[6] C. D. Manning, *An introduction to information retrieval*. Cambridge university press, 2009.

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

[8] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He *et al.*, "A comprehensive survey on pretrained foundation models: A history from bert to chatgpt," *arXiv preprint arXiv:2302.09419*, 2023.

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

[10] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, and Z. Sui, "A survey for in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.

[11] J. Huang and K. C.-C. Chang, "Towards reasoning in large language models: A survey," *arXiv preprint arXiv:2212.10403*, 2022.

[12] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999.

[13] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Advances in neural information processing systems*, vol. 13, 2000.

[14] H. Schwenk, D. Déchelotte, and J.-L. Gauvain, "Continuous space language models for statistical machine translation," in *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, 2006, pp. 723–730.

[15] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, no. 3. Makuhari, 2010, pp. 1045–1048.

[16] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[17] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 2333–2338.

[18] J. Gao, C. Xiong, P. Bennett, and N. Craswell, *Neural Approaches to Conversational Information Retrieval*. Springer Nature, 2023, vol. 44.

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

[20] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[21] H. Fang, S. Gupta, F. Iandola, R. K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt *et al.*, "From captions to visual concepts and back," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1473–1482.

[22] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[23] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations. corr abs/1802.05365 (2018)," *arXiv preprint arXiv:1802.05365*, 2018.

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

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

[26] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[27] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang *et al.*, "Pre-trained models: Past, present and future," *AI Open*, vol. 2, pp. 225–250, 2021.

[28] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*, vol. 63, no. 10, pp. 1872–1897, 2020.

[29] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," 2022.

[30] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

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

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

[33] OpenAI, "GPT-4 Technical Report," https://arxiv.org/pdf/2303.08774v3.pdf, 2023.

[34] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 24 824–24 837. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[35] G. Mialon, R. Dessì, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Raileanu, B. Rozière, T. Schick, J. Dwivedi-Yu, A. Celikyilmaz *et al.*, "Augmented language models: a survey," *arXiv preprint arXiv:2302.07842*, 2023.

[36] B. Peng, M. Galley, P. He, H. Cheng, Y. Xie, Y. Hu, Q. Huang, L. Liden, Z. Yu, W. Chen, and J. Gao, "Check your facts and try again: Improving large language models with external knowledge and automated feedback," *arXiv preprint arXiv:2302.12813*, 2023.

[37] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[38] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning internal representations by error propagation," 1985.

[39] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[40] M. V. Mahoney, "Fast text compression with neural networks." in *FLAIRS conference*, 2000, pp. 230–234.

[41] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ, "Strategies for training large scale neural network language models," in *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, 2011, pp. 196–201.

[42] tmikolov. rnnlm. [Online]. Available: https://www.fit.vutbr.cz/~imikolov/rnnlm/

[43] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: a comprehensive review," *ACM computing surveys (CSUR)*, vol. 54, no. 3, pp. 1–40, 2021.

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

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

[46] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[47] G. Lample and A. Conneau, "Cross-lingual language model pretraining," *arXiv preprint arXiv:1901.07291*, 2019.

[48] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[49] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H.-W. Hon, "Unified language model pre-training for

natural language understanding and generation," *Advances in neural information processing systems*, vol. 32, 2019.

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

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

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

[53] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel, "mt5: A massively multilingual pre-trained text-to-text transformer," *arXiv preprint arXiv:2010.11934*, 2020.

[54] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, "Mass: Masked sequence to sequence pre-training for language generation," *arXiv preprint arXiv:1905.02450*, 2019.

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

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

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

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

[59] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[60] OpenAI. (2022) Introducing chatgpt. [Online]. Available: https://openai.com/blog/chatgpt

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

[62] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Alpaca: A strong, replicable instruction-following model," *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, vol. 3, no. 6, p. 7, 2023.

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

[64] X. Geng, A. Gudibande, H. Liu, E. Wallace, P. Abbeel, S. Levine, and D. Song, "Koala: A dialogue model for academic research," *Blog post, April*, vol. 1, 2023.

[65] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[66] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[67] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive apis," 2023.

[68] A. Pal, D. Karkhanis, M. Roberts, S. Dooley, A. Sundararajan, and S. Naidu, "Giraffe: Adventures in expanding context lengths in llms," *arXiv preprint arXiv:2308.10882*, 2023.

[69] B. Huang, "Vigogne: French instruction-following and chat models," https://github.com/bofenghuang/vigogne, 2023.

[70] Y. Wang, H. Ivison, P. Dasigi, J. Hessel, T. Khot, K. R. Chandu, D. Wadden, K. MacMillan, N. A. Smith, I. Beltagy *et al.*, "How far can camels go? exploring the state of instruction tuning on open resources," *arXiv preprint arXiv:2306.04751*, 2023.

[71] S. Tworkowski, K. Staniszewski, M. Pacek, Y. Wu, H. Michalewski, and P. Miłoś, "Focused transformer: Contrastive training for context scaling," *arXiv preprint arXiv:2307.03170*, 2023.

[72] D. Mahan, R. Carlow, L. Castricato, N. Cooper, and C. Laforte, "Stable beluga models." [Online]. Available: https://huggingface.co/stabilityai/StableBeluga2

[73] Y. Tay, J. Wei, H. W. Chung, V. Q. Tran, D. R. So, S. Shakeri, X. Garcia, H. S. Zheng, J. Rao, A. Chowdhery *et al.*, "Transcending scaling laws with 0.1% extra compute," *arXiv preprint arXiv:2210.11399*, 2022.

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

[75] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[76] K. Singhal, S. Azizi, T. Tu, S. S. Mahdavi, J. Wei, H. W. Chung, N. Scales, A. Tanwani, H. Cole-Lewis, S. Pfohl *et al.*, "Large language models encode clinical knowledge," *arXiv preprint arXiv:2212.13138*, 2022.

[77] K. Singhal, T. Tu, J. Gottweis, R. Sayres, E. Wulczyn, L. Hou, K. Clark, S. Pfohl, H. Cole-Lewis, D. Neal *et al.*, "Towards expert-level medical question answering with large language models," *arXiv preprint arXiv:2305.09617*, 2023.

[78] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," *arXiv preprint arXiv:2109.01652*, 2021.

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

[80] V. Sanh, A. Webson, C. Raffel, S. H. Bach, L. Sutawika, Z. Alyafeai, A. Chaffin, A. Stiegler, T. L. Scao, A. Raja *et al.*, "Multitask prompted training enables zero-shot task generalization," *arXiv preprint arXiv:2110.08207*, 2021.

[81] Y. Sun, S. Wang, S. Feng, S. Ding, C. Pang, J. Shang, J. Liu, X. Chen, Y. Zhao, Y. Lu *et al.*, "Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation," *arXiv preprint arXiv:2107.02137*, 2021.

[82] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark *et al.*, "Improving language models by retrieving from trillions of tokens," in *International conference on machine learning*. PMLR, 2022, pp. 2206–2240.

[83] O. Lieber, O. Sharir, B. Lenz, and Y. Shoham, "Jurassic-1: Technical details and evaluation," *White Paper. AI21 Labs*, vol. 1, p. 9, 2021.

[84] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat *et al.*, "Glam: Efficient scaling of language models with mixture-of-experts," in *International Conference on Machine Learning*. PMLR, 2022, pp. 5547–5569.

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

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

[87] R. Taylor, M. Kardas, G. Cucurull, T. Scialom, A. Hartshorn, E. Saravia, A. Poulton, V. Kerkez, and R. Stojnic, "Galactica: A large language model for science," *arXiv preprint arXiv:2211.09085*, 2022.

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

[89] S. Soltan, S. Ananthakrishnan, J. FitzGerald, R. Gupta, W. Hamza, H. Khan, C. Peris, S. Rawls, A. Rosenbaum, A. Rumshisky *et al.*, "Alexatm 20b: Few-shot learning using a large-scale multilingual seq2seq model," *arXiv preprint arXiv:2208.01448*, 2022.

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

[91] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo *et al.*, "Solving quantitative reasoning problems with language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 3843–3857, 2022.

[92] Y. Tay, M. Dehghani, V. Q. Tran, X. Garcia, D. Bahri, T. Schuster, H. S. Zheng, N. Houlsby, and D. Metzler, "Unifying language learning paradigms," *arXiv preprint arXiv:2205.05131*, 2022.

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

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

[95] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff *et al.*, "Pythia: A suite for analyzing large language models across training and scaling," in *International Conference on Machine Learning*. PMLR, 2023, pp. 2397–2430.

[96] S. Mukherjee, A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and A. Awadallah, "Orca: Progressive learning from complex explanation traces of gpt-4," *arXiv preprint arXiv:2306.02707*, 2023.

[97] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim *et al.*, "Starcoder: may the source be with you!" *arXiv preprint arXiv:2305.06161*, 2023.

[98] S. Huang, L. Dong, W. Wang, Y. Hao, S. Singhal, S. Ma, T. Lv, L. Cui, O. K. Mohammed, Q. Liu *et al.*, "Language is not all you need: Aligning perception with language models," *arXiv preprint arXiv:2302.14045*, 2023.

[99] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[100] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar *et al.*, "Inner monologue: Embodied reasoning through planning with language models," *arXiv preprint arXiv:2207.05608*, 2022.

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

[102] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[103] S. Iyer, X. V. Lin, R. Pasunuru, T. Mihaylov, D. Simig, P. Yu, K. Shuster, T. Wang, Q. Liu, P. S. Koura *et al.*, "Opt-iml: Scaling language model instruction meta learning through the lens of generalization," *arXiv preprint arXiv:2212.12017*, 2022.

[104] Y. Hao, H. Song, L. Dong, S. Huang, Z. Chi, W. Wang, S. Ma, and F. Wei, "Language models are general-purpose interfaces," *arXiv preprint arXiv:2206.06336*, 2022.

[105] Z. Sun, Y. Shen, Q. Zhou, H. Zhang, Z. Chen, D. Cox, Y. Yang, and C. Gan, "Principle-driven self-alignment of language models from scratch with minimal human supervision," *arXiv preprint arXiv:2305.03047*, 2023.

[106] W. E. team, "Palmyra-base Parameter Autoregressive Language Model," https://dev.writer.com, 2023.

[107] ——, "Camel-5b instructgpt," https://dev.writer.com, 2023.

[108] Yandex. Yalm. [Online]. Available: https://github.com/yandex/YaLM-100B

[109] M. Team *et al.*, "Introducing mpt-7b: a new standard for open-source, commercially usable llms," 2023.

[110] A. Mitra, L. D. Corro, S. Mahajan, A. Codas, C. Simoes, S. Agarwal, X. Chen, A. Razdaibiedina, E. Jones, K. Aggarwal, H. Palangi, G. Zheng, C. Rosset, H. Khanpour, and A. Awadallah, "Orca 2: Teaching small language models how to reason," 2023.

[111] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "Pal: Program-aided language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 10 764–10 799.

[112] Anthropic. claude. [Online]. Available: https://www.anthropic.com/news/introducing-claude

[113] E. Nijkamp, H. Hayashi, C. Xiong, S. Savarese, and Y. Zhou, "Codegen2: Lessons for training llms on programming and natural languages," *arXiv preprint arXiv:2305.02309*, 2023.

[114] L. Tunstall, E. Beeching, N. Lambert, N. Rajani, K. Rasul, Y. Belkada, S. Huang, L. von Werra, C. Fourrier, N. Habib *et al.*, "Zephyr: Direct distillation of lm alignment," *arXiv preprint arXiv:2310.16944*, 2023.

[115] X. team. Grok. [Online]. Available: https://grok.x.ai/

[116] J. Bai, S. Bai, S. Yang, S. Wang, S. Tan, P. Wang, J. Lin, C. Zhou, and J. Zhou, "Qwen-vl: A frontier large vision-language model with versatile abilities," *arXiv preprint arXiv:2308.12966*, 2023.

[117] mixtral. mixtral. [Online]. Available: https://mistral.ai/news/mixtral-of-experts/

[118] D. Wang, N. Raman, M. Sibue, Z. Ma, P. Babkin, S. Kaur, Y. Pei, A. Nourbakhsh, and X. Liu, "Docllm: A layout-aware generative language model for multimodal document understanding," 2023.

[119] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang, "Deepseek-coder: When the large language model meets programming – the rise of code intelligence," 2024.

[120] F. Wan, X. Huang, D. Cai, X. Quan, W. Bi, and S. Shi, "Knowledge fusion of large language models," 2024.

[121] P. Zhang, G. Zeng, T. Wang, and W. Lu, "Tinyllama: An open-source small language model," 2024.

[122] C. Wu, Y. Gan, Y. Ge, Z. Lu, J. Wang, Y. Feng, P. Luo, and Y. Shan, "Llama pro: Progressive llama with block expansion," 2024.

[123] X. Amatriain, A. Sankar, J. Bing, P. K. Bodigutla, T. J. Hazen, and M. Kazi, "Transformer models: an introduction and catalog," 2023.

[124] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay, "The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only," *arXiv preprint arXiv:2306.01116*, 2023.

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

[126] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018.

[127] J. Su, Y. Lu, S. Pan, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *arXiv preprint arXiv:2104.09864*, 2021.

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

[129] G. Ke, D. He, and T.-Y. Liu, "Rethinking positional encoding in language pre-training," *arXiv preprint arXiv:2006.15595*, 2020.

[130] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[131] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 5232–5270, 2022.

[132] R. K. Mahabadi, S. Ruder, M. Dehghani, and J. Henderson, "Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks," 2021.

[133] S. Zhang, L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu, and G. Wang, "Instruction tuning for large language models: A survey," 2023.

[134] S. Mishra, D. Khashabi, C. Baral, and H. Hajishirzi, "Cross-task generalization via natural language crowdsourcing instructions," *arXiv preprint arXiv:2104.08773*, 2021.

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

[136] K. Ethayarajh, W. Xu, D. Jurafsky, and D. Kiela. Kto. [Online]. Available: https://github.com/ContextualAI/HALOs/blob/main/assets/report.pdf

[137] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *Advances in neural information processing systems*, vol. 30, 2017.

[138] H. Lee, S. Phatale, H. Mansoor, K. Lu, T. Mesnard, C. Bishop, V. Carbune, and A. Rastogi, "Rlaif: Scaling reinforcement learning from human feedback with ai feedback," *arXiv preprint arXiv:2309.00267*, 2023.

[139] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *arXiv preprint arXiv:2305.18290*, 2023.

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

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

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

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

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

[145] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, mar 2023. [Online]. Available: https://doi.org/10.1145/3571730

[146] N. McKenna, T. Li, L. Cheng, M. J. Hosseini, M. Johnson, and M. Steedman, "Sources of hallucination by large language models on inference tasks," 2023.

[147] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013

[148] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, P. Isabelle, E. Charniak, and D. Lin, Eds. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. [Online]. Available: https://aclanthology.org/P02-1040

[149] B. Dhingra, M. Faruqui, A. Parikh, M.-W. Chang, D. Das, and W. Cohen, "Handling divergent reference texts when evaluating table-to-text generation," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 4884–4895. [Online]. Available: https://aclanthology.org/P19-1483

[150] Z. Wang, X. Wang, B. An, D. Yu, and C. Chen, "Towards faithful neural table-to-text generation with content-matching constraints," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds. Online: Association for Computational Linguistics, Jul. 2020, pp. 1072–1086. [Online]. Available: https://aclanthology.org/2020.acl-main.101

[151] H. Song, W.-N. Zhang, J. Hu, and T. Liu, "Generating persona consistent dialogues by exploiting natural language inference," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, pp. 8878–8885, Apr. 2020.

[152] O. Honovich, L. Choshen, R. Aharoni, E. Neeman, I. Szpektor, and O. Abend, "q^2: Evaluating factual consistency in knowledge-grounded dialogues via question generation and question answering," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 7856–7870. [Online]. Available: https://aclanthology.org/2021.emnlp-main.619

[153] N. Dziri, H. Rashkin, T. Linzen, and D. Reitter, "Evaluating attribution in dialogue systems: The BEGIN benchmark," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 1066–1083, 2022. [Online]. Available: https://aclanthology.org/2022.tacl-1.62

[154] S. Santhanam, B. Hedayatnia, S. Gella, A. Padmakumar, S. Kim, Y. Liu, and D. Z. Hakkani-Tür, "Rome was built in 1776: A case study on factual correctness in knowledge-grounded response generation," *ArXiv*, vol. abs/2110.05456, 2021.

[155] S. Min, K. Krishna, X. Lyu, M. Lewis, W. tau Yih, P. W. Koh, M. Iyyer, L. Zettlemoyer, and H. Hajishirzi, "Factscore: Fine-grained atomic evaluation of factual precision in long form text generation," 2023.

[156] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, "Machine learning: The high interest credit card of technical debt," in *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.

[157] Z. Zhang, A. Zhang, M. Li, and A. Smola, "Automatic chain of thought prompting in large language models," 2022.

[158] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," 2023.

[159] P. Manakul, A. Liusie, and M. J. F. Gales, "Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models," 2023.

[160] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," 2023.

[161] S. J. Zhang, S. Florin, A. N. Lee, E. Niknafs, A. Marginean, A. Wang, K. Tyser, Z. Chin, Y. Hicke, N. Singh, M. Udell, Y. Kim, T. Buonassisi, A. Solar-Lezama, and I. Drori, "Exploring the mit mathematics and eecs curriculum using large language models," 2023.

[162] T. Wu, E. Jiang, A. Donsbach, J. Gray, A. Molina, M. Terry, and C. J. Cai, "Promptchainer: Chaining large language model prompts through visual programming," 2022.

[163] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large language models are human-level prompt engineers," 2023.

[164] P. S. H. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *CoRR*, vol. abs/2005.11401, 2020. [Online]. Available: https://arxiv.org/abs/2005.11401

[165] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[166] A. W. Services. (Year of publication, e.g., 2023) Question answering using retrieval augmented generation with foundation models in amazon sagemaker jumpstart. Accessed: Date of access, e.g., December 5, 2023. [Online]. Available: https://shorturl.at/dSV47

[167] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *arXiv preprint arXiv:2306.08302*, 2023.

[168] Z. Jiang, F. F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig, "Active retrieval augmented generation," 2023.

[169] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," 2023.

[170] B. Paranjape, S. Lundberg, S. Singh, H. Hajishirzi, L. Zettlemoyer, and M. T. Ribeiro, "Art: Automatic multi-step reasoning and tool-use for large language models," 2023.

[171] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface," *arXiv preprint arXiv:2303.17580*, 2023.

[172] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou *et al.*, "The rise and potential of large language model based agents: A survey," *arXiv preprint arXiv:2309.07864*, 2023.

[173] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, "A survey on large language model based autonomous agents," *arXiv preprint arXiv:2308.11432*, 2023.

[174] Z. Durante, Q. Huang, N. Wake, R. Gong, J. S. Park, B. Sarkar, R. Taori, Y. Noda, D. Terzopoulos, Y. Choi, K. Ikeuchi, H. Vo, L. Fei-Fei, and J. Gao, "Agent ai: Surveying the horizons of multimodal interaction," *arXiv preprint arXiv:2401.03568*, 2024.

[175] B. Xu, Z. Peng, B. Lei, S. Mukherjee, Y. Liu, and D. Xu, "Rewoo: Decoupling reasoning from observations for efficient augmented language models," 2023.

[176] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023.

[177] V. Nair, E. Schumacher, G. Tso, and A. Kannan, "Dera: Enhancing large language model completions with dialog-enabled resolving agents," 2023.

[178] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," 2023.

[179] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, K. Toutanova, L. Jones, M. Kelcey, M.-W. Chang, A. M. Dai, J. Uszkoreit, Q. Le, and S. Petrov, "Natural questions: A benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 452–466, 2019. [Online]. Available: https://aclanthology.org/Q19-1026

[180] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," 2021.

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

[182] E. Choi, H. He, M. Iyyer, M. Yatskar, W.-t. Yih, Y. Choi, P. Liang, and L. Zettlemoyer, "QuAC: Question answering in context," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2174–2184. [Online]. Available: https://aclanthology.org/D18-1241

[183] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, and J. Steinhardt, "Measuring coding challenge competence with apps," *NeurIPS*, 2021.

[184] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017.

[185] M. Joshi, E. Choi, D. Weld, and L. Zettlemoyer, "TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, R. Barzilay and M.-Y. Kan, Eds. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1601–1611. [Online]. Available: https://aclanthology.org/P17-1147

[186] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy, "RACE: Large-scale ReAding comprehension dataset from examinations," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, M. Palmer, R. Hwa, and S. Riedel, Eds. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 785–794. [Online]. Available: https://aclanthology.org/D17-1082

[187] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, J. Su, K. Duh, and X. Carreras, Eds. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2383–2392. [Online]. Available: https://aclanthology.org/D16-1264

[188] C. Clark, K. Lee, M. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "Boolq: Exploring the surprising difficulty of natural yes/no questions," *CoRR*, vol. abs/1905.10044, 2019. [Online]. Available: http://arxiv.org/abs/1905.10044

[189] D. Khashabi, S. Chaturvedi, M. Roth, S. Upadhyay, and D. Roth, "Looking beyond the surface:a challenge set for reading comprehension over multiple sentences," in *Proceedings of North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

[190] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," *CoRR*, vol. abs/2110.14168, 2021. [Online]. Available: https://arxiv.org/abs/2110.14168

[191] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the MATH dataset," *CoRR*, vol. abs/2103.03874, 2021. [Online]. Available: https://arxiv.org/abs/2103.03874

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

[193] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the AI2 reasoning challenge," *CoRR*, vol. abs/1803.05457, 2018. [Online]. Available: http://arxiv.org/abs/1803.05457

[194] Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi, "PIQA: reasoning about physical commonsense in natural language," *CoRR*, vol. abs/1911.11641, 2019. [Online]. Available: http://arxiv.org/abs/1911.11641

[195] M. Sap, H. Rashkin, D. Chen, R. L. Bras, and Y. Choi, "Socialiqa: Commonsense reasoning about social interactions," *CoRR*, vol. abs/1904.09728, 2019. [Online]. Available: http://arxiv.org/abs/1904.09728

[196] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? A new dataset for open book question answering," *CoRR*, vol. abs/1809.02789, 2018. [Online]. Available: http://arxiv.org/abs/1809.02789

[197] S. Lin, J. Hilton, and O. Evans, "Truthfulqa: Measuring how models mimic human falsehoods," *arXiv preprint arXiv:2109.07958*, 2021.

[198] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning, "Hotpotqa: A dataset for diverse, explainable multi-hop question answering," *CoRR*, vol. abs/1809.09600, 2018. [Online]. Available: http://arxiv.org/abs/1809.09600

[199] Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang, "Toolqa: A dataset for llm question answering with external tools," *arXiv preprint arXiv:2306.13304*, 2023.

[200] D. Chen, J. Bolton, and C. D. Manning, "A thorough examination of the cnn/daily mail reading comprehension task," in *Association for Computational Linguistics (ACL)*, 2016.

[201] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang *et al.*, "Abstractive text summarization using sequence-to-sequence rnns and beyond," *arXiv preprint arXiv:1602.06023*, 2016.

[202] Y. Bai and D. Z. Wang, "More than reading comprehension: A survey on datasets and metrics of textual question answering," *arXiv preprint arXiv:2109.12264*, 2021.

[203] H.-Y. Huang, E. Choi, and W.-t. Yih, "Flowqa: Grasping flow in history for conversational machine comprehension," *arXiv preprint arXiv:1810.06683*, 2018.

[204] S. Lee, J. Lee, H. Moon, C. Park, J. Seo, S. Eo, S. Koo, and H. Lim, "A survey on evaluation metrics for machine translation," *Mathematics*, vol. 11, no. 4, p. 1006, 2023.

[205] J. Li, X. Cheng, W. X. Zhao, J.-Y. Nie, and J.-R. Wen, "Halueval: A large-scale hallucination evaluation benchmark for large language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 6449–6464.

[206] Simon Mark Hughes, "Hughes hallucination evaluation model (hhem) leaderboard," 2024, https://huggingface.co/spaces/vectara/Hallucination-evaluation-leaderboard, Last accessed on 2024-01-21.

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

[208] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. Del Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi *et al.*, "Textbooks are all you need," *arXiv preprint arXiv:2306.11644*, 2023.

[209] Y. Li, S. Bubeck, R. Eldan, A. Del Giorno, S. Gunasekar, and Y. T. Lee, "Textbooks are all you need ii: phi-1.5 technical report," *arXiv preprint arXiv:2309.05463*, 2023.

[210] M. Poli, S. Massaroli, E. Nguyen, D. Y. Fu, T. Dao, S. Baccus, Y. Bengio, S. Ermon, and C. Ré, "Hyena hierarchy: Towards larger convolutional language models," 2023.

[211] M. Poli, J. Wang, S. Massaroli, J. Quesnelle, E. Nguyen, and A. Thomas, "StripedHyena: Moving Beyond Transformers with Hybrid Signal Processing Models," 12 2023. [Online]. Available: https://github.com/togethercomputer/stripedhyena

[212] D. Y. Fu, S. Arora, J. Grogan, I. Johnson, S. Eyuboglu, A. W. Thomas, B. Spector, M. Poli, A. Rudra, and C. Ré, "Monarch mixer: A simple sub-quadratic gemm-based architecture," 2023.

[213] G. J. McLachlan, S. X. Lee, and S. I. Rathnayake, "Finite mixture models," *Annual review of statistics and its application*, vol. 6, pp. 355–378, 2019.

[214] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *arXiv preprint arXiv:2304.08485*, 2023.

[215] S. Liu, H. Cheng, H. Liu, H. Zhang, F. Li, T. Ren, X. Zou, J. Yang, H. Su, J. Zhu, L. Zhang, J. Gao, and C. Li, "Llava-plus: Learning to use tools for creating multimodal agents," *arXiv preprint arXiv:2311.05437*, 2023.

[216] S. Wu, H. Fei, L. Qu, W. Ji, and T.-S. Chua, "Next-gpt: Any-to-any multimodal llm," *arXiv preprint arXiv:2309.05519*, 2023.

[217] N. N. Khasmakhi, M. Asgari-Chenaghlu, N. Asghar, P. Schaer, and D. Zühlke, "Convgenvismo: Evaluation of conversational generative vision models," 2023.

[218] N. Alshahwan, J. Chheda, A. Finegenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, "Automated unit test improvement using large language models at meta," *arXiv preprint arXiv:2402.09171*, 2024.

[219] L. Sun, Y. Huang, H. Wang, S. Wu, Q. Zhang, C. Gao, Y. Huang, W. Lyu, Y. Zhang, X. Li *et al.*, "Trustllm: Trustworthiness in large language models," *arXiv preprint arXiv:2401.05561*, 2024.

[220] Microsoft. Deepspeed. [Online]. Available: https://github.com/microsoft/DeepSpeed

[221] HuggingFace. Transformers. [Online]. Available: https://github.com/huggingface/transformers

[222] Nvidia. Megatron. [Online]. Available: https://github.com/NVIDIA/Megatron-LM

[223] BMTrain. Bmtrain. [Online]. Available: https://github.com/OpenBMB/BMTrain

[224] EleutherAI. gpt-neox. [Online]. Available: https://github.com/EleutherAI/gpt-neox

[225] microsoft. Lora. [Online]. Available: https://github.com/microsoft/LoRA

[226] ColossalAI. Colossalai. [Online]. Available: https://github.com/hpcaitech/ColossalAI

[227] FastChat. Fastchat. [Online]. Available: https://github.com/lm-sys/FastChat

[228] skypilot. skypilot. [Online]. Available: https://github.com/skypilot-org/skypilot

[229] vllm. vllm. [Online]. Available: https://github.com/vllm-project/vllm

[230] huggingface. text-generation-inference. [Online]. Available: https://github.com/huggingface/text-generation-inference

[231] langchain. langchain. [Online]. Available: https://github.com/langchain-ai/langchain

[232] bentoml. Openllm. [Online]. Available: https://github.com/bentoml/OpenLLM

[233] embedchain. embedchain. [Online]. Available: https://github.com/embedchain/embedchain

[234] microsoft. autogen. [Online]. Available: https://github.com/microsoft/autogen

[235] babyagi. babyagi. [Online]. Available: https://github.com/yoheinakajima/babyagi

[236] guidance. guidance. [Online]. Available: https://github.com/guidance-ai/guidance

[237] prompttools. prompttools. [Online]. Available: https://github.com/hegelai/prompttools

[238] promptfoo. promptfoo. [Online]. Available: https://github.com/promptfoo/promptfoo

[239] facebook. faiss. [Online]. Available: https://github.com/facebookresearch/faiss

[240] milvus. milvus. [Online]. Available: https://github.com/milvus-io/milvus

[241] qdrant. qdrant. [Online]. Available: https://github.com/qdrant/qdrant

[242] weaviate. weaviate. [Online]. Available: https://github.com/weaviate/weaviate

[243] llama index. llama-index. [Online]. Available: https://github.com/run-llama/llama_index

APPENDIX

1. Open Source Toolkits For LLM Development and Deployment

There are various frameworks and libraries developed for LLM training, evaluation, and deployment, and covering every single framework is out of this paper's scope. But we try to provide a brief introduction of some of the most popular ones, grouped into different categories.

A. LLM Training/Inference Frameworks

Some of the popular frameworks which are useful for LLM training includes (note that some of them can be used beyond LLM training too):

DeepSpeed [220] is a deep learning optimization library that makes distributed training and inference easy, efficient, and effective. DeepSpeed enables world's most powerful language models like MT-530B and BLOOM. It is an easy-to-use deep learning optimization software suite that powers unprecedented scale and speed for both training and inference. With DeepSpeed you can:

Transformers [221] is library by HuggingFace which provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio. Using pretrained models one can reduce compute costs, carbon footprint, and save the time and resources required to train a model from scratch.

Megatron-LM [222] is a large, powerful transformer developed by the Applied Deep Learning Research team at NVIDIA. It contains efficient, model-parallel (tensor, sequence, and pipeline), and multi-node pre-training of transformer based models such as GPT, BERT, and T5 using mixed precision.

BMTrain [223] is an efficient large model training toolkit that can be used to train large models with tens of billions of parameters. It can train models in a distributed manner while keeping the code as simple as stand-alone training.

GPT-NeoX [224] leverages many of the same features and technologies as the popular Megatron-DeepSpeed library but with substantially increased usability and novel optimizations.

LoRA [225] library provides the support for Low-Rank Adaptation of Large Language Models. It reduces the number of trainable parameters by learning pairs of rank-decompostion matrices while freezing the original weights. This vastly

reduces the storage requirement for large language models adapted to specific tasks and enables efficient task-switching during deployment all without introducing inference latency. LoRA also outperforms several other adaptation methods including adapter, prefix-tuning, and fine-tuning.

ColossalAI library [226] provides a collection of parallel components. It aims to support developers to write their distributed deep learning models just like how they write their model on their laptop. They provide user-friendly tools to kickstart distributed training and inference in a few lines. In terms of Parallelism strategies, they support: Data Parallelism, Pipeline Parallelism, Sequence Parallelism, Zero Redundancy Optimizer (ZeRO) [140], and Auto-Parallelism.

B. Deployment Tools

We provide an overview of some of the most popular LLM deployment tools here.

FastChat [227] is an open platform for training, serving, and evaluating large language model based chatbots. FastChat's core features include: The training and evaluation code for state-of-the-art models (e.g., Vicuna, MT-Bench), and a distributed multi-model serving system with web UI and OpenAI-compatible RESTful APIs.

Skypilot [228] is a framework for running LLMs, AI, and batch jobs on any cloud, offering maximum cost savings, highest GPU availability, and managed execution.

vLLM [229] is a fast and easy-to-use library for LLM inference and serving. vLLM seamlessly supports many Hugging Face models, including the following architectures: Aquila, Baichuan, BLOOM, ChatGLM, DeciLM, Falcon, GPT BigCode, LLaMA, LLaMA 2, Mistral, Mixtral, MPT, OPT, Qwen, Yi, and many more.

text-generation-inference [230] is a toolkit for deploying and serving Large Language Models (LLMs). TGI enables high-performance text generation for the most popular open-source LLMs, including Llama, Falcon, StarCoder, BLOOM, GPT-NeoX, and more.

LangChain [231] is a framework for developing applications powered by language models. It enables applications that:

- Are context-aware: connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)
- Reason: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)

OpenLLM [232] is an open-source platform designed to facilitate the deployment and operation of large language models (LLMs) in real-world applications. With OpenLLM, you can run inference on any open-source LLM, deploy them on the cloud or on-premises, and build powerful AI applications.

Embedchain [233] is an Open Source RAG Framework that makes it easy to create and deploy AI apps. Embedchain streamlines the creation of RAG applications, offering a seamless process for managing various types of unstructured data. It efficiently segments data into manageable chunks, generates relevant embeddings, and stores them in a vector database for optimized retrieval.

Autogen [234] is a framework that enables the development of LLM applications using multiple agents that can converse with each other to solve tasks. AutoGen agents are customizable, conversable, and seamlessly allow human participation. They can operate in various modes that employ combinations of LLMs, human inputs, and tools.

BabyAGI [235] is an autonomous Artificial Intelligence agent, that is designed to generate and execute tasks based on given objectives. It harnesses cutting-edge technologies from OpenAI, Pinecone, LangChain, and Chroma to automate tasks and achieve specific goals. In this blog post, we will dive into the unique features of BabyAGI and explore how it can streamline task automation.

C. Prompting Libraries

Guidance [236] is a programming paradigm that offers superior control and efficiency compared to conventional prompting and chaining. It allows users to constrain generation (e.g. with regex and CFGs) as well as to interleave control (conditional, loops) and generation seamlessly.

PromptTools [237] offers a set of open-source, self-hostable tools for experimenting with, testing, and evaluating LLMs, vector databases, and prompts. The core idea is to enable developers to evaluate using familiar interfaces like code, notebooks, and a local playground.

PromptBench [**?**] is a Pytorch-based Python package for Evaluation of Large Language Models (LLMs). It provides user-friendly APIs for researchers to conduct evaluation on LLMs.

Promptfoo [238] is a tool for testing and evaluating LLM output quality. It systematically test prompts, models, and RAGs with predefined test cases.

D. VectorDB

Faiss [239] is a library developed by Facebook AI Research that provides efficient similarity search and clustering of dense vectors. It is designed for use with large-scale, high-dimensional data and supports several index types and algorithms for various use cases.

Milvus [240] is an open-source vector database built to power embedding similarity search and AI applications. Milvus makes unstructured data search more accessible, and provides a consistent user experience regardless of the deployment environment.

Qdrant [241] is a vector similarity search engine and vector database. It provides a production-ready service with a convenient API to store, search, and manage points—vectors with an additional payload Qdrant is tailored to extended filtering support. environment.

Weaviate [242] is an open-source, GraphQL-based vector search engine that enables similarity search on high-dimensional data. While it is open-source, the commercial version offers additional features, support, and managed services.

Some of the other popular options includes **LlamaIndex** [243] and **Pinecone**.