A Comprehensive Overview of Large Language Models

Humza Naveed[a], Asad Ullah Khan[a,*], Shi Qiu[b,*], Muhammad Saqib[c,d,*], Saeed Anwar[e,f], Muhammad Usman[e,f], Naveed Akhtar[g,i], Nick Barnes[h], Ajmal Mian[i]

[a]*University of Engineering and Technology (UET), Lahore, Pakistan*
[b]*The Chinese University of Hong Kong (CUHK), HKSAR, China*
[c]*University of Technology Sydney (UTS), Sydney, Australia*
[d]*Commonwealth Scientific and Industrial Research Organisation (CSIRO), Sydney, Australia*
[e]*King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia*
[f]*SDAIA-KFUPM Joint Research Center for Artificial Intelligence (JRCAI), Dhahran, Saudi Arabia*
[g]*The University of Melbourne (UoM), Melbourne, Australia*
[h]*Australian National University (ANU), Canberra, Australia*
[i]*The University of Western Australia (UWA), Perth, Australia*

Abstract

Large Language Models (LLMs) have recently demonstrated remarkable capabilities in natural language processing tasks and beyond. This success of LLMs has led to a large influx of research contributions in this direction. These works encompass diverse topics such as architectural innovations, better training strategies, context length improvements, fine-tuning, multi-modal LLMs, robotics, datasets, benchmarking, efficiency, and more. With the rapid development of techniques and regular breakthroughs in LLM research, it has become considerably challenging to perceive the bigger picture of the advances in this direction. Considering the rapidly emerging plethora of literature on LLMs, it is imperative that the research community is able to benefit from a concise yet comprehensive overview of the recent developments in this field. This article provides an overview of the existing literature on a broad range of LLM-related concepts. Our self-contained comprehensive overview of LLMs discusses relevant background concepts along with covering the advanced topics at the frontier of research in LLMs. This review article is intended to not only provide a systematic survey but also a quick comprehensive reference for the researchers and practitioners to draw insights from extensive informative summaries of the existing works to advance the LLM research.

Keywords:
Large Language Models, LLMs, chatGPT, Augmented LLMs, Multimodal LLMs, LLM training, LLM Benchmarking

1. Introduction

Language plays a fundamental role in facilitating communication and self-expression for humans, and their interaction with machines. The need for generalized models stems from the growing demand for machines to handle complex language tasks, including translation, summarization, information retrieval, conversational interactions, etc. Recently, significant breakthroughs have been witnessed in language models, primarily attributed to transformers [1], increased computational capabilities, and the availability of large-scale training data. These developments have brought about a revolutionary transformation by enabling the creation of LLMs that can approximate human-level performance on various tasks [2, 3]. Large

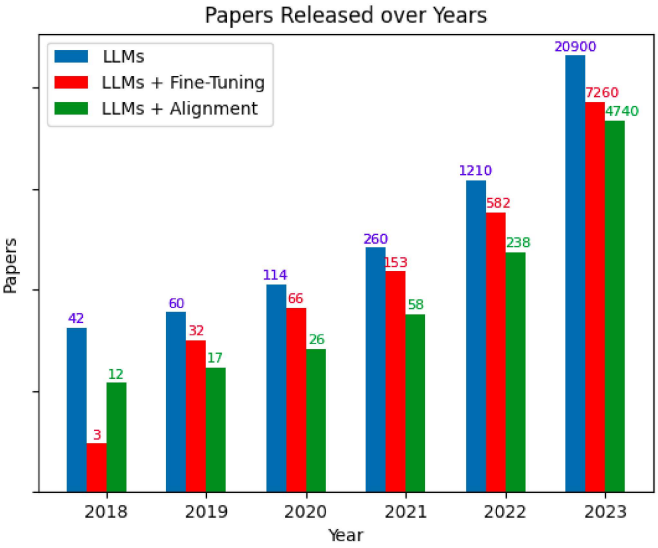

Figure 1: The trend of papers released over years containing keywords "Large Language Model", "Large Language Model + Fine-Tuning", and "Large Language Model + Alignment".

*Equal contribution
Email addresses: humza_naveed@yahoo.com (Humza Naveed), aukhanee@gmail.com (Asad Ullah Khan), shiqiu@cse.cuhk.edu.hk (Shi Qiu), muhammad.saqib@data61.csiro.au (Muhammad Saqib), saeed.anwar@kfupm.edu.sa (Saeed Anwar), muhammad.usman@kfupm.edu.sa (Muhammad Usman), naveed.akhtar1@unimelb.edu.au (Naveed Akhtar), nick.barnes@anu.edu.au (Nick Barnes), ajmal.mian@uwa.edu.au (Ajmal Mian)

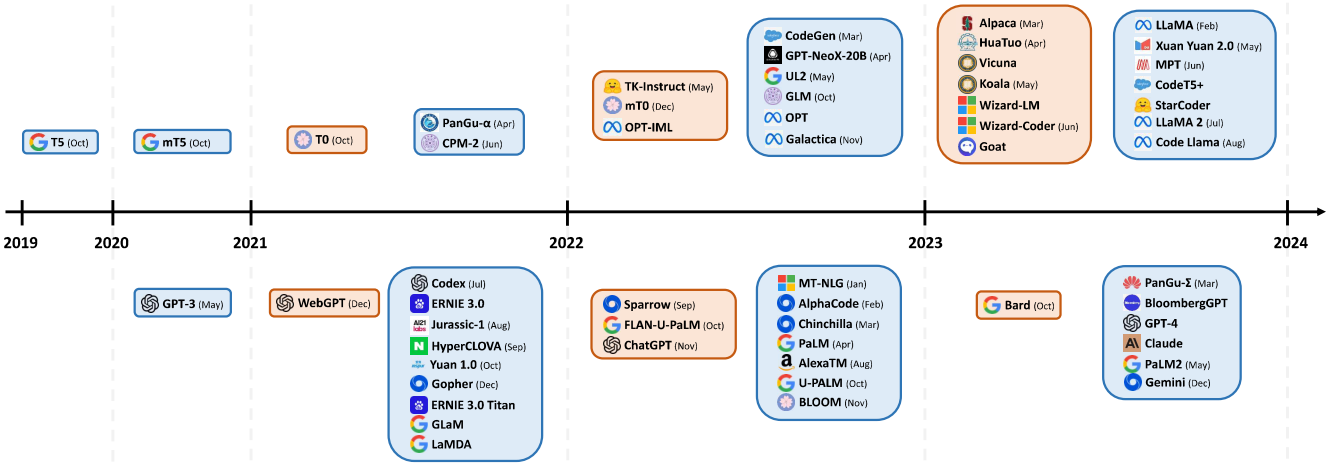

Figure 2: Chronological display of LLM releases: blue cards represent 'pre-trained' models, while orange cards correspond to 'instruction-tuned' models. Models on the upper half signify open-source availability, whereas those on the bottom half are closed-source. The chart illustrates the increasing trend towards instruction-tuned models and open-source models, highlighting the evolving landscape and trends in natural language processing research.

Language Models (LLMs) have emerged as cutting-edge artificial intelligence systems that can process and generate text with coherent communication [4], and generalize to multiple tasks [5, 6].

The historical progress in natural language processing (NLP) evolved from statistical to neural language modeling and then from pre-trained language models (PLMs) to LLMs. While conventional language modeling (LM) trains task-specific models in supervised settings, PLMs are trained in a self-supervised setting on a large corpus of text [7, 8, 9] with the aim of learning a generic representation that is shareable among various NLP tasks. After fine-tuning for downstream tasks, PLMs surpass the performance gains of traditional language modeling (LM). The larger PLMs bring more performance gains, which has led to the transitioning of PLMs to LLMs by significantly increasing model parameters (tens to hundreds of billions) [10] and training dataset (many GBs and TBs) [10, 11]. Following this development, numerous LLMs have been proposed in the literature [10, 11, 12, 6, 13, 14, 15]. An increasing trend in the number of released LLMs and names of a few significant LLMs proposed over the years are shown in Fig 1 and Fig 2, respectively.

The early work on LLMs, such as T5 [10] and mT5 [11] employed transfer learning until GPT-3 [6] showed LLMs are zero-shot transferable to downstream tasks without fine-tuning. LLMs accurately respond to task queries when prompted with task descriptions and examples. However, pre-trained LLMs fail to follow user intent and perform worse in zero-shot settings than in few-shot. Fine-tuning them with task instructions data [16, 17, 18, 19] and aligning with human preferences [20, 21] enhances generalization to unseen tasks, improving zero-shot performance significantly and reducing misaligned behavior.

In addition to better generalization and domain adaptation, LLMs appear to have emergent abilities, such as reasoning, planning, decision-making, in-context learning, answering in zero-shot settings, etc. These abilities are known to be acquired by them due to their gigantic scale even when the pre-trained LLMs are not trained specifically to possess these attributes [22, 23, 24]. Such abilities have led LLMs to be widely adopted in diverse settings including, multi-modal, robotics, tool manipulation, question answering, autonomous agents, etc. Various improvements have also been suggested in these areas either by task-specific training [25, 26, 27, 28, 29, 30, 31] or better prompting [32].

The LLMs abilities to solve diverse tasks with human-level performance come at a cost of slow training and inference, extensive hardware requirements, and higher running costs. Such requirements have limited their adoption and opened up opportunities to devise better architectures [15, 33, 34, 35] and training strategies [36, 37, 21, 38, 39, 40, 41]. Parameter efficient tuning [38, 41, 40], pruning [42, 43], quantization [44, 45], knowledge distillation, and context length interpolation [46, 47, 48, 49] among others are some of the methods widely studied for efficient LLM utilization.

Due to the success of LLMs on a wide variety of tasks, the research literature has recently experienced a large influx of LLM-related contributions. Researchers have organized the LLMs literature in surveys [50, 51, 52, 53], and topic-specific surveys in [54, 55, 56, 57, 58]. In contrast to these surveys, our contribution focuses on providing a comprehensive yet concise overview of the general direction of LLM research. This article summarizes architectural and training details of pre-trained LLMs and delves deeper into the details of concepts like fine-tuning, multi-modal LLMs, augmented LLMs, datasets, evaluation, applications, challenges, and others to provide a self-contained comprehensive overview. Our key contributions are summarized as follows.

- We present a survey on the developments in LLM research providing a concise comprehensive overview of the direction.
- We present extensive summaries of pre-trained models that include fine-grained details of architecture and training details.
- We summarize major findings of the popular contributions and provide a detailed discussion on the key design and development aspects of LLMs to help practitioners effectively leverage this technology.
- In this self-contained article, we cover a range of concepts to present the general direction of LLMs compre-

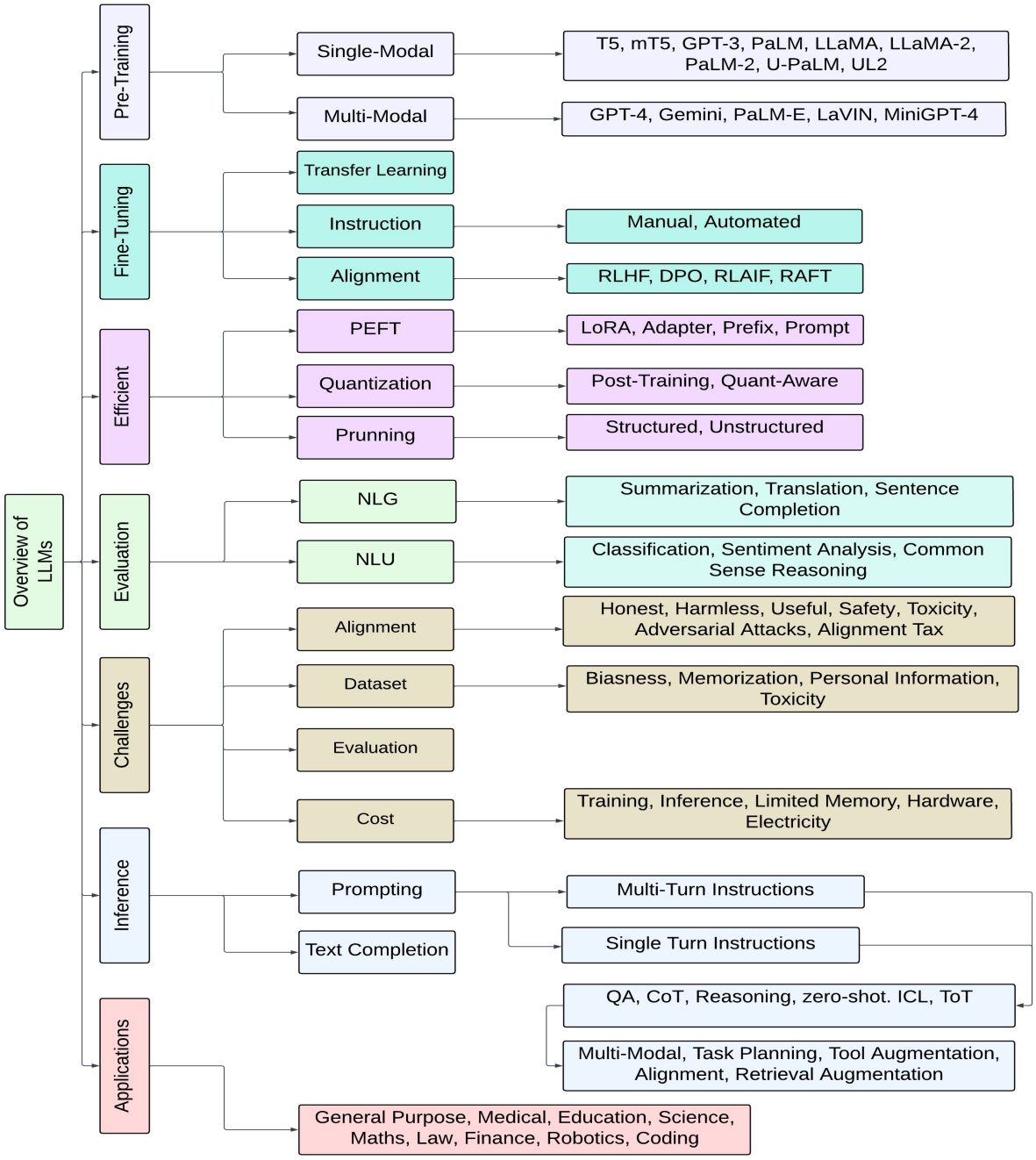

Figure 3: A broader overview of LLMs, dividing LLMs into seven branches: 1. Pre-Training 2. Fine-Tuning 3. Efficient 4. Inference 5. Evaluation 6. Applications 7. Challenges

hensively, including background, pre-training, fine-tuning, multi-modal LLMs, augmented LLMs, LLMs-powered agents, datasets, evaluation, etc.

We loosely follow the existing terminology to ensure a standardized outlook of this research direction. For instance, following [50], our survey discusses pre-trained LLMs with 10B parameters or more. We refer the readers interested in smaller pre-trained models to [51, 52, 53].

The organization of this paper is as follows. Section 2 discusses the background of LLMs. Section 3 focuses on LLMs overview, architectures, training pipelines and strategies, fine-tuning, and utilization in different domains. Section 4 highlights the configuration and parameters that play a crucial role in the functioning of these models. Summary and discussions are presented in section 3.8. The LLM training and evaluation, datasets, and benchmarks are discussed in section 5, followed by challenges and future directions, and conclusion in sections 7 and 8, respectively.

2. Background

We provide the relevant background to understand the fundamentals related to LLMs in this section. We briefly discuss necessary components in LLMs and refer the readers interested in details to the original works.

2.1. Tokenization

Tokenization [59] is an essential pre-processing step in LLM training that parses the text into non-decomposing units called tokens. Tokens can be characters, subwords [60], symbols [61], or words, depending on the tokenization process. Some of the commonly used tokenization schemes in LLMs include wordpiece [62], byte pair encoding (BPE) [61], and unigramLM [60]. Readers are encouraged to refer to [63] for a detailed survey.

2.2. Encoding Positions

The transformer processes input sequences in parallel and independently of each other. Moreover, the attention module in the transformer does not capture positional information. As a result, positional encodings were introduced in transformer [64], where a positional embedding vector is added to the token embedding. Variants of positional embedding include absolute, relative, or learned positional encodings. Within relative encoding, Alibi and RoPE are two widely used positional embeddings in LLMs.

Alibi [65]: It subtracts a scalar bias from the attention score that increases with the distance between token positions. This favors using recent tokens for attention.

RoPE [66]: It rotates query and key representations at an angle proportional to the token absolute position in the input sequence, resulting in a relative positional encoding scheme which decays with the distance between the tokens.

2.3. Attention in LLMs

Attention assigns weights to input tokens based on importance so that the model gives more emphasis to relevant tokens. Attention in transformers [64] calculates query, key, and value mappings for input sequences, where the attention score is obtained by multiplying the query and key, and later used to weight values. We discuss different attention strategies used in LLMs below.

Self-Attention [64]: Calculates attention using queries, keys, and values from the same block (encoder or decoder).

Cross Attention: It is used in encoder-decoder architectures, where encoder outputs are the queries, and key-value pairs come from the decoder.

Sparse Attention [67]: Self-attention has $O(n^2)$ time complexity which becomes infeasible for large sequences. To speed up the computation, sparse attention [67] iteratively calculates attention in sliding windows for speed gains.

Flash Attention [68]: Memory access is the major bottleneck in calculating attention using GPUs. To speed up, flash attention employs input tiling to minimize the memory reads and writes between the GPU high bandwidth memory (HBM) and the on-chip SRAM.

2.4. Activation Functions

The activation functions serve a crucial role in the curve-fitting abilities of neural networks [69]. We discuss activation functions used in LLMs in this section.

ReLU [70]: The Rectified linear unit (ReLU) is defined as:

$$ReLU(x) = max(0, x) \tag{1}$$

GeLU [71]: The Gaussian Error Linear Unit (GeLU) is the combination of ReLU, dropout [72] and zoneout [73].

GLU variants [74]: The Gated Linear Unit [75] is a neural network layer that is an element-wise product ($\otimes$) of a linear transformation and a sigmoid transformed (σ) linear projection of the input given as:

$$GLU(x, W, V, b, c) = (xW + b) \otimes \sigma(xV + c), \tag{2}$$

where X is the input of layer and l, W, b, V and c are learned parameters. Other GLU variants [74] used in LLMs are:

$$\begin{aligned} ReGLU(x, W, V, b, c) &= max(0, xW + b)\otimes, \\ GEGLU(x, W, V, b, c) &= GELU(xW + b) \otimes (xV + c), \\ SwiGLU(x, W, V, b, c, \beta) &= Swish\beta(xW + b) \otimes (xV + c). \end{aligned}$$

2.5. Layer Normalization

Layer normalization leads to faster convergence and is an integrated component of transformers [64]. In addition to LayerNorm [76] and RMSNorm [77], LLMs use pre-layer normalization [78], applying it before multi-head attention (MHA). Pre-norm is shown to provide training stability in LLMs. Another normalization variant, DeepNorm [79] fixes the issue with larger gradients in pre-norm.

2.6. Distributed LLM Training

This section describes distributed LLM training approaches briefly. More details are available in [13, 37, 80, 81].

Data Parallelism: Data parallelism replicates the model on multiple devices where data in a batch gets divided across devices. At the end of each training iteration weights are synchronized across all devices.

Tensor Parallelism: Tensor parallelism shards a tensor computation across devices. It is also known as horizontal parallelism or intra-layer model parallelism.

Pipeline Parallelism: Pipeline parallelism shards model layers across different devices. This is also known as vertical parallelism.

Model Parallelism: A combination of tensor and pipeline parallelism is known as model parallelism.

3D Parallelism: A combination of data, tensor, and model parallelism is known as 3D parallelism.

Optimizer Parallelism: Optimizer parallelism also known as zero redundancy optimizer [37] implements optimizer state partitioning, gradient partitioning, and parameter partitioning across devices to reduce memory consumption while keeping the communication costs as low as possible.

2.7. Libraries

Some commonly used libraries for LLMs training are:
Transformers [82]: The library provides access to various pre-trained transformer models with APIs to train, fine-tune, infer, and develop custom models.
DeepSpeed [36]: A library for scalable distributed training and inference of deep learning models.
Megatron-LM [80]: It provides GPU-optimized techniques for large-scale training of LLMs.
JAX [83]: A Python library for high-performance numerical computing and scaleable machine learning. It can differentiate native Python and NumPy functions and execute them on GPUs.
Colossal-AI [84]: A collection of components to write distributed deep learning models.
BMTrain [81]: A library to write efficient stand-alone LLMs training code.
FastMoE [85]: Provides API to build mixture-of-experts (MoE) model in PyTorch.
MindSpore [86]: A deep learning training and inference framework extendable to mobile, edge, and cloud computing.
PyTorch [87]: A framework developed by Facebook AI Research lab (FAIR) to build deep learning models. The main features of PyTorch include a dynamic computation graph and a pythonic coding style.
Tensorflow [88]: A deep learning framework written by Google. The key features of TensorFlow are graph-based computation, eager execution, scalability, etc.
MXNet [89]: Apache MXNet is a deep learning framework with support to write programs in multiple languages, including, Python, C++, Scala, R, etc. It also provides support for dynamic and static computation graphs.

2.8. Data PreProcessing

This section briefly summarizes data preprocessing techniques used in LLMs training.
Quality Filtering: For better results, training data quality is essential. Some approaches to filtering data are: 1) classifier-based and 2) heuristics-based. Classifier-based approaches train a classifier on high-quality data and predict the quality of text for filtering, whereas heuristics-based employ some rules for filtering like language, metrics, statistics, and keywords.
Data Deduplication: Duplicated data can affect model performance and increase data memorization; therefore, to train LLMs, data deduplication is one of the preprocessing steps. This can be performed at multiple levels, like sentences, documents, and datasets.
Privacy Reduction: Most of the training data for LLMs is collected through web sources. This data contains private information; therefore, many LLMs employ heuristics-based methods to filter information such as names, addresses, and phone numbers to avoid learning personal information.

2.9. Architectures

Here we discuss the variants of the transformer architectures used in LLMs. The difference arises due to the application of the attention and the connection of transformer blocks. An illustration of attention patterns of these architectures is shown in Figure 4.

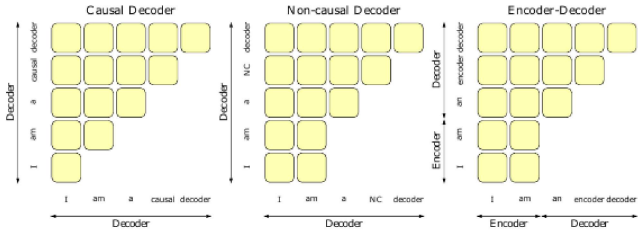

Figure 4: An example of attention patterns in language models, image is taken from [93].

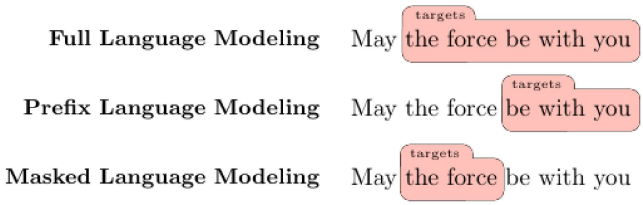

Figure 5: An example of language model training objectives, image from [93].

Encoder Decoder: This architecture processes inputs through the encoder and passes the intermediate representation to the decoder to generate the output. Here, the encoder sees the complete sequence utilizing self-attention whereas the decoder processes the sequence one after the other with implementing cross-attention.
Causal Decoder: A type of architecture that does not have an encoder and processes and generates output using a decoder, where the predicted token depends only on the previous time steps.
Prefix Decoder: It is also known as a non-causal decoder, where the attention calculation is not strictly dependent on the past information and the attention is bidirectional. An example of a non-causal attention mask is shown in Figure 4.
Mixture-of-Experts: It is a variant of transformer architecture with parallel independent experts and a router to route tokens to experts. These experts are feed-forward layers after the attention block [90]. Mixture-of-Experts (MoE) is an efficient sparse architecture that offers comparable performance to dense models and allows increasing the model size without increasing the computational cost by activating only a few experts at a time [91, 92].

2.10. Pre-Training Objectives

This section describes LLMs pre-training objectives. For more details see the paper [93].
Full Language Modeling: An autoregressive language modeling objective where the model is asked to predict future tokens given the previous tokens, an example is shown in Figure 5.
Prefix Language Modeling: A non-causal training objective, where a prefix is chosen randomly and only remaining target tokens are used to calculate the loss. An example is shown in Figure 5.

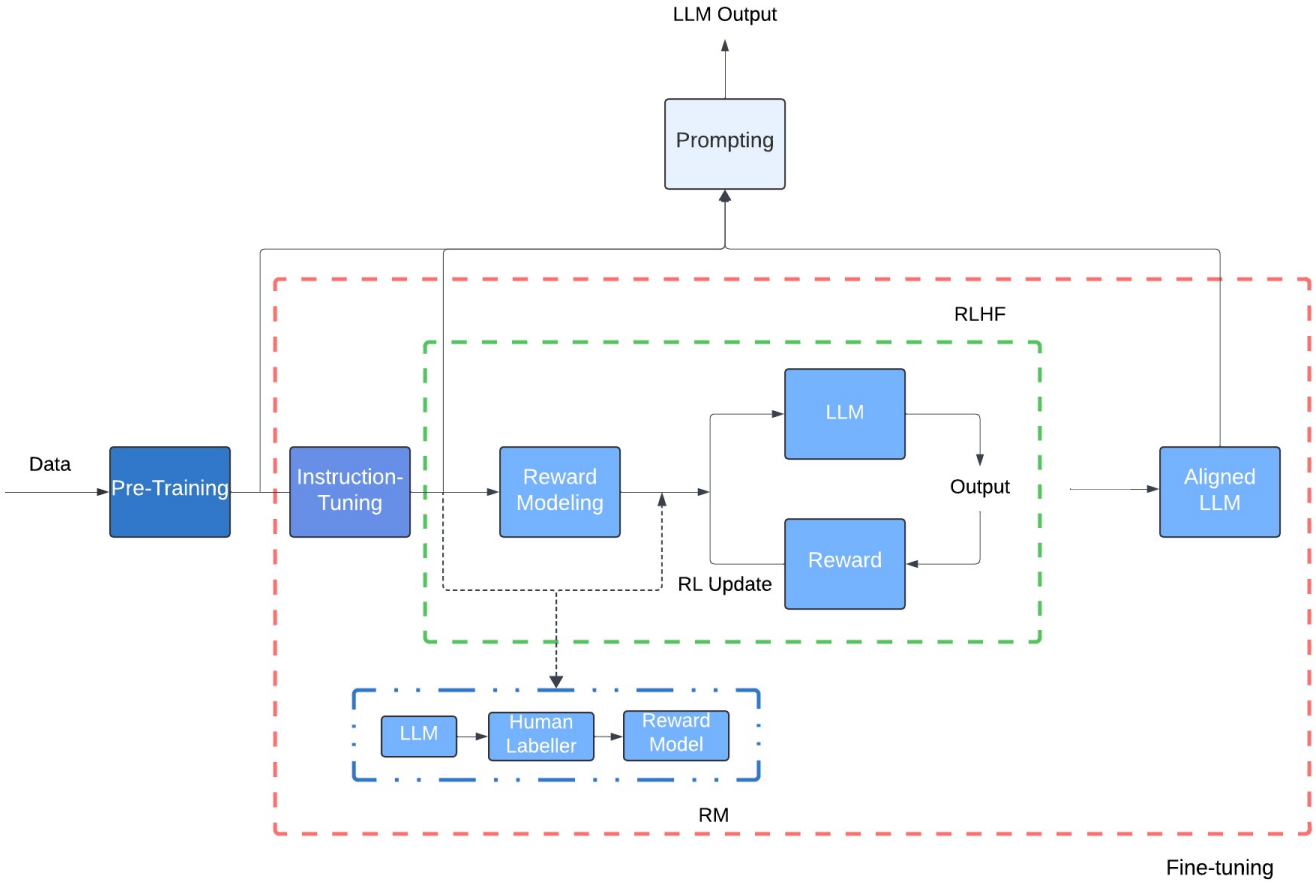

Figure 6: A basic flow diagram depicting various stages of LLMs from pre-training to prompting/utilization. Prompting LLMs to generate responses is possible at different training stages like pre-training, instruction-tuning, or alignment tuning. "RL" stands for reinforcement learning, "RM" represents reward-modeling, and "RLHF" represents reinforcement learning with human feedback.

Masked Language Modeling: In this training objective, tokens or spans (a sequence of tokens) are masked randomly and the model is asked to predict masked tokens given the past and future context. An example is shown in Figure 5.

Unified Language Modeling: Unified language modeling [94] is a combination of causal, non-causal, and masked language training objectives. Here in masked language modeling, the attention is not bidirectional but unidirectional, attending either left-to-right or right-to-left context.

2.11. LLMs Scaling Laws

Scaling laws study the optimal combination of model parameters, dataset size, and computational resources that predict the improvement in the model performance. It has been shown that the loss scales according to the power-law with model size, dataset size, and compute resources [95]. This study suggests larger models are more important than big data for better performance. Another variant of scaling law [96] suggests the model size and the number of training tokens should be scaled equally.

2.12. LLMs Adaptation Stages

This section discusses the fundamentals of LLMs adaptation stages, from pre-training to fine-tuning for downstream tasks and utilization. An example of different training stages and inference in LLMs is shown in Figure 6. In this paper, we refer to alignment-tuning as aligning with human preferences, while occasionally the literature uses the term alignment for different purposes.

2.12.1. Pre-Training

In the very first stage, the model is trained in a self-supervised manner on a large corpus to predict the next tokens given the input. The design choices of LLMs vary from encoder-decoder to decoder-only architectures with different building blocks and loss functions in sections 2.5, 2.4, 2.10.

2.12.2. Fine-Tuning

There are different styles to fine-tune an LLM. This section briefly discusses fine-tuning approaches.

Transfer Learning: The pre-trained LLMs perform well for various tasks [6, 15]. However, to improve the performance for

a downstream task, pre-trained models are fine-tuned with the task-specific data [10, 11], known as transfer learning.

Instruction-tuning: To enable a model to respond to user queries effectively, the pre-trained model is fine-tuned on instruction formatted data i.e., instruction and an input-output pair. Instructions generally comprise multi-task data in plain natural language, guiding the model to respond according to the prompt and the input. This type of fine-tuning improves zero-shot generalization and downstream task performance. Details on formatting instruction data and its various styles are available in [16, 50, 97].

Alignment-tuning: LLMs are prone to generating false, biased, and harmful text. To make them helpful, honest, and harmless, models are aligned using human feedback. Alignment involves asking LLMs to generate unexpected responses and then updating their parameters to avoid such responses [20, 21, 98].

It ensures LLMs operate according to human intentions and values. A model is defined to be an "aligned" model if the model fulfills three criteria of helpful, honest, and harmless or "HHH" [99].

Researchers employ reinforcement learning with human feedback (RLHF) [100] for model alignment. In RLHF, a fine-tuned model on demonstrations is further trained with reward modeling (RM) and reinforcement learning (RL), shown in Figure 6. Below we briefly discuss RM and RL pipelines in RLHF.

Reward modeling: trains a model to rank generated responses according to human preferences using a classification objective. To train the classifier humans annotate LLMs generated responses based on the HHH criteria.

Reinforcement learning: in combination with the reward model is used for alignment in the next stage. The previously trained reward model ranks LLM-generated responses into preferred vs. non-preferred, which is used to align the model with proximal policy optimization (PPO). This process repeats iteratively until convergence.

2.12.3. Prompting/Utilization

Prompting is a method to query trained LLMs for generating responses, as illustrated in Figure 6. LLMs can be prompted in various prompt setups, where they can be adapted to the instructions without fine-tuning and in other cases with fine-tuning on data containing different prompt styles [16, 101, 102]. A good guide on prompt engineering is available at [32]. Below, we will discuss various widely used prompt setups.

Zero-Shot Prompting: LLMs are zero-shot learners and capable of answering queries never seen before. This style of prompting requires LLMs to answer user questions without seeing any examples in the prompt.

In-context Learning: Also known as few-shot learning, here, multiple input-output demonstration pairs are shown to the model to generate the desired response. This adaptation style is also called few-shot learning. A discussion on formatting in-context learning (ICL) templates is available in [54, 50, 18, 16].

Reasoning in LLMs: LLMs are zero-shot reasoners and can be provoked to generate answers to logical problems, task planning, critical thinking, etc. with reasoning. Generating reasons is possible only by using different prompting styles, whereas to improve LLMs further on reasoning tasks many methods [16, 97] train them on reasoning datasets. We discuss various prompting techniques for reasoning below.

Chain-of-Thought (CoT): A special case of prompting where demonstrations contain reasoning information aggregated with inputs and outputs so that the model generates outcomes with step-by-step reasoning. More details on CoT prompts are available in [55, 103, 101].

Self-Consistency: Improves CoT performance by generating multiple responses and selecting the most frequent answer [104].

Tree-of-Thought (ToT): Explores multiple reasoning paths with possibilities to look ahead and backtrack for problem-solving [105].

Single-Turn Instructions: In this prompting setup, LLMs are queried only once with all the relevant information in the prompt. LLMs generate responses by understanding the context either in a zero-shot or few-shot setting.

Multi-Turn Instructions: Solving a complex task requires multiple interactions with LLMs, where feedback and responses from the other tools are given as input to the LLM for the next rounds. This style of using LLMs in the loop is common in autonomous agents.

3. Large Language Models

This section reviews LLMs, briefly describing their architectures, training objectives, pipelines, datasets, and fine-tuning details.

3.1. Pre-Trained LLMs

Here, we provide summaries of various well-known pre-trained LLMs with significant discoveries, changing the course of research and development in NLP. These LLMs have considerably improved the performance in NLU and NLG domains, and are widely fine-tuned for downstream tasks. Moreover, We also identify key findings and insights of pre-trained LLMs in Table 1 and 2 that improve their performance.

3.1.1. General Purpose

T5 [10]: An encoder-decoder model employing a unified text-to-text training for all NLP problems is shown in Figure 7. T5 places layer normalization outside the residual path in a conventional transformer model [64]. It uses masked language modeling as a pre-training objective where spans (consecutive tokens) are replaced with a single mask instead of separate masks for each token. This type of masking speeds up the training as it produces shorter sequences. After pre-training, the model is fine-tuned using adapter layers [106] for downstream tasks.

GPT-3 [6]: The GPT-3 architecture is the same as the GPT-2 [5] but with dense and sparse attention in transformer layers similar to the Sparse Transformer [67]. It shows that large models can train on larger batch sizes with a lower learning rate to decide the batch size during training, GPT-3 uses the gradient noise scale as in [107]. Overall, GPT-3 increases model parameters to 175B showing that the performance of large language

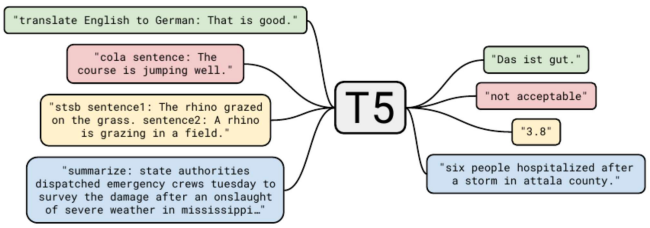

Figure 7: Unified text-to-text training example, source image from [10].

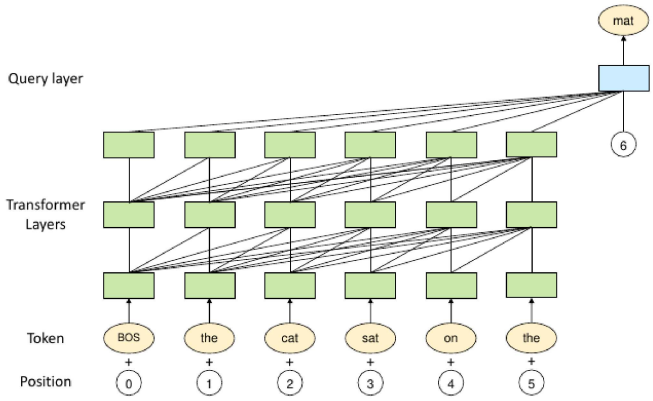

Figure 8: The image is the article of [108], showing an example of PanGu-α architecture.

models improves with the scale and is competitive with the fine-tuned models.

mT5 [11]: A multilingual T5 model [10] trained on the mC4 dataset with 101 languages. The dataset is extracted from the public common crawl scrape. The model uses a larger vocabulary size of 250,000 to cover multiple languages. To avoid over-fitting or under-fitting for a language, mT5 employs a data sampling procedure to select samples from all languages. The paper suggests using a small amount of pre-training datasets, including all languages when fine-tuning for a task using English language data. This allows the model to generate correct non-English outputs.

PanGu-α [108]: An autoregressive model that has a query layer at the end of standard transformer layers, example shown in Figure 8, to predict the next token. Its structure is similar to the transformer layer but with an additional embedding for the next position in the attention mechanism, given in Eq. 3.

$$a = p_n W_h^q W_h^k T H_L^T \tag{3}$$

CPM-2 [12]: Cost-efficient Pre-trained language Models (CPM-2) pre-trains bilingual (English and Chinese) 11B and 198B mixture-of-experts (MoE) models on the WuDaoCorpus [109] dataset. The tokenization process removes "_" white space tokens in the sentencepiece tokenizer. The models are trained with knowledge inheritance, starting with only the Chinese language in the first stage and then adding English and Chinese data. This trained model gets duplicated multiple times to initialize the 198B MoE model. Moreover, to use the model for downstream tasks, CPM-2 experimented with both complete fine-tuning and prompt fine-tuning as in [40] where only prompt-related parameters are updated by inserting prompts at various positions, front, middle, and back. CPM-2 also proposes the INFMOE, a memory-efficient framework with a strategy to dynamically offload parameters to the CPU for inference at a 100B scale. It overlaps data movement with inference computation for lower inference time.

ERNIE 3.0 [110]: ERNIE 3.0 takes inspiration from multi-task learning to build a modular architecture using Transformer-XL [111] as the backbone. The universal representation module is shared by all the tasks, which serve as the basic block for task-specific representation modules, which are all trained jointly for natural language understanding, natural language generation, and knowledge extraction. This LLM is primarily focused on the Chinese language. It claims to train on the largest Chinese text corpora for LLM training, and achieved state-of-the-art in 54 Chinese NLP tasks.

Jurassic-1 [112]: A pair of auto-regressive language models, including a 7B-parameter J1-Large model and a 178B-parameter J1-Jumbo model. The training vocabulary of Jurassic-1 comprise word pieces, complete words, and multi-word expressions without any word boundaries, where possible out-of-vocabulary instances are interpreted as Unicode bytes. Compared to the GPT-3 counterparts, the Jurassic-1 models apply a more balanced depth-to-width self-attention architecture [113] and an improved tokenizer for a faster prediction based on broader resources, achieving a comparable performance in zero-shot learning tasks and a superior performance in few-shot learning tasks given the ability to feed more examples as a prompt.

HyperCLOVA [114]: A Korean language model with GPT-3 architecture.

Yuan 1.0 [115]: Trained on a Chinese corpus with 5TB of high-quality text collected from the Internet. A Massive Data Filtering System (MDFS) built on Spark is developed to process the raw data via coarse and fine filtering techniques. To speed up the training of Yuan 1.0 to save energy expenses and carbon emissions, various factors that improve the performance of distributed training are incorporated in architecture and training: like increasing the hidden state size improves pipeline and tensor parallelism performance, larger micro batches improve pipeline parallelism performance, and larger global batch size improve data parallelism performance. In practice, the Yuan 1.0 model performs well on text classification, Winograd Schema, natural language inference, and reading comprehension tasks.

Gopher [116]: The Gopher family of models ranges from 44M to 280B parameters in size to study the effect of *scale* on the LLMs performance. The 280B model beats GPT-3 [6], Jurrasic-1 [112], MT-NLG [117], and others on 81% of the evaluated tasks.

ERNIE 3.0 TITAN [35]: ERNIE 3.0 Titan extends ERNIE 3.0 by training a larger model with 26x the number of parameters of the latter. This bigger model outperformed other state-of-the-art models in 68 NLP tasks. LLMs produce text with incorrect facts. In order to have control of the generated text with factual consistency, ERNIE 3.0 Titan adds another task, *Credible and Controllable Generations*, to its multi-task learning setup.

It introduces additional self-supervised adversarial and controllable language modeling losses to the pre-training step, which enables ERNIE 3.0 Titan to beat other LLMs in their manually selected Factual QA task set evaluations.

GPT-NeoX-20B [118]: An auto-regressive model that largely follows GPT-3 with a few deviations in architecture design, trained on the Pile dataset without any data deduplication. GPT-NeoX has parallel attention and feed-forward layers in a transformer block, given in Eq. 4, that increases throughput by 15%. It uses rotary positional embedding [66], applying it to only 25% of embedding vector dimension as in [119]. This reduces the computation without performance degradation. As opposed to GPT-3, which uses dense and sparse layers, GPT-NeoX-20B uses only dense layers. The hyperparameter tuning at this scale is difficult; therefore, the model chooses hyperparameters from the method [6] and interpolates values between 13B and 175B models for the 20B model. The model training is distributed among GPUs using both tensor and pipeline parallelism.

$$x + Attn(LN_1(x)) + FF(LN_2(x)) \tag{4}$$

OPT [14]: It is a clone of GPT-3, developed to open-source a model that replicates GPT-3 performance. Training of OPT employs dynamic loss scaling [120] and restarts from an earlier checkpoint with a lower learning rate whenever loss divergence is observed. Overall, the performance of OPT-175B models is comparable to the GPT3-175B model.

BLOOM [13]: A causal decoder model trained on the ROOTS corpus to open-source an LLM. The architecture of BLOOM is shown in Figure 9, with differences like ALiBi positional embedding, an additional normalization layer after the embedding layer as suggested by the bitsandbytes[1] library. These changes stabilize training with improved downstream performance.

GLaM [91]: Generalist Language Model (GLaM) represents a family of language models using a sparsely activated decoder-only mixture-of-experts (MoE) structure [121, 90]. To gain more model capacity while reducing computation, the experts are sparsely activated where only the best two experts are used to process each input token. The largest GLaM model, GLaM (64B/64E), is about 7× larger than GPT-3 [6], while only part of the parameters are activated per input token. The largest GLaM (64B/64E) model achieves better overall results as compared to GPT-3 while consuming only one-third of GPT-3's training energy.

MT-NLG [117]: A 530B causal decoder based on the GPT-2 architecture that has roughly 3× GPT-3 model parameters. MT-NLG is trained on filtered high-quality data collected from various public datasets and blends various types of datasets in a single batch, which beats GPT-3 on several evaluations.

Chinchilla [96]: A causal decoder trained on the same dataset as the Gopher [116] but with a little different data sampling distribution (sampled from MassiveText). The model architecture is similar to the one used for Gopher, with the exception of AdamW optimizer instead of Adam. Chinchilla identifies the relationship that model size should be doubled for every doubling of training tokens. Over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens are trained to get the estimates for compute-optimal training under a given budget. The authors train a 70B model with the same compute budget as Gopher (280B) but with 4 times more data. It outperforms Gopher [116], GPT-3 [6], and others on various downstream tasks, after fine-tuning.

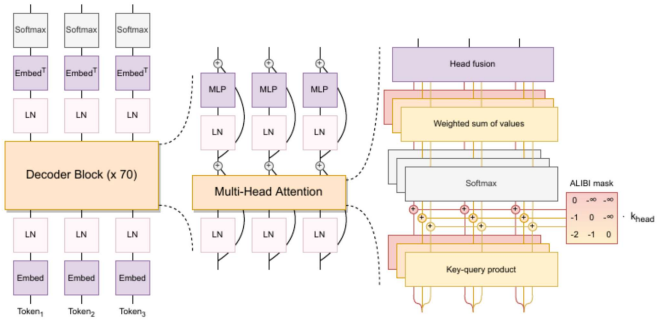

Figure 9: The BLOOM architecture example sourced from [13].

AlexaTM [122]: An encoder-decoder model, where encoder weights and decoder embeddings are initialized with a pre-trained encoder to speed up training. The encoder stays frozen for the initial 100k steps and is later unfrozen for end-to-end training. The model is trained on a combination of denoising and causal language modeling (CLM) objectives, concatenating a [*CLM*] token at the beginning for mode switching. During training, the CLM task is applied for 20% of the time, which improves the in-context learning performance.

PaLM [15]: A causal decoder with parallel attention and feed-forward layers similar to Eq. 4, speeding up training by a factor of 15. Additional changes to the conventional transformer model include SwiGLU activation, RoPE embeddings, multi-query attention that saves computation cost during decoding, and shared input-output embeddings. During training, loss spiking was observed, and to fix it, model training was restarted from a 100-step earlier checkpoint by skipping 200-500 batches around the spike. Moreover, the model was found to memorize around 2.4% of the training data at the 540B model scale, whereas this number was lower for smaller models.

PaLM-2 [123]: A smaller multi-lingual variant of PaLM, trained for larger iterations on a better quality dataset. PaLM-2 shows significant improvements over PaLM, while reducing training and inference costs due to its smaller size. To lessen toxicity and memorization, it appends special tokens with a fraction of pre-training data, which shows a reduction in generating harmful responses.

U-PaLM [124]: This method trains PaLM for 0.1% additional compute with the UL2 (also named as UL2Restore) objective [125], using the same dataset it outperforms the baseline significantly on various NLP tasks, including zero-shot, few-shot, commonsense reasoning, CoT, etc. Training with UL2R involves converting a causal decoder PaLM to a non-causal decoder PaLM and employing 50% sequential denoising, 25% regular denoising, and 25% extreme denoising loss functions.

[1] https://github.com/TimDettmers/bitsandbytes

UL2 [125]: An encoder-decoder architecture trained using a mixture of denoisers (MoD) objective. Denoisers include 1) R-Denoiser: a regular span masking, 2) S-Denoiser: which corrupts consecutive tokens of a large sequence and 3) X-Denoiser: which corrupts a large number of tokens randomly. During pre-training, UL2 includes a denoiser token from R, S, X to represent a denoising setup. It helps improve fine-tuning performance for downstream tasks that bind the task to one of the upstream training modes. This MoD style of training outperforms the T5 model on many benchmarks.

GLM-130B [33]: GLM-130B is a bilingual (English and Chinese) model trained using an auto-regressive mask infilling pre-training objective similar to the GLM [126]. This training style makes the model bidirectional as compared to GPT-3, which is unidirectional. As opposed to GLM, the training of GLM-130B includes a small amount of multi-task instruction pre-training data (5% of the total data) along with self-supervised mask infilling. To stabilize the training, it applies embedding layer gradient shrink.

LLaMA [127, 21]: A set of decoder-only language models varying from 7B to 70B parameters. LLaMA models series is the most famous among the community for parameter efficiency and instruction tuning.

LLaMA-1 *[127]:* Implements efficient causal attention [128] by not storing and computing masked attention weights and key/query scores. Another optimization is reducing the number of activations recomputed in the backward pass, as in [129].

LLaMA-2 *[21]:* This work is more focused on fine-tuning a safer and better LLaMA-2-Chat model for dialogue generation. The pre-trained model has 40% more training data with a larger context length and grouped-query attention.

PanGu-Σ [92]: An autoregressive model with parameters copied from PanGu-α and extended to a trillion scale with Random Routed Experts (RRE), the architectural diagram is shown in Figure 10. RRE is similar to the MoE architecture, with distinctions at the second level, where tokens are randomly routed to experts in a domain instead of using a learnable gating method. The model has bottom layers densely activated and shared across all domains, whereas top layers are sparsely activated according to the domain. This training style allows extracting task-specific models and reduces catastrophic forgetting effects in the case of continual learning.

3.1.2. Coding

CodeGen [130]: CodeGen has similar architecture to PaLM [15], i.e., parallel attention, MLP layers, and RoPE embeddings. The model is trained on both natural language and programming language data sequentially (trained on the first dataset, then the second and so on) on the following datasets 1) PILE, 2) BIGQUERY and 3) BIGPYTHON. CodeGen proposed a multi-step approach to synthesizing code. The purpose is to simplify the generation of long sequences where the previous prompt and generated code are given as input with the next prompt to generate the next code sequence. CodeGen open-source a Multi-Turn Programming Benchmark (MTPB) to evaluate multi-step program synthesis.

Codex [131]: This LLM is trained on a subset of public Python Github repositories to generate code from docstrings. Computer programming is an iterative process where the programs are often debugged and updated before fulfilling the requirements. Similarly to this, Codex generates 100 versions of a program by repetitive sampling for a given description, which produces a working solution for 77.5% of the problems passing unit tests. Its powerful version powers Github Copilot[2].

AlphaCode [132]: A set of large language models, ranging from 300M to 41B parameters, designed for competition-level code generation tasks. It uses the multi-query attention [133] to reduce memory and cache costs. Since competitive programming problems highly require deep reasoning and an understanding of complex natural language algorithms, the AlphaCode models are pre-trained on filtered GitHub code in popular languages and then fine-tuned on a new competitive programming dataset named CodeContests. The CodeContests dataset mainly contains problems, solutions, and test cases collected from the Codeforces platform[3]. The pre-training employs standard language modeling objectives, while GOLD [134] with tempering [135] serves as the training objective for the fine-tuning on CodeContests data. To evaluate the performance of AlphaCode, simulated programming competitions are hosted on the Codeforces platform: overall, AlphaCode ranks at the top 54.3% among over 5000 competitors, where its Codeforces rating is within the top 28% of recently participated users.

CodeT5+ [34]: CodeT5+ is based on CodeT5 [136], with shallow encoder and deep decoder, trained in multiple stages initially unimodal data (code) and later bimodal data (text-code pairs). Each training stage has different training objectives and activates different model blocks encoder, decoder, or both according to the task. The unimodal pre-training includes span denoising and CLM objectives, whereas bimodal pre-training objectives contain contrastive learning, matching, and CLM for text-code pairs. CodeT5+ adds special tokens with the text to enable task modes, for example, $[CLS]$ for contrastive loss, $[Match]$ for text-code matching, etc.

StarCoder [137]: A decoder-only model with the SantaCoder architecture, employing Flash attention to scale up the context length to 8k. The StarCoder trains an encoder to filter names, emails, and other personal data from the training data. Its fine-tuned variant outperforms PaLM, LLaMA, and LAMDA on HumanEval and MBPP benchmarks.

3.1.3. Scientific Knowledge

Galactica [138]: A large curated corpus of human scientific knowledge with 48 million papers, textbooks, lecture notes, millions of compounds and proteins, scientific websites, encyclopedias, and more are trained using the metaseq library3, which is built on PyTorch and fairscale [139]. The model wraps reasoning datasets with the $< work >$ token to provide step-by-step reasoning context to the model, which has been shown to improve the performance on reasoning tasks.

[2] https://github.com/features/copilot
[3] https://codeforces.com/