A Survey of Large Language Models

Wayne Xin Zhao, Kun Zhou*, Junyi Li*, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyou Liu, Jian-Yun Nie and Ji-Rong Wen

Abstract—Ever since the Turing Test was proposed in the 1950s, humans have explored the mastering of language intelligence by machine. Language is essentially a complex, intricate system of human expressions governed by grammatical rules. It poses a significant challenge to develop capable artificial intelligence (AI) algorithms for comprehending and grasping a language. As a major approach, language modeling has been widely studied for language understanding and generation in the past two decades, evolving from statistical language models to neural language models. Recently, pre-trained language models (PLMs) have been proposed by pre-training Transformer models over large-scale corpora, showing strong capabilities in solving various natural language processing (NLP) tasks. Since researchers have found that model scaling can lead to performance improvement, they further study the scaling effect by increasing the model size to an even larger scale. Interestingly, when the parameter scale exceeds a certain level, these enlarged language models not only achieve a significant performance improvement but also show some special abilities (e.g., in-context learning) that are not present in small-scale language models (e.g., BERT). To discriminate the difference in parameter scale, the research community has coined the term large language models (LLMs) for the PLMs of significant size (e.g., containing tens or hundreds of billions of parameters). Recently, the research on LLMs has been largely advanced by both academia and industry, and a remarkable progress is the launch of ChatGPT (a powerful AI chatbot developed based on LLMs), which has attracted widespread attention from society. The technical evolution of LLMs has been making an important impact on the entire AI community, which would revolutionize the way we develop and use AI algorithms. Considering this rapid technical progress, in this survey, we review the recent advances of LLMs by introducing the background, key findings, and mainstream techniques. In particular, we focus on four major aspects of LLMs, namely pre-training, adaptation tuning, utilization, and capacity evaluation. Besides, we also summarize the available resources for developing LLMs and discuss the remaining issues for future directions. This survey provides an up-to-date review of the literature on LLMs, which can be a useful resource for both researchers and engines.

Index Terms—Large Language Models; Emergent Abilities; Adaptation Tuning; Utilization; Alignment; Capacity Evaluation

1 INTRODUCTION

Language is a prominent ability of human beings for expression and communication, which develops in early childhood and evolves over a lifetime [1,2]. Whereas for machines, they cannot naturally grasp the abilities of understanding and communicating in the form of human language, unless equipped with powerful artificial intelligence (AI) algorithms. To achieve this goal, it has been a longstanding research challenge that enables machines to read, write, and communicate like humans [3].

Technically, language modeling (LM) is one of the major approaches to advancing language intelligence of machines. In general, LM aims to model the generative likelihood of word sequences, so as to predict the probabilities of future (or missing) tokens. The research of LM has received extensive research attention in the literature, which can be roughly divided into four major development stages:

- **Statistical language models (SLM).** SLMs [4–7] are developed based on statistical learning methods that rose in the 1990s. The basic idea is to build the word prediction model based on the Markov assumption, e.g., predicting the next word based on the most recent context. The SLMs with a fixed context length \( n \) are also called \( n \)-gram language models, e.g., bigram and trigram language models. SLMs have been widely applied to enhance task performance in information retrieval (IR) [8,9] and natural language processing (NLP) [10,12]. However, they often suffer from the curse of dimensionality: it is difficult to accurately estimate high-order language models since an exponential number of transition probabilities need to be estimated. Thus, specially designed smoothing strategies such as back-off estimation [13] and Good–Turing estimation [14] have been introduced to alleviate the data sparsity problem.

- **Pre-trained language models (PLM).** As an early attempt, ELMo [21] was proposed to capture context-aware word representations by first pre-training a bidirectional LSTM (biLSTM) network (instead of learning fixed word representations) and fine-tuning the biLSTM network ac-
According to specific downstream tasks. Further, based on the highly parallelizable Transformer architecture [22] with self-attention mechanisms, BERT [23] was proposed by pre-training bidirectional language models with specially designed pre-training tasks on large-scale unlabeled corpora. These pre-trained context-aware word representations are very effective as general-purpose semantic features, which have largely raised the performance bar of NLP tasks. This work has inspired a large number of follow-up work, which sets the “pre-training and fine-tuning” learning paradigm. Following this paradigm, a great number of studies on PLMs have been developed, introducing either different architectures [24, 25] (e.g., GPT-2 [26] and BART [24]) or improved pre-training strategies [27, 29]. In this paradigm, it often requires fine-tuning the PLM for adapting to different downstream tasks.

- **Large language models (LLM).** Researchers find that scaling PLM (e.g., scaling model size or data size) often leads to an improved model capacity on downstream tasks (i.e., following the scaling law [30]). A number of studies have explored the performance limit by training an ever larger PLM (e.g., the 175B-parameter GPT-3 and the 540B-parameter PaLM). Although scaling is mainly conducted in model size (with similar architectures and pre-training parameter PaLM). Although scaling is mainly conducted in model size (with similar architectures and pre-training), these large-sized PLMs display different behaviors from smaller PLMs (e.g., 330M-parameter BERT and 1.5B-parameter GPT-2) and show surprising abilities (called emergent abilities [31]) in solving a series of complex tasks. For example, GPT-3 can solve few-shot tasks through in-context learning, whereas GPT-2 cannot do well. Thus, the research community coins the term “large language models (LLM)” for these large-sized PLMs [32–35]. A remarkable application of LLMs is ChatGPT [36] that adapts the LLMs from the GPT series for dialogue, which presents an amazing conversation ability with humans.

In the existing literature, PLMs have been widely discussed and surveyed [36, 39], while LLMs are seldom reviewed in a systematic way. To motivate our survey, we first highlight three major differences between LLMs and PLMs. First, LLMs display some surprising emergent abilities that may not be observed in previous smaller PLMs. These abilities are key to the performance of language models on complex tasks, making AI algorithms unprecedentedly powerful and effective. Second, LLMs have revolutionized the way that humans develop and use AI algorithms. Unlike small PLMs, the major approach to accessing LLMs is through the prompting interface (e.g., GPT-4 API). Humans have to understand how LLMs work and format their tasks in a way that LLMs can follow. Third, the development of LLMs no longer draws a clear distinction between research and engineering. The training of LLMs requires extensive practical experiences in large-scale data processing and distributed parallel training. To develop capable LLMs, researchers have to solve complicated engineering issues, working with engineers or being engineers.

Nowadays, LLMs are posing a significant impact on the AI community, and the advent of ChatGPT and GPT-4 even leads to the rethinking of the possibilities of artificial general intelligence (AGI). OpenAI has published a technical article entitled “Planning for AGI and beyond”, which discussed the short-term and long-term plans to approach AGI [40], and a more recent paper has argued that GPT-4 might be considered as an early version of an AGI system [41]. The research areas of AI have been revolutionized by the rapid progress of LLMs. In the field of NLP, LLMs can serve as a general-purpose language task solver (to some extent), and the research paradigm has been shifting towards the use of LLMs. In the field of IR, traditional search engines are challenged by the new information seeking way through AI chatbots (i.e., ChatGPT), and New Bing [42] presents an initial attempt that enhances the search results based on LLMs. In the field of CV, the researchers try to develop ChatGPT-like vision-language models that can better serve multimodal dialogues [42–45], and GPT-4 [46] has supported multimodal input by integrating the visual information. This new wave of technology would potentially lead to a prosperous ecosystem of real-world applications based on LLMs. For instance, Microsoft 365 is being empowered by LLMs (i.e., Copilot) to automate office work, and ChatGPT enables the integration of useful plugins for solving complex tasks.

Despite the progress and impact, the underlying principles of LLMs are still not well explored. Firstly, it is still mysterious why emergent abilities occur in LLMs, instead of smaller PLMs. As a more general issue, there still lacks a deep, detailed investigation of the key factors that contribute to the abilities of LLMs. It is important to study when and how LLMs obtain such abilities [47]. Although there are some meaningful discussions about this problem [31, 47], more principled investigations are needed to uncover the “secrets” of LLMs. Secondly, it is difficult to train capable LLMs for the research community. Due to the huge cost of model pre-training, it is difficult to carry out repetitive, ablating studies for the research community. Indeed, LLMs are mainly trained by industry, where many important training details (e.g., data collection and cleaning) are not revealed to the public. Thirdly, it is very challenging to align LLMs with human values or preferences. Despite the capacities, LLMs are also likely to produce toxic, fictitious, or harmful contents. It requires effective and efficient control approaches to eliminating the potential risk of the use of LLMs [48].

Faced with both opportunities and challenges, it needs more attention on the research and development of LLMs. In order to provide a basic understanding of LLMs, this survey provides a literature review of the recent advances in LLMs from four major aspects, including pre-training (how to pre-train a capable LLM), adaptation tuning (how to effectively tune pre-trained LLMs from the two perspectives of effectiveness and safety), utilization (how to use LLMs for solving various downstream tasks) and capability evaluation (how to evaluate the abilities of LLMs and existing empirical findings). We thoroughly comb the literature and summarize the key findings, techniques, and methods of LLMs. For this survey, we also create a GitHub project website [4], by collecting the supporting resources for LLMs.

---

1. Note that a LLM is not necessarily more capable than a small PLM, and some emergent abilities may not always occur in LLMs.
2. https://openai.com/blog/chatgpt/
4. https://github.com/RUCAIBox/LLMSurvey
We are also aware of several related review articles on PLMs or LLMs \[32, 36, 38, 39, 43, 45-54\]. These papers either discuss PLMs or some specific (or general) aspects of LLMs. Compared with them, we focus on the techniques and methods to develop and use LLMs and provide a relatively comprehensive reference to important aspects of LLMs.

The remainder of the survey is organized as follows: Section 2 introduces the background for the survey article, with the terminology, settings, resources, and organization outline, followed by the summarization of available resources for developing LLMs in Section 3. Sections 4, 5, 6, and 7 review and summarize the recent progress from the four aspects of pre-training, adaptation tuning, utilization, and capacity evaluation, respectively. Finally, we conclude this survey in Section 8 by summarizing the major findings and discuss the remaining issues for future work.

2 OVERVIEW

In this section, we introduce the background of LLMs with key terminologies, abilities and techniques.

**Background.** Typically, large language models (LLMs) refer to language models that contain hundreds of billions (or more) of parameters which are trained on massive text data \[32\], such as GPT-3 \[55\], PaLM \[56\], Galactica \[38\], and LLaMA \[57\]. Specifically, LLMs are built upon the Transformer architecture \[22\], where multi-head attention layers are stacked in a very deep neural network. Existing LLMs mainly adopt similar model architectures (i.e., Transformer) and pre-training objectives (i.e., language modeling) as small language models. As the major difference, LLMs largely scale the model size, pre-training data, and total compute (orders of magnification). They can better understand the natural language and generate high-quality text based on the given context (i.e., prompts). Such a capacity improvement can be partially described by the scaling law, where the performance roughly follows a substantial increase with respect to the model size \[30\]. However, some abilities (e.g., in-context learning \[55\]) are unpredictable according to the scaling law, which can be observed only when the model size exceeds a certain level (as discussed below).

**Emergent Abilities of LLMs.** In the literature \[31\], emergent abilities of LLMs are formally defined as “the abilities that are not present in small models but arise in large models”, which is one of the most distinctive features that distinguish LLMs from previous PLMs. It also introduces a notable characteristic when emergent abilities occur: performance rises significantly above random when the scale reaches a certain level. By analogy, such an emergent pattern has close connections with the phenomenon of phase transition in physics \[31, 58\]. In principle, emergent abilities can be also defined in relation to some complex tasks \[31, 59\], while we are more concerned with general abilities that can be applied to solve multiple tasks. Here, we briefly introduce three representative emergent abilities for LLMs, described as follows.

5. In existing literature, there is no a formal consensus on the minimum parameter scale for LLMs. In this survey, we mainly focus on discussing LLMs with a model size larger than 10B.

6. In some recent studies \[50\], it also shows that in-context learning implicitly performs meta-optimization through the attention mechanism.

- **In-context learning.** The in-context learning ability is formally introduced by GPT-3 \[55\]: assuming that the language model has been provided with a natural language instruction and/or several task demonstrations, it can generate the expected output for the test instances by completing the word sequence of input text, without requiring additional training or gradient update \[55\].

- **Instruction following.** By fine-tuning with a mixture of multi-task datasets formatted via natural language descriptions (i.e., instructions), LLMs are shown to perform well on unseen tasks that are also described in the form of instructions \[28, 61, 62\]. With this ability, instruction tuning enables LLMs to perform new tasks by understanding the task instructions without using explicit examples, which can largely improve the generalization ability.

- **Step-by-step reasoning.** For small language models, it is usually difficult to solve complex tasks that involve multiple reasoning steps, e.g., mathematical word problems. While, with the chain-of-thought reasoning strategy \[33\], LLMs can solve such tasks by utilizing the prompting mechanism that involves intermediate reasoning steps for deriving the final answer. This ability is speculated to be potentially obtained by training on code \[33, 47\].

**Key Techniques for LLMs.** It has been a long way that LLMs evolve into the current state: general and capable learners. In the development process, a number of important techniques are proposed, which largely improve the capacity of LLMs. Here, we briefly list several important techniques that (potentially) lead to the success of LLMs, as follows.

- **Scaling.** Scaling is the key factor to increase the model capacity of LLMs. As the initial attempt, GPT-3 firstly increases the model size to an extremely large scale of 175B parameters. Later on, PaLM further raises the parameter scale to a new record of 540B. As discussed before, a large model size is essential to emergent abilities. While, scaling is not only conducted on model size but also related to data size and total compute \[31, 63\]. A recent study \[34\] has discussed the optimal schedule among the three aspects of model size, data size, and total compute, given a fixed budget. Further, the quality of the pre-training data plays a key role in achieving good performance, so that data collection and cleaning strategies are very important to consider when scaling the pre-training corpora.

- **Training.** Due to the huge model size, it is very challenging to successfully train a capable LLM. Distributed training algorithms are needed to learn the network parameters of LLMs, in which various parallel strategies are often jointly utilized. To support distributed training, several optimization frameworks have been released to facilitate the implementation and deployment of parallel algorithms, such as DeepSpeed \[64\] and Megatron-LM \[65\]. Besides, optimization tricks are also important for training stability and model performance, e.g., restart with training loss spike \[50\] and mixed precision training \[65\]. More recently, GPT-4 \[46\] proposes to develop special infrastructure and...
optimization methods that reliably predict the performance of large models with much smaller models.

- **Ability eliciting.** After being pre-trained on large-scale corpora, LLMs are endowed with potential abilities as general task solvers. While, these abilities might not be explicitly exhibited when LLMs perform some specific tasks. As a major approach, it is useful to design suitable task instructions or specific in-context strategies to elicit such abilities. For instance, chain-of-thought prompting has been shown to be useful to solve complex reasoning tasks by including intermediate reasoning steps. Besides, we can further perform instruction tuning on LLMs with task descriptions in natural language, for improving the generalizability of LLMs on unseen tasks. While, these techniques mainly correspond to the emergent abilities of LLMs, which may not show the same effect on small language models.

- **Alignment tuning.** Since LLMs are trained to capture the data characteristics of pre-training corpora (including both high-quality and low-quality data), they are likely to generate toxic, biased, or even harmful content for humans. It is necessary to align LLMs with human values, e.g., *helpful, honest, and harmless*. For this purpose, InstructGPT [61] designs an effective tuning approach that enables LLMs to follow the expected instructions, which utilizes the technique of *reinforcement learning with human feedback* [61, 67]. It incorporates human in the training loop with elaborately designed labeling strategies. ChatGPT is indeed developed on a similar technique to InstructGPT, which shows a strong alignment capacity in producing high-quality, harmless responses, e.g., rejecting to answer insulting questions.

- **Tools manipulation.** In essence, LLMs are trained as text generators over massive plain text corpora, thus performing less well on the tasks that are not best formed or expressed in the text (e.g., numerical computation). Besides, their capacities are also limited to the pre-training data, e.g., the inability to capture up-to-date information. To tackle these issues, a recently proposed technique is to employ external tools to compensate for the deficiencies of LLMs [68, 69]. For example, LLMs can utilize the calculator for accurate computation [69] and employ search engines to retrieve unknown information [70]. More recently, ChatGPT has enabled the mechanism of using external plugins (existing or newly created apps) [7], which are by analogy with the “eyes and ears” of LLMs. Such a mechanism can broadly expand the scope of capacities for LLMs.

Besides, many other factors (e.g., the upgrade of hardware) also contribute to the success of LLMs. While, we limit our discussion to the technical approaches and key findings for developing LLMs.

## 3 Resources of LLMs

It is by no means an easy job to develop or reproduce LLMs, considering the challenging technical issues and huge demands of computation resources. A feasible way is to learn experiences from existing LLMs and reuse publicly available resources for incremental development or experimental study. In this section, we will mainly summarize the open-source model checkpoints or APIs, available corpora, and useful libraries for LLMs.

### 3.1 Publicly Available Model Checkpoints or APIs

Given the huge cost of model pre-training, well-trained model checkpoints are critical to the study and development of LLMs for the research community. Since the parameter scale is a key factor to consider for using LLMs, we categorize these public models into two scale levels (i.e., *tens of billions of parameters or hundreds of billions of parameters*), which is useful for users to identify the suitable resources according to their resource budget. Besides, for inference, we can directly employ public APIs to perform our tasks, without running the model locally. In this section, we briefly summarize the publicly available checkpoints and APIs for LLMs.

**Models with Tens of Billions of Parameters.** Most open-source models of this category have a parameter scale ranging from 10B to 200B, except LLaMA [57] (containing 65B parameters in the largest version). Other models within this range include mT5 [72], T0 [28], GPT-NeoX-20B [75], CodeGen [76], UL2 [78], Flan-T5 [81], mT0 [82], and PanGu-α [73]. Among them, Flan-T5 (11B version) can serve as a premier model for research on instruction tuning, since it explores the instruction tuning from three aspects [81]: increasing the number of tasks, scaling the model size, and fine-tuning with chain-of-thought prompting data. Besides, CodeGen (11B version), as an autoregressive language model designed for generating code, can be considered as a good open-source candidate for exploring the code generation ability. It also introduces a new benchmark MTPB [76] specially for multi-turn program synthesis, which is composed by 115 expert-generated problems. To solve these problems, it requires LLMs to acquire sufficient programming knowledge (e.g., math, array operations, and algorithms). As for multilingual tasks, mT0 (13B version) might be a good candidate model, which has been fine-tuned on multilingual tasks with multilingual prompts. Furthermore, PanGu-α [73] shows good performance in Chinese downstream tasks in zero-shot or few-shot settings, which is developed based on the deep learning framework MindSpore [29]. Note that PanGu-α [73] holds multiple versions of models (up to 200B parameters), while the largest public version has 13B parameters. As a more recent release, LLaMA (65B version) [57], which contains approximately five times as many parameters as other models, has exhibited superior performance in tasks related to instruction following. Typically, pre-training models at this scale require hundreds or even thousands of GPUs or TPUs. For instance, GPT-NeoX-20B uses 12 supermicro servers, each equipped with 8 NVIDIA A100-540GB GPUs, while LLaMA utilizes 2,048 A100-80G GPUs as reported in their original publications. To accurately estimate the computation resources needed, it is suggested to use the metrics measuring the number of involved computations such as FLOPS (i.e., FLoating point number Operations Per Second) [30].

**Models with Hundreds of Billions of Parameters.** For models in this category, only a handful of models have been publicly released. For example, OPT [79], OPT-IML [83], BLOOM [69], and BLOOMZ [82] have nearly the same number of parameters as GPT-3 (175B version), while GLM [80]...
3.2 Commonly Used Corpora

In contrast to earlier PLMs, LLMs which consist of a significantly larger number of parameters require a higher volume of training data that covers a broad range of content. For this need, there are increasingly more accessible training datasets that have been released for research. In this section, we will briefly summarize several widely used corpora for training LLMs. Based on their content types, we categorize these corpora into six groups: Books, CommonCrawl, Reddit links, Wikipedia, Code, and others.

**Books.** BookCorpus [100] is a commonly used dataset in previous small-scale models (e.g., GPT [110] and GPT-2 [26]), consisting of over 11,000 books covering a wide range of topics and genres (e.g., novels and biographies). Another large-scale book corpus is Project Gutenberg [101], consisting of over 70,000 literary books including novels, essays, poetry, drama, history, science, philosophy, and other types of works in the public domain. It is currently one of the largest open-source book collections, which is used in training of MT-NLG [90] and LLaMA [57]. As for Books1 [55] and Books2 [55] used in GPT-3 [55], they are much larger than BookCorpus but have not been publically released so far.

**CommonCrawl.** CommonCrawl [111] is one of the largest open-source web crawling databases, containing a petabyte-scale data volume, which has been widely used as training...
data scale (either in the number of tokens or storage size) and hardware resource costs. Here, “Adaptation” indicates whether the model has been with subsequent fine-tuning: IT denotes instruction tuning and RLHF denotes reinforcement learning with human feedback. “Evaluation” indicates whether the model has been evaluated with corresponding abilities in their original paper: ICL denotes in-context learning and CoT denotes chain-of-thought. *** denotes the largest publicly available version.

### TABLE 1
Statistics of large language models (having a size larger than 10B in this survey) in recent years, including the capacity evaluation, pre-training data scale (either in the number of tokens or storage size) and hardware resource costs. Here, “Adaptation” indicates whether the model has been with subsequent fine-tuning: IT denotes instruction tuning and RLHF denotes reinforcement learning with human feedback. “Evaluation” indicates whether the model has been evaluated with corresponding abilities in their original paper: ICL denotes in-context learning and CoT denotes chain-of-thought. *** denotes the largest publicly available version.

<table>
<thead>
<tr>
<th>Model</th>
<th>Release Time</th>
<th>Size (B)</th>
<th>Base Model</th>
<th>Adaptation</th>
<th>IT</th>
<th>RLHF</th>
<th>Pre-train Data Scale</th>
<th>Latest Data Timestamp</th>
<th>Hardware (GPUs / TPUs)</th>
<th>Training Time</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GShard [54]</td>
<td>Jan-2020</td>
<td>600</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GPT-3 [55]</td>
<td>May-2020</td>
<td>175</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LaMDa [85]</td>
<td>May-2021</td>
<td>137</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HyperCLOVA [86]</td>
<td>Jun-2021</td>
<td>82</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Codex [57]</td>
<td>Jul-2021</td>
<td>12</td>
<td>GPT-3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ERNIE 3.0  [88]</td>
<td>Jul-2021</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Jurasic-1  [89]</td>
<td>Aug-2021</td>
<td>178</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FLAN [90]</td>
<td>Oct-2021</td>
<td>137</td>
<td>LaMDA</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MT-XLG [91]</td>
<td>Oct-2021</td>
<td>530</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Yuan 1 0.1  [92]</td>
<td>Oct-2021</td>
<td>245</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WebGPT-70 [93]</td>
<td>Dec-2021</td>
<td>175</td>
<td>GPT-3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gopher [59]</td>
<td>Dec-2021</td>
<td>280</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ERNIE 3.0 Titan [92]</td>
<td>Dec-2021</td>
<td>260</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GLaM [95]</td>
<td>Dec-2021</td>
<td>1200</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>InstructGPT-70 [96]</td>
<td>Jan-2022</td>
<td>175</td>
<td>GPT-3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AlphaCode  [97]</td>
<td>Feb-2022</td>
<td>41</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Chinchilla [98]</td>
<td>Mar-2022</td>
<td>70</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PaLM [56]</td>
<td>Apr-2022</td>
<td>540</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AlexaTM [99]</td>
<td>Aug-2022</td>
<td>20</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Sparrow [100]</td>
<td>Sep-2022</td>
<td>70</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>U-PaLM [101]</td>
<td>Oct-2022</td>
<td>540</td>
<td>PaLM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Flan-U-PaLM [102]</td>
<td>Oct-2022</td>
<td>540</td>
<td>U-PaLM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GPT-4 [103]</td>
<td>Mar-2023</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PanGu-Σ [98]</td>
<td>Mar-2023</td>
<td>1085</td>
<td>PanGu-o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### TABLE 2
Statistics of commonly-used data sources.

<table>
<thead>
<tr>
<th>Corpora</th>
<th>Size</th>
<th>Source</th>
<th>Latest Update Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>BookCorpus</td>
<td>5GB</td>
<td>Books</td>
<td>Dec-2015</td>
</tr>
<tr>
<td>Gutenberg</td>
<td>10GB</td>
<td>Books</td>
<td>Dec-2021</td>
</tr>
<tr>
<td>C4 [71]</td>
<td>800GB</td>
<td>CommonCrawl</td>
<td>Apr-2019</td>
</tr>
<tr>
<td>CC-stories-R [102]</td>
<td>31GB</td>
<td>CommonCrawl</td>
<td>Sep-2019</td>
</tr>
<tr>
<td>CC-NEWS [27]</td>
<td>78GB</td>
<td>CommonCrawl</td>
<td>Feb-2019</td>
</tr>
<tr>
<td>REALNews [103]</td>
<td>120GB</td>
<td>CommonCrawl</td>
<td>Apr-2019</td>
</tr>
<tr>
<td>OpenWebText [104]</td>
<td>38GB</td>
<td>Reddit links</td>
<td>Mar-2023</td>
</tr>
<tr>
<td>Pushshift.io [105]</td>
<td>-</td>
<td>Reddit links</td>
<td>Mar-2023</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>10GB</td>
<td>Wikipedia</td>
<td>Mar-2023</td>
</tr>
<tr>
<td>BigQuery [107]</td>
<td>8GB</td>
<td>Codes</td>
<td>Mar-2023</td>
</tr>
<tr>
<td>the Pile [108]</td>
<td>800GB</td>
<td>Other</td>
<td>Dec-2020</td>
</tr>
<tr>
<td>ROOTS [109]</td>
<td>1.6TB</td>
<td>Other</td>
<td>Jun-2022</td>
</tr>
</tbody>
</table>

10. https://www.tensorflow.org/datasets/catalog/c4
posed of a subset of CommonCrawl data, in which the contents are made in a story-like way. While, the original source of CC-Stories is not available now, so a reproduction version, CC-Stories-R [112], has been included in Table 2. Moreover, two news corpora extracted from CommonCrawl, i.e., REALNEWS (120G) and CC-News (76G), are also commonly used as the pre-training data.

**Reddit Links.** Reddit is a social media platform that enables users to submit links and text posts, which can be voted on by others through “upvotes” or “downvotes”. Highly upvoted posts are often considered useful, and can be utilized to create high-quality datasets. WebText [25] is a well-known corpus composed of highly upvoted links from Reddit, but it is not publicly available. As a surrogate, there is a readily accessible open-source alternative called OpenWebText [104]. Another corpus extracted from Reddit is PushShift.io [105], a real-time updated dataset that consists of historical data from Reddit since its creation day. Pushshift provides not only monthly data dumps but also useful utility tools to support users in searching, summarizing, and conducting preliminary investigations on the entire dataset. This makes it easy for users to collect and process Reddit data.

**Wikipedia.** Wikipedia [106] is an online encyclopedia containing a large volume of high-quality articles on diverse topics. Most of these articles are composed in an expository style of writing (with supporting references), covering a wide range of languages and fields. Typically, the English-only filtered versions of Wikipedia are widely used in most LLMs (e.g., GPT-3 [55], LaMDA [53], and LLaMA [57]). Wikipedia is available in multiple languages, so it can be used in multilingual settings.

**Code.** To collect code data, existing work mainly crawls open-source licensed codes from the Internet. Two major sources are public code repositories under open-source licenses (e.g., GitHub) and code-related question-answering platforms (e.g., StackOverflow). Google has publicly released the BigQuery dataset [107], which includes a substantial number of open-source licensed code snippets in various programming languages, serving as a representative code dataset. CodeGen has utilized BIGQUERY [25], a subset of the BigQuery dataset, for training the multilingual version of CodeGen (CodeGen-Multi).

**Others.** The Pile [108] is a large-scale, diverse, and open-source text dataset consisting of over 800GB of data from multiple sources, including books, websites, codes, scientific papers, and social media platforms. It is constructed from 22 diverse high-quality subsets. The Pile dataset is widely used in models with different parameter scales, such as GPT-J (6B) [113], CodeGen (16B) [25], and Megatron-Turing NLG (530B) [65]. Besides, ROOTS [109] is composed of various smaller datasets (totally 1.61 TB of text) and covers 59 different languages (containing natural languages and programming languages), which have been used for training BLOOM [65].

As we can see from Figure 2, LLMs no longer rely on a single corpus, but instead utilize multiple data sources for pre-training. Therefore, existing studies commonly mix several ready-made datasets (e.g., C4, OpenWebText, and the Pile), and then perform further processing to obtain the pre-training corpus. Besides, to train the LLMs that are adaptive to specific applications, it is also important to extract data from relevant sources (e.g., Wikipedia and BigQuery) for enriching the corresponding information in pre-training data. To have a quick reference of the data sources used in existing LLMs, we present the pre-training corpora of three representative LLMs:

- **GPT-3** (175B) [55] was trained on a mixed dataset of 300B tokens, including CommonCrawl [111], WebText2 [55], Books1 [55], Books2 [55], and Wikipedia [106].
- **PaLM** (540B) [56] uses a pre-training dataset of 780B tokens, which is sourced from social media conversations, filtered webpages, books, Github, multilingual Wikipedia, and news.
- **LLaMA** [57] extracts training data from various sources, including CommonCrawl, C4 [71], Github, Wikipedia, books, ArXiv, and StackExchange. The training data size for LLaMA (6B) and LLaMA (13B) is 1.0T tokens, while 1.4T tokens are used for LLaMA (32B) and LLaMA (65B).

### 3.3 Library Resource

In this part, we briefly introduce a series of available libraries for developing LLMs.

- **Transformers** [114] is an open-source Python library for building models using the Transformer architecture, which is developed and maintained by Hugging Face. It has a simple and user-friendly API, making it easy to use and customize various pre-trained models, as well as tools for dataset processing and evaluation. It is a powerful library with a large and active community of users and developers who regularly update and improve the models and algorithms.
- **DeepSpeed** [64] is a PyTorch-based deep learning optimization library developed by Microsoft, which has been used to train a number of LLMs, such as GPT-Neo [115] and BLOOM [65]. It provides various optimization techniques for distributed training, such as memory optimization (ZeRO technique), gradient checkpointing, and pipeline parallelism. Additionally, it provides the API for fine-tuning and evaluating these models.
- **Megatron-LM** [116] is a PyTorch-based deep learning library developed by NVIDIA for training large-scale language models. It also provides rich optimization techniques for distributed training, including model and data parallelism, mixed-precision training, FlashAttention, and gradient checkpointing. These optimization techniques can significantly improve the training efficiency and speed, enabling efficient distributed training across GPUs and machines.
- **JAX** [117] is a Python library for high-performance machine learning developed by Google Brain, allowing users to easily perform computations on arrays with hardware acceleration (GPU or TPU) support. It supports computation on various devices and also provides several convenient functions, such as just-in-time compilation acceleration and automatic batching.
- **Colossal-AI** [118] is a deep learning library developed by EleutherAI for training large-scale language models. It is built on top of JAX and supports optimization strategies for
training such as mixed-precision training and parallelism. Recently, a ChatGPT-like model called ColossalChat \cite{119} has been publicly released with two versions (7B and 13B), which are developed using Colossal-AI based on LLaMA \cite{57}.

- **BMTrain** \cite{120} is an efficient library developed by OpenBMB for training models with large-scale parameters in a distributed manner, which emphasizes code simplicity, low resource, and high availability. BMTrain has already incorporated several common LLMs (e.g., Flan-T5 \cite{81} and GLM \cite{80}) into its ModelCenter, where developers can use these models directly.

- **FastMoE** \cite{121} is a specialized training library for MoE (i.e., mixture-of-experts) models. It is developed on top of PyTorch, prioritizing both efficiency and user-friendliness in its design. FastMoE simplifies the process of transferring Transformer models to MoE models and supports both data parallelism and model parallelism during training.

Besides the above libraries, existing deep learning frameworks (e.g., PyTorch \cite{122}, TensorFlow \cite{123}, MXNet \cite{124}, PaddlePaddle \cite{125}, MindSpore \cite{99}, and OneFlow \cite{126}) have also provided support for parallelism algorithms, which are commonly used for training large-scale models.

## 4 Pre-training

Pre-training establishes the basis of the abilities of LLMs. By pre-training on large-scale corpora, LLMs can acquire essential language understanding and generation skills \cite{55, 56}. In this process, the scale and quality of the pre-training corpus are critical for LLMs to attain powerful capabilities. Besides, to effectively pre-train LLMs, model architectures, acceleration methods, and optimization techniques need to be well designed. In what follows, we first discuss the data collection and processing in Section 4.1, then introduce the commonly used model architectures in Section 4.2, and finally present the training techniques to stably and efficiently optimize LLMs in Section 4.3.

### 4.1 Data Collection

Compared with small-scale language models, LLMs have a stronger demand for high-quality data for model pre-training, and their model capacities largely rely on the pre-training corpus and how it has been preprocessed. In this part, we discuss the collection and processing of pre-training data, including data sources, preprocessing methods, and important analysis of how pre-training data affects the performance of LLMs.

#### 4.1.1 Data Source

To develop a capable LLM, it is key to collect a large amount of natural language corpus from various data sources. Existing LLMs mainly leverage a mixture of diverse public textual datasets as the pre-training corpus. Figure 2 shows the distribution of the sources of pre-training data for several existing LLMs.

The source of pre-training corpus can be broadly categorized into two types: general data and specialized data. General data, such as webpages, books, and conversational text, is utilized by most LLMs \cite{55, 56, 79} due to its large, diverse, and accessible nature, which can enhance the language modeling and generalization abilities of LLMs. In light of the impressive generalization capabilities exhibited by LLMs, there are also studies that extend their pre-training corpus to more specialized datasets, such as multilingual data, scientific data, and code, endowing LLMs with specific task-solving capabilities \cite{55, 56, 76}. In what follows, we describe these two types of pre-training data sources and their effects on LLMs. For a detailed introduction to the commonly used corpus, one can refer to Section 3.2.

#### General Data

As we can see in Figure 2, the vast majority of LLMs adopt general-purpose pre-training data, such as webpages, books, and conversational text, which provides rich text sources on a variety of topics. Next, we briefly summarize three important kinds of general data.

- **Webpages.** Owing to the proliferation of the Internet, various types of data have been created, which enables LLMs to gain diverse linguistic knowledge and enhance their generalization capabilities \cite{26, 71}. For convenient use of these data resources, a large amount of data is crawled from the web in previous work, such as CommonCrawl \cite{111}. However, the crawled web data tends to contain both high-quality text, such as Wikipedia and low-quality text, like spam mail, thus it is important to filter and process webpages for improving the data quality.

- **Conversation text.** Conversation data can enhance the conversational competence of LLMs \cite{29} and potentially improve their performance on a range of question-answering tasks \cite{56}. Researchers can utilize subsets of public conversation corpus (e.g., PushShift.io Reddit corpus) \cite{105, 127} or collect conversation data from online social media. Since online conversational data often involves discussions among multiple participants, an effective processing way is to transform a conversation into a tree structure, where the utterance is linked to the one it responds to. In this way, the multi-party conversation tree can be divided into multiple sub-conversations, which can be collected in the pre-training corpus. Furthermore, a potential risk is that the excessive integration of dialogue data into LLMs may result in a side effect \cite{79}: declarative instructions and direct interrogatives are erroneously perceived as the beginning of conversations, thus leading to a decline in the efficacy of the instructions.

- **Books.** Compared to other corpus, books provide an important source of formal long texts, which are potentially beneficial for LLMs to learn linguistic knowledge, model long-term dependency, and generate narrative and coherent texts. To obtain open-source book data, existing studies usually adopt the Books3 and Bookcorpus2 datasets, which are available in the Pile dataset \cite{108}.

#### Specialized Data

Specialized datasets are useful to improve the specific capabilities of LLMs on downstream tasks. Next, we introduce three kinds of specialized data.

- **Multilingual text.** Besides the text in the target language, integrating a multilingual corpus can enhance the multilingual abilities of language understanding and generation. For example, BLOOM \cite{66} and PaLM \cite{56} have curated multilingual data covering 46 and 122 languages, respectively, within their pre-training corpora. These models demonstrate impressive performance in multilingual tasks,
such as translation, multilingual summarization, and multilingual question answering, and achieve comparable or superior performance to the state-of-the-art models that are fine-tuned on the corpus in the target language(s).

- **Scientific text.** The exploration of science by humans has been witnessed by the increasing growth of scientific publications. In order to enhance the understanding of scientific knowledge for LLMs, it is useful to incorporate a scientific corpus for model pre-training. By pre-training on a vast amount of scientific text, LLMs can achieve impressive performance in scientific and reasoning tasks. To construct the scientific corpus, existing efforts mainly collect arXiv papers, scientific textbooks, math webpages, and other related scientific resources. Due to the complex nature of data in scientific fields, such as mathematical symbols and protein sequences, specific tokenization and preprocessing techniques are usually required to transform these different formats of data into a unified form that can be processed by language models.

- **Code.** Program synthesis has been widely studied in the research community, especially the use of PLMs trained on code. However, it remains challenging for these PLMs (e.g., GPT-J) to generate high-quality and accurate programs. Recent studies have found that training LLMs on a vast code corpus can lead to a substantial improvement in the quality of the synthesized programs. The generated programs can successfully pass expert-designed unit-test cases or solve competitive programming questions. In general, two types of code corpora are commonly used for pre-training LLMs. The first source is from program question answering communities like Stack Exchange. The second source is from public software repositories such as GitHub, where code data (including comments and docstrings) are collected for utilization. Compared to natural language text, code is in the format of a programming language, corresponding to long-range dependencies and accurate execution logic. A recent study also speculates that training on code might be a source of complex reasoning abilities (e.g., chain-of-thought ability). Besides, it has been shown that formatting reasoning tasks into code can help LLMs generate more accurate results.

4.1.2 **Data Preprocessing**

After collecting a large amount of text data, it is essential to preprocess it for constructing the pre-training corpus, especially removing noisy, redundant, irrelevant, and potentially toxic data, which may largely affect the capacity and performance of LLMs. In this part, we review the detailed data preprocessing strategies to improve the quality of the collected data. A typical pipeline of preprocessing the pre-training data for LLMs has been illustrated in Figure 3.

**Quality Filtering.** To remove low-quality data from the collected corpus, existing work generally adopts two approaches: (1) classifier-based, and (2) heuristic-based. The former approach trains a selection classifier based on high-quality texts and leverages it to identify and filter out low-quality data. Typically, these methods train a binary classifier with well-curated data (e.g., Wikipedia pages) as positive instances and sample candidate data as negative instances, and predict the score that measures the quality of each data example. However, several studies also find that a classifier-based approach may result in the unintentional removal of high-quality texts in dialectal, colloquial, and sociolectal languages, which potentially leads to bias in the pre-training corpus and diminishes the corpus diversity. As the second approach, several studies employ heuristic-based approaches to eliminate low-quality texts through a set of well-designed rules, which can be summarized as follows:

- **Language filtering.** If a LLM would be mainly used in the tasks of certain languages, the text in other languages can be filtered.

- **Metric filtering.** Evaluation metrics about the generated texts, e.g., perplexity, can be employed to detect and remove unnatural sentences.

- **Statistic filtering.** Statistical features of a corpus, e.g., the punctuation distribution, symbol-to-word ratio, and sen-
tence length, can be utilized to measure the text quality and filter the low-quality data.

- **Keyword filtering.** Based on specific keyword set, the noisy or useless elements in the text, such as HTML tags, hyperlinks, boilerplates, and offensive words, can be identified and removed.

**De-duplication.** Existing work [138] has found that duplicate data in a corpus would reduce the diversity of language models, which may cause the training process unstable and thus affect the model performance. Therefore, it is necessary to de-duplicate the pre-training corpus. Specially, de-duplication can be performed at different granularities, including sentence-level, document-level, and dataset-level de-duplication. First, low-quality sentences that contain repeated words and phrases should be removed, as they may introduce repetitive patterns in language modeling [139]. At the document level, existing studies mostly rely on the overlap ratio of surface features (e.g., words and n-grams overlap) between documents to detect and remove duplicate documents containing similar contents [57, 59, 66, 140]. Furthermore, to avoid the dataset contamination problem, it is also crucial to prevent the overlap between the training and evaluation sets [56], by removing the possible duplicate texts from the training set. It has been shown that the three levels of de-duplication are useful to improve the training of LLMs [56, 141], which should be jointly used in practice.

**Privacy Redaction.** The majority of pre-training text data is obtained from web sources, including user-generated content involving sensitive or personal information, which may increase the risk of privacy breaches [142]. Thus, it is necessary to remove the personally identifiable information (PII) from the pre-training corpus. One direct and effective approach is to employ rule-based methods, such as keyword spotting, to detect and remove PII such as names, addresses, and phone numbers [109]. Furthermore, researchers also find that the vulnerability of LLMs under privacy attacks can be attributed to the presence of duplicate PII data in the pre-training corpus [143]. Therefore, de-duplication can also reduce privacy risks to some extent.

**Tokenization.** Tokenization is also a crucial step for data preprocessing. It aims to segment raw text into sequences of individual tokens, which are subsequently used as the inputs of LLMs. Although it is expedient to leverage an existing tokenizer (e.g., OPT [29] and GPT-3 [55] utilize the tokenizer of GPT-2 [28]), using a tokenizer specially designed for the pre-training corpus can be highly beneficial [66], especially for the corpus that consists of diverse domains, languages, and formats. Therefore, several recent LLMs train the customized tokenizers specifically for the pre-training corpus with SentencePiece [144]. The byte-level **Byte Pair Encoding (BPE)** algorithm [145] is utilized to ensure that the information after tokenization is lossless [58, 59]. Notably, normalization techniques in BPE, such as NFKC [146], may even degrade the tokenization performance [34, 59, 66].

### 4.1.3 Effect of Pre-training Data on LLMs

Unlike small-scale PLMs, it is usually infeasible to iterate the pre-training of LLMs multiple times, due to the huge demand for computational resources. Thus, it is particularly important to construct a well-prepared pre-training corpus before training a LLM. In this part, we discuss how the quality and distribution of the pre-training corpus potentially influence the performance of LLMs.

**Mixture of Sources.** As discussed before, pre-training data from different domains or scenarios has distinct linguistic characteristics or semantic knowledge. By pre-training on a mixture of text data from diverse sources, LLMs can acquire a broad scope of knowledge and may exhibit a strong generalization capacity. When mixing different sources, one needs to carefully set the distribution of pre-training data, since it is also likely to affect the performance of LLMs on downstream tasks [59]. Gopher [59] conducts the ablation experiment on data distribution to examine the impact of mixed sources on downstream tasks. Experimental results on the LAMBADA dataset [147] show that increasing the proportion of books data can improve the capacity of the model in capturing long-term dependencies from text, and increasing the proportion of the C4 dataset [21] leads to performance improvement on the C4 validation dataset [59]. While, as a side effect, training on excessive data about a certain domain would affect the generalization capability of LLMs on other domains [35, 59]. Therefore, it is suggested that researchers should carefully determine the proportion of data from different domains in the pre-training corpus, in order to develop LLMs that better meet their specific needs. The readers can refer to Figure 2 for a comparison of the data sources for different LLMs.

**Amount of Pre-training Data.** For pre-training an effective LLM, it is important to collect sufficient high-quality data that satisfies the data quantity demand of the LLM. Existing studies have found that with the increasing parameter scale in the LLM, more data is also required to train the
model \cite{34,57}: a similar scaling law as model size is also observed in data size, with respect to model performance. Chinchilla \cite{34} demonstrates that a number of existing LLMs suffer from sub-optimal training due to inadequate pre-training data. By conducting extensive experiments, it further shows that it is necessary to adopt equal scales of the model parameters and training tokens for a given compute budget. More recently, LLaMA \cite{57} shows that with more data and longer training, smaller models can also achieve good performance. Therefore, it is suggested that researchers should pay more attention to the amount of high-quality data for adequately training the model, especially when scaling the model parameters.

**Quality of Pre-training Data.** Existing work has shown that pre-training on the low-quality corpus, such as noisy, toxic, and duplicate data, may hurt the performance of models \cite{59,138,140,143}. For developing a well-performing LLM, it is crucial to consider not only the quantity but also the quality of the collected training data. Recent studies, such as T5 \cite{21}, GLaM \cite{93}, and Gopher \cite{59}, have investigated the influence of data quality on the performance of downstream tasks. By comparing the performances of models trained on the filtered and unfiltered corpus, they reach the same conclusion that pre-training LMs on cleaned data can improve the model performance. More specifically, the duplication of data may result in the “double descent” (referring to the phenomenon of performance initially deteriorating and subsequently improving) \cite{138,148}, or even overwhelm the training process \cite{138}. Besides, it has been shown that duplicate data degrades the ability of LLMs to copy from the context, which might further affect the generalization capacity of LLMs using in-context learning \cite{138}. Therefore, as suggested in existing work \cite{59,59,69}, it is essential to incorporate preprocessing methods on the pre-training corpus carefully (as illustrated in Section 4.1.2), to improve stability of the training process and avoid affecting the model performance.

### 4.2 Architecture

In this section, we review the architecture design of LLMs, \textit{i.e.}, mainstream architecture, pre-training objective, and detailed configuration. Table 3 presents the model cards of several representative LLMs with public details.

#### 4.2.1 Mainstream Architectures

Due to the excellent parallelizability and capacity, the Transformer architecture \cite{22} has become the de facto backbone to develop various LLMs, making it possible to scale language models to hundreds or thousands of billions of parameters. In general, the mainstream architectures of existing LLMs can be roughly categorized into three major types, namely encoder-decoder, casual decoder, and prefix decoder.

**Encoder-decoder Architecture.** The vanilla Transformer model is built on the encoder-decoder architecture \cite{22}, which consists of two stacks of Transformer blocks as the encoder and decoder, respectively. The encoder adopts stacked multi-head self-attention layers to encode the input sequence for generating its latent representations, while the decoder performs cross-attention on these representations and autoregressively generates the target sequence. Encoder-decoder PLMs (\textit{e.g.}, T5 \cite{21} and BART \cite{24}) have shown effectiveness on a variety of NLP tasks. So far, there are only a small number of LLMs that are built based on the encoder-decoder architecture, \textit{e.g.}, Flan-T5 \cite{51}. We leave a detailed discussion about the architecture selection in Section 4.2.3.

**Casual Decoder Architecture.** The casual decoder architecture incorporates the uni-directional attention mask, to guarantee that each input token can only attend to the past tokens and itself. The input and output tokens are processed in the same fashion through the decoder. As representative language models of this architecture, the GPT-series models \cite{26,55,110} are developed based on the casual-decoder architecture. In particular, GPT-3 \cite{55} has successfully demonstrated the effectiveness of this architecture, also showing an amazing in-context learning capability of LLMs. Interestingly, GPT-1 \cite{110} and GPT-2 \cite{25} do not exhibit such superior abilities as those in GPT-3, and it seems that scaling plays an important role in increasing the model capacity of this model architecture. So far, the casual decoders have been widely adopted as the architecture of LLMs by various existing LLMs, such as OPT \cite{79}, BLOOM \cite{66}, and Gopher \cite{59}. Note that both the casual decoder and prefix decoder discussed next belong to decoder-only architectures. While, when mentioning ”decoder-only architecture”, it mainly refers to the casual decoder architecture in existing literature, unless specified.

**Prefix Decoder Architecture.** The prefix decoder architecture (\textit{a.k.a.}, non-casual decoder \cite{149}) revises the masking mechanism of casual decoders, to enable performing bidirectional attention over the prefix tokens \cite{150} and unidirectional attention only on generated tokens. In this way, like the encoder-decoder architecture, the prefix decoders can bidirectionally encode the prefix sequence and autoregressively predict the output tokens one by one, where the same parameters are shared during encoding and decoding. Instead of pre-training from scratch, a practical suggestion is to continually train casual decoders and then convert them into prefix decoders for accelerating convergence \cite{29}. \textit{E.g.}, U-PaLM \cite{97} is derived from PaLM \cite{56}. Existing representative LLMs based on prefix decoders include GLM-130B \cite{50} and U-PaLM \cite{97}.

For the three types of architectures, we can also consider extending them via the mixture-of-experts (MoE) scaling, in which a subset of neural network weights for each input are sparsely activated, \textit{e.g.}, Switch Transformer \cite{25} and GLaM \cite{93}. It has been shown that substantial performance improvement can be observed by increasing either the number of experts or the total parameter size \cite{151}.

#### 4.2.2 Detailed Configuration

Since the launch of Transformer \cite{22}, various improvements have been proposed to enhance its training stability, performance, and computational efficiency. In this part, we will discuss the corresponding configurations for four major parts of the Transformer, including normalization, position embeddings, activation functions, attention, and bias.
**Normalization.** Training instability is a challenging issue for pre-training LLMs. To alleviate this problem, layer normalization (Layer Norm, LN) is widely employed in Transformer architectures. The position of LN is vital to the performance of LLMs. While the initial Transformer uses post-LN, most LLMs employ pre-LN for more stable training in spite of decreasing performance. Based on pre-LN, Sandwich-LN adds extra LN before the residual connections to avoid value explosion. However, it has been found that Sandwich-LN sometimes fails to stabilize the training of LLMs and may lead to the collapse of training. Recently, several advanced normalization techniques have been proposed as alternatives to LN. In Gopher and Chinchilla, RMS Norm is employed due to its superiority in training speed and performance. Compared with LN, DeepNorm has shown a better capability to ensure the stability in training, which has been adopted by GLM-130B with post normalization. In addition, adding an extra LN after the embedding layer can also stabilize the training of LLMs. However, it tends to incur a significant performance drop, which has been removed in several recent LLMs.

**Activation Functions.** To obtain good performance, activation functions also need to be properly set in feed-forward networks. In existing LLMs, GeLU activations is widely used. Besides, in the latest LLMs (e.g., PaLM and LaMDA), variants of GLU activation have also been utilized, especially the SwiGLU and GeGLU variants, which often achieve better performance than GeLU. However, compared with GeLU, they require extra parameters (about 50%) in the feed-forward networks.

**Position Embeddings.** Since the self-attention modules in Transformer are permutation equivariant, position embeddings are employed to inject absolute or relative position information for modeling sequences. There are two variants of absolute position embeddings in the vanilla Transformer, i.e., sinusoids and learned position embeddings, where the latter is commonly employed in LLMs. Unlike absolute position embeddings, relative positional encodings generate embeddings according to the offsets between keys and queries, so it can perform well on sequences longer than those it has seen during training, i.e., extrapolation. ALiBi biases attention scores using a penalty based on the distance between keys and queries. Empirical results have shown that it has better zero-shot generalization with a stronger extrapolation capacity than other position embeddings. Besides, by setting specific rotary matrices based on the absolute position, the scores between keys and queries in RoPE can be computed with relative position information, which is useful to model long sequences. As a result, RoPE has been widely adopted in several latest LLMs.

**Attention and Bias.** Beyond the full self-attention in the original Transformer, sparse attention with lower computation complexity is employed in GPT-3 (i.e., Factorized Attention). In order to effectively and efficiently model longer sequences, more attempts have been made by either introducing special attention patterns or considering GPU memory access (i.e., FlashAttention). Besides, following the original Transformer, most LLMs keep the biases in each dense kernel and Layer Norm. However, in PaLM and Galactica, biases are removed. It demonstrates that no biases can enhance training stability for LLMs.

To put all these discussions together, we summarize the suggestions from existing literature for detailed configuration. For stronger generalization and training stability, it is suggested to choose the pre RMS Norm for layer normalization, and SwiGLU or GeGLU as the activation function. While, Layer Norm may not be used immediately after embedding layers, which is likely to incur performance degradation. Besides, as for position embeddings, RoPE or ALiBi is a better choice since it performs better on long sequences.

### 4.2.3 Pre-training Tasks
Pre-training plays a key role that encodes general knowledge from large-scale corpus into the massive model parameters. For training LLMs, there are two commonly used pre-training tasks, namely language modeling and denoising autoencoding.

**Language Modeling.** The language modeling task (LM) is the most commonly used objective to pre-train decoder-only models. The objective is to maximize the probability of generating the next word in a sequence given the previous words. This task helps LLMs learn to predict the next word accurately, which is crucial for generating coherent and meaningful sequences. Pre-training on language modeling not only enhances the model’s ability to understand the context and meaning of words but also enables it to generalize well to various downstream tasks.

In this context, the model is trained to predict the next word from a sequence of words. The loss function is typically the cross-entropy loss, which measures the difference between the predicted distribution and the actual distribution. By minimizing this loss, the model learns to adjust its parameters to better fit the training data. This process is repeated for a large number of iterations, and the model’s weights are updated using an optimization algorithm, usually stochastic gradient descent. The model’s performance is evaluated using metrics such as perplexity, which measures the average log-probability of the model’s predictions. The lower the perplexity, the better the model’s performance.

The model is pre-trained on language modeling tasks using large-scale corpora such as Wikipedia and Books1/2. This training involves feeding the model with long sequences of text, and minimizing the cross-entropy loss between the predicted word distribution and the ground-truth word distribution. This helps the model learn to capture the statistical patterns and dependencies in the language, enabling it to generate coherent and contextually relevant text.

### Table 3: Model Cards of Several Selected LLMs

<table>
<thead>
<tr>
<th>Model</th>
<th>Category</th>
<th>Size</th>
<th>Normalization</th>
<th>PE</th>
<th>Activation</th>
<th>Bias</th>
<th>#L</th>
<th>#H</th>
<th>d_{model}</th>
<th>MCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT3</td>
<td>Casual decoder</td>
<td>127B</td>
<td>Pre Layer Norm</td>
<td>Learned</td>
<td>GeLU</td>
<td>✓</td>
<td>96</td>
<td>96</td>
<td>12288</td>
<td>2048</td>
</tr>
<tr>
<td>PaLM</td>
<td>Casual decoder</td>
<td>207B</td>
<td>Pre Layer Norm</td>
<td>Learned</td>
<td>GeLU</td>
<td>✓</td>
<td>64</td>
<td>128</td>
<td>16384</td>
<td>1024</td>
</tr>
<tr>
<td>OPT</td>
<td>Casual decoder</td>
<td>175B</td>
<td>Pre Layer Norm</td>
<td>Learned</td>
<td>RelLU</td>
<td>✓</td>
<td>96</td>
<td>96</td>
<td>12288</td>
<td>2048</td>
</tr>
<tr>
<td>BLOOM</td>
<td>Casual decoder</td>
<td>176B</td>
<td>Pre Layer Norm</td>
<td>ALiBi</td>
<td>GeLU</td>
<td>✓</td>
<td>70</td>
<td>112</td>
<td>14336</td>
<td>2048</td>
</tr>
<tr>
<td>MT-NLG</td>
<td>Casual decoder</td>
<td>530B</td>
<td>Pre Layer Norm</td>
<td>RoPE</td>
<td>SwiGLU</td>
<td>×</td>
<td>118</td>
<td>48</td>
<td>18432</td>
<td>2048</td>
</tr>
<tr>
<td>Gopher</td>
<td>Casual decoder</td>
<td>510B</td>
<td>Pre Layer Norm</td>
<td>Relative</td>
<td>-</td>
<td>-</td>
<td>105</td>
<td>128</td>
<td>20480</td>
<td>2048</td>
</tr>
<tr>
<td>Chinchilla</td>
<td>Casual decoder</td>
<td>280B</td>
<td>Pre Layer Norm</td>
<td>Relative</td>
<td>-</td>
<td>-</td>
<td>80</td>
<td>128</td>
<td>16384</td>
<td>1024</td>
</tr>
<tr>
<td>Galactica</td>
<td>Casual decoder</td>
<td>120B</td>
<td>Pre Layer Norm</td>
<td>Learned</td>
<td>GeLU</td>
<td>×</td>
<td>96</td>
<td>80</td>
<td>10240</td>
<td>2048</td>
</tr>
<tr>
<td>LaMDA</td>
<td>Casual decoder</td>
<td>137B</td>
<td>Relative</td>
<td>GeGLU</td>
<td>-</td>
<td>64</td>
<td>128</td>
<td>8192</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Jurassic1</td>
<td>Casual decoder</td>
<td>178B</td>
<td>Pre Layer Norm</td>
<td>Learned</td>
<td>GeLU</td>
<td>✓</td>
<td>76</td>
<td>96</td>
<td>13824</td>
<td>2048</td>
</tr>
<tr>
<td>LLaMA</td>
<td>Casual decoder</td>
<td>65B</td>
<td>Pre Layer Norm</td>
<td>RoPE</td>
<td>SwiGLU</td>
<td>✓</td>
<td>80</td>
<td>64</td>
<td>8192</td>
<td>2048</td>
</tr>
<tr>
<td>GLM-130B</td>
<td>Prefix decoder</td>
<td>130B</td>
<td>Post Deep Norm</td>
<td>RoPE</td>
<td>GeGLU</td>
<td>✓</td>
<td>70</td>
<td>96</td>
<td>12288</td>
<td>2048</td>
</tr>
<tr>
<td>T5</td>
<td>Encoder-decoder</td>
<td>11B</td>
<td>Pre Layer Norm</td>
<td>Relative</td>
<td>RelLU</td>
<td>×</td>
<td>24</td>
<td>128</td>
<td>1024</td>
<td>-</td>
</tr>
</tbody>
</table>
LLMs, e.g., GPT3 [55] and PaLM [56]. Given a sequence of tokens \( x = \{x_1, \ldots, x_n\} \), the LM task aims to autoregressively predict the target tokens \( x_i \) based on the preceding tokens \( x_{<i} \) in a sequence. A general training objective is to maximize the following likelihood:

\[
\mathcal{L}_{LM}(x) = \sum_{i=1}^{n} \log P(x_i|x_{<i}).
\]

(1)

Since most language tasks can be cast as the prediction problem based on the input, these decoder-only LLMs might be potentially advantageous to implicitly learn how to accomplish these tasks in a unified LM way. Some studies have also revealed that decoder-only LLMs can be naturally transferred to certain tasks by autoregressively predicting the next tokens [26, 55], without fine-tuning. An important variant of LM is the prefix language modeling task, which is designed for pre-training models with the prefix decoder architecture. The tokens within a randomly selected prefix would be not used in computing the loss of prefix language modeling. With the same amount of tokens seen during pre-training, prefix language modeling performs slightly worse than language modeling, since fewer tokens in the sequence are involved for model pre-training [28].

Denoising Autoencoding. Besides conventional LM, the denoising autoencoding task (DAE) has also been widely used to pre-train language models [24, 27]. The inputs \( x_{\hat{k}} \) for DAE task are corrupted text with randomly replaced spans. Then, the language models are trained to recover the replaced tokens \( \hat{x} \). Formally, the training objective of DAE is denoted as follows:

\[
\mathcal{L}_{DAE}(x) = \log P(\hat{x}|x_{\hat{k}}).
\]

(2)

However, the DAE task seems to be more complicated in implementation than LM task. As a result, it has not been widely used to pre-train large language models. Existing LLMs that take DAE as pre-training objectives include T5 [27] and GLM-130B [80]. These models are mainly trained to recover the replaced spans in an autoregressive way.

4.2.4 Summary and Discussion

The choice of architecture and pre-training tasks may incur different inductive biases for LLMs, which would lead to different model capacities. In this part, we summarize some important findings or discussions in the existing literature on this issue.

- By pre-training with the LM objective, it seems that casual decoder architecture can achieve a superior zero-shot and few-shot generalization capacity. Existing research has shown that without multi-task fine-tuning, the casual decoder has better zero-shot performance than other architectures [28]. The success of GPT-3 [55] has demonstrated that the large casual decoder model can be a good few-shot learner. In addition, instruction tuning and alignment tuning discussed in Section 5 have been proven to further enhance the capability of large casual decoder models [61, 62, 51].

- Scaling law has been widely observed in casual decoders. By scaling the model size, the dataset size, and the total computation, the performance of casual decoders can be substantially improved [50, 55]. Thus, it has become an important strategy to increase the model capacity of the casual decoder via scaling. However, more detailed investigation on encoder-decoder models is still lacking, and more efforts are needed to investigate the performance of encoder-decoder models at a large scale.

More research efforts about the discussions on architectures and pre-training objectives are in need to analyze how the choices of the architecture and pre-training tasks affect the capacity of LLMs, especially for encoder-decoder architectures. Besides the major architecture, the detailed configuration of LLM is also worth attention, which has been discussed in Section 4.2.2.

4.3 Model Training

In this part, we review the important settings, techniques, or tricks for training LLMs.

4.3.1 Optimization Setting

For parameter optimization of LLMs, we present the commonly used settings for batch training, learning rate, optimizer, and training stability.

Batch Training. For language model pre-training, existing work generally sets the batch size to a large number (e.g., 8,192 examples or 1.6M tokens) to improve the training stability and throughput. For LLMs such as GPT-3 and PaLM, they have introduced a new strategy that dynamically increases the batch size during training, ultimately reaching a million scale. Specifically, the batch size of GPT-3 is gradually increasing from 32K to 3.2M tokens. Empirical results have demonstrated that the dynamic schedule of batch size can effectively stabilize the training process of LLMs [56].

Learning Rate. Existing LLMs usually adopt a similar learning rate schedule with the warm-up and decay strategies during pre-training. Specifically, in the initial 0.1% to 0.5% of the training steps, a linear warm-up schedule is employed for gradually increasing the learning rate to the maximum value that ranges from approximately 5 \( \times \) 10^{-5} to 1 \( \times \) 10^{-4} (e.g., 6 \( \times \) 10^{-5} for GPT-3). Then, a cosine decay strategy is adopted in the subsequent steps, gradually reducing the learning rate to approximately 10% of its maximum value, until the convergence of the training loss.

Optimizer. The Adam optimizer [168] and AdamW optimizer [169] are widely utilized for training LLMs (e.g., GPT-3), which are based on adaptive estimates of lower-order moments for first-order gradient-based optimization. Commonly, its hyper-parameters are set as follows: \( \beta_1 = 0.9 \), \( \beta_2 = 0.95 \) and \( \epsilon = 10^{-8} \). Meanwhile, the AdaFactor optimizer [170] has also been utilized in training LLMs (e.g., PaLM and T5), which is a variant of the Adam optimizer specially designed for conserving GPU memory during training. The hyper-parameters of the AdaFactor optimizer are set as: \( \beta_1 = 0.9 \) and \( \beta_2 = 1.0 - k^{-0.8} \), where \( k \) denotes the number of training steps.

Stabilizing the Training. During the pre-training of LLMs, it often suffers from the training instability issue, which may cause the model collapse. To address this issue, weight decay and gradient clipping have been widely utilized,
where existing studies [55, 66, 79, 80, 90] commonly set the threshold of gradient clipping to 1.0 and weight decay rate to 0.1. However, with the scaling of LLMs, the training loss spike is also more likely to occur, leading to unstable training. To mitigate this problem, PaLM [56] and OPT [79] use a simple strategy that restarts the training process from an earlier checkpoint before the occurrence of the spike and skips over the data that may have caused the problem. Further, GLM [80] finds that the abnormal gradients of the embedding layer usually lead to spikes, and proposes to shrink the embedding layer gradients to alleviate it.

4.3.2 Scalable Training Techniques

As the model and data sizes increase, it has become challenging to efficiently train LLMs under a limited computational resource. Especially, two primary technical issues are required to be resolved, i.e., increasing training throughput and loading larger models into GPU memory. In this part, we review several widely used approaches in existing work to address the above two challenges, namely 3D parallelism, and mixed precision training [174], and also give general suggestions about how to utilize them for training.

3D Parallelism. 3D parallelism is actually a combination of three commonly used parallel training techniques, namely data parallelism, pipeline parallelism [171, 172], and tensor parallelism [65]. We next introduce the three parallel training techniques.

• Data parallelism. Data parallelism is one of the most fundamental approaches to improving the training throughput. It replicates the model parameters and optimizer states across multiple GPUs and then distributes the whole training corpus into these GPUs. In this way, each GPU only needs to process the assigned data for it, and performs the forward and backward propagation to obtain the gradients. The computed gradients on different GPUs will be further aggregated to obtain the gradients of the entire batch for updating the models in all GPUs. In this way, as the calculations of gradients are independently performed on different GPUs, the data parallelism mechanism is highly scalable, enabling the way that increases the number of GPUs to improve training throughput. Furthermore, this technique is simple in implementation, and most of existing popular deep learning libraries have already implemented data parallelism, such as TensorFlow and PyTorch.

• Pipeline parallelism. Pipeline parallelism aims to distribute the different layers of a LLM into multiple GPUs. Especially, in the case of a Transformer model, pipeline parallelism loads consecutive layers onto the same GPU, to reduce the cost of transmitting the computed hidden states or gradients between GPUs. However, a naive implementation of pipeline parallelism may result in a lower GPU utilization rate as each GPU has to wait for the previous one to complete the computation, leading to the unnecessary cost of bubbles overhead [171]. To reduce these bubbles in pipeline parallelism, GPipe [171] and PipeDream [172] propose the techniques of padding multiple batches of data and asynchronous gradient update to improve the pipeline efficiency.

• Tensor parallelism. Tensor parallelism is also a commonly used technique that aims to decompose the LLM for multi-GPU loading. Unlike pipeline parallelism, tensor parallelism focuses on decomposing the tensors (the parameter matrices) of LLMs. For a matrix multiplication operation \( Y = X A \) in the LLM, the parameter matrix \( A \) can be split into two submatrices, \( A_1 \) and \( A_2 \), by column, which can be expressed as \( Y = [X A_1, X A_2] \). By placing matrices \( A_1 \) and \( A_2 \) on different GPUs, the matrix multiplication operation would be invoked at two GPUs in parallel, and the final result can be obtained by combining the outputs from the two GPUs through across-GPU communication. Currently, tensor parallelism has been supported in several open-source libraries, e.g., Megatron-LM [65], and can be extended to higher-dimensional tensors. Besides, Colossal-AI has also implemented tensor parallelism for higher-dimensional tensors [175, 177] and proposed sequence parallelism [178] especially for sequence data, which can further decompose the attention operation of the Transformer model.

ZeRO. ZeRO [173] technique, proposed by the Deep-
Mixed Precision Training. In previous PLMs (e.g., BERT [23]), 32-bit floating-point numbers, also known as FP32, have been predominantly used for pre-training. In recent years, to pre-train extremely large language models, FP32 have been increasingly used for pre-training. In practice, one can further improve the training efficiency and reduce the time and space costs of LLMs during the inference stage [183]. With some loss in model performance, quantized language models have smaller model sizes and can achieve and faster inference speed [80, 184, 185]. For model quantization, a popular choice is INT8-quantization [184]. Further, some research work attempts to develop more aggressive INT4-quantization methods [80]. Among these open-source LLMs, BLOOM [12], GPT-3 [13], and GLM [14] have released the corresponding quantized model copies.

5 Adaptation Tuning of LLMs

After pre-training, LLMs can acquire the general abilities for solving various tasks. However, increasing studies have shown that LLM’s abilities can be further adapted according to specific goals. In this section, we introduce two major approaches to adapting pre-trained LLMs, namely instruction tuning and alignment tuning. The former approach mainly aims to enhance (or unlock) the abilities of LLMs, while the latter approach aims to align the behaviors of LLMs with human values or preferences. In what follows, we will introduce the two approaches in detail.

<table>
<thead>
<tr>
<th>Collections</th>
<th>Time</th>
<th>#Task types</th>
<th>#Tasks</th>
<th>#Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nat. Inst. [156]</td>
<td>Apr-2021</td>
<td>6</td>
<td>61</td>
<td>193K</td>
</tr>
<tr>
<td>CrossFit [157]</td>
<td>Apr-2021</td>
<td>13</td>
<td>160</td>
<td>7.1M</td>
</tr>
<tr>
<td>P3 [188]</td>
<td>Oct-2021</td>
<td>13</td>
<td>267</td>
<td>12.1M</td>
</tr>
<tr>
<td>ExMix [189]</td>
<td>Nov-2021</td>
<td>11</td>
<td>107</td>
<td>18M</td>
</tr>
<tr>
<td>UnifiedSKG [190]</td>
<td>Jan-2022</td>
<td>6</td>
<td>21</td>
<td>812K</td>
</tr>
<tr>
<td>Super Nat. Inst. [27]</td>
<td>Apr-2022</td>
<td>76</td>
<td>1616</td>
<td>5M</td>
</tr>
<tr>
<td>MVPCorpus [191]</td>
<td>Jan-2023</td>
<td>11</td>
<td>77</td>
<td>41M</td>
</tr>
<tr>
<td>xP3 [18]</td>
<td>Nov-2022</td>
<td>17</td>
<td>85</td>
<td>81M</td>
</tr>
</tbody>
</table>

5.1 Instruction Tuning

In essence, instruction tuning is the approach to fine-tuning pre-trained LLMs on a collection of formatted instances in the form of natural language [62], which is highly related to supervised fine-tuning [61] and multi-task prompted training [28]. In order to perform instruction tuning, we first need to collect or construct instruction-formatted instances. Then, we employ these formatted instances to fine-tune LLMs in a supervised learning way (e.g., training with the sequence-to-sequence loss). After instruction tuning, LLMs can demonstrate superior abilities to generalize to unseen tasks [28, 62, 81], even in a multilingual setting [82].
A recent survey [192] presents a systematic overview of the research on instruction tuning. In comparison to that, we mainly focus on the effect of instruction tuning on LLMs and provide detailed guidelines or strategies for instance collection and tuning. Besides, we also discuss the use of instruction tuning for satisfying the real needs of users, which has been widely applied in existing LLMs, e.g., InstructGPT [61] and GPT-4 [46].

### 5.1.1 Formatted Instance Construction

Generally, an instruction-formatted instance consists of a task description (called an instruction), an input-output pair, and a small number of demonstrations (optional). As important public resources, existing studies have released a large number of labeled data formatted in natural language (see the list of available resources in Table 5). Next, we introduce two major methods for constructing formatted instances (see an illustration in Figure 4) and then discuss several key factors for instance construction.

#### Formatting Existing Datasets

Before instruction tuning was proposed, several early studies [189, 191, 192, 194] collected the instances from a diverse range of tasks (e.g., text summarization, text classification, and translation) to create supervised multi-task training datasets. As a major source of instruction tuning instances, it is convenient to format these multi-task training datasets with natural language task descriptions. Specifically, recent work [28, 61, 62, 77] augments the labeled datasets with human-written task descriptions, which instructs LLMs to understand the tasks by explaining the task goal. For example, in Figure 4(b), a task description “Please answer this question” is added for each example in the question-answering task. After instruction tuning, LLMs can generalize well to other unseen tasks by following their task descriptions [28, 62, 81]. In particular, it has been shown that instructions are the crucial factor in task generalization ability for LLMs [62]. By fine-tuning the model on labeled datasets with the task descriptions removed, it results in a dramatic drop in model performance. To better generate labeled instances for instruction tuning, a crowd-sourcing platform, PromptSource [188] has been proposed to effectively create, share, and verify the task descriptions for different datasets. To enrich the training instances, several studies [28, 191, 195] also try to invert the input-output pairs of existing instances with specially designed task descriptions for instruction tuning. For instance, given a question-answer pair, we can create a new instance by predicting the question-conditioned answer and some task description (e.g., “Please generate a question based on the answer.”). Besides, some work [196] also leverages heuristic task templates to convert massive unlabeled texts into labeled instances.

#### Formatting Human Needs

Despite that a large number of training instances have been formatted with instructions, they mainly come from public NLP datasets, either lacking instruction diversity or mismatching with real human needs [61]. To overcome this issue, InstructGPT [61] proposes to take the queries that real users have submitted to the OpenAI API as the task descriptions. User queries are expressed in natural languages, which are particularly suitable for eliciting the ability of instruction following for LLMs. Additionally, to enrich the task diversity, human labelers are also asked to compose the instructions for real-life tasks, including open-ended generation, open question answering, brainstorming, and chatting. Then, they let another group of labelers directly answer these instructions as the output. Finally, they pair one instruction (i.e., the collected user query) and the expected output (i.e., the human-written answer) as a training instance. Note that InstructGPT also employs these real-world tasks formatted in natural language for alignment tuning (discussed in Section 5.2). Further, GPT-4 [46] has designed potentially high-risk instructions and guided the model to reject these instructions through supervised fine-tuning for safety concerns. Besides, to reduce the burden of human annotation, several semi-automated approaches [197, 199] have also been proposed for constructing instances by feeding existing instances into LLMs to generate diverse task descriptions and instances.

#### Key Factors for Instance Construction

The quality of
instruction instances has an important impact on the performance of the model. Here, we discuss some essential factors for instance construction.

- **Scaling the instructions.** It has been widely shown that scaling the number of tasks can largely enhance the generalization ability of LLMs [28, 62, 77]. When increasing the parameter scale, the performance initially shows a continuous growth pattern with the number of tasks, while the gain becomes negligible when it reaches a certain level [77, 81]. A plausible speculation is that a certain number of representative tasks can provide relatively sufficient knowledge and adding more tasks may not bring additional gains [81]. Besides, it is also beneficial to enhance the diversity of the task descriptions in several aspects, such as length, structure, and creativity [28]. As for the number of instances per task, it has been found that a small number of instances can usually saturate the generalization performance of the model [62, 81]. Whereas, increasing the number of instances for some tasks to a large number (e.g., hundreds of thousands) could potentially result in the overfitting issue and impair the model performance [77].

- **Formatting design.** As an important factor, the design of natural language format also highly impacts the generalization performance of LLMs [77]. Typically, we can add task descriptions and optional demonstrations to the input-output pairs of existing datasets, where the task description is the most key part for LLMs to understand the task [77]. Further, it can lead to substantial improvements by using an appropriate number of exemplars as demonstrations [81], which also alleviates the model sensitivity to instruction engineering [62, 81]. However, incorporating other components (e.g., things to avoid, reasons, and suggestions) into instructions may have a negligible or even adverse effect on the performance of LLMs [77, 186]. Recently, to elicit the step-by-step reasoning ability of LLMs, some work [81] proposes to include chain-of-thought (CoT) examples for some reasoning datasets, such as arithmetic reasoning. It has been shown that fine-tuning LLMs with both CoT and non-CoT examples can lead to a good performance across various reasoning tasks, including those that require multi-hop reasoning ability (e.g., commonsense question answering and arithmetic reasoning) as well as those without the need for such a reasoning way (e.g., sentiment analysis and extractive question answering) [81, 83].

To summarize, it seems that the diversity of instructions is more important than the number of instances since the well-performing InstructGPT [61] and Alpaca [199] utilize fewer but more diverse instructions (or instances) than the Flan-series LLMs [62, 81]. Further, it is more useful to invite labelers to compose human-need tasks than using dataset-specific tasks. While, it still lacks the guidelines to annotate human-need instances, making the task composition somehow heuristic. To reduce human efforts, we can either reuse existing formatted datasets (Table 5) or automatically construct the instructions using existing LLMs [197].

### 5.1.2 Instruction Tuning Strategies

Unlike pre-training, instruction tuning is often more efficient since only a moderate number of instances are used for training. Although instruction tuning can be considered as a supervised training process, its optimization is different from pre-training in several aspects [81], such as the training objective (i.e., sequence-to-sequence loss) and optimization configuration (e.g., smaller batch size and learning rate), which require special attention in practice. In addition to these optimization configurations, there are also two important aspects to consider for instruction tuning:

**Balancing the Data Distribution.** Since instruction tuning involves a mixture of different tasks, it is important to balance the proportion of different tasks during fine-tuning. A widely used method is the examples-proportional mixing strategy [71], i.e., combining all the datasets and sampling each instance equally from the mixed datasets. Furthermore, increasing the sampling ratio of high-quality collections (e.g., FLAN [62] and P3 [188]) can generally lead to performance improvement according to recent findings [81, 83]. While, it is common to set a maximum cap to control the maximum number of examples that a dataset can contain during instruction tuning [71], which is set to prevent larger datasets from overwhelming the entire distribution [71, 83]. In practice, the maximum cap is typically set to several thousands or tens of thousands according to different datasets [62, 81].

**Combining Instruction Tuning and Pre-Training.** To make the tuning process more effective and stable, OPT-IML [83] incorporates pre-training data during instruction tuning, which can be regarded as regularization for model tuning. Further, instead of using a separate two-stage process (pre-training then instruction tuning), some studies attempt to train a model from scratch with a mixture of pre-training data (i.e., plain texts) and instruction tuning data (i.e., formatted datasets) using multi-task learning [71, 189]. Specifically, GLM-130B [80] and Galactica [35] integrate instruction-formatted datasets as a small proportion of the pre-training corpora to pre-train LLMs, which potentially achieves the advantages of pre-training and instruction tuning at the same time.

### 5.1.3 The Effect of Instruction Tuning

In this part, we discuss the effect of instruction tuning on LLMs in two major aspects.

**Performance Improvement.** Despite being tuned on a moderate number of instances, instruction tuning has become an important way to improve or unlock the abilities of LLMs [81]. Recent studies have experimented with language models in multiple scales (ranging from 77M to 540B), showing that the models of different scales can all benefit from instruction tuning [81, 195], yielding improved performance as the parameter scale increases [82]. Further, smaller models with instruction tuning can even perform better than larger models without fine-tuning [28, 81]. Besides the model scale, instruction tuning demonstrates consistent improvements in various model architectures, pre-training objectives, and model adaptation methods [81]. In practice, instruction tuning offers a general approach to enhancing the abilities of existing language models [81] (including small-sized PLMs). Besides, it is also more efficient than pre-training, since the amount of labeled instruction data required by LLMs is much smaller than pre-training data.
Task Generalization. Instruction tuning encourages the model to understand natural language instructions for task completion. It endows LLMs with the ability (often considered as an emergent ability) to follow human instructions [31] to perform specific tasks without demonstrations, even on unseen tasks [83]. A large number of studies have confirmed the effectiveness of instruction tuning to achieve superior performance on both seen and unseen tasks [83, 152]. Besides, instruction tuning has been shown to be useful in alleviating several weaknesses of LLMs (e.g., repetitive generation or complementing the input without accomplishing a certain task) [61, 81], leading to a superior capacity to solve real-world tasks for LLMs. Furthermore, LLMs trained with instruction tuning can generalize to related tasks across languages. For example, BLOOM-P3 [82] is fine-tuned based on BLOOM [56] using English-only task collection P3 [188]. Interestingly, BLOOM-P3 can achieve a more than 50% improvement in multilingual sentence completion tasks compared to BLOOM, which shows that instruction tuning can help LLMs acquire general task skills from English-only datasets and transfer such skills into other languages [82]. In addition, it has been found that using English-only instructions can produce satisfactory results on multilingual tasks [82], which helps reduce the effort of instruction engineering for a specific language.

5.2 Alignment Tuning

This part first presents the background of alignment with its definition and criteria, then focuses on the collection of human feedback data for aligning LLMs, and finally discusses the key technique of reinforcement learning from human feedback for alignment tuning.

5.2.1 Background and Criteria for Alignment

Background. LLMs have shown remarkable capabilities in a wide range of NLP tasks [55, 56, 62, 79]. However, these models may sometimes exhibit unintended behaviors, e.g., fabricating false information, pursuing inaccurate objectives, and producing harmful, misleading, and biased expressions [61, 201]. For LLMs, the language modeling objective pre-trains the model parameters by word prediction while lacking the consideration of human values or preferences. To avert these unexpected behaviors, human alignment has been proposed to make LLMs act in line with preferences. To avert these unexpected behaviors, human alignment has been proposed to make LLMs act in line with preferences (e.g., helpfulness, honesty, and harmlessness). It has been shown that alignment might harm the general abilities of LLMs to some extent, which is called alignment tax in related literature [61, 201, 202].

Alignment Criteria. Recently, there is increasing attention on developing multifarious criteria to regulate the behaviors of LLMs. Here, we take three representative alignment criteria (i.e., helpful, honest, and harmless) as examples for discussion, which have been widely adopted in existing literature [61, 201, 202]. Besides, there are also other alignment criteria for LLMs from different perspectives including behavior, intent, incentive, and inner aspects [201], which are essentially similar (or at least with similar alignment techniques) to the above three criteria. It is also feasible to modify the three criteria according to specific needs, e.g., substituting honesty with correctness [96] or focusing on some specified criteria [202]. Next, we give brief explanations about the three representative alignment criteria:

- **Helpfulness.** To be helpful, the LLM should demonstrate a clear attempt to assist users in solving their tasks or answering questions in a concise and efficient manner as possible. At a higher level, when further clarification is needed, the LLM should demonstrate the capability of eliciting additional relevant information through pertinent inquiries and exhibit suitable levels of sensitivity, perceptiveness, and prudence [201]. Realizing the alignment of helpful behavior is challenging for LLMs since it is difficult to precisely define and measure the intention of users [203].

- **Honesty.** At a basic level, a LLM aligned to be honest should present accurate content to users instead of fabricating information. Additionally, it is crucial for the LLM to convey appropriate degrees of uncertainty in its output, in order to avoid any form of deception or misrepresentation of information. This requires the model to know about its capabilities and levels of knowledge (e.g., “know unknowns”). According to the discussion in [201], honesty is a more objective criterion compared to helpfulness and harmlessness, hence honesty alignment could potentially be developed with less reliance on human efforts.

- **Harmlessness.** To be harmless, it requires that the language produced by the model should not be offensive or discriminatory. To the best of its abilities, the model should be capable of detecting covert endeavors aimed at soliciting requests for malicious purposes. Ideally, when the model was induced to conduct a dangerous action (e.g., committing a crime), the LLM should politely refuse. Nonetheless, what behaviors are deemed harmful and to what extent vary amongst individuals or societies [201] heavily depend on who is using the LLM, the type of the posed question, and the context (e.g., time) at which the LLM is being used.

As we can see, these criteria are quite subjective, and are developed based on human cognition. Thus, it is difficult to directly formulate them as optimization objectives for LLMs. In existing work, there are many ways to fulfill these criteria when aligning LLMs. A promising technique is red teaming [203, 204], which involves using manual or automated means to probe LLMs in an adversarial way to generate harmful outputs and then updates LLMs to prevent such outputs.

5.2.2 Collecting Human Feedback

During the pre-training stage, LLMs are trained using the language modeling objective on a large-scale corpus. However, it cannot take into account the subjective and qualitative evaluations of LLM outputs by humans (called human feedback in this survey). High-quality human feedback is extremely important for aligning LLMs with human preferences and values. In this part, we discuss how to select a team of human labelers for feedback data collection.

Human Labeler Selection. In existing work, the dominant method for generating human feedback data is human annotation [61, 96, 205]. This highlights the critical role
of selecting appropriate human labelers. To provide high-quality feedback, human labelers are supposed to have a qualified level of education and excellent proficiency in English. For example, Sparrow [95] requires human labelers to be UK-based native English speakers who have obtained at least an undergraduate-level educational qualification. Further, in [202], half of the human labelers were recruited from the US-based Amazon Mechanical Turk workforce with a master’s qualification. Even then, several studies [205, 206] have found that there still exists a mismatch between the intentions of researchers and human labelers, which may lead to low-quality human feedback and cause LLMs to produce unexpected output. To address this issue, InstructGPT [61] further conducts a screening process to filter labelers by assessing the agreement between human labelers and researchers. Specifically, researchers first label a small amount of data and then measure the agreement between themselves and human labelers. The labelers with the highest agreement will be selected to proceed with the subsequent annotation work. In some other work, [202, 207], “super raters” are used to ensure the high quality of human feedback. They evaluate the performance of human labelers and select a group of well-performing human labelers (e.g., high agreement) as super raters. The super raters will be given priority to collaborate with the researchers in the subsequent study. When human labelers annotate the output of LLMs, it is helpful to specify detailed instructions and provide instant guidance for human labelers [206], which can further regulate the annotation of labelers.

**Human Feedback Collection.** In existing work, there are mainly three kinds of approaches to collecting feedback and preference data from human labelers.

- **Ranking-based collection.** In early work [205, 208], human labelers often evaluate model-generated outputs in a coarse-grained manner (i.e., only selecting the best) without taking into account more fine-grained alignment criteria. Nonetheless, different labelers may hold diverse opinions on the selection of the best candidate output, and this method disregards the unselected samples, which may lead to inaccurate or incomplete human feedback. To address this issue, subsequent studies [61] introduce the Elo rating system to derive the preference ranking by comparing candidate outputs. The ranking of outputs serves as the training signal that guides the model to prefer certain outputs over others, thus inducing outputs that are more reliable and safer.

- **Question-based collection.** Further, human labelers can provide more detailed feedback by answering certain questions designed by researchers [70], covering the alignment criteria as well as additional constraints for LLMs. Specially, in WebGPT [70], to assist the model in filtering and utilizing relevant information from retrieved documents, human labelers are required to answer questions with multiple options about whether the retrieved documents are useful for answering the given input.

- **Rule-based collection.** Besides, many studies develop rule-based methods to provide more detailed human feedback. As a typical case, Sparrow [96] not only selects the response that labelers consider the best but also uses a series of rules to test whether model-generated responses meet the alignment criteria of being helpful, correct, and harmless. In this way, two kinds of human feedback data can be obtained: (1) the response preference feedback is obtained by comparing the quality of model-generated output in pairs, and (2) the rule violation feedback is obtained by collecting the assessment from human labelers (i.e., a score indicating to what extent the generated output has violated the rules). Furthermore, GPT-4 [46] utilizes a set of zero-shot classifiers (based on GPT-4 itself) as rule-based reward models, which can automatically determine whether the model-generated outputs violate a set of human-written rules.

In the following, we focus on a well-known technique, reinforcement learning from human feedback (RLHF), which has been widely used in the recent powerful LLMs such as ChatGPT. As discussed below, the alignment criteria introduced in Section 5.2.1 can be fulfilled by learning from human feedback on the responses of LLMs to users’ queries.

5.2.3 Reinforcement Learning from Human Feedback

To align LLMs with human values, reinforcement learning from human feedback (RLHF) [67, 205] has been proposed to fine-tune LLMs with the collected human feedback data, which is useful to improve the alignment criteria (e.g., helpfulness, honesty, and harmlessness). RLHF employs reinforcement learning (RL) algorithms (e.g., Proximal Policy Optimization (PPO) [209]) to adapt LLMs to human feedback by learning a reward model. Such an approach incorporates humans in the training loop for developing well-aligned LLMs, as exemplified by InstructGPT [61].

**RLHF System.** The RLHF system mainly comprises three key components: a pre-trained LM to be aligned, a reward model learning from human feedback, and a RL algorithm training the LM. Specifically, the pre-trained LM is typically a generative model that is initialized with existing pre-trained LM parameters. For example, OpenAI uses 175B GPT-3 for its first popular RLHF model, InstructGPT [61], and DeepMind uses the 280 billion parameter model Gopher [59] for its GopherCite model [207]. Further, the reward model (RM) provides (learned) guidance signals that reflect human preferences for the text generated by the LM, usually in the form of a scalar value. The reward model can take on two forms: a fine-tuned LM or a LM trained de novo using human preference data. Existing work typically employs reward models having a parameter scale different from that of the aligned LM [61, 207]. For example, OpenAI uses 6B GPT-3 and DeepMind uses 7B Gopher as the reward model, respectively. Finally, to optimize the pre-trained LM using the signal from the reward model, a specific RL algorithm is designed for large-scale model tuning. Specifically, Proximal Policy Optimization (PPO) [209] is a widely used RL algorithm for alignment in existing work [61, 86, 207].

**Key Steps for RLHF.** Figure illustrates the overall methodology of RLHF, which follows a three-step process in existing work [61, 206] as introduced below.

- **Supervised fine-tuning.** To make the LM initially perform desired behaviors, it usually needs to collect a supervised dataset containing input prompts (instruction) and desired outputs for fine-tuning the LM. These prompts and outputs can be written by human labelers for some specific tasks while ensuring the diversity of tasks. For example, InstructGPT [61] asks human labelers to compose prompts (e.g.,
“List five ideas for how to regain enthusiasm for my career”
and desired outputs for several generative tasks such as open QA, brainstorming, chatting, and rewriting. Note that the first step is not necessarily used and can be optional in specific settings or scenarios.

- **Reward model training.** The second step is to train the RM using human feedback data. Specifically, we employ the LM to generate a certain number of output texts using sampled prompts (from either the supervised dataset or the human-generated prompt) as input. We then invite human labelers to annotate the preference for these pairs. The annotation process can be conducted in multiple forms, and a common approach is to annotate by ranking the generated candidate texts, which can reduce the inconsistency among annotators. Then, the RM is trained to predict the human-preferred output. In InstructGPT, labelers rank model-generated outputs from best to worst, and the RM (i.e., 6B GPT-3) is trained to predict the ranking.

- **RL fine-tuning.** At this step, aligning (i.e., fine-tuning) the LM is formalized as an RL problem. In this setting, the pre-trained LM acts as the policy that takes as input a prompt and returns an output text, the action space of this policy is all the terms in the vocabulary of the LM, the state is characterized by the currently generated token sequence, and the reward is provided by the RM. To avoid deviating significantly from the initial (before tuning) LM, a penalty term is commonly incorporated into the reward function. For example, InstructGPT optimizes the LM against the RM using the PPO algorithm. For each input prompt, InstructGPT calculates the KL divergence between the generated results from the current LM and the initial LM as the penalty. It is noted that the second and final steps can be iterated in multiple turns for better aligning LLMs.

## 6 Utilization

After pre-training or adaptation tuning, a major approach to using LLMs is to design suitable prompting strategies for solving various tasks. A typical prompting method is **in-context learning** [50, 55], which formulates the task description and/or demonstrations in the form of natural language text. In addition, **chain-of-thought prompting** [53] can be employed to enhance in-context learning by involving a series of intermediate reasoning steps into prompts. Next, we will elaborate on the details of the two techniques.

### 6.1 In-Context Learning

As a special prompting form, in-context learning (ICL) is first proposed along with GPT-3 [55], which has become a typical approach to utilizing LLMs.

#### 6.1.1 Prompting Formulation

As stated in [55], ICL uses a formatted natural language prompt, consisting of the task description and/or a few task examples as demonstrations. Figure 6 presents the illustration of ICL. First, starting with a task description, a few examples are selected from the task dataset as demonstrations. Then, they are combined in a specific order to form natural language prompts with specially designed templates. Finally, the test instance is appended to the demonstration as the input for LLMs to generate the output. Based on task demonstrations, LLMs can recognize and perform a new task without explicit gradient update.

Formally, let $D_k = \{f(x_1, y_1), \ldots, f(x_k, y_k)\}$ represent a set of demonstrations with $k$ examples, where $f(x_k, y_k)$ is the prompt function that transforms the $k$-th task example into natural language prompts. Given the task description $I$, demonstration $D_k$, and a new input query $x_{k+1}$, the prediction of the output $\hat{y}_{k+1}$ generated from LLMs can be formulated as follow:

$$
\text{LLM}(I, f(x_1, y_1), \ldots, f(x_k, y_k), f(x_{k+1}, \_ \_ \_) \rightarrow \hat{y}_{k+1}.
$$

where the actual answer $y_{k+1}$ is left as a blank to be predicted by the LLM. Since the performance of ICL heavily relies on demonstrations, it is an important issue to properly design them in the prompts. According to the construction process in Equation (3), we focus on three major aspects in formulating demonstrations in the prompts, including how to select examples that make up demonstrations, format each example into the prompt with the function $f(\cdot)$, and arrange demonstrations in a reasonable order.

A comprehensive review of ICL has been presented in the survey paper [50], and we suggest the readers refer to it for a more general, detailed discussion on this topic. Compared with this survey, we especially focus on the discussion of applying ICL to LLMs in two major aspects, i.e., demonstration design and the underlying mechanism of LLMs. Besides, ICL also has a close connection with instruction tuning (discussed in Section 5.1) in that both utilize natural language to format the task or instances.

16. When ICL was introduced in the GPT-3’s paper [55], it was originally defined to be a combination of the task description and demonstration examples, wherein either component is dispensable. Following this definition, when a LLM is required to solve an unseen task by using only task descriptions, it can be also considered to perform ICL for task solving, whereas the ICL ability can be enhanced by instruction tuning.
However, instruction tuning needs to fine-tune LLMs for adaptation, while ICL only prompts LLMs for utilization. Furthermore, instruction tuning can enhance the ICL ability of LLMs to perform target tasks, especially in the zero-shot setting (only using task descriptions) [81].

### 6.1.2 Demonstration Design

Several studies have shown that the effectiveness of ICL is highly affected by the design of demonstrations [210, 212]. Following the discussion in Section 6.1.1, we will introduce the demonstration design of ICL from three major aspects, i.e., demonstration selection, format, and order.

**Demonstration Selection.** The performance of ICL tends to have a large variance with different demonstration examples [213], so it is important to select a subset of examples that can effectively leverage the ICL capability of LLMs. There are two main demonstration selection approaches, namely heuristic and LLM-based approaches:

- **Heuristic approaches.** Due to the simplicity and low costs, existing work widely adopts heuristic methods to select demonstrations. Several studies employ a $k$-NN based retriever to select examples that are semantically relevant to the query [213, 214]. However, they perform the selection individually for each example, rather than evaluating the example set as a whole. To resolve this issue, diversity-based selection strategies are proposed to choose the most representative set of examples for specific tasks [215, 216]. Furthermore, in [217], both relevance and diversity are taken into consideration when selecting demonstrations.

- **LLM-based approaches.** Another line of work selects demonstrations by making use of LLMs. For example, LLMs can be utilized to directly measure the informativeness of each example according to the performance gain after adding the example [218]. Besides, EPR [219] proposes a two-stage retrieval approach that first recalls similar examples with an unsupervised method (e.g., BM25) and then ranks them using a dense retriever (trained with positive and negative examples labeled by LLMs). As an alternative approach, the task of demonstration selection can be formulated into a RL problem, where LLMs serve as the reward function to provide feedback for training the policy model [220]. Since LLMs perform well for text annotation [221], some recent studies employ LLM itself as the demonstration generator without human intervention [222, 223].

To summarize, as discussed in [224], the selected demonstration examples in ICL should contain sufficient information about the task to solve as well as be relevant to the test query, for the above two selection approaches.

**Demonstration Format.** After selecting task examples, the next step is to integrate and format them into a natural language description, several demonstrations, and a test query. While CoT prompting involves a series of intermediate reasoning steps in prompts.

**Demonstration Order.** LLMs are shown to sometimes suffer from overfitting to previously solved ones. As two representative methods, Auto-CoT [225] leverages LLMs with the zero-shot prompt “Let’s think step by step” for generating intermediate reasoning steps, while least-to-most prompting [226] first queries LLMs to perform problem decomposition and then utilizes LLMs to sequentially solve sub-problems based on the intermediate answers to previously solved ones.
from the recency bias, i.e., they are prone to repeat answers that are near the end of demonstrations [212]. Thus, it is important to arrange demonstrations (i.e., task examples) in a reasonable order. Early work proposes several heuristic methods to quickly find a good order. For example, demonstrations can be directly organized according to their similarity to the query in the embedding space [213]; the more similar, the closer to the end. Besides, global and local entropy metrics can be used to score different demonstration orders [211]. To integrate more task information, some recent studies propose to minimize the code length required to compress and transmit task labels, which is inspired by information theory [227]. However, these methods need additional labeled data as the validation set to evaluate the performance of specific demonstration orders. To eliminate this need, the authors in [211] propose to sample the validation data from the LLM itself.

6.1.3 Underlying Mechanism
After pre-training, LLMs can exhibit intriguing ICL capability without being updated. In what follows, we discuss two key questions about the ICL ability of LLMs, i.e., “how does pre-training affect the ICL ability” and “how do LLMs perform ICL during inference”.

How Pre-Training Affects ICL? ICL is first proposed in GPT-3 [35], and it has shown that the ICL ability becomes more significant with a larger model size. While, some studies reveal that small-scale PLMs can also demonstrate a strong ICL ability with specially designed training tasks (e.g., learning to predict the label with task examples and the query as the input), and may even surpass larger models [228]. It suggests that the design of training tasks is an important influence factor of the ICL capability of LLMs. Besides training tasks, recent studies have also investigated the relationship between ICL and the pre-training corpora [224, 229, 230]. It has been shown that the performance of ICL heavily depends on the source of pre-training corpora rather than the scale [280]. Another study [229] provides an in-depth analysis of the impact of training data distribution. They find that ICL emerges when the training data can be clustered into numerous infrequent classes, instead of being uniformly distributed. Furthermore, the authors in [224] theoretically explain ICL as the product of pre-training on documents that exhibit long-range coherence.

How LLMs Perform ICL? At the inference stage, researchers focus on analyzing how the ICL capability operates based on given demonstrations since no explicit learning or updating is involved. They typically analyze from the perspective of gradient descent and consider ICL as implicit fine-tuning [60, 231]. Under this framework, the ICL process can be explained as follows: by means of forward computation, LLMs generate meta-gradients with respect to demonstrations and implicitly perform gradient descent via the attention mechanism. Experiments also show that certain attention heads in LLMs are capable of performing task-agnostic atomic operations (e.g., copying and prefix matching), which are closely related to the ICL ability [232, 233]. To further explore the working mechanism of ICL, some studies abstract ICL as an algorithm learning process [234–236]. Specifically, the authors in [235] find that LLMs essentially encode implicit models through their parameters during pre-training. With the examples provided in ICL, LLMs can implement learning algorithms such as gradient descent or directly compute the closed-form solution to update these models during forward computation. Under this explanation framework, it has been shown that LLMs can effectively learn simple linear functions and even some complex functions like decision trees with ICL [234–236].

6.2 Chain-of-Thought Prompting
Chain-of-Thought (CoT) [83] is an improved prompting strategy to boost the performance of LLMs on complex reasoning tasks, such as arithmetic reasoning [237–239], commonsense reasoning [240, 241], and symbolic reasoning [83]. Instead of simply constructing the prompts with input-output pairs as in ICL, CoT incorporates intermediate reasoning steps that can lead to the final output into the prompts. In the following, we will elaborate on the usage of CoT with ICL and discuss when and why CoT prompting works.

6.2.1 In-context Learning with CoT
Typically, CoT can be used with ICL in two major settings, namely the few-shot and zero-shot settings, as introduced below.

Few-shot CoT. Few-shot CoT is a special case of ICL, which augments each demonstration (input, output) as (input, CoT, output) by incorporating the CoT reasoning steps. To apply this strategy, we next discuss two key issues, i.e., how to design appropriate CoT prompts and how to utilize the generated CoTs for deriving the final answer.

• CoT prompt design. It is critical to design appropriate CoT prompts for effectively eliciting the complex reasoning ability of LLMs. As a direct approach, it is shown that using diverse CoTs (i.e., multiple reasoning paths for each problem) can effectively enhance their performance [242].
Another intuitive idea is that prompts with more complex reasoning paths are more likely to elicit the reasoning ability of LLMs [243], which can result in higher accuracy in generating correct answers. However, both of these two approaches rely on annotated CoT datasets, which limits their use in practice. To overcome this limitation, Auto-CoT [244] proposes to utilize Zero-shot-CoT [225] (detailed in the following part “Zero-shot-CoT”) to generate CoT reasoning paths by specially prompting LLMs, thus eliminating manual efforts. In order to boost the performance, Auto-CoT further divides the questions in the training set into different clusters and then chooses the questions that are closest to the centroid of each cluster, which is supposed to well represent the questions in the training set. Although few-shot CoT can be considered as a special prompt case in ICL, the ordering of demonstrations seems to have a relatively small impact compared to the standard prompt in ICL: reordering the demonstrations only results in a performance variation of less than 2% in most tasks [33].

• Enhanced CoT strategies. Besides enriching the contextual information, CoT prompting also provides more options to infer the answer given a question. Existing studies mainly focus on generating multiple reasoning paths, and
try to find a consensus among the derived answers [245–247]. For instance, self-consistency [245] is proposed as a new decoding strategy when generating CoT and the final answer. It first generates several reasoning paths and then takes an ensemble over all the answers (e.g., selecting the most consistent answer by voting among these paths). Self-consistency boosts the performance in CoT reasoning by a large margin, and can even improve some tasks where CoT prompting is usually worse than standard promoting (e.g., closed-book question answering and natural language inference). Further, the authors in [246] expand the self-consistency strategy to a more general framework, and they find that diverse reasoning paths are the key to the performance improvement in CoT reasoning. The above methods can be easily integrated into CoT prompting to enhance the performance without additional training. In contrast, other studies train a scoring model to measure the reliability of the generated reasoning paths [242] or continually train LLMs on the reasoning paths generated by themselves [245] to improve the performance.

Zero-shot CoT. Different from few-shot CoT, zero-shot CoT does not include human-annotated task demonstrations in the prompts. Instead, it directly generates reasoning steps and then employs the generated CoTs to derive the answers. Zero-shot CoT is first proposed in [227], where the LLM is first prompted by “Let’s think step by step” to generate reasoning steps and then prompted by “Therefore, the answer is” to derive the final answer. They find that such a strategy drastically boosts the performance when the model scale exceeds a certain size, but is not effective with small-scale models, showing a significant pattern of emergent abilities. In order to unlock the CoT ability on more tasks, Flan-T5 and Flan-PaLM [81] further perform instruction tuning on CoT annotations and the zero-shot performance on unseen tasks has been improved.

6.2.2 Further Discussion on CoT

In this part, we present discussions regarding two fundamental questions related to CoT, i.e., “when does CoT work for LLMs” and “why can LLMs perform CoT reasoning”.

When CoT works for LLMs? Since CoT is an emergent ability [31], it only has a positive effect on sufficiently large models (e.g., typically containing 10B or more parameters [33]) but not on small models. Moreover, since CoT augments the standard prompting with reasoning steps, it is mainly effective to improve the tasks that require step-by-step reasoning, such as arithmetic reasoning, commonsense reasoning, and symbolic reasoning. Whereas, for other tasks that do not rely on complex reasoning, it might show worse performance than standard prompting [246], e.g., MNLI-m/mm, SST-2, and QQP from GLUE [250]. Interestingly, it seems that the performance gain brought by CoT prompting could be significant only when standard prompting yields poor results [33].

Why LLMs Can Perform CoT Reasoning? As the second question, we discuss the underlying mechanism of CoT in the following two aspects.

- The source of CoT ability. Regarding the source of CoT capability, it is widely hypothesized that it can be attributed to training on code since models trained on it show a strong reasoning ability [47] [251]. Intuitively, code data is well organized with algorithmic logic and programming flow, which may be useful to improve the reasoning performance of LLMs. However, this hypothesis still lacks publicly reported evidence of ablation experiments with and without training on code. Besides, instruction tuning seems not to be the key to obtaining the CoT ability, since it has been empirically shown that instruction tuning on non-CoT data does not improve the performance on held-out CoT benchmarks [81].

- The effect of prompting components. The major distinction between CoT prompting and standard prompting is the incorporation of reasoning paths prior to the final answer. Thus, some researchers investigate the effect of different components in the reasoning paths. Specifically, a recent study identifies three key components in CoT prompting, namely symbols (e.g., numerical quantities in arithmetic reasoning), patterns (e.g., equations in arithmetic reasoning), and text (i.e., the rest of tokens that are not symbols or patterns) [252]. It is shown that the latter two parts (i.e., patterns and text) are essential to the model performance, and removing either one would lead to a significant performance drop. However, the correctness of symbols and patterns does not seem critical. Further, there exists a symbiotic relationship between text and patterns: the text helps LLMs to generate useful patterns, and patterns aid LLMs to understand tasks and generate texts that help solve them [252].

In summary, CoT prompting provides a general yet flexible approach to eliciting the reasoning ability of LLMs. There are also some preliminary attempts that extend this technique to solve multimodal tasks [253] and multilingual tasks [254]. In addition to directly utilizing LLMs with ICL and CoT, some recent studies explore how to specialize the ability of LLMs towards specific tasks [255–257], which is called model specialization [258]. For example, the researchers in [258] specialize the ability of mathematical reasoning from LLMs through fine-tuning the small-scale Flan-T5 [81] on CoT reasoning paths generated by LLMs. Model specialization can also be applied to solve a variety of tasks like question answering [259], code synthesis [260], and information retrieval [261].

7 Capacity Evaluation

to examine the effectiveness and superiority of LLMs, a surge of tasks and benchmarks have been leveraged for conducting empirical evaluation and analysis. We first introduce three types of basic evaluation tasks of LLMs for language generation and understanding, then present several advanced tasks of LLMs with more complicated settings or goals, and finally discuss existing benchmarks and empirical analyses.

7.1 Basic Evaluation Tasks

In this part, we mainly focus on three types of evaluation tasks for LLMs, i.e., language generation, knowledge utilization, and complex reasoning. It is noted that we do not intend to have complete coverage of all the related tasks, but instead only focus on the most widely discussed or studied tasks for LLMs. Next, we introduce these tasks in detail.
7.1.1 Language Generation

According to the task definition, existing tasks about language generation can be roughly categorized into language modeling, conditional text generation, and code synthesis tasks. Note that code synthesis is not a typical NLP task, we include it for discussion because it can be directly solved by most LLMs (trained on code data) in a similar generation approach as natural language text.

Language Modeling. As the most fundamental ability of LLMs, language modeling aims to predict the next token based on the previous tokens \[15\], which mainly focuses on the capacity of basic language understanding and generation. For evaluating such an ability, typical language modeling datasets that existing work uses include Penn Treebank \[262\], WikiText-103 \[263\], the Pile \[108\], LAMBADA \[147\], WMT’14-16,19,20,21,22 \[264-269\], Flores-101 \[270\], DiaBLA \[271\], CNN/DailyMail \[272\], XSum \[273\], WikiLingua \[274\], OpenDialKG \[275\], SuperGLUE \[276\], MMLU \[277\], Big-bench Hard \[278\], CLUE \[279\], APPS \[280\], HumanEval \[287\], MBPP \[133\], CodeContest \[94\], MTPB \[76\], DS-1000 \[261\], ODEX \[282\].

Conditional Text Generation

Closed-Book QA

Natural Questions \[283\], ARC \[284\], TruthfulQA \[285\], Web Questions \[286\], TriviaQA \[287\], PIQA \[288\], LC-quad2.0 \[289\], GrillQA \[290\], KQAPro \[291\], CWQ \[292\], MKQA \[293\], ScienceQA \[294\].

Knowledge Utilization

Open-Book QA

Natural Questions \[283\], OpenBookQA \[295\], ARC \[284\], Web Questions \[286\], TriviaQA \[287\], PIQA \[288\], MS MARCO \[296\], QASC \[297\], SquAD \[298\], WikiMovies \[299\].

Knowledge Completion

WikiFact \[300\], FB15k-237 \[301\], Freebase \[302\], WN18RR \[303\], WordNet \[304\], LAMA \[305\], YAGO3-10 \[306\], YAGO \[307\].

Knowledge Reasoning

CSQA \[240\], StrategyQA \[241\], ARC \[284\], BoolQ \[308\], PIQA \[288\], SIQA \[309\], HellaSwag \[310\], Winograd \[311\], OpenBookQA \[295\], COPA \[312\], ScienceQA \[294\], proScript \[313\], PiQA \[314\], ExplaGraphs \[315\], ProofWriter \[316\], EntailmentBank \[317\], ProOntoQA \[318\].

Complex Reasoning

Symbolic Reasoning

CoinFlip \[33\], ReverseList \[33\], LastLetter \[33\], Boolean Assignment \[33\], Parity \[319\], Colored Object \[320\], Penguins in a Table \[320\], Repeat Copy \[35\], Object Counting \[68\].

Mathematical Reasoning

MATH \[277\], GSM8k \[278\], SVAMP \[283\], MultiArith \[271\], ASDiv \[239\], MathQA \[322\], AQUA-RAT \[324\], MAWPS \[272\], DROP \[273\], NaturalProofs \[326\], PISA \[327\], miniF2F \[288\], ProofNet \[289\].

TABLE 6

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language Modeling</td>
<td>Penn Treebank [262], WikiText-103 [263], the Pile [108], LAMBADA [147], WMT’14-16,19,20,21,22 [264-269], Flores-101 [270], DiaBLA [271], CNN/DailyMail [272], XSum [273], WikiLingua [274], OpenDialKG [275], SuperGLUE [276], MMLU [277], Big-bench Hard [278], CLUE [279], APPS [280], HumanEval [287], MBPP [133], CodeContest [94], MTPB [76], DS-1000 [261], ODEX [282]</td>
</tr>
<tr>
<td>Conditional Text Generation</td>
<td>Natural Questions [283], ARC [284], TruthfulQA [285], Web Questions [286], TriviaQA [287], PIQA [288], LC-quad2.0 [289], GrillQA [290], KQAPro [291], CWQ [292], MKQA [293], ScienceQA [294]</td>
</tr>
<tr>
<td>Code Synthesis</td>
<td>APPS [280], HumanEval [287], MBPP [133], CodeContest [94], MTPB [76], DS-1000 [261], ODEX [282]</td>
</tr>
<tr>
<td>Knowledge Utilization</td>
<td>Natural Questions [283], ARC [284], TruthfulQA [285], Web Questions [286], TriviaQA [287], PIQA [288], MS MARCO [296], QASC [297], SquAD [298], WikiMovies [299]</td>
</tr>
<tr>
<td>Knowledge Completion</td>
<td>WikiFact [300], FB15k-237 [301], Freebase [302], WN18RR [303], WordNet [304], LAMA [305], YAGO3-10 [306], YAGO [307]</td>
</tr>
<tr>
<td>Knowledge Reasoning</td>
<td>CSQA [240], StrategyQA [241], ARC [284], BoolQ [308], PIQA [288], SIQA [309], HellaSwag [310], Winograd [311], OpenBookQA [295], COPA [312], ScienceQA [294], proScript [313], PiQA [314], ExplaGraphs [315], ProofWriter [316], EntailmentBank [317], ProOntoQA [318]</td>
</tr>
<tr>
<td>Complex Reasoning</td>
<td>CoinFlip [33], ReverseList [33], LastLetter [33], Boolean Assignment [33], Parity [319], Colored Object [320], Penguins in a Table [320], Repeat Copy [35], Object Counting [68]</td>
</tr>
<tr>
<td>Mathematical Reasoning</td>
<td>MATH [277], GSM8k [278], SVAMP [283], MultiArith [271], ASDiv [239], MathQA [322], AQUA-RAT [324], MAWPS [272], DROP [273], NaturalProofs [326], PISA [327], miniF2F [288], ProofNet [289]</td>
</tr>
</tbody>
</table>
**Code Synthesis.** Besides generating high-quality natural language, existing LLMs also show strong abilities to generate formal language, especially computer programs (i.e., code) that satisfy specific conditions, called code synthesis \[337\]. Unlike natural language generation, as the generated code can be directly checked by execution with corresponding compilers or interpreters, existing work mostly evaluated the quality of the generated code from LLMs by calculating the pass rate against the test cases, i.e., pass@\(k\). Recently, several code benchmarks focusing on functional correctness are proposed to assess the code synthesis abilities of LLMs, such as APPS [280], HumanEval [87], and MBPP [133]. Typically, they consist of diverse programming problems, with text specification and test cases for correctness checking. To improve such an ability, it is key to fine-tuning (or pre-training) LLMs on code data, which can effectively adapt LLMs to code synthesis tasks [76]. Besides, existing work has proposed new strategies to generate code, e.g., sampling multiple candidate solutions [133] and planning-guided decoding [336], which can be considered as the imitation of bug-fixing and code-planning processes by programmers. Impressively, LLMs have recently shown competitive performance with humans by achieving a ranking of the top 28% among users on the programming contest platform Codeforces [94]. Further, GitHub Copilot has been released to assist programmers in coding IDEs (e.g., Visual Studio and JetBrains IDEs), which can support a variety of languages including Python, JavaScript, and Java. A viewpoint article entitled “The End of Programming” \[339\] in Communications of the ACM discussed the impact of AI programming in the field of computer science, emphasizing an important shift towards the highly adaptive LLM as a new atomic unit of computation.

**Major Issues.** Although LLMs have achieved splendid performance in generating human-like text, they are susceptible to suffering from two major issues in language generation as discussed below:

- **Controllable generation.** For LLMs, the mainstream way to generate texts under given conditions is through the use of natural language instructions or prompts. Despite the simplicity, such a mechanism poses significant challenges in terms of exerting fine-grained or structural constraints over the generated outputs of these models. Existing work [41] shows that, when generating texts with complex constraints on their structures, LLMs can handle local planning (e.g., interactions between proximal sentences) very well but might struggle with global planning (i.e., long-range relatedness). For example, to generate a complex long passage with several paragraphs, it is still difficult to directly ensure specific text structure (e.g., the order of concepts and the logical flow), considering the whole text. This case will become even more challenging for generation tasks that require formal rules or grammar, e.g., code synthesis. To tackle this issue, a potential solution is to extend the one-pass generation into the iterative prompting of LLMs. This simulates the human writing process to break down language generation into multiple steps such as planning, drafting, rewriting, and editing [335]. Several studies have proven that iterative prompting can elicit relevant knowledge to achieve better performance in sub-tasks [540, 541]. In essence, chain-of-thought prompting has utilized the idea of decomposing complex tasks into multi-step reasoning chains. Besides, the safety control of generated texts is also important for practical deployment. It has been shown that LLMs may generate texts that contain sensitive information or offensive expressions [46]. Although the RLHF algorithm [61] can alleviate this problem to some extent, it still relies on considerable human-labeled data for tuning LLMs, without an objective optimization goal to follow. Thus, it is imperative to explore effective methods to overcome these limitations and enable safer control over the outputs of LLMs.

- **Specialized generation.** Although LLMs have learned general language patterns to generate coherent text, their proficiency in generation might be constrained when dealing with a specialized domain or task. For instance, a language model that has been trained on general web articles may face challenges when generating a medical report which involves many medical jargon and methods. Intuitively, domain knowledge should be critical for model specialization. Whereas, it is not easy to inject such specialized knowledge into LLMs. As discussed in recent analyses [47, 342], when LLMs are trained to exhibit some specific ability that allows them to excel in some areas, they might struggle in others. Such an issue is related to catastrophic forgetting [343, 344] in training neural networks, which refers to the conflict phenomenon of integrating new and old knowledge. Similar cases also occur in human alignment of LLMs, where “alignment tax” \[61\] (e.g., a potential loss in the in-context learning ability) has to be paid for aligning to human values and needs. Therefore, it is important to develop effective model specialization methods that can flexibly adapt LLMs to various task scenarios, meanwhile retaining the original abilities as possible.

### 7.1.2 Knowledge Utilization

Knowledge utilization is an important ability of intelligent systems to accomplish knowledge-intensive tasks (e.g., commonsense question answering and fact completion) based on supporting factual evidence. Concretely, it requires LLMs to properly utilize the rich factual knowledge from the pre-training corpus or retrieve external data when necessary. In particular, question answering (QA) and knowledge completion have been two commonly used tasks for evaluating this ability. According to the test tasks (question answering or knowledge completion) and evaluation settings (with or without external resources), we categorize existing knowledge utilization tasks into three types, namely closed-book QA, open-book QA \[18\] and knowledge completion.

**Closed-Book QA.** Closed-book QA tasks [345] test the acquired factual knowledge of LLMs from the pre-training corpus, where LLMs should answer the question only based on the given context without using external resources. For

17. Given \(k\) programs generated by the LLM, pass@\(k\) is computed as 1 when at least one program passes all test cases, or else 0.

18. In this part, open-book QA refers to the QA tasks that require to extract and utilize useful information from external knowledge resources, as the antithesis of closed-book QA (only using the encoded information from pre-training corpus). Note that there is a dataset also named OpenBookQA [295], which follows the settings of open-book QA tasks by extracting and utilizing external science facts.
evaluating this ability, there are several datasets that can be leveraged, including Natural Questions [283], Web Questions [286], and TriviaQA [287], where the accuracy metric is widely adopted. Empirical results have revealed that LLMs can perform well in this setting and even match the performance of state-of-the-art open-domain QA systems [59]. Besides, the performance of LLMs on closed-book QA tasks also shows a scaling law pattern in terms of both model size and data size: scaling the parameters and training tokens can increase the capacity of LLMs and help them learn (or memorize) more knowledge from the pre-training data [56]. Further, under a similar parameter scale, LLMs with more pre-training data relevant to the evaluated tasks would achieve better performance [70]. Besides, the closed-book QA setting also provides a testbed for probing the accuracy of the factual knowledge encoded by LLMs. However, as shown in existing work [55], LLMs might perform less well on QA tasks relying on fine-grained knowledge, even when it exists in the pre-training data.

Open-Book QA. Unlike closed-book QA, in open-book QA tasks, LLMs can extract useful evidence from the external knowledge base or document collections, and then answer the question based on the extracted evidence [346, 349]. Typical open-book QA datasets (e.g., Natural Questions [283], OpenBookQA [295], and SQuAD [298]) have overlaps with closed-book QA datasets, but they incorporate external data sources, e.g., Wikipedia. The metrics of accuracy and F1 score are widely used in open-book QA tasks for evaluation. To select relevant knowledge from external resources, LLMs are often paired with a text retriever (or even a search engine), which is trained independently or jointly with LLMs [70, 346, 350]. In evaluation, existing studies mainly focus on testing how LLMs utilize the extracted knowledge to answer the question and show that the retrieved evidence can largely improve the accuracy of the generated answers, even enabling a smaller LLM to outperform 10× larger ones [346, 350]. Besides, open-book QA tasks can also evaluate the recency of knowledge information. Pre-training or retrieving from outdated knowledge resources may cause LLMs to generate incorrect answers for time-sensitive questions [346].

Knowledge Completion. In knowledge completion tasks, LLMs might be (to some extent) considered as a knowledge base [305], which can be leveraged to complete or predict the missing parts of knowledge units (e.g., knowledge triples). Such tasks can probe and evaluate how much and what kind of knowledge LLMs have learned from the pre-training data. Existing knowledge completion tasks can be roughly divided into knowledge graph completion tasks (e.g., FB15k-237 [301] and WN18RR [303] and fact completion tasks (e.g., WikiFact [300]), which aim to complete the triples from a knowledge graph and incomplete sentences about specific facts, respectively. Empirical studies have revealed that it is difficult for existing LLMs to accomplish knowledge completion tasks in specific domains [251]. As shown in the evaluation results on WikiFact, LLMs perform well on several frequent relations that occur in the pre-training data (e.g., currency and author), while not well on rare ones (e.g., discoverer_or_inventor and place_of_birth). Interestingly, under the same evaluation settings (e.g., in-context learning), InstructGPT (i.e., text-davinci-002) outperforms GPT-3 in all subsets of WikiFact. It indicates that instruction tuning is helpful for LLMs to accomplish knowledge completion tasks.

Major Issues. Although LLMs have achieved key progress in capturing and utilizing knowledge information, they suffer from two major issues as discussed below.

- **Hallucination.** In generating factual texts, a challenging issue is hallucination generations [336], where the generated information is either in conflict with the existing source (intrinsic hallucination) or cannot be verified by the available source (extrinsic hallucination), which are illustrated with two examples in Figure 7. Hallucination widely occurs in existing LLMs, even the most superior LLMs such as GPT-4 [46]. In essence, LLMs seem to “unconsciously” utilize the knowledge in task solving, which still lack an ability to accurately control the use of intrinsic or external knowledge. Hallucination would mislead LLMs to generate undesired outputs and mostly degrade the performance, leading to potential risks when deploying LLMs in real-world applications. To alleviate this problem, the alignment tuning strategies (as discussed in Section 5.2) have been widely utilized in existing works [61], which rely on tuning LLMs on high-quality data or using human feedback. For the evaluation of the hallucination problem, a set of hallucination detection tasks have been proposed, e.g., TruthfulQA [285], for detecting human falsehood mimicked by models.

---

**Fig. 7. Examples of intrinsic and extrinsic hallucination for a public LLM (access date: March 19, 2023).** As an example of intrinsic hallucination, the LLM gives a conflicting judgment about the relationship between Cindy and Amy, which contradicts the input. For extrinsic hallucination, in this example, the LLM seems to have an incorrect understanding of the meaning of RLHF (reinforcement learning from human feedback), though it can correctly understand the meaning of LLMs (in this context).
● Knowledge recency. As another major challenge, LLMs would encounter difficulties when solving tasks that require the latest knowledge beyond the training data. To tackle this issue, a straightforward approach is to regularly update LLMs with new data. However, it is very costly to fine-tune LLMs, and also likely to cause the catastrophic forgetting issue when incrementally training LLMs. Therefore, it is necessary to develop efficient and effective approaches that can integrate new knowledge into existing LLMs, making them up-to-date. Existing studies have explored how to utilize the external knowledge source (e.g., search engine) to complement LLMs, which can be either jointly optimized with LLMs or used as a plug-and-play module. For instance, ChatGPT utilizes a retrieval plugin to access up-to-date information sources. By incorporating the extracted relevant information into the context, LLMs can acquire new factual knowledge and perform better on relevant tasks. However, such an approach seems to be still at a superficial level. It has been revealed that it is difficult to directly amend intrinsic knowledge or inject specific knowledge into LLMs, which remains an open research problem.

7.1.3 Complex Reasoning

Complex reasoning refers to the ability of understanding and utilizing supporting evidence or logic to derive conclusions or make decisions. According to the type of involved logic and evidence in the reasoning process, we consider dividing existing evaluation tasks into three major categories, namely knowledge reasoning, symbolic reasoning, and mathematical reasoning.

Knowledge Reasoning. The knowledge reasoning tasks rely on logical relations and evidence about factual knowledge to answer the given question. Existing work mainly uses specific datasets to evaluate the reasoning capacity of the corresponding type of knowledge, e.g., CSQA/StrategyQA for commonsense knowledge reasoning and ScienceQA for science knowledge reasoning. In addition to the accuracy of the predicted results, existing work has also evaluated the quality of the generated reasoning process, via automatic metrics (e.g., BLEU) or human evaluation. Typically, these tasks require LLMs to perform step-by-step reasoning based on factual knowledge, until reaching the answer to the given question. To elicit the step-by-step reasoning ability, chain-of-thought (CoT) prompting strategy has been proposed for enhancing the complex reasoning capacity of LLMs. As discussed in Section 6.2, CoT involves the intermediate reasoning steps, which can be manually created or automatically generated, into the prompts to guide LLMs to perform multi-step reasoning. Such a way largely improves the reasoning performance of LLMs, leading to new state-of-the-art results on several complex knowledge reasoning tasks. Further, after reformulating knowledge reasoning tasks into code generation tasks, researchers have found that the performance of LLMs can be further improved, especially with the LLMs pretrained on code. However, due to the complexity of knowledge reasoning tasks, the current performance of LLMs still lags behind human results. As one of the most common mistakes, LLMs might generate inaccurate intermediate steps based on wrong factual knowledge, leading to a wrong final result. To address this issue, existing work has proposed special decoding or ensemble strategies to improve the accuracy of the whole reasoning chain. More recently, an empirical study reveals that LLMs may have difficulty in explicitly inferring the commonsense knowledge required by a specific task, though they can successfully solve it. Further, it seems that leveraging self-generated knowledge is not beneficial for improving the reasoning performance.

Symbolic Reasoning. The symbolic reasoning tasks mainly focus on manipulating the symbols in a formal rule setting to fulfill some specific goal, where the operations and rules may have never been seen by LLMs during pre-training. Existing work commonly evaluates LLMs on the task of last letter concatenation and coin flip, where the evaluation examples require the same reasoning steps as the in-context examples (called in-domain test) or more steps (called out-of-domain test). For an example of the out-of-domain test, LLMs could only see the examples with two words in context, but it requires LLMs to concatenate the last letters of three or more words. Typically, the accuracy of the generated symbols is adopted to evaluate the performance of LLMs on these tasks. Thus, LLMs need to understand the semantic relations among the symbolic operations and their composition in complex scenarios. However, under the out-of-domain setting, as LLMs have not seen the complex compositions of symbolic operations and rules (e.g., twice the number of operations in context examples), it is hard for LLMs to capture their accurate meanings. To solve this issue, existing studies incorporate scratchpad, interpreter, and tutor strategies to help LLMs better manipulate symbolic operations, for generating longer and more complex reasoning processes. Another line of research work utilizes the formal programming language to represent the symbolic operations and rules, which requires LLMs to generate code and perform the reasoning process by executing it with external interpreters. Such a way can decompose the complex reasoning process into code synthesis and program execution for LLMs and interpreters, respectively, leading to a simplified reasoning process with yet more accurate results.

Mathematical Reasoning. The mathematical reasoning tasks need to comprehensively utilize mathematical knowledge, logic, and computation for solving problems or generating proof statements. Existing mathematical reasoning tasks can be mainly categorized into math problem solving and automated theorem proving. For math problem solving tasks, SVAMP, GSM8k, and MATH datasets are commonly used for evaluation, where LLMs need to generate accurate concrete numbers or equations to answer the mathematical problem. As these tasks also require multi-step reasoning, the chain-of-thought prompting strategy has been widely adopted for LLMs to improve the reasoning performance. As a practical strategy, continually pre-
training LLMs on large-scale mathematical corpora can largely boost their performance on mathematical reasoning tasks. Further, since math problems in different languages share the same mathematical logic, researchers also propose a multilingual math word problem benchmark to evaluate the multilingual mathematical reasoning capacity of LLMs. As another challenging task, automated theorem proving (ATP) requires the reasoning model to strictly follow the reasoning logic and mathematical skills. To evaluate the performance on this task, PISA and miniF2F are two typical ATP datasets with the success rate of proving as the evaluation metric. As a typical approach, existing work on ATP utilizes LLMs to aid the search for proofs using an interactive theorem prover (ITP), such as Lean and Metamath. A major limitation of ATP research is the lack of related corpora in formal language. To tackle it, several studies utilize LLMs to convert informal statements into formal proofs for augmenting new data or generate drafts and proof sketches to reduce the search space of the proofs.

**Major Issues.** In spite of the advancements, LLMs still have several limitations in solving complex reasoning tasks.

- **Inconsistency.** With improved reasoning strategies (e.g., CoT prompting), LLMs can solve some complex reasoning tasks, by performing step-by-step reasoning based on the supporting logic and evidence. Despite the effectiveness, the inconsistency issue often occurs in the decomposed reasoning process. Concretely, LLMs may generate the correct answer following an invalid reasoning path, or produce a wrong answer after correct reasoning, leading to inconsistency between the derived answer and the reasoning process. To alleviate this problem, existing work has proposed to guide the whole generation process of LLMs via external tools or models, or re-check the reasoning process and final answer for correcting them. As a promising solution, recent approaches reformulate the complex reasoning tasks into code generation tasks, where the strict execution of the generated code ensures the consistency between the reasoning process and the outcome. Besides, it has been revealed that there might also exist inconsistency between tasks with similar inputs, where small changes in the task description may cause the model to produce different results. To mitigate this problem, the ensemble of multiple reasoning paths can be applied to enhance the decoding process of LLMs.

- **Numerical computation.** For complex reasoning tasks, LLMs still face difficulties in the involved numerical computation, especially for the symbols that are seldom encountered during pre-training, such as arithmetic with large numbers. To tackle this issue, a direct way is to tune LLMs on synthesis arithmetic problems. A surge of studies follow this approach and further improve the numerical computation performance by special training and inference strategies, e.g., scratchpad tracing. Besides, existing work has also incorporated external tools (e.g., calculator), especially for handling arithmetic operations. More recently, ChatGPT has provided a plugin mechanism to use external tools. In this way, LLMs need to learn how to properly manipulate the tools. For this purpose, researchers have augmented the examples using tools even the LLM itself for tuning the LLM, or devised instructions and exemplars for in-context learning. While, these LLMs still rely on the text context to capture the semantic meanings of mathematical symbols (during the pre-training stage), which is not best suited for numerical computation in essence.

### 7.2 Advanced Ability Evaluation

In addition to the above basic evaluation tasks, LLMs also exhibit some superior abilities that require special considerations for evaluation. In this part, we discuss several representative advanced abilities and the corresponding evaluation approaches, including human alignment, interaction with the external environment, and tool manipulation. Next, we discuss these advanced abilities in detail.

#### 7.2.1 Human Alignment

It is desired that LLMs could well conform to human values and needs, i.e., human alignment, which is a key ability for the broad use of LLMs in real-world applications. To evaluate this ability, existing studies consider multiple criteria for human alignment, such as helpfulness, honesty, and safety. For helpfulness and honesty, adversarial question answering tasks (e.g., TruthfulQA) can be utilized to examine LLM's ability in detecting possible falsehood in the text. Furthermore, harmlessness can be also evaluated by several existing benchmarks, e.g., CrowS-Pairs and Winogender. Despite the automatic evaluation with the above datasets, human evaluation is still a more direct way to effectively test the human alignment ability of LLMs. OpenAI invites many experts in domains related to AI risks to evaluate and improve the behaviors of GPT-4 when encountering risky contents. Besides, for other aspects of human alignment (e.g., truthfulness), several studies propose to use specific instructions and devise annotation rules to guide the annotation process. Empirical studies have revealed that these strategies can greatly improve the human alignment ability of LLMs. OpenAI invites many experts in domains related to AI risks to evaluate and improve the behaviors of GPT-4 when encountering risky contents. For instance, after alignment tuning on data collected through interactions with experts, the incorrect behavior rate of GPT-4 can be largely reduced when it deals with sensitive or disallowed prompts. In addition, high-quality pre-training data can reduce the effort required for alignment. For instance, Galactica is potentially more harmless due to the less biased contents in the scientific corpus.

#### 7.2.2 Interaction with External Environment

Besides standard evaluation tasks, LLMs have the ability to receive feedback from the external environment and perform actions according to the behavior instruction, e.g., generating action plans in natural language to manipulate agents. Such an ability is also emergent in LLMs that can generate detailed and highly realistic action plans, while smaller models (e.g., GPT-2) tend to generate shorter or meaningless plans.

To test this ability, several embodied AI benchmarks can be used for evaluation, described as follows. Virtual-Home builds a 3D simulator for household tasks such as cleaning and cooking, in which the agent can execute
natural language actions generated by LLMs. ALFRED [376] includes more challenging tasks that require LLMs to accomplish compositional targets. BEHAVIOR [377] focuses on rearrangement tasks in simulation environments and requires LLMs to generate complex solutions, e.g., changing the internal status of objects. Based on the generated action plans from LLMs, existing work either adopts the regular metrics (e.g., executability and correctness of the generated action plans) [375] in the benchmark or directly conducts real-world experiments and measures the success rate [378], to evaluate such ability. Existing work has shown the effectiveness of LLMs in interacting with the external environment and generating accurate action plans [379]. Recently, several improved methods have been proposed to enhance the interaction ability of LLMs, e.g., designing code-like prompts [380] and providing real-world grounding [378].

7.2.3 Tool Manipulation

When solving complex problems, LLMs can turn to external tools if they determine it is necessary. By encapsulating available tools with API calls, existing work has involved a variety of external tools, e.g., search engine [20], calculator [69], and compiler [68], to enhance the performance of LLMs on several specific tasks. Recently, OpenAI has supported the use of plugins in ChatGPT [352], which can equip LLMs with broader capacities beyond language modeling. For example, the web browser plugin enables ChatGPT to access fresh information. Further, incorporating third-party plugins is particularly key for creating a prosperous ecosystem of applications based on LLMs.

To examine the ability of tool manipulation, existing work mostly adopts complex reasoning tasks for evaluation, such as mathematical problem solving (e.g., GSM8k [237] and SVAMP [238]) or open-book QA (e.g., TruthfulQA [285]), where the successful utilization of tools is very important for enhancing the required skills that LLMs are incapable of (e.g., numerical calculation). In this way, the evaluated performance on these tasks can reflect the ability of LLMs in tool manipulation. To teach LLMs to utilize tools, existing studies add exemplars using tools in context to elicit LLMs [69], or fine-tune LLMs on simulated data about tool utilization [69] [370]. Existing work has found that with the help of tools, LLMs become more capable of handling the issues that they are not good at, e.g., equation calculation and utilizing real-time information, and eventually improve the final performance [69].

Summary. The above three abilities are of great value to the practical performance of LLMs: conforming to human values and preferences (human alignment), acting properly in real-world scenarios (interaction with the external environment), and expanding the ability scope (tool manipulation). In addition to the above three advanced abilities, LLMs might also show other abilities that are specially related to some tasks (e.g., data annotation [221] or learning mechanisms (e.g., self-improvement [249]). It will be an open direction to discover, measure and evaluate these newly emerging abilities, so as to better utilize and improve LLMs.

7.3 Public Benchmarks and Empirical Analysis

In the aforementioned parts, we have discussed the evaluation tasks of LLMs and their corresponding settings. Next, we will introduce existing evaluation benchmarks and empirical analyses for LLMs, which focus on exploring more comprehensive discussions from a general perspective.

7.3.1 Evaluation Benchmarks

Recently, several comprehensive benchmarks [251, 277, 320] have been released for the evaluation of LLMs. In this part, we introduce several representative and widely used benchmarks, i.e., MMLU, BIG-bench, and HELM.

- **MMLU [277]** is a versatile benchmark for large-scale evaluation of multi-task knowledge understanding, covering a wide range of knowledge domains from mathematics and computer science to humanities and social sciences. The difficulties of these tasks vary from basic to advanced. As shown in existing work, LLMs mostly outperform small models by a substantial margin on this benchmark [35, 55, 57, 81], which shows the scaling law in model size.

  Furthermore, GPT-4 achieves a remarkable record (86.4% in 5-shot setting) in MMLU, which is significantly better than the previous state-of-the-art models [46].

- **BIG-bench [320]** is a collaborative benchmark intended to probe existing LLMs from various aspects. It comprises 204 tasks that encompass a broad range of topics, including linguistics, childhood development, mathematics, commonsense reasoning, biology, physics, social bias, software development, and so on. By scaling the model size, LLMs can even outperform the average human performance under the few-shot setting on 65% of tasks in BIG-bench [56].

  Considering the high evaluation cost of the entire benchmark, existing work also proposed a lightweight benchmark BIG-bench-Lite that includes 24 small yet diverse and challenging tasks from BIG-bench. Additionally, the BIG-bench hard (BBH) benchmark has been proposed to concentrate on investigating the currently unsolvable tasks of LLMs by selecting the challenging tasks in which LLMs exhibit inferior performance compared to humans. Since BBH becomes more difficult, small models mostly achieve performance close to random. As a comparison, CoT prompting can elicit the abilities of LLMs to perform step-by-step reasoning for enhancing the performance, even exceeding the average human performance in BBH [278].

- **HELM [251]** is a comprehensive benchmark that currently implements a core set of 16 scenarios and 7 categories of metrics. It is built on top of many prior studies, conducting a holistic evaluation of language models. As shown in the experimental results of HELM [251], instruction tuning can consistently boost the performance of LLMs in terms of accuracy, robustness, and fairness. Further, for reasoning tasks, the LLMs that have been pre-trained on code corpus show superior performance.

  The above benchmarks cover a variety of mainstream evaluation tasks for the evaluation of LLMs. Besides, there are also several benchmarks that focus on evaluating specific abilities of LLMs, such as TyDiQA [381] for multilingual knowledge utilization and MGSMS [254] for multilingual mathematical reasoning. To conduct the evaluation, one can select suitable benchmarks according to specific goals.

20. In fact, there are also other benchmarks that are specially utilized for instruction tuning LLMs (e.g., Muffin and T0-SF). We do not list them here as we mainly focus on the benchmarks for the evaluation of LLMs.
In addition, there are also several open-source evaluation frameworks for researchers to conduct evaluations of LLMs on existing benchmarks or new evaluation tasks, such as Language Model Evaluation Harness \[382\] and OpenAI Evals \[46\].

7.3.2 Comprehensive Analyses on LLMs’ Capacities

In addition to constructing large-scale evaluation benchmarks, a surge of studies has conducted comprehensive analyses to investigate the strengths and limitations of LLMs. In this part, we briefly discuss them in major aspects, namely generalist (general-purpose capacity) and specialist (domain-specific capacity).

**Generalist.** Due to the remarkable performance, existing work \[41\] \[46\] \[324\] \[383\] \[385\] has systematically evaluated the general capacities of LLMs, to explore their competences in a variety of different tasks or applications. Typically, these studies mainly focus on the newly emerged LLMs (e.g., ChatGPT and GPT-4) that have not been well investigated before, which are discussed as follows:

- **Mastery.** To evaluate the mastery level of LLMs in solving general tasks, existing work \[385\] typically collects a set of datasets covering a range of tasks and domains, and then tests LLMs under the few/zero-shot setting. Empirical results \[41\] \[46\] \[342\] \[385\] have shown the superior capacities of being a general-purpose task solver. As a remarkable progress, GPT-4 has surpassed the state-of-the-art capacities of being a general-purpose task solver. As a result, recent studies have widely explored the use of LLMs for solving domain-specific tasks and evaluated the adaptation capacity of LLMs. Typically, these studies collect or construct domain-specific datasets to evaluate the performance of LLMs using in-context learning. Since our focus is to cover all the possible application domains, we briefly discuss three representative domains receiving considerable attention from the research community, namely healthcare, education, and law.

- **Healthcare** is a vital application field closely related to human life. Since the advent of ChatGPT, a series of studies have applied ChatGPT or other LLMs to the medical domain. It has been shown that LLMs are capable of handling a variety of healthcare tasks, e.g., biology information extraction \[389\], medical advice consultation \[390\] \[392\], and report simplification \[393\], and can even pass the medical license exams \[394\] \[396\] specially designed for professional doctors. However, LLMs may fabricate medical misinformation \[391\] \[393\], e.g., misinterpreting medical terms and suggesting advice inconsistent with medical guidelines. Besides, it would also raise privacy concerns to upload the health information of patients \[389\].

- **Education** is also an important application domain where LLMs potentially exert significant influence. Existing work has found that LLMs can achieve student-level performance on standardized tests \[46\] \[397\] \[398\] in the subjects of mathematics, physics, computer science and so on, in both multiple-choice and free-response problems. Besides, empirical studies have shown that LLMs can serve as writing or reading assistant for education \[399\] \[401\]. A recent study \[401\] reveals that ChatGPT is capable of generating logically consistent answers across disciplines, balancing both depth and breadth. Another quantitative analysis \[400\] shows that students utilizing ChatGPT perform better than average students with different usage methods (e.g., keeping or refining the results from LLMs as their own answers) in some courses from the computer security field. However, the increasing popularity of LLMs has been raising concerns (e.g., cheating on homework) on the rational use of such intelligent assistants for education.

- **Law** is a specialized domain that is built on professional domain knowledge. Recently, a number of studies have applied LLMs to solve various legal tasks, e.g., legal document analysis \[402\] \[403\], legal judgment prediction \[404\], and legal document writing \[403\]. A recent study \[406\] has found that LLMs own powerful abilities of legal interpretation and reasoning. Moreover, the latest GPT-4 model achieves a top 10% score in a simulated bar exam compared with...
human test-takers. However, the use of LLMs in law also raises concerns about legal challenges, including copyright issues, personal information leakage, or bias and discrimination.

Besides the aforementioned work, the capacities of LLMs have been also analyzed from other perspectives. For instance, some recent work has studied the human-like characteristics of LLMs, such as self-awareness, theory of mind (ToM), and affective computing. In particular, an empirical evaluation of ToM conducted on two classic false-belief tasks speculates that LLMs may have ToM-like abilities since the model in the GPT-3.5 series achieves comparable performance with nine-year-old children in ToM task. Further, another line of work has investigated the fairness and accuracy of existing evaluation settings about LLMs, e.g., the large-scale mixture-of-source pre-training data may contain the data in test sets.

8 Conclusion and Future Directions

In this survey, we have reviewed the recent progress of large language models (LLMs), and introduced the key concepts, findings, and techniques for understanding and utilizing LLMs. We focus on the discussion of large-sized models (i.e., having a size larger than 10B) while excluding the contents of early pre-trained language models (e.g., BERT and GPT-2) that have been well covered in the existing literature. In particular, our survey has discussed four important aspects of LLMs, i.e., pre-training, adaptation tuning, utilization, and evaluation. For each aspect, we highlight the techniques or findings that are key to the success of LLMs. Besides, we also summarize the available resources for developing LLMs and discuss important implementation guidelines for reproducing LLMs. This survey tries to cover the most recent literature about LLMs and provides a good reference resource on this topic for both researchers and engineers.

In this section, we summarize the discussions of this survey, and introduce the challenges and future directions for LLMs, in the following aspects.

Theory and Principle. To understand the underlying working mechanism of LLMs, one of the greatest mysteries is how information is distributed, organized, and utilized through the very large, deep neural network. It is important to reveal the basic principles or elements that establish the foundation of the abilities of LLMs. In particular, scaling seems to play an important role in increasing the capacity of LLMs. It has been shown that some emergent abilities would occur in an unexpected way (a sudden performance leap) when the parameter scale of language models increases to a critical size (e.g., 10B), typically including in-context learning, instruction following, and step-by-step reasoning. These emergent abilities are fascinating yet perplexing: when and how they are obtained by LLMs. Recent studies either conduct extensive experiments for investigating the effect of emergent abilities and the contributing factors to such abilities, or explain some specific abilities with existing theoretical frameworks. An insightful technical post also specially discusses this topic, taking the GPT-series models as the target. However, more formal theories and principles to understand, characterize, and explain the abilities or behaviors of LLMs are still missing. Since emergent abilities bear a close analogy to phase transitions in nature, cross-discipline theories or principles (e.g., whether LLMs can be considered as some kind of complex systems) might be useful to explain and understand the behaviors of LLMs. These fundamental questions are worth exploring for the research community, which are important for developing next-generation LLMs.

Model Architecture. Due to the scalability and effectiveness, Transformer, consisting of stacked multi-head self-attention layers, has become the de facto architecture for building LLMs. Various strategies have been proposed to improve the performance of this architecture, such as neural network configuration and scalable parallel training (see discussions in Section 2.2.2). To further enhance the model capacity (e.g., the multi-turn conversation ability), existing LLMs typically maintain a long context length, e.g., GPT-4-32k has an extremely large context length of 32,768 tokens. Thus, a practical consideration is to reduce the time complexity (originally to be quadratic costs) incurred by the standard self-attention mechanism. It is important to investigate the effect of more efficient Transformer variants in building LLMs, e.g., sparse attention has been used in GPT-3. Besides, catastrophic forgetting has been a long-standing challenge for neural networks, which also has a negative impact on LLMs. When tuning LLMs with new data, the originally learned knowledge is likely to be damaged, e.g., fine-tuning a LLM according to some specific tasks will affect the general ability of LLMs. A similar case occurs when LLMs are aligned with human values (called alignment tax). Thus, it is necessary to consider extending existing architectures with more flexible mechanisms or modules that can effectively support data update and task specialization.

Model Training. In practice, it is very difficult to pre-train capable LLMs, due to the huge computation consumption and the sensitivity to data quality and training tricks. It becomes particularly important to develop more systemic, economical pre-training approaches for optimizing LLMs, considering the factors of model effectiveness, efficiency optimization, and training stability. More model checking or performance diagnosis methods (e.g., predictable scaling in GPT-4) should be developed in order to detect early abnormal issues during training. Furthermore, it also calls for more flexible mechanisms of hardware support or resource schedule, so as to better organize and utilize the resources in a computing cluster. Since it is very costly to pre-train a LLM from scratch, it is important to design a suitable mechanisms for continually pre-training or fine-tuning the LLM based on publicly available model checkpoints (e.g., LLaMA and Flan-T5). For this purpose, a number of technical issues have to be resolved, including data inconsistency, catastrophic forgetting, and task specialization. While, to date, there still lack open-source model checkpoints for LLMs with complete pre-processing and training logs (e.g., the scripts to prepare the pre-training data) for reproducing. We believe that it will be of great value to have more open-source models for the
research of LLMs. Besides, it is also important to develop more improvement tuning strategies and investigate the mechanism that effectively elicits the model abilities.

**Model Utilization.** Since fine-tuning is very costly in real applications, *prompting* has become the prominent approach to using LLMs. By combining task descriptions and demonstration examples into prompts, in-context learning (a special form of prompting) endows LLMs with the ability to perform well on new tasks, even outperforming full-data fine-tuned models in some cases. Further, to enhance the ability of complex reasoning, advanced prompting techniques have been proposed, exemplified by the chain-of-thought (CoT) strategy, which includes the intermediate reasoning steps into prompts. However, existing prompting approaches still have several deficiencies described as follows. Firstly, it involves considerable human efforts in the design of prompts. It would be quite useful to automatically generate effective prompts for solving various tasks. Secondly, some complex tasks (e.g., formal proof and numerical computation) require specific knowledge or logic rules, which may not be best described in natural language or demonstrated by examples. Thus, it is important to develop more informative, flexible task formatting methods for prompts. Thirdly, existing prompting strategies mainly focus on single-turn performance. It is useful to develop interactive prompting mechanisms (e.g., through natural language conversations) for solving complex tasks, which have been demonstrated to be very useful by ChatGPT.

**Safety and Alignment.** Despite their capacities, LLMs pose similar safety challenges as small language models. For example, LLMs exhibit a tendency to generate hallucination texts [416], which refers to the texts that seem plausible but may be factually incorrect. What is worse, LLMs might be elicited by intentional instructions to produce harmful, biased, or toxic texts for malicious systems, leading to the potential risks of misuse [55][61]. To have a detailed discussion of other safety issues of LLMs (e.g., privacy, overreliance, disinformation, and influence operations), the readers can refer to the GPT-3/4 technical reports [46][55]. As the major approach to averting these issues, reinforcement learning from human feedback (RLHF) [51][56] has been widely used by incorporating humans in the training loop for developing well-aligned LLMs. To improve the model safety, it is also important to include safety-relevant prompts during RLHF, as shown by GPT-4 [46]. However, RLHF heavily relies on high-quality human feedback data from professional labelers, making it difficult to be properly implemented in practice. Therefore, it is necessary to improve the RLHF framework for reducing the efforts of human labelers and seek a more efficient annotation approach with guaranteed data quality, e.g., LLMs can be employed to assist the labeling work. More recently, red teaming [203][204] has been adopted for improving the model safety of LLMs, which utilizes the collected adversarial prompts to refine the LLMs (i.e., avoiding the attacks from red teaming). Furthermore, it is also meaningful to establish the learning mechanism for

21. While, it seems that an alternative approach to this issue is to invoke external tools, e.g., the plugins for ChatGPT, when the task is difficult to solve via text generation.

**Application and Ecosystem.** As LLMs have shown a strong capacity in solving various tasks, they can be applied in a broad range of real-world applications (e.g., following specific natural language instructions). As a remarkable progress, ChatGPT has potentially changed the way how humans access information, which leads to the release of New Bing. In the near future, it can be foreseen that LLMs would have a significant impact on information-seeking techniques, including both search engines and recommender systems. Furthermore, the development and use of intelligent information assistants would be highly promoted with the technology upgrade from LLMs. At a broader scope, this wave of technical innovation tends to build an ecosystem of LLM-empowered applications (e.g., the support of plugins by ChatGPT), which will have a close connection with human life. Lastly, the rising of LLMs sheds light on the exploration of artificial general intelligence (AGI). It is promising to develop more smart intelligent systems (possibly with multi-modality signals) than ever. While, in this development process, AI safety should be one of the primary concerns, i.e., making AI lead to good for humanity but not bad [40].

**Coda:** This survey was planned during a discussion meeting held by our research team, and we aimed to summarize the recent advances of large language models as a highly readable report for our team members. The first draft was finished on March 13, 2023, in which our team members tried their best to include the related studies about LLMs in a relatively objective, comprehensive way. Then, we have extensively revised the writing and contents in several passes. While, this survey is still far from perfect: we are likely to miss important references or topics, and might also have non-rigorous expressions or discussions. We will continuously update this survey, and improve the quality as possible as we can. For us, survey writing is also a learning process for LLMs by ourselves. For readers with constructive suggestions to improve this survey, you are welcome to leave comments on the GitHub page of our survey or directly email our authors. We will make according revisions following the received comments or suggestions in a future version, and acknowledge the readers that have contributed constructive suggestions in our survey.

**Acknowledgments**
The authors would like to thank Yankai Lin and Yutao Zhu for their constructive feedback.

**References**


2022, Baltimore, Maryland, USA, pp. 4057–4086.


[290] Y. Gu, S. Kase, M. Vanni, B. M. Sadler, P. Liang, X. Yan,


Y. Ma, Y. Cao, Y. Hong, and A. Sun, “Large language model is not a good few-shot information extractor, but a good reranker for hard samples!” CoRR, vol. abs/2303.08559, 2023.


