A Survey of Large Language Models

Tran Bao Chi

April 21, 2023

Outline

Introduction

- Language Modeling (LM)
- Overview of LLMs

Language Modelling

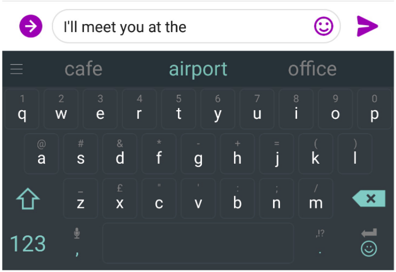


Figure 1: LM application

- Language Modeling is the task of predicting what word comes next
- A system that does this is called a Language Model

Language Modelling

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): e.g n-gram LM
- Neural language models (NLM): e.g Word2vec, Glove,...
- Pretrained language models (NLM): e.g BERT, GPT-2, BART,...
- Large language models (LLM): e.g PaLM, GPT-3,LLaMA, FlanT5,...

Introduction

- Language Modeling (LM)
- Overview of LLMs

Overview

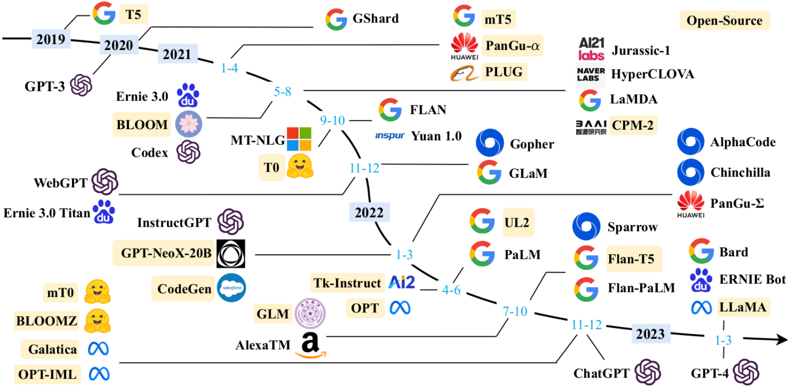


Figure 2: Models

Overview

- **Background:** Typically, large language models (LLMs) refer to language models that contain hundreds of billions (or more) of parameters (In existing literature, there is no a formal consensus on the min-imum parameter scale for LLMs. In this survey, we mainly focus on discussing LLMs with a model size larger than 10B.)
- **Overview**:
  - **In-context learning:** generate the expected output without requiring additional training or gradient updates (GPT-3, Why ICL?)
  - **Instruction following:** fine-tuning with a mixture of multi-task datasets formatted via instructions (1, 2, 3)
  - **Step-by-step reasoning** (COT)
- **Key Techniques for LLMs**
  - **Scaling:** model size, data size, and total compute, given a fixed budget; data collection and cleaning strategies (Chinchilla) recommends training a 10B model on 200B tokens, we find that the performance of a 7B model continues to improve even after 1T tokens. **However**, this objective disregards the inference budget (LLaMa)
  - **Training:** Distributed training + Optimization tricks (DeepSpeed)
  - **Ability eliciting:** COT, Instruction,...

Overview

- **Alignment tuning:** Instruction, RLHF,...
- **Tools manipulation:** e.g: ChatGPT plugins

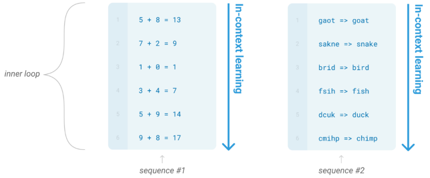


Figure 3: ICL

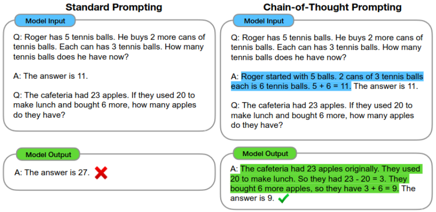


Figure 4: COT

Overview

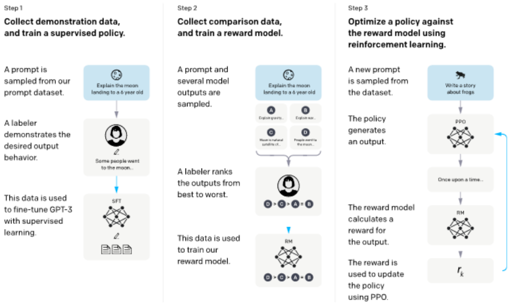


Figure 5: Instruction

Resources

We will mainly summarize the open- source model checkpoints or APIs, available corpora, and useful libraries for LLMs.

- Model
- Corpora
- Library

Models

The research of LM has received extensive research attention in the literature, which can be roughly divided into four major development stages:

- **Tens of Billions**: Flan-T5 (premier), LLaMA (Hoffman rule contradiction), CodeGen (Open-source + Benchmark), mT0 (multilingual). 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)
- **Hundreds of Billions:** OPT, GPT-3, BLOOM-¿BLOOMZ, PaLM)
- **Public APIs:** OpenAI

Operation	Parameters	FLOPs per Token
Embed	$(n_{\text{vocab}} + n_{\text{ctx}})\, d_{\text{model}}$	$4d_{\text{model}}$
Attention: QKV	$n_{\text{layer}} d_{\text{model}} 3 d_{\text{attn}}$	$2 n_{\text{layer}} d_{\text{model}} 3 d_{\text{attn}}$
Attention: Mask	—	$2 n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$
Attention: Project	$n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$	$2 n_{\text{layer}} d_{\text{attn}} d_{\text{embd}}$
Feedforward	$n_{\text{layer}} 2 d_{\text{model}} d_{\text{ff}}$	$2 n_{\text{layer}} 2 d_{\text{model}} d_{\text{ff}}$
De-embed	—	$2 d_{\text{model}} n_{\text{vocab}}$
Total (Non-Embedding)	$N = 2 d_{\text{model}} n_{\text{layer}} (2 d_{\text{attn}} + d_{\text{ff}})$	$C_{\text{forward}} = 2N + 2 n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$

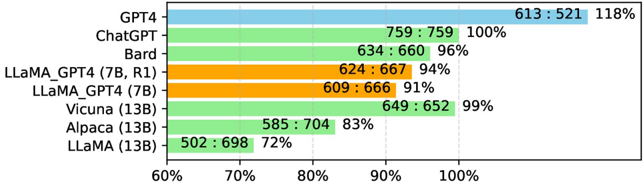
Table 1 Parameter counts and compute (forward pass) estimates for a Transformer model. Sub-leading terms such as nonlinearities, biases, and layer normalization are omitted.

Models

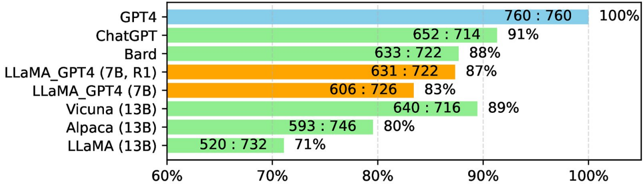
	Model	Release Time	Size (B)	Base Model	Adaptation IT	Adaptation RLHF	Pre-train Data Scale	Latest Data Timestamp	Hardware (GPUs / TPUs)	Training Time	Evaluation ICL	Evaluation CoT
Open Source	T5 [71]	Oct-2019	11	-	-	-	1T tokens	Apr-2019	1024 TPU v3	-	✓	-
	mT5 [72]	Mar-2021	13	-	-	-	1T tokens	Apr-2019	-	-	✓	-
	PanGu-α [73]	May-2021	13*	-	-	-	1.1TB	-	2048 Ascend 910	-	✓	-
	CPM-2 [74]	May-2021	198	-	-	-	2.6TB	-	-	-	-	-
	T0 [28]	Oct-2021	11	T5	✓	-	-	-	512 TPU v3	27 h	✓	-
	GPT-NeoX-20B [75]	Feb-2022	20	-	-	-	825GB	Dec-2022	96 40G A100	-	✓	-
	CodeGen [76]	Mar-2022	16	-	-	-	577B tokens	-	-	-	✓	-
	Tk-Instruct [77]	Apr-2022	11	T5	✓	-	-	-	256 TPU v3	4 h	✓	-
	UL2 [78]	Apr-2022	20	-	✓	-	1T tokens	Apr-2019	512 TPU v4	-	✓	✓
	OPT [79]	May-2022	175	-	-	-	180B tokens	-	992 80G A100	-	✓	-
	BLOOM [66]	Jul-2022	176	-	-	-	366B	-	384 80G A100	105 d	✓	-
	GLM [80]	Aug-2022	130	-	-	-	400B tokens	-	768 40G A100	60 d	✓	-
	Flan-T5 [81]	Oct-2022	11	T5	✓	-	-	-	-	-	✓	✓
	mT0 [82]	Nov-2022	13	mT5	✓	-	-	-	-	-	✓	-
	Galactica [35]	Nov-2022	120	-	-	-	106B tokens	-	-	-	✓	✓
	BLOOMZ [82]	Nov-2022	176	BLOOM	✓	-	-	-	-	-	✓	-
	OPT-IML [83]	Dec-2022	175	OPT	✓	-	-	-	128 40G A100	-	✓	✓
	LLaMA [57]	Feb-2023	65	-	-	-	1.4T tokens	-	2048 80G A100	21 d	✓	-
Closed Source	GShard [84]	Jan-2020	600	-	-	-	1T tokens	-	2048 TPU v3	4 d	-	-
	GPT-3 [55]	May-2020	175	-	-	-	300B tokens	-	-	-	✓	-
	LaMDA [85]	May-2021	137	-	-	-	2.81T tokens	-	1024 TPU v3	57.7 d	-	-
	HyperCLOVA [86]	Jun-2021	82	-	-	-	300B tokens	-	1024 A100	13.4 d	✓	-
	Codex [87]	Jul-2021	12	GPT-3	-	-	100B tokens	May-2020	-	-	✓	-
	ERNIE 3.0 [88]	Jul-2021	10	-	-	-	375B tokens	-	384 V100	-	✓	-
	Jurassic-1 [89]	Aug-2021	178	-	-	-	300B tokens	-	800 GPU	-	✓	-
	FLAN [62]	Oct-2021	137	LaMDA	✓	-	-	-	128 TPU v3	60 h	✓	-
	MT-NLG [90]	Oct-2021	530	-	-	-	270B tokens	-	4480 80G A100	-	✓	-
	Yuan 1.0 [91]	Oct-2021	245	-	-	-	180B tokens	-	2128 GPU	-	✓	-
	WebGPT [70]	Dec-2021	175	GPT-3	-	✓	-	-	-	-	✓	-
	Gopher [59]	Dec-2021	280	-	-	-	300B tokens	-	4096 TPU v3	920 h	✓	-
	ERNIE 3.0 Titan [92]	Dec-2021	260	-	-	-	300B tokens	-	2048 V100	28 d	✓	-
	GLaM [93]	Dec-2021	1200	-	-	-	280B tokens	-	1024 TPU v4	574 h	✓	-
	InstructGPT [61]	Jan-2022	175	GPT-3	✓	✓	-	-	-	-	✓	-
	AlphaCode [94]	Feb-2022	41	-	-	-	967B tokens	Jul-2021	-	-	-	-
	Chinchilla [34]	Mar-2022	70	-	-	-	1.4T tokens	-	-	-	✓	-
	PaLM [56]	Apr-2022	540	-	-	-	780B tokens	-	6144 TPU v4	-	✓	✓
	AlexaTM [95]	Aug-2022	20	-	-	-	1.3T tokens	-	128 A100	120 d	✓	✓
	Sparrow [96]	Sep-2022	70	-	-	✓	-	-	64 TPU v3	-	✓	-
	U-PaLM [97]	Oct-2022	540	PaLM	-	-	-	-	512 TPU v4	5 d	✓	✓
	Flan-PaLM [81]	Oct-2022	540	PaLM	✓	-	-	-	512 TPU v4	37 h	✓	✓
	Flan-U-PaLM [81]	Oct-2022	540	U-PaLM	✓	-	-	-	-	-	✓	✓
	GPT-4 [46]	Mar-2023	-	-	✓	✓	-	-	-	-	✓	✓
	PanGu-Σ [98]	Mar-2023	1085	PanGu-α	-	-	329B tokens	-	512 Ascend 910	100 d	✓	-

Figure 7: Details

Models



(c) All chatbots against ChatGPT



(d) All chatbots against GPT-4

Figure 4: Performance comparisons evaluated by GPT-4.

Figure 8: Compares

Models

	GPU Type	GPU Power consumption	GPU-hours	Total power consumption	Carbon emitted (tCO_2eq)
OPT-175B	A100-80GB	400W	809,472	356 MWh	137
BLOOM-175B	A100-80GB	400W	1,082,880	475 MWh	183
LLaMA-7B	A100-80GB	400W	82,432	36 MWh	14
LLaMA-13B	A100-80GB	400W	135,168	59 MWh	23
LLaMA-33B	A100-80GB	400W	530,432	233 MWh	90
LLaMA-65B	A100-80GB	400W	1,022,362	449 MWh	173

Figure 9: cost + co2 footprint

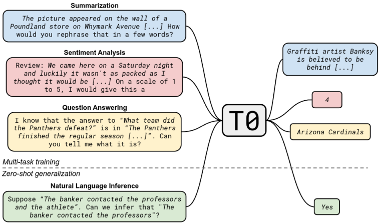
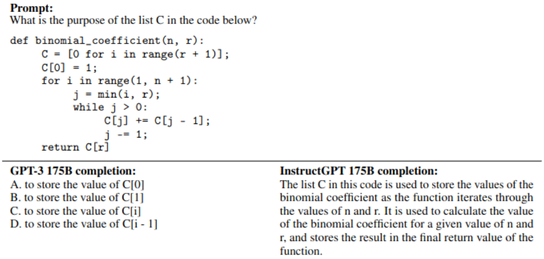


Figure 10: multi-task tuning

Models



Prompt:
What is the purpose of the list C in the code below?

```
def binomial_coefficient(n, r):
    C = [0 for i in range(r + 1)];
    C[0] = 1;
    for i in range(1, n + 1):
        j = min(i, r);
        while j > 0:
            C[j] += C[j - 1];
            j -= 1;
    return C[r]
```

GPT-3 175B completion:
A. to store the value of C[0]
B. to store the value of C[1]
C. to store the value of C[i]
D. to store the value of C[i - 1]

InstructGPT 175B completion:
The list C in this code is used to store the values of the binomial coefficient as the function iterates through the values of n and r. It is used to calculate the value of the binomial coefficient for a given value of n and r, and stores the result in the final return value of the function.

Figure 11: instruction tuning

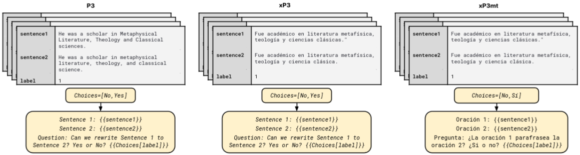


Figure 12: cross-lingual tuning

Resources

We will mainly summarize the open- source model checkpoints or APIs, available corpora, and useful libraries for LLMs.

- Model
- Corpora
- Library

Copora

- **Books**: BookCorpus, Project Gutenberg; Books1, Books2 (not public)
- **CommonCrawl**: C4, CC-News, ...
- **Reddit**
- **Wiki**
- **Code**: GitHub, Stackoverflow, BIGQUERY, ...
- **Others**: The Pile

Copora

Corpora	Size	Source	Latest Update Time
BookCorpus [100]	5GB	Books	Dec-2015
Gutenberg [101]	-	Books	Dec-2021
C4 [71]	800GB	CommonCrawl	Apr-2019
CC-stories-R [102]	31GB	CommonCrawl	Sep-2019
CC-NEWS [27]	78GB	CommonCrawl	Feb-2019
REALNEWs [103]	120GB	CommonCrawl	Apr-2019
OpenWebText [104]	38GB	Reddit links	Mar-2023
Pushift.io [105]	-	Reddit links	Mar-2023
Wikipedia [106]	-	Wikipedia	Mar-2023
BigQuery [107]	-	Codes	Mar-2023
the Pile [108]	800GB	Other	Dec-2020
ROOTS [109]	1.6TB	Other	Jun-2022

Figure 13: Details

• *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).

Figure 14: Typical models

Resources

We will mainly summarize the open- source model checkpoints or APIs, available corpora, and useful libraries for LLMs.

- Model
- Corpora
- Library

Library

- **Transformers**
- **DeepSpeed**
- **Megatron-LM**
- **Colossal-AI**
- **BMTrain**
- **FastMoE**

Pretraining

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.

- Data
- Architecture Design
- Training

Data

- **Source**
  - General: General-purpose pre-training e.g Webpages, Conversation text, Books,...
  - Specialized: useful for downstream task fine-tuning e.g Multilingual text, Scientific text, Code,...
- **Preprocessing**
  - classifier-based
  - heuristic-based

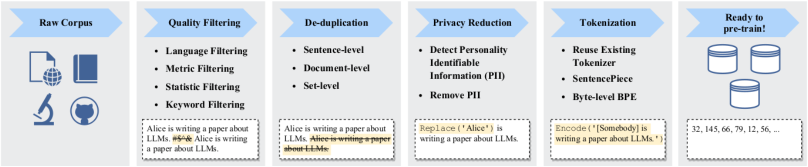


Figure 15: Pipeline

Data

- **Effect**
  - Mixture: While, as a side effect, training on excessive data about a certain domain would affect the generalization capability of LLMs on other domains
  - Amount: Chinchilla demonstrates that a number of existing LLMs suffer from sub-optimal training due to inadequate pre-training data. LLaMA shows that with more data and longer training, smaller models can also achieve good performance.
  - Quality: 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"

Data

9

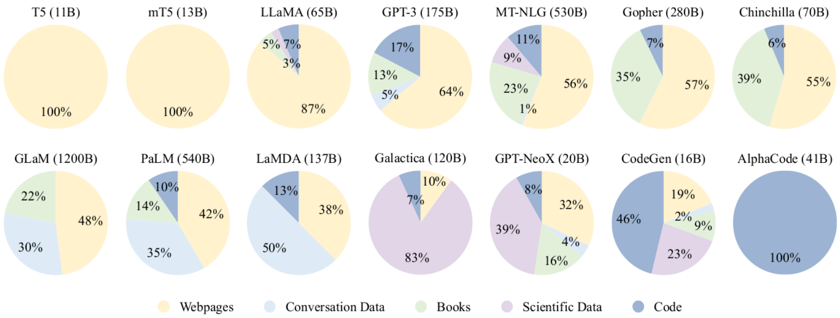


Figure 16: Data propotion

Pretraining

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.

- Data
- Architecture Design
- Training

Architecture

- **Encoder-Decoder:** So far, there are only a small number of LLMs that are built based on the encoder-decoder architecture, e.g., Flan-T5
- **Decoder only:**
  - **Casual Decoder:** So far, the casual decoders have been widely adopted as the architecture of LLMs by var- ious existing LLMs, such as GPT-3+, OPT, BLOOM and Gopher
  - **Prefix Decoder:** More like finetuning (U-PaLM)

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, e.g., Switch Transformer [25] and GLaM [93].

Config

- **Normalization:** Pre-Normalization (Stability + Decrease Performance), RMSNorm, DeepNorm, Embedding Norm (eliminated).
- **Activation function:** GeLU; SwiGLU, GeGLU (extra params 1.5x)
- **Position Embeddings:** Absolute PE, Relative PE, RoPE
- **Attention and Bias:** Sparse attention (GPT-3), FlashAttention (LLaMA), no biases can enhance training stability.

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.

Figure 17: Config

Objective

- **Language Modelling** 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
- **Denoising Autoencoding** 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.

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

Figure 18: Language Modelling

$$\mathcal{L}_{DAE}(\mathbf{x}) = \log P(\tilde{\mathbf{x}}|\mathbf{x}_{\setminus\tilde{\mathbf{x}}}).$$

Figure 19: DAE

Pretraining

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.

- Data
- Architecture Design
- Training

Optimization

- **Batch Training:** For language model pre-training, existing work generally sets the batch size to a large number (e.g., 8,196 examples or 1.6M tokens). The batch size of GPT-3 is gradually increasing from 32K to 3.2M tokens
- **Learning Rate:** Existing LLMs usually adopt a similar learn- ing rate schedule with the warm-up and decay strategies during pre-training
- **Optimizer:** AdamW optimizer [169] are widely utilized for training LLMs (e.g., GPT-3). Meanwhile, the Adafactor optimizer [170] has also been utilized in training LLMs (e.g., PaLM and T5)
- **Stabilizing the Training:** To address training instability, weight decay and gradient clipping have been widely utilized. To mitigate training loss spike, 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 (shrink).

Training Techniques

- **3D Parallelism:**
  - Data parallelism
  - Pipeline Parallelism
  - Tensor parallelism
- **ZeRO** To resolve it, the ZeRO technique aims to retain only a fraction of data on each GPU (DeepSpeed)
  - optimizer state partitioning
  - gradient partitioning
  - parameter partitioning
- **Mixed Precision Training** FP16, FP8, FP4, BF16

FYI: HuggingFace Parallelism

Training Techniques

Model	Batch Size (#tokens)	Learning Rate	Warmup	Decay Method	Optimizer	Precision Type	Weight Decay	Grad Clip	Dropout
GPT3 (175B)	32K→3.2M	6×10^{-5}	yes	cosine decay to 10%	Adam	FP16	0.1	1.0	-
PanGu-α (200B)	-	2×10^{-5}	-	-	Adam	-	0.1	-	-
OPT (175B)	2M	1.2×10^{-4}	yes	manual decay	AdamW	FP16	0.1	-	0.1
PaLM (540B)	1M→4M	1×10^{-2}	no	inverse square root	Adafactor	BF16	lr^2	1.0	0.1
BLOOM (176B)	4M	6×10^{-5}	yes	cosine decay to 10%	Adam	BF16	0.1	1.0	0.0
MT-NLG (530B)	64 K→3.75M	5×10^{-5}	yes	cosine decay to 10%	Adam	BF16	0.1	1.0	-
Gopher (280B)	3M→6M	4×10^{-5}	yes	cosine decay to 10%	Adam	BF16	-	1.0	-
Chinchilla (70B)	1.5M→3M	1×10^{-4}	yes	cosine decay to 10%	AdamW	BF16	-	-	-
Galactica (120B)	2M	7×10^{-6}	yes	linear decay to 10%	AdamW	-	0.1	1.0	0.1
LaMDA (137B)	256K	-	-	-	-	BF16	-	-	-
Jurassic-1 (178B)	32 K→3.2M	6×10^{-5}	yes	-	-	-	-	-	-
LLaMA (65B)	4M	1.5×10^{-4}	yes	cosine decay to 10%	AdamW	-	0.1	1.0	-
GLM (130B)	0.4M→8.25M	8×10^{-5}	yes	cosine decay to 10%	AdamW	FP16	0.1	1.0	0.1
T5 (11B)	64K	1×10^{-2}	no	inverse square root	AdaFactor	-	-	-	0.1
ERNIE 3.0 Titan (260B)	-	1×10^{-4}	-	-	Adam	FP16	0.1	1.0	-
PanGu-Σ (1.085T)	0.5M	2×10^{-5}	yes	-	Adam	FP16	-	-	-

Figure 20: Hyperparams

Adaptation Tuning

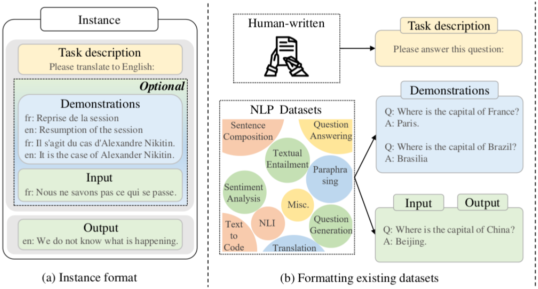
After pre-training, LLMs can acquire the general abilities for solving various tasks. However, increasing studies have shown that LLM's abilities can be further adapted according to specific goals. In this section, we introduce two major ap- proaches to adapting pre-trained LLMs, namely *instruction tuning* and *alignment tuning*.

- Instruction Tuning
- Alignment Tuning

Instruction Tuning

- **Formatted Instance Construction**
  - Existing Datasets: re-format and inverted format e.g *"Please answer this question" is added for each example in the question-answering task*, *"Please generate a question based on the answer:"*
  - Human Needs: Human annotation, Reject High-risk instructions (GPT-4).
  - Key factors: Instruction Scaling + Design

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].



(a) Instance format (b) Formatting existing datasets

Figure 21: Paper Recommendation

Instruction Tuning

Collections	Time	#Task types	#Tasks	#Examples
Nat. Inst. [186]	Apr-2021	6	61	193K
CrossFit [187]	Apr-2021	13	160	7.1M
FLAN [62]	Sep-2021	12	62	4.4M
P3 [188]	Oct-2021	13	267	12.1M
ExMix [189]	Nov-2021	11	107	18M
UnifiedSKG [190]	Jan-2022	6	21	812K
Super Nat. Inst. [77]	Apr-2022	76	1616	5M
MVPCorpus [191]	Jun-2022	11	77	41M
xP3 [82]	Nov-2022	17	85	81M
OIG[15]	Mar-2023	-	-	43M

Figure 23: Instruction Data

Table 6: Dataset sizes, in terms of number of prompts.

SFT Data			RM Data			PPO Data		
split	source	size	split	source	size	split	source	size
train	labeler	11,295	train	labeler	6,623	train	customer	31,144
train	customer	1,430	train	customer	26,584	valid	customer	16,185
valid	labeler	1,550	valid	labeler	3,488			
valid	customer	103	valid	customer	14,399			

Figure 24: InstructGPT Data

Instruction Tuning

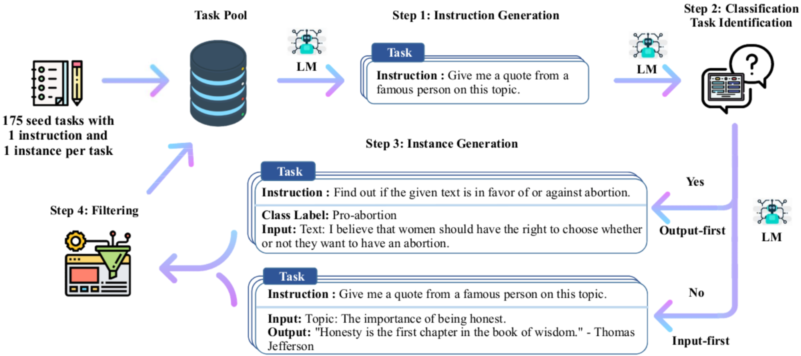


Figure 25: self-instruct

Instruction Tuning

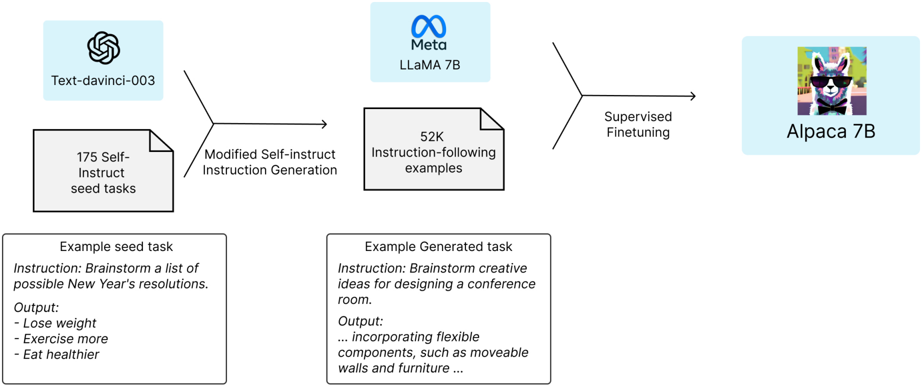


Figure 26: Alpaca

Instruction Tuning

- **Instruction Tuning Strategies**
  - Balancing the Data Distribution: examples-proportional mixing strategy, increasing the sampling ratio of high-quality collections, maximum cap to control the maximum number of examples
  - Combining Instruction Tuning and Pre-Training: GLM-130B [80] and Galactica [35] integrate instruction-formatted datasets as a small proportion of the pre-training corpora to pre-train LLMs
- **The Effect of Instruction Tuning**
  - Performance Improvement: 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. Smaller models with instruction tuning can even perform better than larger models without fine-tuning; it is also more efficient than pre-training.
  - Task Generalization: generalize to related tasks across languages; effectiveness of instruction tuning to achieve superior performance on both seen and unseen tasks

Adaptation Tuning

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 ap- proaches to adapting pre-trained LLMs, namely *instruction tuning* and *alignment tuning.*

- Instruction Tuning
- Alignment Tuning

Alignment Tuning

- **Background and Criteria**
  - Background: However, these models may sometimes exhibit unintended behav- iors, e.g., fabricating false information, pursuing inaccurate objectives, and producing harmful, misleading, and biased expressions; alignment requires considering very different criteria; might harm LLMs performance. 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.
  - Criteria: Helpfulness, Honesty, Harmlessness

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.

Alignment Tuning

- **RLHF System**
  - Supervised fine-tuning: the first step is not necessarily used and can be optional in specific settings or scenarios.
  - Reward model training: the RM (i.e., 6B GPT-3) is trained to predict the human-preferred output.
  - RL fine-tuning: Aligning (i.e., fine-tuning) the LM is formalized as an RL problem. To avoid deviating significantly from the initial (before tuning) LM, a penalty term is commonly incorporated into the reward function.

It is noted that the second and final steps can be iterated in multiple turns for better aligning LLMs.

Figure 27: Tweet

Alignment tuning

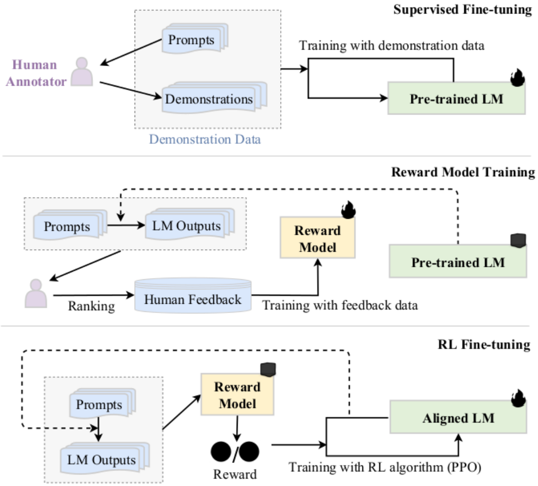


Figure 28: RHLF

Utilization

After pre-training or adaptation tuning, a major approach to using LLMs is to design suitable prompting strategies.

- In-Context Learning
- Chain-of-Thought Prompting

ICL

- **Formulation**

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

Figure 29: Caption

- **Demonstration Design**
  - Selection: Heuristic approaches, LLM-based approaches for examples retrieval.
  - Format: To construct more informative tem- plates, recent studies consider adding task descriptions [81] or enhancing the reasoning capability of LLMs with chain- of-thought prompts
  - Order: LLMs are shown to sometimes suffers from the recency bias

ICL

- **Underlying Mechanism**
  - Pretraining: Training tasks, Scaling, Copora
  - Why ICL? Meta Optimization, Dual Form, Stanford, Bayesian Inference, Induction Heads, LLMs can effectively learn Linear Function and even some complex functions like Decision tree with ICL

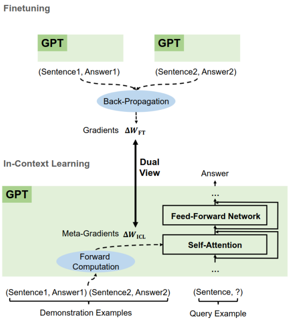


Figure 30: Meta-optimization

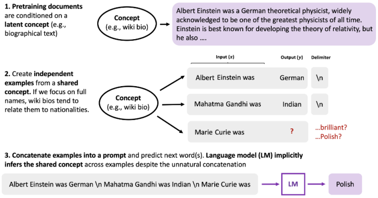


Figure 31: Bayesian Inference

Utilization

After pre-training or adaptation tuning, a major approach to using LLMs is to design suitable prompting strategies.

- In-Context Learning
- Chain-of-Thought Prompting

In-context Learning with CoT

- **Few-shot CoT**: multiple reasoning paths for each problem; the ordering of demonstrations seems to have a relatively small impact compared to the standard prompt in ICL (less than 2
- **Zero-shot CoT** Strategy drastically boosts the performance when the model scale exceeds a certain size, but is not effective with small-scale models; "Let's think step by step" to generate reasoning steps and then prompted by "Therefore, the answer is" to derive the final answer.
- Why ? Pretraining data, Prompting components

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].

Figure 32: CoT

Evaluation

- Tasks(LM)
- Benchmarks and Empirical Analysis

Tasks

- **Basic**

Task		Dataset
Language Generation	Language Modeling	Penn Treebank [262], WikiText-103 [263], the Pile [108], LAMBADA [147]
	Conditional Text Generation	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]
	Code Synthesis	APPS [280], HumanEval [87], MBPP [133], CodeContest [94], MTPB [76], DS-1000 [281], ODEX [282]
Knowledge Utilization	Closed-Book QA	Natural Questions [283], ARC [284], TruthfulQA [285], Web Questions [286], TriviaQA [287], PIQA [288], LC-quad2.0 [289], GrailQA [290], KQApro [291], CWQ [292], MKQA [293], ScienceQA [294]
	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]
Complex Reasoning	Knowledge Reasoning	CSQA [240], StrategyQA [241], ARC [284], BoolQ [308], PIQA [288], SIQA [309], HellaSwag [310], WinoGrande [311], OpenBookQA [295], COPA [312], ScienceQA [294], proScript [313], ProPara [314], ExplaGraphs [315], ProofWriter [316], EntailmentBank [317], ProOntoQA [318]
	Symbolic Reasoning	CoinFlip [33], ReverseList [33], LastLetter [33], Boolean Assignment [319], Parity [319], Colored Object [320], Penguins in a Table [320], Repeat Copy [68], Object Counting [68]
	Mathematical Reasoning	MATH [277], GSM8k [237], SVAMP [238], MultiArith [321], ASDiv [239], MathQA [322], AQUA-RAT [323], MAWPS [324], DROP [325], NaturalProofs [326], PISA [327], miniF2F [328], ProofNet [329]

Figure 33: Basic Tasks

Tasks

- **Advanced**
  - Human Alignment: TruthfulQA (helpfulness + honesty), CrowS-Pairsm, Winogender(harmlessness)
  - External Environment Interaction: BEHAVIOR, ALFRED; e.g., generating action plans in natural language to manipulate agents
  - Tool Manipulation: turn to external tools if they determine it is necessary.

Evaluation

- Tasks(LM)
- Benchmarks and Empirical Analysis

Benchmarks

- **MMLU**: Large-scale evaluation of multi-task knowledge understanding of mathematics and computer science to humanities and social sciences.
- **BIG-bench**: 204 tasks that encompass a broad range of topics, including linguistics, childhood development, mathematics, commonsense reasoning,...
- **HELM**: holistic evaluation with a core set of 16 scenarios and 7 categories stanford HELM

Analysis

- **Generalist.**
  - Mastery: GPT-4 approaches human-level performance in a variety of challenging tasks (e.g., mathematics, vision, and coding), and considered it as "an early version of an artificial general intelligence system". Despite the promising results, this analysis has also revealed that GPT-4 still has severe limitations.
  - Robustness: Concretely, LLMs are prone to provide different answers when using varied expressions of the same input
- **Specialist.**
  - Healthcare: It has been shown that LLMs are capable of han- dling a variety of healthcare tasks, e.g., biology information extraction,... However, fabricate medical misinfor- mation, raise privacy concerns.
  - Education: an important application domain where LLMs potentially exert significant influence. 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: Recently, a number of studies have ap- plied LLMs to solve various legal tasks, e.g., legal document analysis, ... However, the use of LLMs in law also raises concerns about legal challenges, including copyright issues, personal information leakage, or bias and

Fine-tuning

- **Tradition**

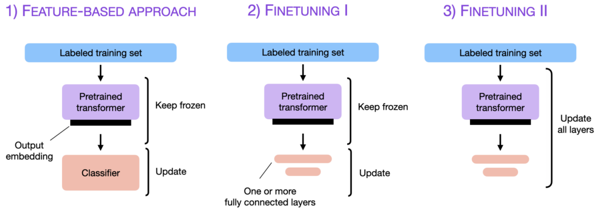


Figure 34: fine-tune

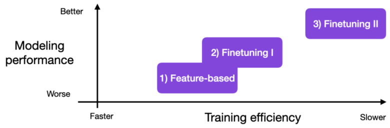


Figure 35: performance

Fine-tuning

- **Params-efficient**

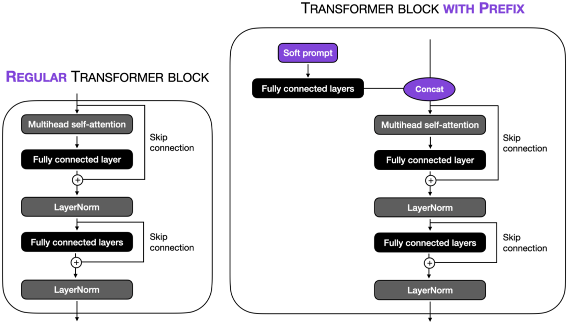


Figure 36: prefix

Fine-tuning

- **Params-efficient**

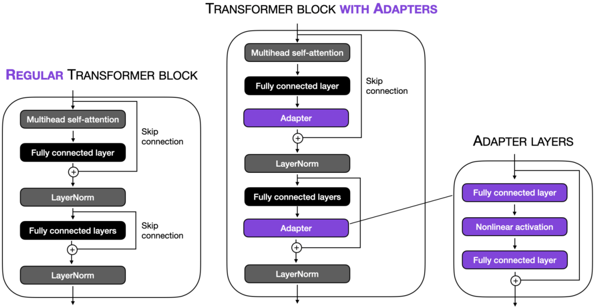


Figure 37: adapter

Fine-tuning

- **Params-efficient**

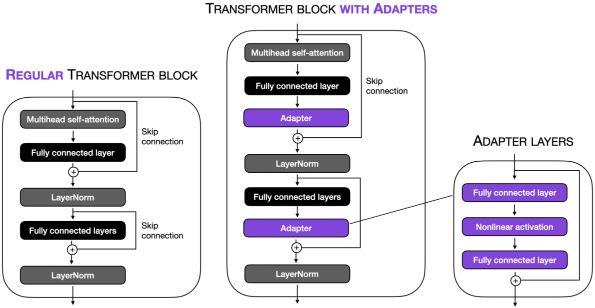


Figure 38: adapter

Fine-tuning

- **Params-efficient**

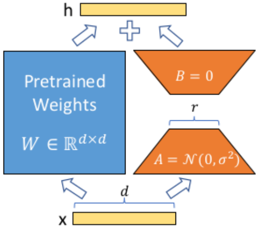


Figure 39: lora

New concepts