

A Survey on Large Language Models with some Insights on their Capabilities and Limitations

Andrea Matarazzo
Expedia Group
Italy

a.matarazzo@gmail.com

Riccardo Torlone
Roma Tre University
Italy

riccardo.torlone@uniroma3.it

Abstract

The rapid advancement of artificial intelligence, particularly with the development of Large Language Models (LLMs) built on the transformer architecture, has redefined the capabilities of natural language processing. These models now exhibit remarkable performance across various language-related tasks, such as text generation, question answering, translation, and summarization, often rivaling human-like comprehension. More intriguingly, LLMs have demonstrated emergent abilities extending beyond their core functions, showing proficiency in tasks like commonsense reasoning, code generation, and arithmetic.

This survey paper explores the foundational components, scaling mechanisms, and architectural strategies that drive these capabilities. Emphasizing models like GPT and LLaMA, we analyze the impact of exponential data and computational growth on LLM performance, while also addressing the trade-offs associated with scaling. We also examine LLM applications across sectors, such as healthcare, finance, education, and law, highlighting their adaptability and potential to solve domain-specific challenges.

Central to this work are the questions of how LLMs generalize across diverse tasks, exhibit planning, and reasoning abilities, and whether these emergent abilities can be systematically elicited or enhanced. In particular, we provide some insights into the CoT (Chain of Thought) and PoT (Plan of Thought) abilities within LLMs, focusing on how pre-training data influences their emergence. Additionally, we investigate LLM-modulo frameworks that integrate external systems, allowing LLMs to handle complex, dynamic tasks. By analyzing these factors, this paper aims to foster the ongoing discussion on the capabilities and limits of LLMs, promoting their responsible development and application in novel and increasingly complex environments.

Contents

1	Introduction	4
1.1	Motivations	4
1.2	Goals of the paper	4
1.3	Content and organization	5
2	Large Language Models	6
2.1	Definition and Overview	6
2.2	Scaling Law	10
2.3	Prominent Model Families	12
2.3.1	BERT	12
2.3.2	T5	14
2.3.3	GPT Series	15
2.3.4	Llama	19
2.3.5	Gemma	23
2.3.6	Claude	25
2.4	Specialized Large Language Models	26
2.4.1	LLMs in Healthcare	27
2.4.2	LLMs in Finance	28
2.4.3	LLMs in Education	38
2.4.4	LLMs in Law	39
2.4.5	LLMs in Scientific Research	40
3	Foundations of Large Language Models	42
3.1	Pre-training	42
3.1.1	Unsupervised pre-training	43
3.1.2	Supervised pre-training	43
3.1.3	Semi-supervised pre-training	44
3.2	Data sources	46
3.2.1	General Data	46
3.2.2	Specialized Data	47
3.2.3	Commonly-used data sources.	48
3.3	Data preprocessing	50
3.3.1	Quality Filtering.	50
3.3.2	Deduplication.	51
3.3.3	Privacy reduction.	51
3.3.4	Tokenization.	51
3.4	LLM Adaptation	53
3.4.1	Instruction Tuning	53
3.4.2	Alignment Tuning	60
3.5	Architecture	62
3.5.1	Encoder-decoder	63
3.5.2	Casual decoder	64
3.5.3	Prefix decoder	64
3.5.4	Transformer Architecture	65
3.5.5	Emerging architectures	74
3.6	Tuning and Optimization	75
3.6.1	Parameter-efficient model adaptation	75
3.6.2	Memory-efficient model adaptation	80

4 Utilization Strategies and Techniques	81
4.1 In-Context Learning	81
4.1.1 ICL strategy	81
4.1.2 ICL performance and origins	86
4.1.3 ICL future research	88
4.2 Chain-of-Thought	89
4.2.1 CoT strategy	89
4.2.2 CoT performance and origins	93
4.3 Program-of-Thoughts	94
4.4 Planning for complex tasks	95
4.4.1 Commonsense knowledge	95
4.4.2 Prompt and code based planning	99
4.4.3 Plan generation	100
4.4.4 Feedback and plan refinement	108
4.4.5 LLM-modulo Framework	114
4.5 Retrieval-Augmented Generation	117
5 Testing the CoT Capabilities of LLMs	124
5.1 What is eliciting the Chain-of-Thought?	124
5.2 Empirical evidences	125
5.3 Prompting	127
5.4 Examples of generated text	127
6 Conclusions	150

1 Introduction

1.1 Motivations

In recent years, the field of artificial intelligence has witnessed an extraordinary transformation, fueled mainly by the development of Large Language Models (LLMs) based on the Transformer architecture. These models, exemplified by OpenAI’s GPT series and Meta’s LLaMA, have revolutionized how we approach natural language processing tasks, achieving comprehension, learning, and generation levels that were once considered unattainable. Their impressive performance spans a variety of tasks, including text generation, question answering, language translation, and summarization, showcasing their potential in tackling intricate language challenges. Surprisingly, these models have also exhibited some abilities that go beyond their primary task of text generation, such as commonsense reasoning, code generation, arithmetic operations, and other complex tasks in various domains.

Several key factors have driven the evolution of LLMs, most notably the exponential growth in available data and computational resources. Indeed, on the one hand, social media platforms, digital libraries, and other sources have provided vast amounts of textual and multimedia information, enabling LLMs to be trained on extensive and diverse datasets. On the other hand, the availability of powerful GPUs, TPUs, and distributed computing frameworks has made it feasible to train models with billions, and even trillions, of parameters. Together, these two factors have led LLMs to capture nuanced linguistic patterns, cultural context, and domain-specific knowledge, enhancing their ability to generate coherent, contextually appropriate, and highly versatile outputs.

However, with their increasing complexity and capabilities, these models have introduced new challenges and raised critical questions about their applicability, limitations, and potential for future development. Questions surrounding their ethical use and long-term impact not only to the AI landscape but also to our own lives have become central to discussions about their future. Addressing these concerns is critical as researchers and practitioners continue to explore the transformative possibilities that LLMs can offer.

1.2 Goals of the paper

The goal of this paper is twofold.

We first aim to provide an in-depth survey on LLMs and their applications, beginning with a foundational overview of their development, pre-training strategies, and architectural variations. This includes an examination of the progression from early language models to the sophisticated architectures of LLMs, such as BERT, GPT, and Llama. In particular, we explore the concept of scaling laws, which have been instrumental in understanding how the size and complexity of LLMs contribute to their performance and capabilities, as well as the trade-offs and challenges associated with building increasingly larger and more powerful models. We will also investigate their application across various domains, such as healthcare, finance, education, law, and scientific research. Each of these domains presents unique challenges and opportunities for LLMs, highlighting the versatility and adaptability of these models. For instance, in healthcare, LLMs have shown promise in assisting with clinical decision-making, while in finance, they are being utilized for tasks such as sentiment analysis and market prediction.

The second objective of the present paper is to deepen some of the mechanisms that enable LLMs to perform tasks previously deemed impossible for machine learning systems. In particular, we will try to address some fundamental questions. How do these models learn and generalize across tasks and domains? What are these emergent abilities, and how can they be elicited? Which factors contribute to their development (e.g., model size, data, architecture)? What are the inherent limitations of these models and how can they be addressed?

The central motivation of this work is therefore to investigate the current capabilities and boundaries of LLMs, focusing on their ability to generalize, plan, and execute tasks autonomously.

1.3 Content and organization

Below, is a summary of the paper organized by its structure.

- Section 2 introduces LLMs, tracing their development from early statistical language models to modern transformer-based architectures. It underscores the significant role of the scaling law in LLM development, where increasing model size, data volume, and computational resources leads to substantial performance enhancements across a wide range of language tasks. The section also illustrates prominent LLM families like BERT, T5, GPT series, and LLaMA, highlighting their distinctive architectures, strengths, and contributions to the advancement of natural language processing. Additionally, it emphasizes the transformative impact of LLMs across various domains, including healthcare, finance, education, law, and scientific research.
- Section 3 focuses on the fundamental building blocks of LLMs, covering data preprocessing techniques, pre-training methodologies, and model adaptation strategies. It explores various pre-training approaches, including unsupervised, supervised, and semi-supervised learning, emphasizing their impact on model performance and adaptability. The section also examines different data sources used in LLM training, categorizing them into general data like Web pages, books, and conversation text, specialized data such as scientific literature and code, and widely used datasets like Wikipedia, BookCorpus, and CommonCrawl. It details the critical data preprocessing steps, such as quality filtering, data cleaning, deduplication, and tokenization, and their role in preparing data for effective LLM training. Moreover, it discusses model adaptation techniques like instruction tuning and alignment tuning, which fine-tune models for specific tasks and align their behaviour with desired human values. Crucially, the section provides a comprehensive analysis of the Transformer architecture, the dominant framework for modern LLMs, detailing its components (encoder, decoder, self-attention mechanisms), normalization methods, activation functions, positional embeddings, and optimization strategies.
- Section 4 addresses the effective strategies and techniques for utilizing LLMs, emphasizing in-context learning (ICL), chain-of-thought prompting (CoT), and planning capabilities. It explains ICL as a unique prompting technique that empowers LLMs to learn from examples presented within the prompt, allowing them to tackle new tasks without requiring explicit gradient updates. It elaborates on various ICL strategies, such as demonstration design, prompt engineering, and the selection of appropriate scoring functions, while also exploring the factors influencing ICL performance. It then introduces CoT prompting as a powerful method for enhancing LLM reasoning abilities. This involves integrating intermediate reasoning steps within the prompt, guiding the model to adopt a structured thought process, particularly beneficial for tasks requiring logical deduction, problem-solving, and mathematical calculations. Finally, the section explores the planning capabilities of LLMs, focusing on prompt-based planning. This technique involves decomposing complex tasks into manageable sub-tasks and generating a plan of action for execution. Different planning approaches, including text-based and programmatic methods, are discussed and the critical role of feedback and plan refinement mechanisms in achieving successful plan execution is highlighted.

- Section 5 investigates the origins of CoT capabilities in LLMs, exploring the hypothesis that the presence of code in pre-training data may contribute to the emergence of these reasoning abilities. For this, it presents empirical evidence obtained from experiments conducted on publicly available Llama family models using LMStudio software on the HuggingFace platform. The analysis focuses on the performance of these models on reasoning tasks derived from the GSM8k and gsm-hard datasets, evaluating their capabilities in utilizing CoT and Program of Thought (PoT) approaches.
- Finally, section 6 summarizes the key points of the paper, reiterating the transformative potential of LLMs across diverse fields. It also acknowledges the existing ethical, technical, and practical challenges associated with LLM development and advocates for continued research to ensure their responsible and beneficial application in the future.

2 Large Language Models

2.1 Definition and Overview

At their core, LLMs are designed to comprehend, learn, and generate coherent and contextually relevant language on an unparalleled scale.

Historically, the development of Language Models (LMs) has been rooted in the quest to understand and replicate human language, and four main stages can be identified:

1. **Statistical Language Models:** These models were developed to capture the statistical properties of language, such as word frequencies and co-occurrences, to predict the likelihood of a given sequence of words based on the Markov assumption, which states that the probability of a word depends only on the previous n words. If the context length n is fixed, the model is called an n -gram model.
However, these models are limited by the exponential number of transition probabilities to be estimated and the Markov assumption¹, which may not always hold true in the complexity of natural languages. Language understanding often involves capturing dependencies over longer distances than the Markov assumption allows. Models considering broader contexts, such as recurrent neural networks (RNNs) and transformers, have been developed to address these long-range dependencies in language processing tasks.
2. **Neural Language Models:** The advent of neural networks led to the development of language models that utilised neural architectures to capture language's complex patterns and dependencies. These models, such as recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, could capture long-range dependencies and contextual information, enabling them to generate coherent and contextually relevant text. Bengio et al. [6] introduced the concept of *distributed representation* of words and built the word prediction function of the distributed word vectors. Later, word2vec [21, 22] introduced the word2vec model, a shallow, two-layer neural network trained to reconstruct the linguistic contexts of words. These models were a significant leap forward in the development of language models, representing a shift from word sequencing to learning representation.

¹The Markov assumption proposes that the future state of a process relies solely on the current state by disregarding the journey to the current state. Mathematically, it is expressed in terms the conditional probability $P(S_t)$ – the likelihood of an event occurring given the past states – as $P(S_{t+1}|S_t, S_{t-1}, \dots, S_1) = P(S_{t+1}|S_t)$, where S_t is the state at time t . The Markov assumption simplifies the modelling process by reducing the number of parameters to estimate.

3. **Pre-trained language models (PLM):** The development of pre-trained language models (PLMs) marked a significant milestone in the evolution of language models. These models were trained on large data corpora in an unsupervised or self-supervised manner before being fine-tuned on specific tasks. The idea is to pre-train a model on a diverse data set and then transfer its knowledge to a narrower task by fine-tuning it on a smaller, task-specific dataset. ELMo ² [50] was one of the first PLMs which used a bidirectional LSTM to generate word embeddings instead of learning fixed word representations. Devlin et al. [65] introduced BERT (Bidirectional Encoder Representations from Transformers), a transformer-based model pre-trained on a large corpus of text and then fine-tuned it on specific tasks. BERT was a significant advancement in natural language processing, as it demonstrated the potential of pre-trained language models to achieve state-of-the-art performance on a wide range of tasks. These studies introduced the “*pre-training and fine-tuning*” paradigm, which has become a standard practice in the development of language models and inspired a significant number of models, such as GPT-2 [75]), GPT-3 (Brown et al. [88]), T5 (Raffel et al. [99], and many others.
4. **Large Language Models (LLM):** The emergence of large language models, characterised by their immense scale and complexity, has redefined the capabilities of language processing systems. Studies find that language models’ performance improves as the number of parameters (e.g., model size) or data size increases, a phenomenon known as the scaling law in large language models. Many LLMs are built on the transformer architecture, designed to capture long-range dependencies and contextual information in language. The transformer architecture has become the foundation for many state-of-the-art language models. Unlike earlier models that were unidirectional (e.g., traditional RNNs), LLMs, especially those based on transformers, are bidirectional. They consider the context of preceding and following words, enhancing their language understanding. LLMs find applications across various domains, including but not limited to:

- **Text Generation:** Producing coherent and contextually relevant text.
- **Question Answering:** Answering questions based on provided context.
- **Language Translation:** Translating text from one language to another.
- **Summarization:** Creating concise summaries of longer texts.
- **Sentiment Analysis:** Determining the sentiment expressed in a text.

These large-sized PLMs have been shown to outperform their smaller (e.g., 330M-parameters vs 1.5B-parameters) and show surprising capabilities ³, also called emergent abilities by Wei et al. [232].

Emergence is when quantitative changes in a system result in qualitative changes in behavior [1].

These emergent abilities include but are not limited to, the ability to perform tasks for which they were not explicitly trained, such as translation, summarisation, and question-answering, and to generalise to new tasks and domains, such as zero-shot learning ⁴,

²Embeddings from Language Models

³Note that a LLM is not necessarily more capable than a small PLM, and emergent abilities may not occur in some LLMs.

⁴It refers to a machine learning scenario where a model makes predictions or performs tasks for classes or examples it has never seen during training

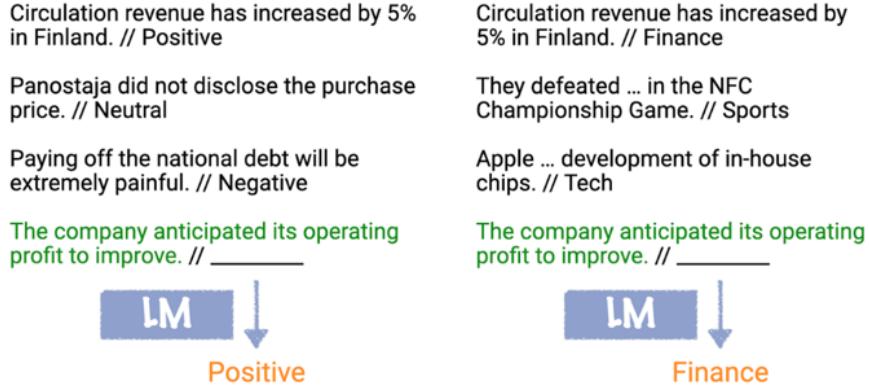


Figure 1: Two examples of in-context learning, where a language model (LM) is given a list of training examples (black) and a test input (green) and asked to make a prediction (orange) by predicting the next tokens/words to fill in the blank. Source: Lab [288].

few-shot learning⁵, and even one-shot⁶ learning⁷. Three typical examples of emergent abilities are:

- (a) **In-context learning:** this ability has been formally observed in GPT-3, which is provided with a natural language instruction or task demonstrations; it can generate the expected output for test instances by completing the word sequence of the input text (as shown in Figure 1). Importantly, this can be achieved without requiring additional training or gradient updates⁸. The surprising fact is that the LM isn't trained to learn from examples. Because of this, there's seemingly a mismatch between pretraining (what it's trained to do, which is next token prediction) and in-context learning (what we're asking it to do).
- (b) **Instruction following:** Through the process called instruction tuning – that we will see more in-depth in Section 3.4.1 – LLMs exhibit strong performance on unseen tasks described through natural language instructions [209, 205, 231]. This approach involves fine-tuning the model using diverse multitask datasets, each accompanied by detailed natural language descriptions. The result is an LLM that effectively interprets and follows instructions for new and unseen tasks without relying on explicit examples. Experiments detailed in Wei et al. [231] demonstrate that LaMDA-PT, fine-tuned with instructions, begins to outperform its untuned counterpart significantly when the model size reaches 68 billion parameters. However, this performance gain is not observed for 8 billion or smaller model sizes. Furthermore, Chung et al. [156] highlights that a model size of at least 62 billion parameters is necessary for PaLM to excel across various tasks in evaluation benchmarks like MMLU, BBH, TyDiQA, and MGSM. Nevertheless, it is noted that certain specific tasks, such as MMLU, might suffice with much smaller model size, emphasising the nuanced relationship between model size and task performance.

⁵It involves training a model with a minimal number of examples per class, usually much fewer than what traditional machine learning models require

⁶It is a specific case of few-shot learning where the model is trained with only one example per class

⁷A shot is an example or demonstration of what type of prompt and response you expect from a large language model. This term originates from training computer vision models on photographs, where one shot was one example or instance that the model used to classify an image [12].

⁸Dai et al. [158] shows that in-context learning implicitly performs meta optimisation through the attention mechanism

- (c) **Step-by-step reasoning:** For small LMs, it is usually difficult to solve complex tasks that involve multiple reasoning steps (e.g., mathematical word problems). In contrast, the chain-of-thought (CoT) prompting strategy [230] empowers Large Language Models (LLMs) to surmount these challenges. By leveraging the CoT prompting mechanism, which involves intermediate reasoning steps to derive the final solution, LLMs exhibit proficiency in tasks requiring intricate cognitive processes. This capability is speculated to be honed through training on code by Wei et al. [230]. Authors demonstrate that the employment of CoT prompting yields performance gains, particularly on arithmetic reasoning benchmarks, when applied to variants of models like PaLM and LaMDA, especially with a model size surpassing 60B. The advantages of CoT prompting become more pronounced as the model size exceeds 100B. Furthermore, the effectiveness of CoT prompting exhibits variability across different tasks, with performance improvement observed in the order of GSM8k > MAWPS > SWAMP for PaLM [230]. Recent studies have shown that size is not a deciding factor in the model’s ability to perform step-by-step reasoning tasks. We will investigate this further in Section 4.2.2.
5. **Small Language Models:** Small Language Models (SLMs) are a rapidly emerging subset of artificial intelligence designed to provide efficient natural language processing (NLP) capabilities. As outlined in IBM’s analysis, SLMs operate with a fraction of the parameters used by large language models (LLMs), ranging from a few million to several billion parameters. This reduction in size allows them to function in resource-constrained environments such as edge devices, mobile platforms, and offline scenarios, where computational resources and connectivity may be limited. SLMs, like their larger counterparts, leverage a transformer architecture. To reduce model size while retaining functionality, model compression techniques are applied. These include:
- (a) Pruning: Eliminating redundant parameters from neural networks to simplify computations while preserving core performance.
 - (b) Quantization: Representing model weights and activations in lower precision (e.g., 8-bit integers) to improve speed and reduce memory usage.
 - (c) Low-Rank Factorization: Decomposing weight matrices into simpler approximations to lower computational demands.
 - (d) Knowledge Distillation: Transferring knowledge from larger “teacher models” to smaller “student models”, enabling compact versions to retain critical features.

A wide range of SLMs are gaining traction due to their adaptability and efficiency. Some notable examples include DistilBERT, Google Gemma, Minstral and others. SLMs are particularly suited to scenarios where computational efficiency and adaptability are paramount, such as edge computing, mobile applications, and offline settings. The development of Small Language Models marks a transformative step in AI, emphasizing efficiency and accessibility without sacrificing core capabilities. As model compression techniques continue to evolve, SLMs are poised to play a crucial role in shaping the future of AI deployment across diverse domains.

The advent of LLMs has led to a paradigm shift in the field of natural language processing, with applications ranging from machine translation to text summarisation and from question-answering systems to language generation. The development of LLMs has been driven by the exponential growth of data and computational resources, which has enabled the training of models with billions of parameters. The scale of these models has enabled them to capture complex patterns in language and generate coherent and contextually relevant text.

The potential of LLMs is vast, and their impact on natural language processing is profound. The advent of ChatGPT [86] and GPT-4 [370] has further expanded the capabilities of LLMs, leading to the rethinking of the possibilities of artificial general intelligence (AGI).

Regarding NLP, LLMs can serve somewhat as general-purpose language task solvers. In the IR field, LLMs can be used to improve the performance of information retrieval systems through AI chatbots (i.e., ChatGPT), or integrating search engines like the New Bing⁹ or using RAG¹⁰ [375] pipelines. RAG addresses these challenges by combining LLMs with external knowledge bases. This integration allows models to retrieve relevant information during generation, enhancing accuracy and credibility.

In the CV field, LLMs can be used to improve the performance of computer vision systems through multimodal models¹¹ (i.e., CLIP¹² [130] and DALL-E [132]).

This work will mainly focus on model sizes larger than 10B parameters to explore their capabilities, limitations, and potential applications. We will delve into the emergent abilities of LLMs, such as in-context learning, instruction following, and step-by-step reasoning, and how these abilities can be leveraged to solve complex tasks in Section 4. The study will investigate and compare the abilities of different LLMs, focusing on the impact of various parameters on their performance.

LLMs are not without challenges, including ethical concerns, environmental impact, and the potential for bias and hallucination in generated text.

2.2 Scaling Law

The Scaling Law in LLMs constitutes a fundamental principle underlining their development and performance. At its essence, the scaling law posits that as language models increase in size, their capabilities and performance on linguistic tasks exhibit disproportionately positive growth. This concept has become a guiding force in pushing the boundaries of language processing and understanding.

As LLMs scale up in terms of parameters, encompassing tens or hundreds of billions, or even trillions, they demonstrate an unprecedented ability to generalise from diverse datasets and generate contextually coherent text. The essence of the scaling law lies in the direct correlation between the size of a language model and the number of parameters it encompasses. Parameters are the internal variables the model learns during training, representing the connections and weights defining its understanding of language. As the number of parameters increases, so does the model’s capacity to encapsulate complex linguistic structures.

One primary outcome of adhering to the scaling law is the substantial improvement in performance across a spectrum of language-related tasks. From language generation to sentiment analysis, question-answering, and summarization, larger models consistently outperform their smaller counterparts. The increased capacity for learning intricate language features enables LLMs to excel in understanding and producing more human-like text.

When writing, most of the LLMs are based on the transformer architecture, where multi-headed self-attention layers are stacked in a very deep neural network. We’ll dive deep into the transformer architecture in Section 3.5.4, but for now, we can say that self-attention is a mechanism that allows a model to weigh different parts of the input sequence differently, capturing dependencies between words. The multi-headed self-attention mechanism lets the model capture different dependencies and relationships between words, enhancing language

⁹<https://www.microsoft.com/it-it/bing?form=MA13FV>

¹⁰Retrieval-Augmented Generation

¹¹Models are designed to process and understand information from multiple modalities or sources (e.g., text, image, audio, video). Multimodal models aim to handle and integrate data from two or more modalities.

¹²Contrastive Language–Image Pre-training

understanding. The idea is that different attention heads can focus on different aspects or relationships within the data, allowing the model to capture more nuanced patterns. Multiple layers of these multi-headed self-attention mechanisms are stacked in a very deep neural network. Each layer in the stack processes the previous layer’s output, learning hierarchical representations of the input data and capturing increasingly complex relationships and abstractions.

Two representative scaling laws for Transformer-based LLMs are the following [93, 172]:

1. **KM scaling law:** named in this way in Zhao et al. [364] and proposed by the OpenAI team in Kaplan et al. [93]. Given model size M , dataset size D , amount of training compute C , and a compute budget c , the KM scaling law states that the performance of a language model scales as per the following three formulas:

$$\begin{aligned} L(N) &= \left(\frac{N_c}{N}\right)^{\alpha_N}, \alpha_N \approx 0.076, N_c \approx 8.8 \times 10^{13} \\ L(D) &= \left(\frac{D_c}{D}\right)^{\alpha_D}, \alpha_D \approx 0.095, D_c \approx 5.4 \times 10^{13} \\ L(C) &= \left(\frac{C_c}{C}\right)^{\alpha_C}, \alpha_C \approx 0.050, C_c \approx 3.1 \times 10^8 \end{aligned} \quad (1)$$

where $L(N)$, $L(D)$, and $L(C)$ denote the cross-entropy loss of the model, the dataset, and the amount of training computed, respectively. The three laws were formulated by analysing the model’s performance across a range of data sizes (from 22M to 23B tokens), model sizes (from 768M to 1.5B non-embedding parameters), and training compute, with certain assumptions (e.g., ensuring that the other two factors do not constrain the analysis of one factor). The findings demonstrated a robust interdependence among the three factors influencing model performance.

2. **Chinchilla scaling law:** An alternative form of the scaling law has been proposed by the Google DeepMind team in Hoffmann et al. [172] experimenting with an extensive range of model size (70M to 16B) and data sizes (5B to 500B tokens). The Chinchilla scaling law posits that the performance of a language model scales as per the following formula:

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}, \quad (2)$$

where $E = 1.69$, $A = 406.4$, $B = 410.7$, $\alpha = 0.34$, $\beta = 0.28$

Authors showed that optimal allocation of compute budget to model size and data size can be derived as follows ¹³:

$$N_{opt}(C) = G\left(\frac{C}{6}\right)^a, D_{opt}(C) = G^{-1}\left(\frac{C}{6}\right)^b, \quad (3)$$

where $a = \frac{\alpha}{\alpha+\beta}$, $b = \frac{\beta}{\alpha+\beta}$ and G is a scaling coefficient. The KM scaling law favours a more significant budget allocation in model size than the data size. In contrast, the Chinchilla scaling law argues that the two sizes should be increased in equal scales [172] (i.e., having similar values for a and b in (3)).

¹³under the constraint $C \approx 6ND$

Scaling boosts performance and addresses inherent limitations in smaller language models. Larger models excel in managing long-range dependencies, comprehending ambiguous language constructs, and displaying a nuanced understanding of context—capabilities that smaller models frequently find challenging. The eliciting of emergent abilities, such as Chain-of-Thought prompting and in-context learning, have shown a phase change in the first Scaling Law, where the performance increases linearly as the model size increases exponentially (Figure 2). Emergency is still a debated topic: Schaeffer, Miranda, and Koyejo [318] shows that different metrics can reveal continuous improvement in LLM performance, challenging the concept of emergent abilities¹⁴, while others argue that the unpredictability of when and which metrics show abrupt improvement still supports the idea of emergence. While the study provides valuable insights, researchers agree that discontinuities and jump-like improvements in model performance still exist as model size increases.

At its core, the scaling law is a guiding principle in the development of LLMs, directing the allocation of resources and the design of models to maximise performance and capabilities.

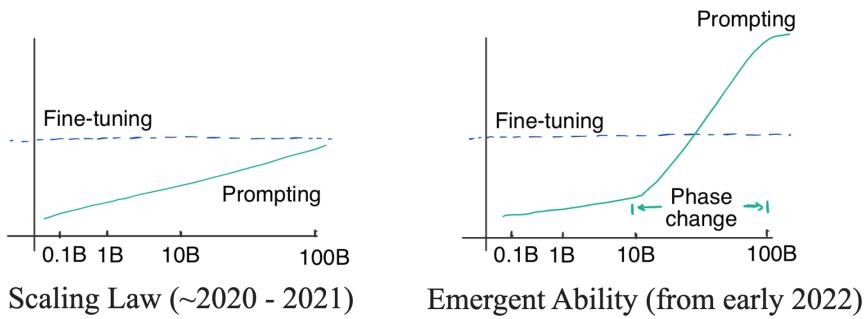


Figure 2: Left: scaling law. Model performance increases linearly as the model size increases exponentially. Right: emergent abilities show a phase change at a certain scale where the performance suddenly increases. Source: Fu [267].

Despite propelling the field of LLMs to new heights, the scaling law comes with computational challenges. Training huge models requires significant computational resources, encompassing processing power and memory. The computational budget is an upper bound limit, demanding innovations in hardware and distributed training techniques to exploit the potential of scaled-up language models fully.

2.3 Prominent Model Families

The development of Large Language Models (LLMs) has been driven by the emergence of prominent model families, each characterised by its unique architecture and capabilities. These model families have played a pivotal role in shaping the landscape of language processing and understanding and have been instrumental in pushing the boundaries of LLMs.

Some of the most prominent large language models (having a size larger than 10B) are depicted in Figure 3.

2.3.1 BERT

Introduced by Google in 2018, BERT [65] marked a significant evolution in LLMs by focusing on bidirectional context in text processing. BERT’s model architecture is a multi-layer

¹⁴Choosing a different metric it’s possible to show that increasing the model size, leads to an improvement in correct sequences prediction in addition problems. Looking at these metrics, the “add” ability is not emergent but gradual and predictable.

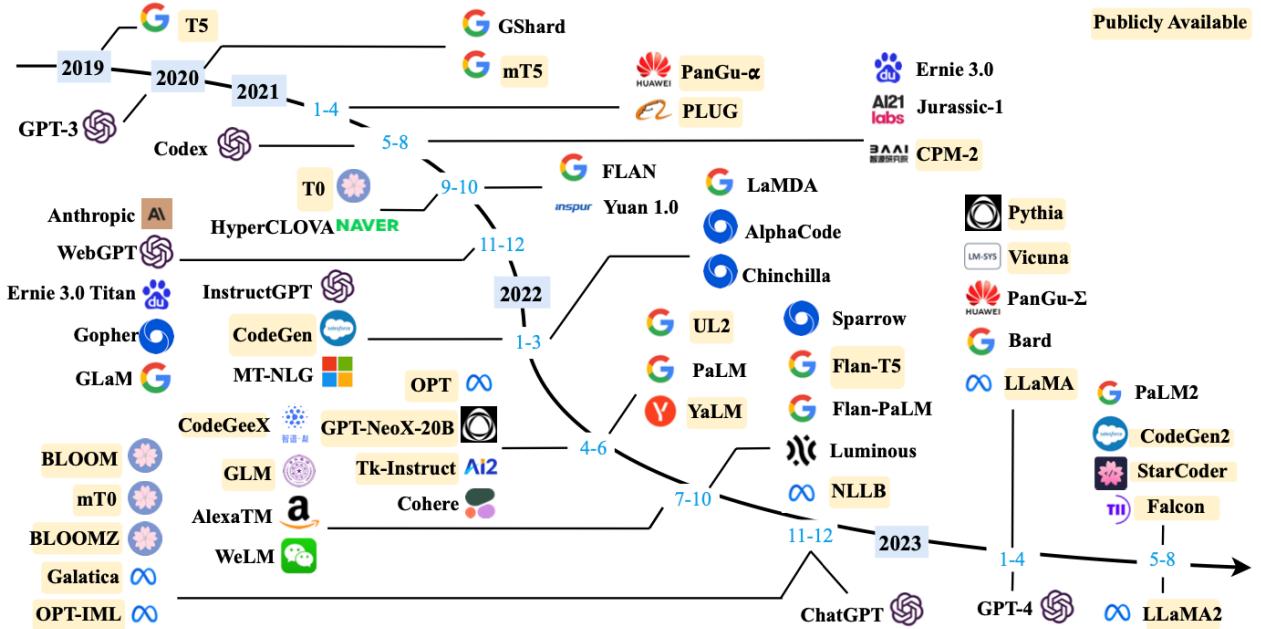


Figure 3: A diagram showing the evolution of publicly available LLMs. Source: Zhao et al. [364].

bidirectional Transformer encoder based on the original transformer architecture introduced by Vaswani et al. [334]. Unlike its predecessors, BERT analyses text in both directions (left-to-right and right-to-left), providing a more nuanced understanding of language context. This bi-directionality enables BERT to achieve state-of-the-art results in various NLP tasks, such as question answering, named entity recognition, and sentiment analysis. BERT's architecture and training methodology have influenced numerous subsequent models and research initiatives [65].

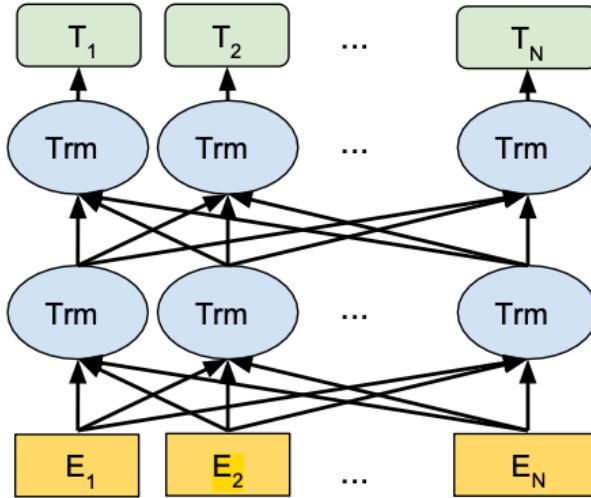


Figure 4: BERT Architecture: The bottom layer contains the embedding representations E_1, E_2, \dots, E_N , which encode input tokens and serve as the input to the transformer layers (Trm). Each transformer bidirectionally processes the input embeddings, and the final output is used for downstream tasks. Source: Devlin et al. [65].

Even BERT is built on the transformer architecture [334], which relies heavily on attention mechanisms to understand the context of words in a sentence. The innovation in BERT is its bidirectional nature and the use of a mechanism called the Masked Language Model (MLM).

In MLM, some percentage of the input tokens are randomly masked, and the objective is to predict these masked tokens based on their context, leveraging information from both sides of the sequence. BERT also incorporates a next-sentence prediction (NSP) task that helps the model learn relationships between sentences, further enhancing its understanding of context.

BERT's bidirectional context understanding significantly improves its performance on various NLP tasks, including sentiment analysis, question answering, and named entity recognition. By pre-training on a large corpus of text and then fine-tuning on specific tasks, BERT can adapt to various domains with relatively little task-specific data, demonstrating impressive transfer learning capabilities. Its architecture has set a new standard in the field, inspiring many subsequent models that build on or modify its foundational structure.

Despite its strengths, BERT is not without limitations. The model's size and complexity require substantial computational resources for training, which can be a barrier for some organisations or researchers. BERT's focus on context from surrounding text does not inherently solve all challenges in language understanding, particularly concerning ambiguity, nuance, or the subtleties of human language. The model can sometimes struggle with tasks requiring extensive world knowledge or reasoning beyond the scope of its training data.

While BERT itself does not exhibit emergent abilities in the same way that scaling up GPT models does, its architecture has enabled new approaches to handling context and language understanding that were not feasible with prior models. Subsequent iterations and variations of BERT, like RoBERTa¹⁵ and ALBERT¹⁶, have sought to optimise and expand upon BERT's foundational principles, exploring how changes in model size, training methodology, and architecture can influence performance and capabilities.

2.3.2 T5

Developed by Google in 2019, T5¹⁷ re-framed all NLP tasks as a unified text-to-text problem, where every task is cast as generating text from input text. This approach simplifies using a single model across diverse tasks, encouraging a more generalised understanding of language.

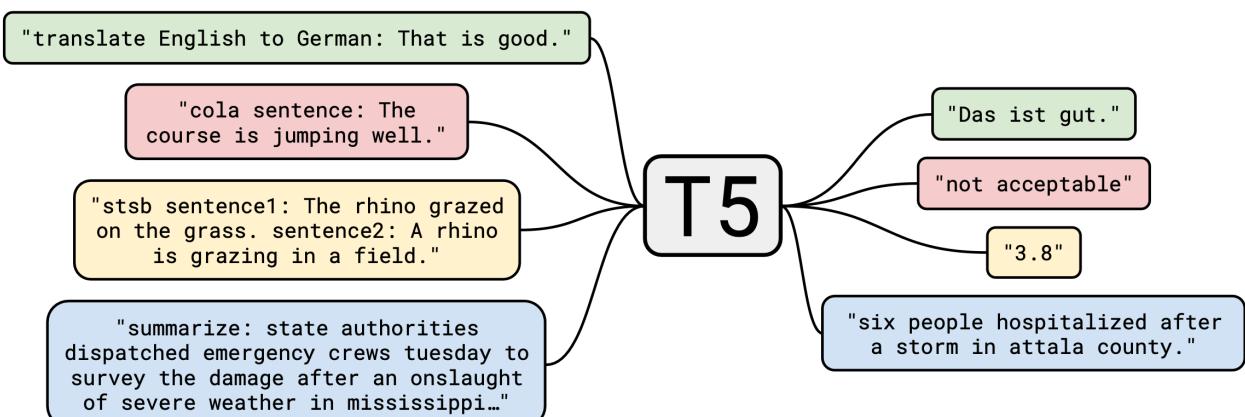


Figure 5: A diagram of the T5 text-to-text framework. Every task – including translation, question answering, and classification – is cast as feeding the model text as input and training it to generate some target text. This approach allows the same model, loss function, hyperparameters, etc., to be used across diverse tasks. Source: Raffel et al. [99].

T5 demonstrated its prowess across a range of benchmarks, setting new standards in the field

¹⁵Robustly Optimized BERT Pre-training Approach

¹⁶A Lite BERT

¹⁷Text-to-Text Transfer Transformer

of NLP [99]. It’s built on the transformer model, similar to its predecessors, BERT and GPT. It leverages the effective self-attention mechanism for processing data sequences. The model is designed to handle various tasks without needing task-specific architectural modifications. It uses a unified text-to-text framework, where tasks are converted into a format where the input and output are always text strings. T5 is pre-trained on a multitask mixture of unsupervised and supervised tasks, utilising a large-scale dataset known as “C4”¹⁸.

T5’s approach simplifies integrating new tasks into the model’s training regime, as they only need to be reformulated into the text-to-text format. While T5’s unified approach offers considerable advantages, it might not be optimal for all types of tasks. Some tasks could potentially benefit from more specialised model architectures or formats. The training process for T5 is resource-intensive, requiring substantial computational power, which could be a limiting factor for smaller organisations or independent researchers. As with other large language models, T5’s outputs can sometimes include biases in the training data, necessitating careful monitoring and potential post-hoc adjustments.

2.3.3 GPT Series

Developed by OpenAI, the GPT series has been at the forefront of LLM research. The original GPT model, introduced in 2018, laid the groundwork with its transformer-based architecture, significantly improving previous models’ understanding of context and generating text. It was developed based on a generative, decoder-only Transformer architecture, and it adopted a hybrid approach of unsupervised pre-training and supervised fine-tuning.

GPT-2 [75], released in 2019, expanded on this with 1.5 billion parameters and was trained with a large webpage dataset, WebText, demonstrating unprecedented text generation capabilities.

The subsequent GPT-3 model, unveiled in 2020, further pushed the boundaries with 175 billion parameters, showcasing remarkable abilities in generating human-like text, performing language translation, question-answering, and more without task-specific training. In the research paper on GPT-3 [88], the authors explained the concept known as in-context learning (ICL). This approach enables Large Language Models (LLMs) to function in few-shot or zero-shot scenarios. ICL empowers LLMs to comprehend tasks when they are described using natural language. This method aligns LLMs’ pre-training and application phases under a unified framework. During pre-training, the model predicts subsequent text sequences based on the prior context. In contrast, during in-context learning, the model generates the appropriate solution to a task in the form of a text sequence using the provided task instructions and examples.

The GPT series is based on the transformer architecture by Vaswani et al. [334]. This architecture leverages self-attention mechanisms to process input data, which allows the model to weigh the importance of different words within the input context, enhancing its ability to understand and generate language. GPT models are characterized by their stacked transformer blocks, which consist of multi-headed self-attention layers followed by fully connected feed-forward neural networks. The series has seen an exponential increase in the number of parameters: GPT with 110 million, GPT-2 with 1.5 billion, and GPT-3 with 175 billion parameters.

GPT models exhibit a remarkable ability to generate coherent and contextually relevant text, simulating human-like writing styles. They demonstrate strong performance in a wide array of NLP tasks without task-specific data training, showcasing their versatility in few-shot, one-shot, or zero-shot learning scenarios. The architecture’s scalability has shown that larger models tend to exhibit better performance and capture subtler patterns in data.

¹⁸Colossal Clean Crawled Corpus

One significant criticism is their data-hungry nature, requiring vast amounts of text data for training, which raises concerns about environmental impact and computational costs. The models can sometimes generate plausible but factually incorrect or nonsensical information, a phenomenon often referred to as “hallucination”. The black-box nature of these models poses challenges in interpretability and transparency, making it difficult to understand how decisions are made or how to correct biases.

GPT-3 demonstrated surprising emergent behaviours, such as improved reasoning, problem-solving, and creative writing, which were not explicitly programmed or observed in their predecessors. These abilities suggest that scaling up model size can lead to qualitative changes in how models understand and interact with language, although the relationship is not yet fully understood. OpenAI has explored two major approaches to further improving the GPT-3 model, i.e., training on code data and alignment with human preference, which are detailed as follows:

1. **Training on code data:** This approach involves fine-tuning the model on a diverse set of programming tasks, such as code completion, code generation, and code summarization. The model is trained on a large corpus of code data, which includes code snippets, programming languages, and software development documentation. The goal is to improve the model’s understanding of programming languages and its ability to generate code, thereby enhancing its performance on programming-related tasks.
2. **Alignment with human preference:** This approach involves training the model to generate outputs that align with human preferences and values and can be dated back to a work that applied reinforcement learning (RL) Christiano et al. [38] (similar to the reward training step in the aligning algorithm of InstructGPT).

GTP-4 GPT-4 [370], the successor to GPT-3, marks a further advancement in the GPT series developed by OpenAI. While specific details about GPT-4’s architecture and capabilities are proprietary, it is known to build upon the foundational concepts of its predecessors, emphasizing scale, capability, and efficiency. GPT-4 is a multimodal model which can accept image and text inputs and produce text outputs. Such models are an important area of study as they have the potential to be used in a wide range of applications, such as dialogue systems, text summarization, and machine translation.

On the MMLU benchmark [113], an English-language suite of multiple-choice questions covering 57 subjects, GPT-4 outperforms existing models by a considerable margin in English and demonstrates strong performance in other languages. GPT-4 development was enabled by deep learning infrastructure and optimization methods that behave predictably across various scales. The approach allowed the authors to predict the expected performance of GPT-4 (based on small runs trained similarly), which was tested against the final run, to increase confidence in the training. The primary reason is that extensive model-specific tuning is not feasible for very large training runs.

The prediction GPT-4’s final loss was predicted by fitting a scaling law with an irreducible loss term (as in Henighan et al. [90]):

$$L(C) = aC^b + c \quad (4)$$

from models trained using the same methodology but using at most $10,000\times$ less compute than GPT-4. The fitted scaling law predicted GPT-4’s final loss with high accuracy. In addition to predicting the final loss, a metric of capability was also predicted. One such metric is the pass rate on HumanEval dataset [108], which measures the ability to write Python functions of various complexity. The approximate power law relationship is

User

What is funny about this image? Describe it panel by panel.



GPT-4

The image shows a “Lightning Cable” adapter package with three panels.

Panel 1: A smartphone with a VGA connector (a large, blue, 15-pin connector typically used for computer monitors) plugged into its charging port.

Panel 2: The package for the “Lightning Cable” adapter with a picture of a VGA connector on it.

Panel 3: A close-up of the VGA connector with a small Lightning connector (used for charging iPhones and other Apple devices) at the end.

The humour in this image comes from the absurdity of plugging a large, outdated VGA connector into a small, modern smartphone charging port.

Table 1: Example prompt demonstrating GPT-4’s visual input capability. The prompt consists of a question about an image with multiple panels.

The model can grasp the context of the image and provide a detailed description of each panel, highlighting the humour in the visual juxtaposition of old and new technology. There are literal websites that explain jokes, and why something is funny. So, it’s not possible to know whether LLM’s explanation of the joke is coming from the true understanding of language or from those retrievals.

$$E_P[\log \text{pass_rate}(C)] = \alpha \times C^{-k} \quad (5)$$

where k and α are positive constants, and P is a subset of problems in the dataset.

GPT-4 accepts prompts consisting of images and text, which lets the user specify any vision or language task in parallel to the text-only setting. Specifically, the model generates text outputs, given inputs consisting of arbitrarily interlaced text and images. Despite its

capabilities, GPT-4 has similar limitations to earlier GPT models: it is not fully reliable (e.g. can suffer from “hallucinations”), has a limited context window, and does not learn from experience. Care should be taken when using the outputs of GPT-4, particularly in contexts where reliability is important.

OpenAI o1 OpenAI o1 [383] is a multimodal model developed by OpenAI, designed to process and generate text and images. It is a new large language model trained with reinforcement learning to perform complex reasoning. The model is able to produce long internal chain of thoughts before responding to a prompt. The o1 model family represents a transition from fast, intuitive thinking to now also using slower, more deliberate reasoning.

The large-scale reinforcement learning algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. Authors found that the performance of o1 consistently improves with more reinforcement learning (train-time compute) and with more time spent thinking (test-time compute). The constraints on scaling this approach differ substantially from those of LLM pretraining, and this approach is still under active research.

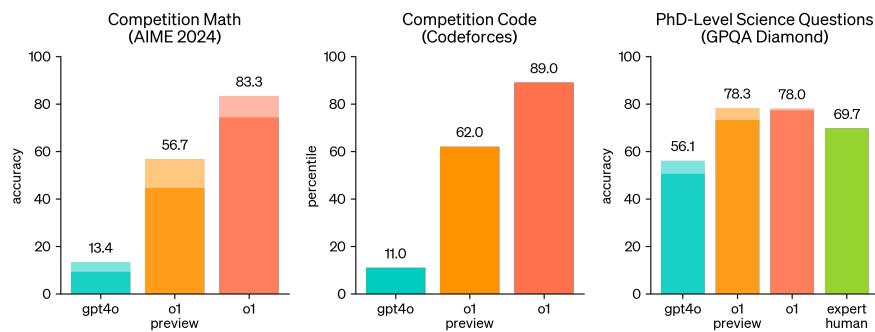


Figure 6: o1 greatly improves over GPT-4o on challenging reasoning benchmarks. Solid bars show pass@1 accuracy and the shaded region shows the performance of majority vote (consensus) with 64 samples. Source: OpenAI [383].

o1 has demonstrated proficiency in various domains, including advanced mathematics, coding, and scientific problem-solving, showcasing its versatility and potential for real-world applications as shown in Figure 6¹⁹. Datasets like MATH and GSM8K are no longer effective at differentiating models for recent frontier models²⁰. o1 version trained for coding also shows a significant improvement in performance on competitive programming questions from 2024 International Olympiad in Informatics (IOI) and in Codeforces competitive programming contests.

Human evaluations show that o1-preview is preferred to gpt-4o by a large margin in reasoning-heavy categories like data analysis, coding, and math. However, o1-preview is not preferred on some natural language tasks, suggesting that it is not well-suited for all use cases.

Integrating policies for model behaviour into chain-of-thought reasoning looks promising for improving model safety and alignment. This approach is a more robust way to teach human values and principles to model, improving performance on known jailbreaks²¹ and safety benchmarks [369].

¹⁹OpenAI o1 ranks in the 89th percentile on competitive programming questions (Codeforces), places among the top 500 students in the US in a qualifier for the USA Math Olympiad (AIME), and exceeds human PhD-level accuracy on a benchmark of physics, biology, and chemistry problems (GPQA).

²⁰Claude 3.5 Sonnet and Gemini

²¹adversarial prompts that purposely try to circumvent model refusals for content it's not supposed to produce

Unfortunately authors decided to not show the raw chains of thoughts generated by the model to the users, as they can be unaligned with human values and principles. The model shows instead a summary of the chain of thoughts, where the summariser is trained to avoid disallowed content.

METR, a nonprofit research organization focused on assessing catastrophic risks from advanced AI systems, evaluated the autonomous capabilities of AI models o1-preview-early, o1-mini, and o1-preview between late August and early September 2024. Their methodology involved testing these models in virtual environments on multi-step tasks. While the models demonstrated strong reasoning and planning abilities, their overall performance in autonomy tasks did not surpass the best public model, *Claude 3.5 Sonnet*. The models struggled with tool usage and feedback responsiveness when placed in basic agent scaffolds²². However, they excelled at one-step code generation, creating coherent plans, and offering useful suggestions. When integrated into optimized agent scaffolds (i.e., where they act as advisors to other agents) the performance aligned with the best public model.

In terms of their planning capabilities, Wang et al. [387] finds that the models excel at following constraints but face difficulties in decision-making and spatial reasoning. The o1 model is evaluated from three key perspectives: feasibility²³, optimality²⁴, and generalizability²⁵. While o1 outperforms GPT-4 in some areas, it struggles with generating optimal solutions and generalizing across various scenarios, such as memory handling and decision-making processes.

The new version of o1, o3, has been recently released, and it is expected to further improve the model’s reasoning capabilities and performance on a wide range of tasks. As reported by New Scientist in New Scientist [382], o3 also scored a record high of 75.7% on the Abstraction and Reasoning Corpus (ARC) developed by Google software engineer François Chollet, a prestigious AI reasoning test, but did not yet complete the requirements for the “Grand Prize” requiring 85% accuracy. Without the computing cost requirements imposing by the test, the model also achieves a new record high of 87.5%, while humans score, on average, 84%.

2.3.4 Llama

Llama²⁶ is a language model developed by Meta AI, designed to be a versatile and efficient foundation for a wide range of natural language processing (NLP) tasks. Llama is built on a transformer architecture [334], similar to other large language models, with a range from 7B to 65B parameters. Main differences between Llama and original Transformer architecture [334] are the following:

1. **Pre-normalization**²⁷ Llama uses pre-normalization²⁸, which means that the normalization layer is placed before the self-attention and feed-forward layers. Pre-normalization has improved training stability and convergence in large language models, making it a popular choice for many state-of-the-art models.

²²Agent scaffolding refers to a structured framework or setup that supports and organizes how an AI agent interacts with tasks, tools, and its environment to achieve specific goals effectively. It serves as a “support structure” - much like scaffolding used in construction - to guide the agent in reasoning, planning, and decision-making.

²³A plan must not only be executable but also ensure goal completion under real-world constraints.

²⁴optimality concerns how efficiently the plan achieves its goal

²⁵Whether a language model can successfully plan across a diverse range of scenarios, including those it may not have explicitly encountered during training.

²⁶Large Language Model Meta AI

²⁷Inspired by GTP-3 model

²⁸See Section 3.5.4

2. **SwiGLU activation function**²⁹ LLAMA uses the SwiGLU³⁰ activation function by Shazeer [100], which is a variant of the Gated Linear Unit (GLU) activation function. SwiGLU has been shown to improve the performance of large language models by enhancing the flow of information through the network.
3. **Rotary Embeddings**³¹ Llama uses rotary embeddings by Su et al. [134], which are a type of positional encoding that helps the model capture long-range dependencies in the input data.

Model	params	dimension	#heads	#layers	learning rate	batch size	#tokens	context
LLaMA	6.7B	4096	32	32	3.0×10^{-4}	4M	1.0T	2k
LLaMA	13.0B	5120	40	40	3.0×10^{-4}	4M	1.0T	2k
LLaMA	32.5B	6656	52	60	1.5×10^{-4}	4M	1.4T	2k
LLaMA	65.2B	8192	64	80	1.5×10^{-4}	4M	1.4T	2k
CodeLlama 2	7B	4096	32	32	2.0×10^{-4}	4M	1.8T	16k
LLaMA 2	7B	4096	32	32	2.0×10^{-4}	4M	1.8T	4k
LLaMA 2	13B	5120	40	40	2.0×10^{-4}	4M	1.8T	4k
LLaMA 2	70B	8192	64	80	1.5×10^{-4}	4M	1.8T	4k
LLaMA 3	8B	4096	32	32	2.5×10^{-4}	4M	15T	8k
LLaMA 3	70B	8192	64	80	1.0×10^{-4}	4M	15T	8k
LLaMA 3.1	8B	4096	32	32	3.0×10^{-4}	4M	15T	128k
LLaMA 3.1	70B	8192	64	80	1.5×10^{-4}	4M	15T	128k
LLaMA 3.1	504B	16384	128	126	8.0×10^{-5}	4M	15T	128k

Table 2: Llama models sizes, architectures, and optimization hyper-parameters. Params: This column represents the total number of parameters in billions. Dimension: The dimension of the model’s hidden layers. # heads: The number of attention heads in the model. # layers: The number of transformer layers in the model. Learning rate: The learning rate used during training. Batch size: The batch size used during training. # tokens: The total number of tokens in the training dataset. Source: Touvron et al. [330].

Based on the Llama paper by Touvron et al. [330], even though Llama 13B is smaller than many competitors, it outperforms GPT-3 on most benchmarks, and the 65B model is competitive with the best large language models available, such as Chinchilla and PaLM-540B, despite being x10 smaller (as shown in Table 3).

Model	Params	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande ARC-e	ARC-c	OBQA
GPT-3	175B	60.5	81.0	-	78.9	70.2	68.8	51.4
Gopher	280B	79.3	81.8	50.6	79.2	70.1	-	-
Chinchilla	70B	83.7	81.8	51.3	80.8	74.9	-	-
PaLM	62B	84.8	80.5	-	79.7	77.0	75.2	52.5
PaLM-cont	62B	83.9	81.4	-	80.6	77.0	-	-
PaLM	540B	88.0	82.3	-	83.4	81.1	76.6	53.0
Llama	7B	76.5	79.8	48.9	76.1	70.1	72.8	47.6
Llama	13B	78.1	80.1	50.4	79.2	73.0	74.8	52.7
Llama	33B	83.1	82.3	50.4	82.8	76.0	80.0	57.8
Llama	65B	85.3	82.8	52.3	84.2	77.0	78.9	56.0

Table 3: Zero-shot performance on Common Sense Reasoning tasks. Source: Touvron et al. [330].

²⁹Inspired by PaLM model

³⁰See Section 3.5.4

³¹Inspired by GPTNeo model

The Llama models were trained exclusively on publicly available data, setting them apart from other models that rely on proprietary datasets³². The dataset is a mixture of several sources (webpages, books, scientific data and code) as reported in Table 4.

Dataset	Classification	Sampling prop.	Epochs	Disk size
CommonCrawl	Webpages	67.0%	1.10	3.3 TB
C4	Web	15.0%	1.06	783 GB
Github	Code	4.5%	0.64	328 GB
Wikipedia	Webpages	4.5%	2.45	83 GB
Books	Books	4.5%	2.23	85 GB
ArXiv	Scientific Data	2.5%	1.06	92 GB
StackExchange	Conversation Data	2.0%	1.03	78 GB

Table 4: Pre-training data. Data mixtures used for pre-training for each subset, the table reports the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion. Source: Touvron et al. [330].

Llama models were designed with efficiency in mind, both in training and inference, allowing even the 13B parameter model to run on a single GPU. A synthetic view of the Llama model family parameters is reported in Table 2. The optimizer used during the training is the same AdamW with the following hyper-parameters: $\beta_1 = 0.9$, $\beta_2 = 0.95$, $eps = 10^{-5}$, a weight decay of 0.1, gradient clipping of 1.0, a cosine learning rate schedule and a warmup of 2000 steps.

Touvron et al. [330] acknowledges the presence of biases and toxicity in the models due to the nature of web data and evaluates these aspects using benchmarks from the responsible AI community.

Llama 2. Llama 2 [329] is a continuation of the Llama series, developed by Meta AI, released in scale from 7B to 70B parameters. The pre-training data of the Llama2 model is a new mix of data from publicly available sources. The training corpus is 40% larger than the one used for Llama 1, and it is composed of a mix of text and a percentage of code data that is roughly 8% of the total. The exact composition of the data mix is not disclosed, but the code percentage is reported in the caption of the Table 5 extracted from the original paper [329]. The pre-training selection focuses on addressing biases and toxicity recognised in the previous version of the model.

Llama 2 adopts most of the pretraining settings and model architecture from Llama 1, including the standard transformer architecture, pre-normalization using RMSNorm, the SwiGLU activation function, and rotary positional embeddings. The optimizer used during the training is the same AdamW with the following hyper-parameters: $\beta_1 = 0.9$, $\beta_2 = 0.95$, $eps = 10^{-5}$, a weight decay of 0.1, gradient clipping of 1.0, a cosine learning rate schedule and a warmup of 2000 steps. The primary architectural differences from Llama 1 include increased context length and grouped-query attention (GQA).

Code Llama. Code Llama [384] is a family of large language models for code generation based on Llama 2 providing infilling³³ capabilities, support for large input contexts and zero-shot instruction following ability for programming tasks. It comes in three flavours: the vanilla

³²Such as “Books — 2TB” or “Social media conversations”

³³With the terms infilling or code completion, we refer to the process of generating code snippets that complete a given code fragment.

Language	Percent	Language	Percent
en	89.70%	uk	0.07%
unknown	8.38%	ko	0.06%
de	0.17%	ca	0.04%
fr	0.16%	sr	0.04%
sv	0.15%	id	0.03%
zh	0.13%	cs	0.03%
es	0.13%	fi	0.03%
ru	0.13%	hu	0.03%
nl	0.12%	no	0.03%
it	0.11%	ro	0.03%
ja	0.10%	bg	0.02%
pl	0.09%	da	0.02%
pt	0.09%	sl	0.01%
vi	0.08%	hr	0.01%

Table 5: Language distribution in pretraining data with percentage $\geq 0.005\%$. Most data is in English, meaning that LLaMA 2 will perform best for English-language use cases. The large unknown category is partially made up of programming code data.

model, the Python specialized model, and the instruction-following model with 7B, 13B, 34B, and 70B parameters each (see Figure 7).

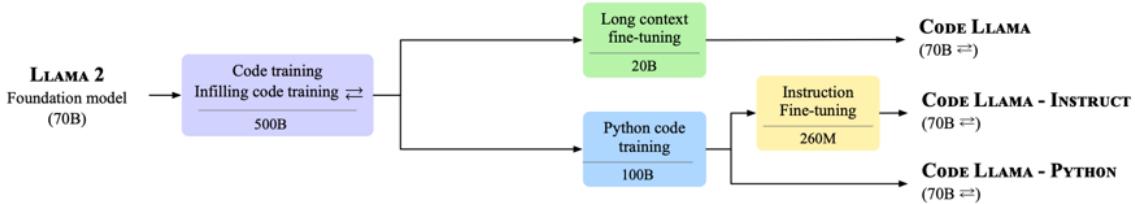


Figure 7: The Code Llama 70B specialization pipeline. The different fine-tuning stages are annotated with the number of tokens seen during training. Infilling-capable models are marked with the \leftrightarrow symbol. Source: Rozière et al. [384].

While most of the code generation models are trained on code only, Code Llama was fine-tuned starting from Llama 2, which was trained on general-purpose text and code data. The comparison in Rozière et al. [384] shows that initializing from Llama 2 leads to better performance on code generation tasks than initializing from a code-only model for a given budget as shown in Figure 8. Code Llama was fine-tuned on 500B extra tokens consisting mostly of code data (85%).

Llama 3. Llama 3 [389] is a continuation of the Llama series, developed by Meta AI, with different model sizes: 8B, 70B, and 405B parameters.

Llama 3 uses a standard, dense Transformer architecture. It does not deviate significantly from Llama and Llama 2 in terms of model architecture; therefore performance gains are primarily driven by improvements in data quality and diversity as well as by increased training scale. Compared to Llama 2, Llama 3 has a few small changes in the model architecture:

1. grouped query attention (GQA) with 8 key-value heads to improve inference speed and to reduce the size of key-value caches during decoding

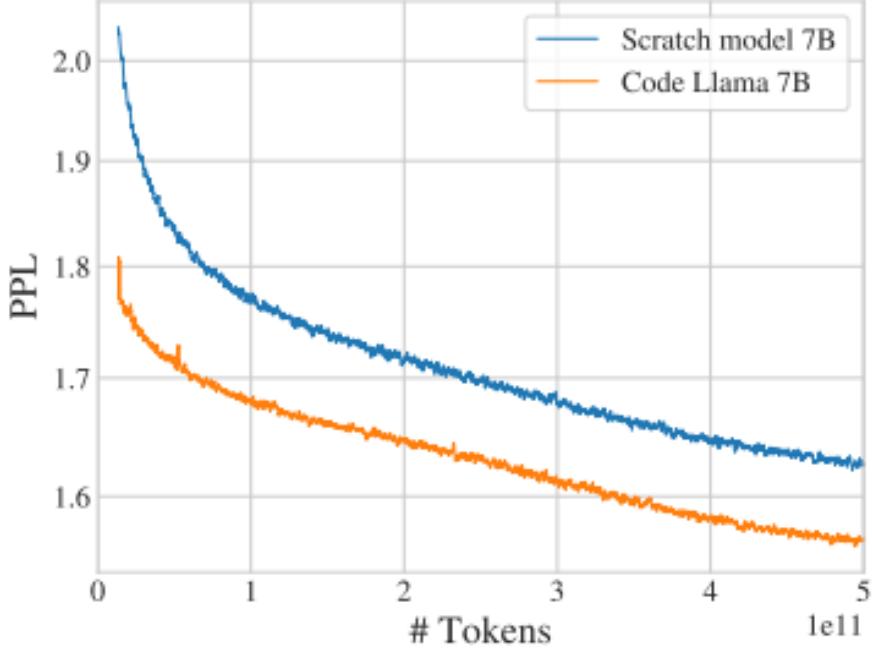


Figure 8: Comparison of Code Llama models versus an identical model trained from scratch. Source: Rozière et al. [384].

2. an attention mask that prevents self-attention between different documents within the same sequence. This change has limited impact during standard pre-training, but it's important in continued pre-training on very long sequences.
3. a vocabulary with 128K tokens. Compared to the Llama 2 tokenizer, the new tokenizer improves compression rates on a sample of English data from 3.17 to 3.94 characters per token.
4. the RoPE base frequency hyper-parameter increased to 500,000 to support longer contexts.

A summary of the key hyper-parameters of Llama 3 is shown in Table 7.

The authors improved the quantity and quality of the data we used for pre-training and post-training compared to prior versions of Llama. These improvements include developing more careful pre-processing and curation pipelines for pre-training data and more rigorous quality assurance and filtering approaches for post-training data. The pre-training corpus consists of about 15T tokens, which is about 50It contains roughly 50% of tokens corresponding to general knowledge, 25% of mathematical and reasoning tokens, 17% code tokens, and 8% multilingual tokens [389]. The resulting models have a rich set of capabilities. They can answer questions in at least eight languages, write high-quality code, solve complex reasoning problems, and use tools out-of-the-box or in a zero-shot way

2.3.5 Gemma

The recent development in the domain of Natural Language Processing has seen Google's introduction of a new family of models named Gemma [372, 385]. Derived from the same research lineage as the renowned Gemini models, Gemma is a testament to the rapid advancements in lightweight, high-performance language models designed for a broad spectrum of computational environments.

Table 6: Overview of the key hyperparameters of Llama 3. We display settings for 8B, 70B, and 405B language models.

	8B	70B	405B
Layers	32	80	126
Model Dimension	4096	8192	16384
FFN Dimension	14336	28672	53248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU	SwiGLU	SwiGLU
Vocabulary Size	128,000	128,000	128,000
Positional Embeddings	RoPE ($\theta = 500,000$)	RoPE ($\theta = 500,000$)	RoPE ($\theta = 500,000$)

Table 7: Overview of the key hyperparameters of Llama 3

Gemma is built upon a transformer-based architecture by Vaswani et al. [334], optimized to deliver state-of-the-art performance with a fraction of the parameter count typically seen in large language models (LLMs). Notable enhancements include the adoption of Multi-Query Attention, RoPE embeddings, GeGLU activations, and RMSNorm, indicating an evolution of the original transformer architecture. The family comprises two main configurations: Gemma 2B and Gemma 7B, available in pre-trained and instruction-tuned variants. The design philosophy targets efficient deployment across diverse hardware platforms, including but not limited to mobile devices, laptops, desktop computers, and servers.

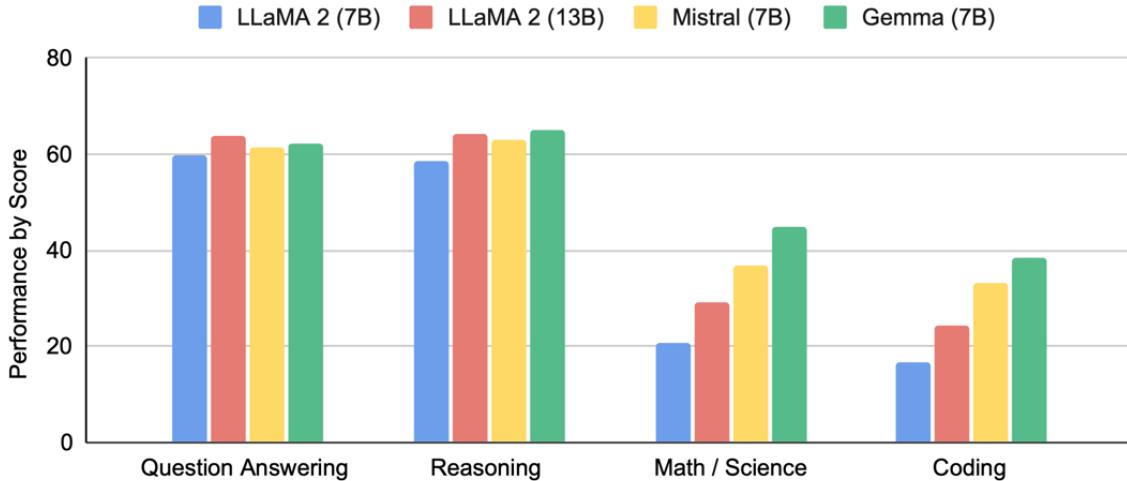


Figure 9: Gemma models exhibit superior performance in language understanding and reasoning tasks compared to larger models. Source: Team et al. [385].

In comparative benchmarks, Gemma models have demonstrated capabilities that exceed those of larger parameter models, such as Llama 2 (13B), indicating a significant efficiency in parameter utilization. Improvements are particularly evident in language understanding and reasoning tasks where Gemma models have been pitted against their contemporaries.

One prominent strength of Gemma models is their deployment efficiency, which democratizes access to state-of-the-art NLP tools. The models are designed to be run on common developer

hardware, eschewing the need for specialized AI accelerators.

The chart compares Gemma and Llama-2 models across four categories: General, Reasoning, Math, and Code. For each category, it lists the benchmark, description, and scores for both models at 7B and 13B parameter counts. Gemma consistently outperforms Llama-2 across all benchmarks and parameter counts.

CAPABILITY	BENCHMARK	DESCRIPTION	Gemma		Llama-2	
			7B	13B	7B	13B
General	MMLU 5-shot, top-1	Representation of questions in 57 subjects (incl. STEM, humanities and others)	64.3		45.3	54.8
	BBH -	Diverse set of challenging tasks requiring multi-step reasoning	55.1		32.6	39.4
Reasoning	HellaSwag 0-shot	Commonsense reasoning for everyday tasks	81.2		77.2	80.7
	GSM8K maj@1	Basic arithmetic manipulations (incl. Grade School math problems)	46.4		14.6	28.7
Math	MATH 4-shot	Challenging math problems (incl. algebra, geometry, pre-calculus, and others)	24.3		2.5	3.9
	HumanEval pass@1	Python code generation	32.3		12.8	18.3
Code						

Figure 10: Gemma models are designed to be lightweight and efficient, making them accessible to a wide range of developers and applications. Source: Banks and Warkentin [372].

Despite their efficiencies, the Gemma models are not without limitations. While the reduced parameter count is advantageous for accessibility and computational efficiency, it may impact performance in complex NLP tasks that can benefit from larger models. Additionally, ethical considerations, such as bias in language models, remain an area of concern and active development.

Google has emphasized the responsible development of AI, which is evident in Gemma's design. Techniques to mitigate sensitive data inclusion and reinforcement learning from human feedback are incorporated to ensure the models' outputs adhere to safety standards. Moreover, Google's release includes a Responsible Generative AI Toolkit to aid developers in prioritizing the creation of ethical AI applications.

2.3.6 Claude

Claude models are a family of large language models developed by Anthropic, a research organization focused on building advanced AI systems [371]. The most advanced model in the Claude series, Claude 3.5 Sonnet, excels at natural language understanding and generation, including summarization, creative writing, and more. It shows marked improvements in logical and mathematical reasoning, outperforming prior versions on benchmarks. The model is capable of writing, debugging, and explaining code snippets. It is optimized for dialogues and interactive workflows, allowing for dynamic and iterative engagement with users.

Claude 3 has demonstrated significant improvements in its ability to perform logical and mathematical reasoning tasks. Logical reasoning, in particular, showcases the model's ability to deduce patterns, validate arguments, and resolve abstract puzzles. For example, tasks involving

syllogistic reasoning or the identification of valid logical structures benefit from the model’s enhanced understanding of formal rules.

In mathematical reasoning, the model has shown its ability to parse and solve complex problems across multiple steps. Benchmarks such as GSM8K, which contains grade-school-level arithmetic and word problems, highlight Claude 3’s ability to provide structured and accurate solutions. The model can further engage in higher-level mathematics, including algebra and basic calculus, as evaluated by the MATH dataset, though challenges remain in more specialized domains.

Beyond formal reasoning, Claude 3 excels in commonsense understanding, a critical aspect of human-like intelligence. Benchmarks such as CommonSenseQA and PIQA demonstrate its ability to reason about everyday scenarios and physical phenomena, respectively. These capabilities are crucial for applications that require intuitive decision-making, such as virtual assistants or educational tools.

Claude 3’s ethical reasoning is a particularly interesting facet. Leveraging training paradigms focused on safety and alignment, the model is adept at identifying and addressing ethical dilemmas. Benchmarks like the Winogender Schema, which tests gender bias, and other ethical reasoning tests confirm the model’s ability to minimize bias and generate responsible outputs.

Despite its strengths, Claude 3 is not without limitations. Contextual understanding can falter in multi-layered or ambiguously phrased tasks. Similarly, abstract reasoning outside the bounds of its training data can present significant hurdles. Another limitation arises in the handling of uncertainty; the model can occasionally overcommit to answers even when the underlying confidence is low. These challenges underscore the need for further improvements, particularly in domains requiring highly abstract thinking or multi-turn contextual reasoning. Integrating enhanced memory mechanisms may help the model process longer or more complex contexts, thereby reducing errors and improving overall coherence.

Claude 3.5 Sonnet shows substantial enhancements in both logical and commonsense reasoning. This improvement is particularly evident in graduate-level problem-solving tasks and other advanced reasoning benchmarks, such as the ARC dataset. The model demonstrates a better ability to:

- Parse complex, multi-step problems and provide structured solutions.
- Handle abstract reasoning with improved accuracy in scenarios involving nuanced logical patterns or uncommon use cases.

Comparing this version to its predecessors Claude 3 and Claude 3 Opus, the advancements in Claude 3.5 Sonnet are clear:

- On reasoning benchmarks, Claude 3.5 Sonnet achieves higher accuracy, particularly in tests like GSM8K and MATH datasets.
- Interaction speeds are significantly faster, improving usability in real-time applications.
- Its coding capabilities surpass earlier versions in complexity and versatility, reflecting deeper training on software development datasets.

2.4 Specialized Large Language Models

Specialized Large Language Models (LLMs) are model checkpoints refined for particular fields or tasks, such as healthcare and finance. The existing domain-specific models are developed by pre-training on specialized datasets [191, 254, 220]), by adapting a very large general-purpose model to domain-specific tasks [213, 185], or mixing both approaches [350]. These models serve

as domain-specific problem solvers and are evaluated based on general competencies, such as fundamental complex reasoning, and more nuanced capabilities, like alignment with human intent, as well as their performance in areas specific to their application. To accurately measure their efficacy, specialized benchmarks are developed that cater to these distinct sectors. These tailored benchmarks are then employed in conjunction with broader assessments to provide a holistic and focused evaluation of the models' capabilities. The following sections highlight some of LLMs' key applications and their impact on different sectors, from healthcare to finance and education to research.

2.4.1 LLMs in Healthcare

The intersection of artificial intelligence (AI) and healthcare has precipitated unparalleled advances in the provision of medical services, diagnosis, treatment, and patient care. Central to these advancements are Large Language Models (LLMs), which have been instrumental in catalyzing transformative changes across the healthcare sector:

1. **Medical image analysis:** Large Language Models (LLMs) have been integrated with medical imaging technologies to enhance diagnostic accuracy and efficiency. By analyzing radiological images and clinical reports, LLMs can assist radiologists in interpreting images, identifying abnormalities, and providing diagnostic insights. These models leverage their natural language processing capabilities to extract information from textual reports and correlate it with visual data, thereby augmenting the diagnostic process [120, 140].
2. **Clinical Decision Support:** LLMs have been pivotal in augmenting clinical decision support systems (CDSS). By analyzing patient data and medical literature, LLMs assist clinicians in diagnosing conditions, suggesting treatment options, and predicting patient outcomes. For instance, models like BERT and its derivatives have been fine-tuned on medical corpora, yielding tools that can parse clinical notes, interpret lab results, and provide evidence-based recommendations [61].
3. **Medical Documentation and Coding:** The onus of medical documentation and billing has traditionally been a significant administrative burden for healthcare providers. LLMs have demonstrated the ability to streamline these processes by automating the translation of clinical dialogue and notes into structured electronic health records (EHRs) and accurately coding medical procedures, thus mitigating errors and saving time [53].
4. **Drug Discovery and Development:** In the domain of pharmaceuticals, LLMs have expedited the drug discovery and development pipelines. By mining through vast chemical libraries and medical databases, these models facilitate the identification of potential drug candidates and the repurposing of existing drugs for new therapeutic uses [84].
5. **Personalized Medicine:** Personalized medicine, which tailors treatment to individual patient characteristics, has benefited from LLMs by generating patient-specific models that predict disease susceptibility and drug response. This personalization extends to creating tailored health interventions based on patient history and genetic information [15].
6. **Patient Engagement and Self-Management:** LLMs are also revolutionizing patient engagement by powering intelligent virtual health assistants capable of providing information, reminders, and motivational support for chronic disease self-management. These AI assistants interact with patients in natural language, thus fostering an environment conducive to patient education and adherence to treatment regimens [70].

Despite these strengths, LLMs face significant challenges within healthcare applications. Concerns regarding patient privacy, data security, and the need for explainability in AI-driven decisions are paramount [43]. Additionally, biases inherent in training data can perpetuate disparities in patient care, necessitating rigorous validation and fairness assessments before clinical deployment [63].

Large Language Models represent a transformative force in healthcare, enhancing efficiency, accuracy, and personalization in various medical domains. Their integration into clinical practice must be pursued with diligent oversight to navigate ethical considerations and ensure equitable and safe applications.

Med-PaLM One of the most advanced LLMs for healthcare is Med-PaLM, a derivative of the PaLM (540B) model developed by Google and its instruction-tuned variant, Flan-PaLM. Using a combination of few-shot [88], chain-of-thought (CoT) (Wei et al. [230]), and self-consistency (Wang et al. [227] prompting strategies, Flan-PaLM achieved state-of-the-art accuracy on every MultiMedQA³⁴ multiple-choice dataset (MedQA, MedMCQA, PubMedQA, MMLU clinical topics and a newly introduced dataset, HealthSearchQA, which consists of commonly searched health questions).

Model (number of parameters)	MedQA (USMLE) Accuracy %
Flan-PaLM (540 B)	67.6
PubMedGPT (2.7 B)	50.3
DRAGON (360 M)	47.5
BioLinkBERT (340 M)	45.1
Galactica (120 B)	44.4
PubMedBERT (100 M)	38.1
GPT-Neo (2.7 B)	33.3

Table 8: Performance comparison of different models on the MedQA (USMLE) benchmark. Source: Singhal et al. [213].

Despite these remarkable results, human evaluation reveals key gaps in Flan-PaLM responses and remains inferior to clinicians [213]. To resolve this issue, researchers introduced “instruction tuning”³⁵ to align the Flan-PaLM model to the medical domain. Thus, Instruction tuning can be seen as a lightweight way (data-efficient, parameter-efficient, compute-efficient during training and inference) of training a model to follow instructions in one or more domains. Instruction tuning adapted LLMs to follow better the specific type of instructions used in the family of medical datasets. The result was Med-PaLM, a model that significantly reduces the gap (or even compares favourably) to clinicians on several evaluation axes, according to clinicians and lay users.

2.4.2 LLMs in Finance

There has been growing interest in applying NLP to various financial tasks, including sentiment analysis, question answering, and stock market prediction. Despite the extensive research into general-domain LLMs and their immense potential in finance, Financial LLM (Fin-LLM)

³⁴Stands for a multi-domain medical question answering benchmark. It has been designed to evaluate the performance of LLMs in the healthcare sector. This benchmark likely encompasses a wide range of medical questions, covering various disciplines, conditions, and scenarios that medical professionals encounter.

³⁵See Section 3.4.1

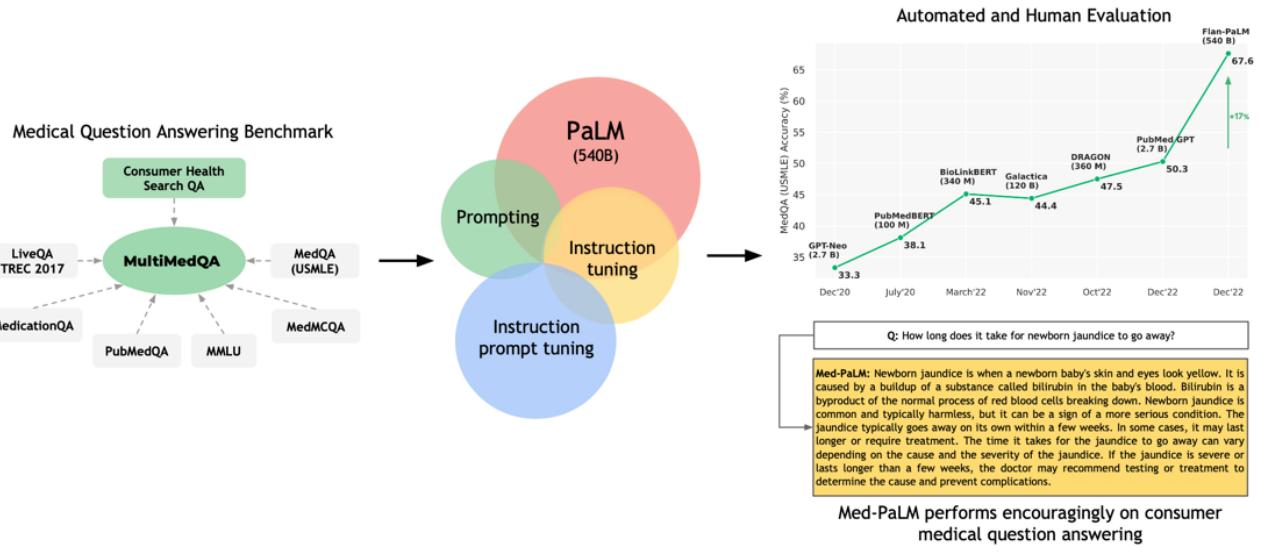


Figure 11: Large Language Models (LLMs) have revolutionized healthcare by enhancing diagnostic accuracy, clinical decision support, and patient engagement. Source: Singhal et al. [213].

research remains limited, and the field of financial LLMs is at an early stage [380]. An overview of the evolution of selected PLM/LLM releases from the general domain to the financial domain is shown in Figure 12.

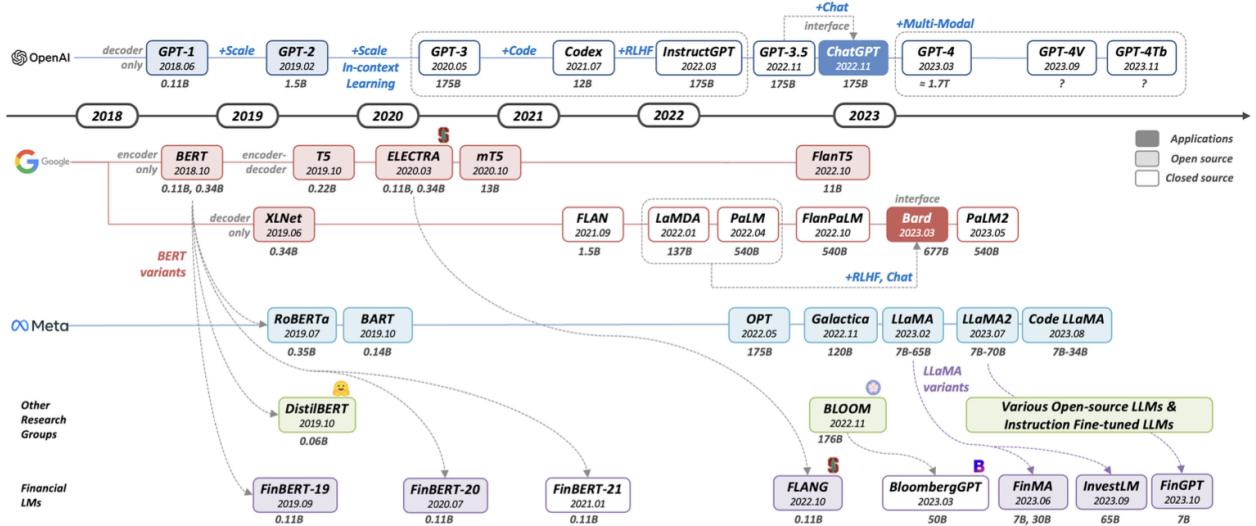


Figure 12: Timeline showing the evolution of selected PLM/LLM releases from the general domain to the financial domain. Source: Lee et al. [380].

Some of these models have demonstrated the potential of LLMs to understand complex financial jargon, generate insights, predict market trends, and enhance customer interaction with unprecedented precision and relevance.

Here are some key applications of LLMs in the financial sector:

1. **Algorithmic Trading:** LLMs analyze vast amounts of unstructured data, including news articles, financial reports, and social media, to gauge market sentiment and predict stock price movements. Their predictive insights enable more informed algorithmic trading strategies [44].

2. **Risk Management:** In risk management, LLMs contribute by parsing and interpreting complex regulatory documents, identifying potential compliance risks, and offering actionable insights to mitigate financial and reputational risks [96].
3. **Customer Service Automation:** Financial institutions leverage LLMs to power chatbots and virtual assistants, providing real-time, personalized customer service. These AI-driven systems can handle inquiries, execute transactions, and offer financial advice, enhancing customer experience and operational efficiency [127].
4. **Fraud Detection:** LLMs enhance fraud detection systems by analyzing transactional data and customer communication to identify patterns indicative of fraudulent activities, thereby bolstering the security of financial transactions [79].

Some of the models in Figure 12 have augmented the accuracy and efficiency of financial analyses and expedited the decision-making processes, enabling more timely and informed decisions. Additionally, their role in risk management is noteworthy, where their data processing and analytical prowess help identify potential risks and adherence issues more effectively than traditional methodologies [44].

Despite their potential, LLMs in finance face challenges, including data privacy concerns, the need for interpretability in model decisions, and the risk of perpetuating biases from training data. Ensuring these models adhere to ethical standards and regulatory compliance is paramount [92, 44].

Let's delve deeper into the techniques used to adapt LLMs for the financial sector to enhance their performance on finance-specific tasks [380]. These techniques enhance the models' understanding of financial language, data, and context, improving their performance on finance-specific tasks. Here's a more detailed look at these techniques:

Model	Backbone	Paras.	PT Techniques	PT Data Size	Evaluation Task	Dataset	O.S. Model	PT	IFT
BloombergGPT [350]	BLOOM	50B	PT, PE	(G) 345B tokens (F) 363B tokens	SA, TC	FPB, FiQA-SA, Headline	N	N	N
FinMA [353]	Llama	7B, 30B	IFT, PE	(G) 1T tokens	SA, TC, NER, QA	FPB, FiQA-SA, Headline FIN, FinQA, Con-vFinQA	Y	Y	Y
InvestLM [358]	Llama	65B	PEFT	(G) 1.4T tokens	SA, SMP	StockNet, CIKM18, BigData22	Y	N	N
FinGPT [339]	6 open-source LLMs	7B	PEFT	(G) 2T tokens	SA, TC, NER, RE	FPB, FiQA-SA, Headline FIN, FinRED	Y	Y	Y

Table 9: The abbreviations correspond to Paras.= Model Parameter Size (Billions); Disc. = Discriminative, Gen. = Generative; Post-PT = Post-Pre-training, PT = Pre-training, FT = Fine-Tuning, PE = Prompt Engineering, IFT = Fine-Tuning, PEFT = Parameter Efficient Fine-Tuning; (G) = General domain, (F) = Financial domain; (in Evaluation) [SA] Sentiment Analysis, [TC] Text Classification, [SBD] Structure Boundary Detection, [NER] Named Entity Recognition, [QA] Question Answering, [SMP] Stock Movement Prediction, [Summ] Text Summarization, [RE] Relation Extraction; O.S. Model = Open Source Model. It is marked as Y if it is publicly accessible as of Dec 2023. Source: Lee et al. [380].

- **Domain-Specific Pre-training:** This technique involves further training a general LLM on a financial corpus. The idea is to refine the model’s language understanding and generation capabilities within the financial domain. By exposing the model to a large volume of financial texts, such as reports, news, and analysis, the model learns the specific jargon, styles, and nuances of financial language.
- **Continual Pre-training:** After initial pre-training on a general dataset, the model undergoes additional pre-training phases on financial data. This step-by-step refinement helps the model gradually adapt from a broad understanding of language to a more specialized comprehension of financial texts. It’s a way to incrementally infuse financial knowledge into the model without losing its general language capabilities.
- **Mixed-Domain Pre-training:** In this approach, the LLM is trained on a mixed dataset comprising both general and financial texts. The goal is to maintain the model’s general language understanding while also equipping it with the ability to process and generate financial content. This method aims to strike a balance, ensuring the model is not overly specialized and retains versatility.
- **Task-Specific Fine-tuning:** Once a model has been pre-trained with financial data, it can be fine-tuned for specific financial tasks. For example, a model could be fine-tuned on a dataset of financial sentiment analysis, stock market prediction, or fraud detection. This fine-tuning process sharpens the model’s skills on tasks that are directly relevant to the financial industry.
- **Transfer Learning:** Techniques from transfer learning can be applied where a model trained on one financial task is adapted for another. This approach leverages the knowledge the model has gained from one context, applying it to a different but related task, thereby enhancing learning efficiency and performance.
- **Custom Tokenization:** Financial texts often contain unique symbols, terms, and numerical expressions. Employing custom tokenization strategies that recognize these peculiarities can significantly enhance the model’s ability to process and understand financial documents.

Within the four FinLLMs in Figure 9, FinMA [353], InvestLM [358], and FinGPT [339] are based on Llama or other open-source based models, while BloombergGPT [350] is a BLOOM-style closed-source model.

Regarding the evaluation tasks, the models are assessed on a range of financial NLP tasks, as shown below:

- **Sentiment Analysis (SA):** This task involves analyzing the sentiment embedded within financial documents, such as market reports and news articles. The capability to accurately discern sentiment is crucial for applications such as market prediction and the formulation of trading strategies.
- **Named Entity Recognition (NER):** Essential for extracting actionable insights from financial documents, this task focuses on the identification and categorization of salient financial entities, including but not limited to company names, stock tickers, and monetary values.
- **Question Answering (QA):** FinLLMs are tasked with providing cogent answers to queries based on an expansive financial corpus. This benchmark often requires the synthesis of information from dense financial reports or news events.

- **Text Classification (TC):** The classification of financial documents into predefined categories aids in the automated sorting and analysis of financial data, an essential task in managing the voluminous data generated by financial markets.
- **Regulatory Compliance (RE):** Given the stringent regulatory environment of the financial sector, FinLLMs are often evaluated on their ability to parse and verify the compliance of financial texts with industry regulations.

To accurately measure the effectiveness of FinLLMs in performing these tasks, several datasets have been curated, each tailored to challenge different aspects of a model’s financial acumen:

- **Financial PhraseBank (FPB):** A dataset comprising sentences from financial news, annotated to reflect sentiment polarity, which is instrumental in the training and testing of models for sentiment analysis [24].
- **FiQA - Financial Opinion Mining and Question Answering Challenge (FiQA-SA, FiQA-QA):** This dataset encompasses annotated financial news and social media texts for sentiment analysis alongside a collection of question-and-answer pairs for the evaluation of QA capabilities [48].
- **FIN: A Financial Document Dataset for NER:** Designed for entity recognition, this dataset consists of financial news articles with annotated entities, testing the model’s capacity to identify and classify financial terms Alvarado, Verspoor, and Baldwin [25]. Another financial NER dataset is FiNER-139, consisting of 1.1M sentences from financial news articles, annotated with 139 eXtensive Business Reporting Language (XBRL) word-level tags [189]. This dataset is designed for Entity Extraction and Numerical Reasoning tasks, predicting the XBRL tags (e.g., cash and cash equivalents) based on numeric input data within sentences (e.g., “24.8” million).
- **ConvFinQA:** A conversational finance QA dataset challenging models to understand and respond within the context of financial dialogues, demonstrating an advanced application of FinLLMs in customer interaction Chen et al. [153]. It’s an extension of FinQA and is a multi-turn conversational hybrid QA dataset consisting of 3,892 conversations with 14,115 questions.
- **StockNet:** This dataset combines historical price data with relevant tweets to comprehensively view SMP tasks. It has been widely used to assess the impact of market sentiment on stock prices [59].
- **CIKM18:** A dataset designed for SMP tasks, CIKM18 comprises stock price data and news headlines, challenging models to predict stock movements based on textual information [58].
- **BigData22:** A dataset for SMP tasks, BigData22 combines financial news articles with stock price data, evaluating models on their ability to predict stock movements based on textual information Soun et al. [216].
- **Headline:** A dataset of financial news headlines, used for text classification [133]. This dataset comprises 11,412 news headlines, labelled with a binary classification across nine labels such as “price up” or “price down”.
- **ECT-Sum:** A dataset for text summarization tasks, ECT-Sum consists of 2,425 document-summary pairs, containing Earnings Call Transcripts (ECTs) and bullet-point summarizations from Reuters [201].

The listed datasets are not exhaustive but represent a comprehensive selection of tasks and benchmarks used to evaluate FinLLMs across a range of financial NLP tasks. As highlighted in Lee et al. [380], in the sentiment analysis task, FLANG-ELECTRA achieved the best results (92% on F1) while FinMA-30B and GPT-4 achieved similar results (87% on F1) with a 5-shot prompting.

These datasets are instrumental in assessing the models' performance, guiding their development, and fostering innovation in the financial sector to address more advanced financial tasks:

- **Relation Extraction (RE):** FinRED [210] is a key dataset curated from financial news and earnings call transcripts, containing 29 finance-specific relation tags (i.e., owned by). It's instrumental in identifying and classifying relationships between entities within financial texts.
- **Event Detection (ED):** The Event-Driven Trading (EDT) dataset, comprising news articles with event labels and stock price information, facilitates the detection of corporate events affecting stock prices [143].
- **Causality Detection (CD):** FinCausal20 from the Financial Narrative Processing (FNP) workshop focuses on identifying cause-and-effect relationships in financial texts, a crucial aspect for generating meaningful financial summaries [98]. It shares two tasks: detecting a causal scheme in a given text and identifying cause-and-effect sentences.
- **Numerical Reasoning (NR):** Datasets like FiNER-139 and ConvFinQA are designed to test a model's ability to perform calculations and understand financial contexts based on numerical data within texts.
- **Structure Recognition (SR):** The FinTabNet [142] dataset, collected from earnings reports, emphasizes the detection of table structures and the recognition of logical relationships within financial documents.
- **Multimodal Understanding (MM):** Datasets like MAEC [95] and MONOPOLY (Mathur et al. [194]) introduce multimodal data (audio, video, text, time series) from earnings calls and monetary policy discussions, challenging models to integrate diverse data formats.
- **Machine Translation (MT) in Finance:** MINDS-14 [112] and MultiFin (Jørgensen et al. [282]) datasets offer multilingual financial text, aiding in the development of models that can translate and comprehend financial information across languages.
- **Market Forecasting (MF):** This task extends beyond stock movement prediction, focusing on broader market trend forecasting³⁶ using datasets that combine sentiment analysis, event detection, and multimodal cues³⁷.

Recent studies have shown that general purpose model can outperform fine-tuned models on some tasks. Still, they fail in some other cases carefully analyzed in Li et al. [292]. Some interesting results are shown in Table 10, Table 11, Table 12, Table 13. For example, in the sentiment analysis task, FinMA-30B and GPT-4 achieved similar results (87% on F1) with a 5-shot prompting, while FLANG-ELECTRA achieved the best results (92% on F1) Lee et al. [380], while GPT-4 could be the first choice for Sentiment Analysis and Relation Extraction tasks.

³⁶Market price, volatility, and risk

³⁷Like StockEmotions [289]), EDT (Zhou, Ma, and Liu [143]), MAEC (Li et al. [95]) and MONOPOLY (Mathur et al. [194])

Data Model	50% Agreement		100% Agreement	
	Accuracy	F1 score	Accuracy	F1 score
ChatGPT ₍₀₎	0.78	0.78	0.90	0.90
ChatGPT ₍₅₎	0.79	0.79	0.90	0.90
GPT-4 ₍₀₎	0.83	0.83	0.96	0.96
GPT-4 ₍₅₎	0.86	0.86	0.97	0.97
BloombergGPT ₍₅₎	/	0.51	/	/
GPT-NeoX ₍₅₎	/	0.45	/	/
OPT6B ₍₅₎	/	0.49	/	/
BLOOM176B ₍₅₎	/	0.50	/	/
FinBert	0.86	0.84	0.97	0.95

Table 10: Results on the Phrasebank dataset. The sub-script (n) following an LLM name represents the number of shots. The best results are marked in bold. The results of other LLMs, like BloombergGPT, are from the corresponding papers. ‘/’ indicates the metrics were not included in the original study. Source: Li et al. [292].

Model	Category	Weighted F1
ChatGPT ₍₀₎	OpenAI LLMs	75.90
ChatGPT ₍₅₎	OpenAI LLMs	78.33
GPT-4 ₍₉₎	OpenAI LLMs	87.15
GPT-4 ₍₅₎	OpenAI LLMs	88.11
BloombergGPT ₍₅₎	Domain LLM	75.07
GPT-NeoX ₍₅₎	Prior LLMs	50.59
OPT 6B ₍₅₎	Prior LLMs	51.60
BLOOM 176B ₍₅₎	Prior LLMs	53.12
RoBERTa-large	Fine-tune	87.09

Table 11: Results for the sentiment analysis task on the FiQA dataset. Source: Li et al. [292].

Model	Weighted F1
ChatGPT ₍₀₎	71.78
ChatGPT ₍₅₎	74.84
GPT-4 ₍₀₎	84.17
GPT-4 ₍₅₎	86.00
BloombergGPT ₍₅₎	82.20
GPT-NeoX ₍₅₎	73.22
OPT6B ₍₅₎	79.41
BLOOM176B ₍₅₎	76.51
BERT	95.36

Table 12: Results on the headline classification task. Source: Li et al. [292].

BloombergGPT The BloombergGPT model, developed by Wu et al. [350], is a specialized LLM tailored for the financial domain. With its 50 billion parameters, is posited to be

Model	Entity F1
ChatGPT ₍₀₎	29.21
ChatGPT ₍₂₀₎	51.52
GPT-4 ₍₀₎	36.08
GPT-4 ₍₂₀₎	56.71
BloombergGPT ₍₂₀₎	60.82
GPT-NeoX ₍₂₀₎	60.98
OPT6B ₍₂₀₎	57.49
BLOOM176B ₍₂₀₎	55.56
CRF _{CoNLL}	17.20
CRF _{FIN5}	82.70

Table 13: Results of few-shot performance on the NER dataset. CRF_{CoNLL} refers to the CRF model trained on general CoNLL data, and CRF_{FIN5} refers to the CRF model trained on FIN5 data. Source: Li et al. [292].

Model	FinQA	ConvFinQA
ChatGPT ₍₀₎	48.56	59.86
ChatGPT ₍₃₎	51.22	/
ChatGPT (CoT)	63.87	/
GPT-4 ₍₀₎	68.79	76.48
GPT-4 ₍₃₎	69.68	/
GPT-4 (CoT)	78.03	/
BloombergGPT ₍₀₎	/	43.41
GPT-NeoX ₍₀₎	/	30.06
OPT6B ₍₀₎	/	27.88
BLOOM176B ₍₀₎	/	36.31
FinQANet (fine-tune)	68.90	61.24
Human Expert	91.16	89.44
General Crowd	50.68	46.90

Table 14: Model performance (accuracy) on the question answering tasks. FinQANet here refers to the best-performing FinQANet version based on RoBERTa-Large [154]. Due to its conservation nature, few-shot and CoT learning cannot be executed on ConvFinQA.

the apex of financial language models, having been trained on a comprehensive dataset of an unprecedented scale within the financial domain. Wu et al. [350] detail the intricacies of BloombergGPT’s training regimen, which employed an amalgamation of financial texts, encompassing a multitude of formats, and a general dataset to ensure versatility³⁸ as shown in Table 15.

The core of BloombergGPT’s training material involved 363 billion tokens of finance-specific data, accompanied by a general corpus of 345 billion tokens. The dataset’s breadth is vast, incorporating textual data spanning web sources, news articles, financial reports, and proprietary content from Bloomberg terminals. This diversified data portfolio enables the model to expertly navigate the financial lexicon and nuances.

³⁸“FINPILE”, a comprehensive dataset consisting of a range of English financial documents including news, filings, press releases, web-scraped financial documents, and social media drawn from the Bloomberg archives augmented with public available data.

Dataset	Docs	C/D	Chars	C/T	Toks	T%
FINPILE	175,886	1,017	17,883	4.92	6,935	51.27%
Web	158,250	933	14,768	4.96	2,978	42.01%
News	10,040	1,665	1,672	4.44	376	5.31%
Filings	3,335	2,340	780	5.39	145	2.04%
Press	1,265	3,443	435	5.06	86	1.21%
Bloomberg	2,996	758	227	4.60	49	0.70%
PUBLIC	50,744	3,314	16,818	4.87	3,454	48.73%
C4	34,832	2,206	7,683	5.56	1,381	19.48%
Pile-CC	5,255	4,401	2,312	5.42	427	6.02%
GitHub	1,428	5,364	766	3.38	227	3.20%
Books3	19	552,398	1,064	4.97	214	3.02%
PubMed Central	294	32,181	947	4.51	210	2.96%
ArXiv	124	47,819	541	3.56	166	2.35%
OpenWebText2	1,684	3,850	648	5.07	128	1.80%
FreeLaw	349	15,381	537	4.99	108	1.80%
StackExchange	1,538	2,201	339	4.17	81	1.15%
DM Mathematics	100	8,193	82	1.92	43	0.60%
Wikipedia (en)	590	2,988	176	4.65	38	0.53%
USPTO Backgrounds	517	4,339	224	6.18	36	0.51%
PubMed Abstracts	1,527	1,333	204	5.77	35	0.50%
OpenSubtitles	38	31,055	119	4.90	24	0.34%
Gutenberg (PG-19)	3	399,351	112	4.89	23	0.32%
Ubuntu IRC	1	539,222	56	3.16	18	0.25%
EuroParl	7	65,053	45	2.93	15	0.21%
YouTubeSubtitles	17	19,831	33	2.54	13	0.19%
BookCorpus2	2	370,384	65	5.36	12	0.17%
HackerNews	82	5,009	41	4.87	8	0.12%
PhilPapers	3	74,827	23	4.21	6	0.08%
NIH ExPorter	92	2,165	20	6.65	3	0.04%
Enron Emails	2	1,882	20	3.90	3	0.04%
Wikipedia (fr/1/22)	2,218	3,271	76	3.06	237	0.32%
TOTAL	226,631	1,531	34,701	4.89	7,089	100.00%

Table 15: Breakdown of the full training set used to train BLOOMBERGGPT. The statistics provided are the average number of characters per document (“C/D”), the average number of characters per token (“C/T”), and the percentage of the overall tokens (“T%”). Source: Wu et al. [350].

Wu et al. [350] proffer insights into their methodological choices and their repercussions on model performance. The authors used parallel tokenizer training strategies because the Unigram tokenizer was found to be inefficient for processing the entire Pile dataset. So the dataset was split into domains, and each domain was further split into chunks. Every chunk was tokenized by a separate tokenizer, and then the tokenizer from each domain was merged hierarchically using a weighted average of the probabilities of corresponding tokens. The result was cut from a tokenizer with 7 million tokens to only 2^{17} tokens, dropping tokens with the smallest probabilities.

The BloombergGPT model is a decoder-only causal language model based on BLOOM [349]. The model contains 70 layers of transformer decoder blocks defined as follows:

$$h_\ell = h_{\ell-1} + \text{SA}(\text{LN}(h_{\ell-1}))$$

$$h_\ell = h_\ell + \text{FFN}(\text{LN}(h_\ell))$$

where SA is multi-head self-attention, LN is layer-normalization, and FFN is a feed-forward network with 1 hidden layer. Inside FFN, the non-linear function is GELU [30]. ALiBi positional encoding is applied through additive biases at the self-attention component of the transformer network [180]. The input token embeddings are tied to the linear mapping before the final softmax. The model also has an additional layer of normalization after token embeddings.

BloombergGPT’s prowess was rigorously benchmarked against a suite of established LLMs’ assessments, financial-specific benchmarks, and a series of internally devised tests. The model exhibited a remarkable ability to outperform existing models on financial NLP tasks, a testament to the efficacy of its specialized training as shown in Table 16, Table 17, and Table 18. BloombergGPT’s performance on standard, general-purpose benchmarks was also evaluated, demonstrating its versatility and proficiency across a range of NLP tasks.

Overall, while BloombergGPT falls behind the much larger PaLM_{540B} (10×parameters) and BLOOM_{176B} (3.5x parameters), it is the best-performing among similarly sized models. In fact, its performance is closer to BLOOM_{176B} than it is to either GPT-NeoX or OPT_{66B}.

In sum, according to benchmarks in Wu et al. [350], developing finance-specific BloombergGPT did not come at the expense of its general-purpose abilities.

	BLOOMBERGGPT	GPT-NeoX	OPT_{66B}	BLOOM_{176B}
ConvFinQA	43.41	30.06	27.88	36.31
FiQA SA	75.07	50.59	51.60	53.12
FPB	51.07	44.64	48.67	50.25
Headline	82.20	73.22	79.41	76.51
NER	60.82	60.98	57.49	55.56
All Tasks (avg)	62.51	51.90	53.01	54.35
All Tasks (WR)	0.93	0.27	0.33	0.47

Table 16: Results on financial domain tasks. Source: Wu et al. [350].

	BLOOMBERGGPT	GPT-NeoX	OPT_{66B}	BLOOM_{176B}
Equity News	79.63	14.17	20.98	19.96
Equity Social Media	72.40	66.48	71.36	68.04
Equity Transcript	65.06	25.08	37.58	34.82
ES News	46.12	26.99	31.44	28.07
Country News	49.14	13.45	17.41	16.06
All Tasks (avg)	62.47	29.23	35.76	33.39
All Tasks (WR)	1.00	0.00	0.67	0.33

Table 17: Results on internal aspect-specific sentiment analysis datasets. BLOOMBERGGPT far outperforms all other models on sentiment analysis tasks. Source: Wu et al. [350].

	BLOOMBERGGPT	GPT-NeoX	OPT_{66B}	BLOOM_{176B}
NER				
BFW	72.04	71.66	72.53	76.87
BN	57.31	52.83	46.87	59.61
Filings	58.84	59.26	59.01	64.88
Headlines	53.61	47.70	46.21	52.17
Premium	60.49	59.39	57.56	61.61
Transcripts	75.50	70.62	72.53	77.80
Social Media	60.60	56.80	51.93	60.88
All Tasks (avg)	62.63	59.75	58.09	64.83
All Tasks (WR)	0.57	0.29	0.19	0.95
NER+NED				
BFW	55.29	34.92	36.73	39.36
BN	60.09	44.71	54.60	49.85
Filings	66.67	31.70	65.63	42.93
Headlines	67.17	36.46	56.46	42.93
Premium	64.11	40.84	57.06	42.11
Transcripts	73.15	23.65	70.44	34.87
Social Media	67.34	62.57	70.57	65.94
All Tasks (avg)	64.83	39.26	58.79	45.43
All Tasks (WR)	0.95	0.00	0.67	0.38

Table 18: Results on internal NER and NED datasets. On NER, while the much larger BLOOM176b model outperforms all other models, results from all models are relatively close, with BLOOMBERGGPT outperforming the other two models. On NER+NED, BLOOMBERGGPT outperforms all other models by a large margin. Source: Wu et al. [350].

2.4.3 LLMs in Education

The advent of LLMs has significantly impacted education. LLMs can be leveraged to create educational content tailored to individual student needs, providing explanations, generating practice problems, and even offering feedback.

Integrating LLMs into educational frameworks offers a rich tapestry of potential enhancements to teaching and learning experiences. The transformative influence of such technology is particularly marked in tasks that can benefit from automation, such as grading and personalized feedback on student work. Through their nuanced understanding of language, LLMs can provide insightful assessments that highlight the strengths and weaknesses in student assignments, which may span essays, research papers, and various other forms of written submissions. An additional benefit is LLMs' capacity to detect plagiarism, bolsters academic evaluation's integrity by mitigating the risk of academic dishonesty. This ability to provide quick and precise feedback can afford educators more time to address individual student needs, leading to a more targeted and effective teaching approach.

LLMs can achieve student-level performance on standardized tests [370] in a variety of subjects of mathematics (e.g., physics, computer science) on both multiple-choice and free-response problems. Additionally, these models can assist in language learning, both for native speakers and language acquisition, due to their deep understanding of linguistic structures and idiomatic expressions.

In the realm of intelligent tutoring systems, LLMs can be applied to simulate one-on-one interaction with a tutor, adapting to the student's learning pace, style, and current level of knowledge. These systems can engage in dialogue, answer student queries, and provide explanations, much like a human tutor would [305, 217].

Furthermore, LLMs have the capacity to automate the grading process by evaluating open-ended responses in exams and assignments. This approach can free up time for educators to focus on more personalized teaching methods and direct student engagement.

The intersection of LLMs and education also extends to research, where these models can aid in summarizing literature, generating hypotheses, and even writing research proposals or papers, albeit with careful oversight to ensure academic integrity.

In administrative and support roles, LLMs can streamline communication with students, handle routine inquiries, and manage scheduling and reminders, enhancing the overall educational experience for students and faculty.

To tap into the full potential of LLMs in education, it is crucial to address challenges such as ensuring the reliability of the information provided, avoiding biases, and maintaining privacy and security, especially in data-sensitive environments like schools and universities.

2.4.4 LLMs in Law

The legal sector is another domain that the advent of LLMs has significantly impacted. A number of tasks in the legal field, such as legal document analysis [253]), legal judgment prediction (Trautmann, Petrova, and Schilder [222]), and legal document writing (Choi et al. [264], can be solved by LLMs with high accuracy and efficiency.

Few-shot learning	Few-shot reasoning	Zero-shot reasoning
<p>We are going to be doing Entailment/Contradiction reasoning applying the statute below:</p> <p>§7703. Determination of marital status (a) General rule (1) the determination of whether an individual is married shall be made as of [...]</p> <p>Premise: Alice and Bob got married on April 5th, 2012. Alice and Bob were legally separated under a decree of divorce on September 16th, 2017.</p> <p>Hypothesis: Section 7703(a)(2) applies to Alice for the year 2012.</p> <p>Answer: Contradiction X 4</p> <p>Premise: Alice and Bob got married on April 5th, 2012. Bob died September 16th, 2017.</p> <p>Hypothesis: Section 7703(a)(1) applies to Alice for the year 2012.</p> <p>Answer: Entailment</p>	<p>§2. Definitions and special rules [...] §63. Taxable income defined [...] §152. Dependent defined [...] §3306. Definitions [...]</p> <p>Premise: Bob is Alice's brother since April 15th, 2014.</p> <p>Hypothesis: Alice bears a relationship to Bob under section 152(d)(2)(B) for the year 2015.</p> <p>Answer: Alice is a sister of Bob since 2014, so Alice is a sister of Bob in 2015. Therefore, the answer is Entailment. X 10</p> <p>Premise: Alice has a son, Bob, who satisfies section 152(c)(1) for the year 2015.</p> <p>Hypothesis: Under section 152(a), Bob is a dependent of Alice for the year 2015.</p> <p>Answer: Bob is Alice's son, and Bob satisfies section 152(c)(1) for the year 2015. Therefore, Bob is a dependent of Alice for the year 2015 under section 152(a). Therefore, the answer is Entailment.</p>	<p>We are going to be doing Entailment/Contradiction reasoning applying the statute below:</p> <p>§152. Dependent defined (a) In general For purposes of this subtitle, the term "dependent" means- (1) a qualifying child, or (2) a qualifying relative. [...]</p> <p>Premise: Bob is Alice's son since April 15th, 2014.</p> <p>Hypothesis: Alice bears a relationship to Bob under section 152(d)(2)(C).</p> <p>Answer: Let's think step by step. Bob is Alice's son, so Alice is Bob's mother. Section 152(d)(2)(C) states that an individual bears a relationship to the taxpayer if the individual is the father or mother, or an ancestor of either. Since Alice is Bob's mother, Alice bears a relationship to Bob under section 152(d)(2)(C). Therefore, the answer is Entailment.</p>

Figure 13: Prompts used in Blair-Stanek, Holzenberger, and Durme [253] to pose SARA test cases to GPT-3. The top boxes, in orange, contain statutes (optional). Example cases are in blue; in zero-shot, no example cases exist. At the bottom, in green, are test cases. The text highlighted in yellow is generated by GPT-3. If GPT-3's first response is unclear, the second prompt with "Therefore the answer is" is used, following Kojima et al. [285]. Source: Trautmann, Petrova, and Schilder [222].

Blair-Stanek, Holzenberger, and Durme [253] evaluates the capacity of OpenAI's GPT-3

model, specifically text-davinci-003, to perform statutory reasoning³⁹, a fundamental skill in legal practice, on an established dataset known as SARA (StAtutory Reasoning Assessment). The investigation includes several approaches like dynamic few-shot prompting, chain-of-thought prompting, and zero-shot prompting (examples in Figure 13).

The model surpasses previous benchmarks yet still exhibits considerable room for improvement, especially when handling simple synthetic statutes, revealing limitations in its current statutory reasoning capabilities even though GPT-3 has some prior knowledge of the U.S. Code.

Method	Constitutional Law	Taxation	Torts	Total
Simple	21/25	24/60	6/10	51/95
CoT	21/25	18/60	5/10	44/95
Rank Order	20/25	21/60	6/10	47/95

Table 19: Comparison of Multiple Choice Methods. Source: Choi et al. [264].

Choi et al. [264] explored ChatGPT’s ability to write law school exams at the University of Minnesota Law School, encompassing multiple choice and essay questions across four courses. ChatGPT generated answers for Constitutional Law, Employee Benefits, Taxation, and Torts exams, with varying question formats across these subjects. These answers were blindly graded in line with the standard grading process. ChatGPT managed to pass all four classes, averaging a C+ grade, demonstrating better performance on essay questions compared to multiple-choice, with notable strengths in organizing and composing essays (Table 19).

Despite its overall passing performance, ChatGPT ranked at or near the bottom in each class. The model’s essays showcased a strong grasp of basic legal rules but struggled with issue spotting and detailed application of rules to facts. The findings suggest that while ChatGPT can assist in legal education and potentially in legal practice, it currently lacks the nuanced understanding and depth of reasoning required for high-level legal analysis.

Recent studies on the latest GPT-4 model have shown that it can achieve a top 10% score in a simulated bar exam compared with human test-takers [370], while Nay [202] and exhibit powerful abilities of legal interpretation and reasoning. To further improve the performance of LLMs in the law domain, specially designed legal prompt engineering is employed to yield advanced performance in long legal document comprehension and complex legal reasoning [364].

2.4.5 LLMs in Scientific Research

LLMs in scientific research can be employed across various stages of the research process, from literature review to hypothesis generation, brainstorming, data analysis, manuscript drafting, proofreading, and peer review. Empirical evidence underscores the aptitude of LLMs in managing tasks dense with scientific knowledge, such as those presented by PubMedQA [69] and BioASQ [178]. It is particularly true for LLMs pre-trained on scientific corpora, including, but not limited to, Galactica [220] and Minerva [182].

Due to their capabilities, LLMs are poised to play an integral role as supportive tools throughout the scientific research process [362]. During the initial stages of research, such as brainstorming, LLMs can assist in generating novel research ideas and hypotheses, thereby fostering creativity and innovation⁴⁰.

³⁹Statutory reasoning is the application of legal rules written by legislative bodies to facts that are also in natural language

⁴⁰Randomness and creativity of a model can be tuned using specific parameters. For example temperature is a OpenAI ChatGPT, GPT-3 and GPT-4 models parameter that govern the randomness and thus the creativity of the responses.

In the literature review phase, LLMs can perform exhaustive reviews, encapsulating the state of advancement within specific scientific disciplines [274, 146] providing explanations for scientific texts and mathematics with follow-up questions.

Progressing to the phase of research ideation, LLMs have displayed potential in formulating compelling scientific hypotheses [311]. In Park et al. [311], the authors show the ability of GPT-4 to generate hypotheses in the field of materials science, showcasing the model's capacity to propose research directions. Through examining conversations, it was evident that GPT-4 generates richer and more specific information than the prompts provided, disproving the mirroring hypothesis. While checking for verbatim copying was more challenging, GPT-4 does seem to reflect current academic trends to an uncanny degree. However, it also combines disciplines and innovates concepts, leading to both errors and genuine creative insights. The authors compared the process to how cosmic rays can drive biological evolution through mutations: radiations break DNA strands and cause cancer and deaths, but can also drive mutations and evolution of the biosphere. Given the highlighted limitation, LLMs can be used to generate hypotheses for further human evaluation and refinement.

In the subsequent data analysis stage, LLMs can be harnessed for automating the examination of data attributes, including exploratory data analysis, visualization, and the extraction of analytical inferences [261]. In Hassan, Knipper, and Santu [276], the authors demonstrate the utility of GPT-4 in automating data analysis tasks, such as data cleaning, feature engineering, and model selection, thereby streamlining the data science workflow.

Regarding proofreading, LLMs can enhance the quality of scientific manuscripts by identifying grammatical errors, improving readability, and ensuring adherence to academic conventions. In addition, LLMs can go beyond helping users check grammar and can further generate reports about document statistics, vocabulary statistics, etc, change the language of a piece to make it suitable for people of any age, and even adapt it into a story [177]. While ChatGPT has some usability issues when it comes to proofreading, such as being over 10 times slower than DeepL and lacking in the ability to highlight suggestions or provide alternative options for specific words or phrases [306], it should be noted that grammar-checking is just the tip of the iceberg. ChatGPT can also be valuable in improving language, text restructuring, and other writing aspects.

Furthermore, in the manuscript drafting phase, the utility of LLMs extends to aiding scientific writing endeavors [279, 251], offering a multitude of services such as condensing existing materials and refining the written prose [255]. As explained in Buruk [255] and Hussam Alkaissi [279], LLMs can assist in generating abstracts, introductions, and conclusions, thereby enhancing the overall quality of scientific manuscripts.

Finally, in the peer review process, LLMs can contribute to automating the peer review process, undertaking tasks like error identification, compliance with checklists, and prioritization of submissions [298].

LLMs' utility spans beyond the aforementioned domains, with their deployment also being explored in the psychological sphere. Here, studies have argued LLMs for human-like traits, encompassing self-perception, Theory of Mind (ToM)⁴¹, and emotional cognition [287, 248]. Kosinski [287] employs classic false-belief tasks⁴², revealing a marked improvement in ToM ca-

⁴¹The ability to impute unobservable mental states to others

⁴²A false-belief task is a psychological test used to assess an individual's ability to understand that others can have beliefs about the world that are different from their own and that these beliefs can be incorrect. This ability is a crucial component of the Theory of Mind (ToM), which is the capacity to attribute mental states—beliefs, intents, desires, emotions, knowledge, etc.—to oneself and others and to understand that others have beliefs, desires, and intentions that are different from one's own.

The classic example of a false-belief task is the Sally-Anne test, used primarily with children. The test involves two dolls, Sally and Anne. Sally has a basket, while Anne has a box. In the presence of Sally, a marble is placed in Sally's basket. Sally then leaves the room, and while she's away, Anne takes the marble from Sally's

pabilities in more recent versions of GPT-3. Specifically, the davinci-002 version solved 70% of ToM tasks, while the davinci-003 version achieved a 93% success rate, demonstrating performances akin to seven and nine-year-old children, respectively. Notably, GPT-3.5’s performance in ToM assessments parallels that of nine-year-olds, suggesting nascent ToM capabilities in LLMs. The study hypothesizes that ToM-like abilities might emerge spontaneously in AI without explicit programming, especially in LLMs trained in human language. In the context of AI, particularly in LLMs like GPT-3, the ability to perform well on false-belief tasks suggests a sophisticated level of language understanding and a rudimentary form of Theory of Mind, albeit not conscious or sentient like in humans. Not surprisingly, initial excitement based on anecdotal performance of LLMs on reasoning tasks has been dissipated to some extent by the recent spate of studies, questioning the robustness of such behaviors—be they planning [333, 379], simple arithmetic and logic[266], theory of mind [331, 386], or general mathematical and abstract benchmarks [269, 307].

Moreover, the application of LLMs in software engineering is also gaining traction, with initiatives in code suggestions [323], code summarizations [325], and automated program repairs [352].

3 Foundations of Large Language Models

Large Language Models (LLMs) have revolutionized the field of Natural Language Processing (NLP) by achieving state-of-the-art performance on a wide range of tasks, such as text generation, text classification, and machine translation. These models are trained on vast amounts of text data to learn the underlying structure of the language and capture the relationships between words.

In the following sections, we will explore the key concepts and techniques that underpin the development of LLMs, including pre-training strategies and major datasets used for training and evaluation, as well as the Transformer architecture, which forms the basis of many modern LLMs.

After that, we will discuss some model adaptation techniques for fine-tuning LLMs for specific tasks or domains.

Finally, we will discuss the tuning and quantization of LLMs, techniques used to reduce the model’s size and computational complexity, making it more efficient for deployment on resource-constrained devices.

3.1 Pre-training

Pre-training constitutes a foundational phase in developing Large Language Models (LLMs). It allows the model to capture the relationships between words and generate coherent and contextually relevant text, laying the groundwork for its subsequent performance on specific NLP tasks [65, 88]. This phase involves training a language model on a vast corpus of text data before fine-tuning it on a smaller, task-specific dataset, such as text generation or text classification, to improve its performance on that task. Moreover, the extensive pre-training on diverse corpora enables LLMs to develop a broad understanding, making them adaptable to a wide range of domains and languages [73, 75]. Despite its advantages, LLM pre-training

basket and puts it in her box. The child is then asked where Sally will look for the marble when she returns. The correct answer is Sally’s basket, where she left the marble. A child who can correctly predict where Sally will look for the marble demonstrates an understanding that Sally holds a false belief about the location of the marble. Successfully completing a false-belief task indicates that the individual can understand that others can hold false beliefs and that these beliefs can influence their actions, a critical step in the development of social cognition and empathy.

is not without its challenges. The process requires substantial computational resources and energy, raising concerns about its environmental impact [80]. Additionally, the data used for pre-training can influence the model’s biases and sensitivities, necessitating careful curation of the training corpus to mitigate potential ethical and fairness issues [106].

The field is evolving towards more efficient pre-training methods, such as transfer learning, where a pre-trained model is adapted to new tasks or languages with minimal additional training [76]. Moreover, emerging approaches aim to enhance LLMs’ contextual awareness and ethical sensitivity during the pre-training phase, addressing the challenges of bias and fairness.

Several pre-training strategies have been used to train large language models, including unsupervised, supervised, and semi-supervised pre-training. Let’s explore each of these strategies in more detail.

3.1.1 Unsupervised pre-training

Unsupervised pre-training is a pre-training strategy involving training a model on a large corpus of text data without labels or annotations.

The model is trained to predict the next word, given the previous words in the sequence [88]. This is done using a technique called Autoregressive Language Modeling (ALM), where the model is trained to predict the probability distribution over the next word in the sequence given the previous words in the sequence in a unidirectional manner.

Models like GPT-3 and its variants use this autoregressive language modelling objective to pre-train over large text corpora and learn the relationships between words in the language. The main idea behind ALM is to predict the next token in a sequence based on the tokens that precede it. The computational realization of this modelling approach is typically achieved through neural networks, particularly transformers, which leverage self-attention mechanisms to encapsulate dependencies across varying distances in the input sequence [334].

During the generation process, a token is sampled based on the probability distribution predicted by the model for the next token position, appended to the sequence, and this augmented sequence is then fed back into the model iteratively to generate subsequent tokens [88]. Despite its prowess, the autoregressive nature of these models imbues them with an intrinsic limitation: the inability to leverage future context in token prediction, constraining their context comprehension to a unidirectional scope.

BERT and its variants, on the other hand, employ a masked language model (MLM) objective, where random words in a sentence are masked, and the model is trained to predict these masked words based on their context, integrating both preceding and succeeding context in representation learning [65].

3.1.2 Supervised pre-training

Supervised pre-training is a pre-training strategy that involves training a model on a large corpus of text data with labels or annotations. This paradigm contrasts with unsupervised pre-training, where models learn from raw text without explicit labels. The supervised approach enables models to learn representations more closely aligned with the end tasks, potentially enhancing their performance and efficiency [89].

In supervised pre-training, LLMs are exposed to a vast array of labelled data across various domains. This training regime involves teaching the model to predict the correct output given an input under the supervision of known input-output pairs. This approach helps in learning general language representations and imbues the model with domain-specific knowledge, which is particularly beneficial when the subsequent fine-tuning task is closely related to the pre-training data [74].

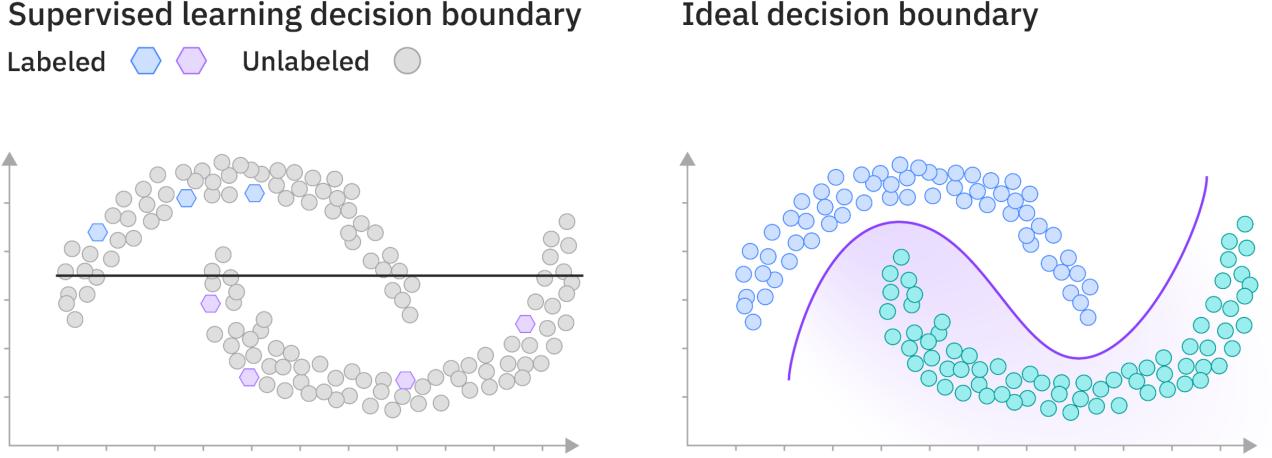


Figure 14: Using only the minimal labelled data points available, a supervised model may learn a decision boundary that will generalize poorly and be prone to misclassifying new examples. Source: Bergmann [252].

One significant advantage of supervised pre-training is its potential to reduce the labelled data required for fine-tuning over specific tasks. By learning robust representations during pre-training, LLMs can achieve high performance on downstream tasks even with relatively smaller datasets, a concept known as transfer learning [76]. Moreover, supervised pre-training can lead to improvements in model generalization, making LLMs more adept at handling unseen data or tasks that diverge from their initial training corpus.

The reliance on large labelled datasets introduces concerns regarding the cost and feasibility of data annotation, especially in specialized domains where expert knowledge is required. Furthermore, as shown in Figure 14, the risk of overfitting to the pre-training data is non-trivial, necessitating careful regularization and validation to ensure the model’s generalizability [45].

3.1.3 Semi-supervised pre-training

Semi-supervised pre-training emerges as a compelling paradigm in the evolution of Large Language Models (LLMs), blending the strengths of supervised and unsupervised learning methodologies. This hybrid training approach leverages a combination of labelled and unlabeled data, optimizing the utilization of available information and enhancing the model’s learning efficacy and adaptability [10, 14].

Semi-supervised pre-training involves the initial training of models using a vast corpus of unlabelled data akin to unsupervised pre-training. This phase allows the model to capture a broad understanding of language structures and patterns. Subsequently, the model undergoes further training or fine-tuning on a smaller labelled dataset, instilling task-specific knowledge and nuances [76, 42]. The rationale behind this approach is to exploit the abundance of readily available unlabeled data to develop a comprehensive language model, which is then refined using the more scarce labelled data to achieve superior performance on target tasks.

Various techniques underpin semi-supervised pre-training in LLMs. One prominent method involves self-training, where the model, initially trained on labelled data, generates pseudo-labels for the unlabeled dataset. These pseudo-labeled data points are then incorporated into further training cycles, iteratively enhancing the model’s accuracy and robustness [19].

Another notable technique is the use of consistency regularization, which ensures that the model produces similar outputs for perturbed versions of the same input data, enhancing the

model's stability and generalization capabilities [34].

Other key techniques in semi-supervised learning include transductive and inductive learning, with practical methods like label propagation and active learning aiding in leveraging unlabeled data. These approaches are instrumental in refining the model's decision-making capabilities [252].

Transductive learning, a concept primarily attributed to Vapnik [3], focuses on predicting specific examples from the training set without attempting to generalize beyond those. In transductive inference, the model is directly applied to the specific test set to infer the correct labels for the given unlabeled data. The key characteristic distinguishing transductive learning from other machine learning methods is its focus on the particular sample rather than a general rule applicable to new, unseen instances. One of the main applications of transductive learning is in the realm of support vector machines (SVMs), where it is employed to predict labels for a given, fixed set of test data, optimizing the margin not only for the training data but also for the test data, despite their labels being unknown [4].

Conversely, inductive learning aims to build a general model that predicts outcomes for new, unseen data based on the patterns learned from the training data. Label propagation (Figure 15) is a common technique in inductive learning, where the model infers the labels of unlabeled data points based on the labels of their neighbours in the feature space.

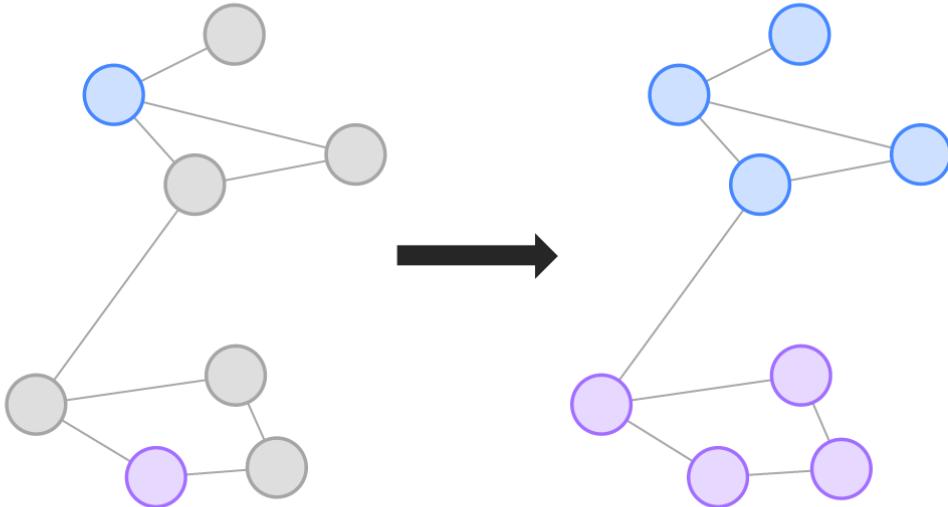


Figure 15: LEFT: original labelled and unlabeled data points. RIGHT: using label propagation, the unlabeled data points have been assigned pseudo-labels. Source: Bergmann [252].

Active learning is another inductive learning method that involves iteratively selecting the most informative data points for labelling and optimizing the model's performance with minimal labelled data. This approach is more general than transductive learning and underpins most supervised learning algorithms. The objective is to infer a function that can generalize well across unseen samples, not just the examples provided during the training phase. Inductive learning is fundamental to numerous machine learning algorithms, from linear regression to deep neural networks, where the model learns an underlying function that maps input data to output predictions, with the hope that this function will perform accurately on data not present in the training set [2].

The semi-supervised approach is predicated on certain assumptions about the underlying structure and distribution of the data, which facilitate the effective integration of unlabeled data into the learning process.

- **Cluster Assumption:** The cluster assumption posits that data points within the same cluster are more likely to share a label. This assumption underpins the idea that data points in high-density regions of the input space belong to the same class, while low-density regions denote boundaries between classes [14]. This principle guides the model in generalizing from labelled data points to nearby unlabeled ones within the same cluster.
- **Continuity Assumption:** Also known as the smoothness assumption, this posits that if two points in the input space are close to each other, then their corresponding outputs are also likely to be similar [9]. In practical terms, if two data points are close in the feature space, they will likely share the same label.
- **Manifold Assumption:** The manifold assumption suggests that high-dimensional data lie on a low-dimensional manifold. This assumption implies that the data points are situated on a manifold of much lower dimensionality embedded within the higher-dimensional space, and learning can be simplified if this manifold structure is discovered and exploited [11]. The manifold assumption often complements the cluster and continuity assumptions, providing a geometric interpretation of the data's distribution.
- **Low-Density Separation Assumption:** This assumption posits that the decision boundary between different classes should lie in regions of low data density [14]. Essentially, there is expected to be a natural separation or gap between classes, and the learning algorithm should prefer hypotheses that place the decision boundary in regions with few data points.

3.2 Data sources

Large Language Models (LLMs) strongly depend on extensive, high-calibre data for pre-training, with their efficacy closely tied to the nature and preprocessing of the utilized corpora. The primary sources of data for training and evaluating LLMs can be broadly categorized into general and specialized datasets, each serving distinct purposes in enhancing the models' capabilities [364].

3.2.1 General Data

This category typically encompasses web content, literary works, and conversational texts, prized for their voluminous, varied, and accessible nature, thereby bolstering LLMs' language modelling and generalization prowess. Including general data, such as web pages and books, offers a rich lexicon spanning various themes, essential for the comprehensive training of LLMs. As shown in Figure 16, general-purpose data are among the most commonly used general data sources for training LLMs.

Three important general data sources are:

- **Webpages:** Web content, extracted from the internet, is a valuable source of diverse and up-to-date text data, encompassing news articles, blog posts, and forum discussions. This data is instrumental in training LLMs to gain different linguistic knowledge and enhance generalization capabilities [88, 99]. Crawled web data tends to contain a mix of high-quality and noisy text, necessitating careful preprocessing to ensure the data's quality and relevance.

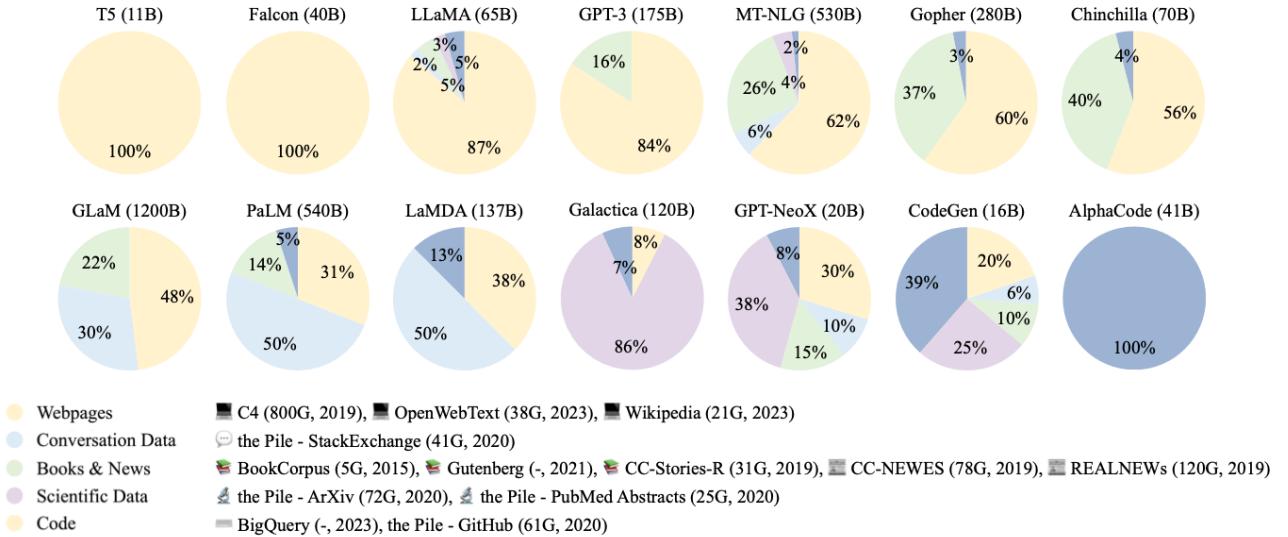


Figure 16: Commonly-used data sources for training and evaluating Large Language Models (LLMs). Source: Zhao et al. [364].

- **Conversation text:** Conversation text, including chat logs and social media interactions, provides a rich source of informal language and colloquial expressions, enabling LLMs to capture the nuances of human communication [241]. This data is particularly useful for training LLMs on question answering [155] and sentiment analysis tasks [82]. Conversational data often involve multiple speakers, so an effective way is to transform the conversation into a tree structure, where the utterance is linked to the one it is replying to. The tree can be divided into multiple subtrees, each one representing a sub-conversation, which can be collected in the pre-training corpus. Overtraining on conversational data can lead to the model to a performance decline since the declarative instructions and direct interrogatives can be erroneously interpreted as the beginning of a conversation [241].
- **Books:** Books, comprising novels, essays, and scientific literature, offer a rich source of long structured and coherent text data, enabling LLMs to learn complex language structures and thematic nuances [27]. This data is instrumental in training LLMs on literary text generation tasks and enhancing their proficiency in narrative comprehension and storytelling [75].

3.2.2 Specialized Data

Tailored to refine LLMs' proficiency in particular tasks, specialized datasets encompass multilingual text, scientific literature, and programming code. Specialized datasets are useful to improve the specific capabilities of LLMs on downstream tasks. Next, we introduce three kinds of specialized data.

- **Multilingual text:** Multilingual text data, spanning multiple languages and dialects, is crucial for training LLMs to understand and generate text in diverse linguistic contexts [364]. This data is instrumental in enhancing the models' cross-lingual capabilities and enabling them to perform translation tasks across different languages [364]. BLOOM [349] and PaLM [155] are two models that have been trained on multilingual text data to improve their performance on cross-lingual tasks. They have impressive performances on translation, multilingual question answering, and cross-lingual summarization tasks, and they achieve comparable or superior results to models fine-tuned in specific languages.

- **Scientific literature:** Scientific literature, encompassing research papers, patents, and technical documents, provides a rich source of domain-specific text data essential for training LLMs on scientific text generation and reasoning tasks [364, 220, 182]. Existing efforts to build the scientific corpus for training LLMs mainly collect arXiv papers, scientific textbooks, math web pages, and other related scientific resources. Data in scientific fields are complex, commonly including mathematical symbols and protein sequences, so specific tokenization and preprocessing techniques are required to transform these different data formats into a unified form that language models can process.
- **Code:** Code, encompassing source code snippets and software documentation, is a valuable source of structured text data, essential for training LLMs on code generation and code completion tasks [364, 203]. Code data is often collected from open-source repositories like GitHub and StackOverflow, and it is used to train LLMs to generate code snippets, complete code fragments, and perform code summarization tasks. Recently Chen et al. [108] and Austin et al. [105] have shown that models trained on code data can be used to generate code with high accuracy and efficiency, and they can be used to improve the performance of code completion tasks. Generated code can successfully pass expert-designed unit-test cases [108] or solve competitive programming problems [184]. In general, two types of code corpora are used: one is question-answering datasets like Stack Exchange [235]; the second is public software repositories like GitHub [108] where code, comments and docstring are collected for utilization.

3.2.3 Commonly-used data sources.

The development and evaluation of Large Language Models (LLMs) rely heavily on the availability of high-quality datasets that span diverse domains and languages. The datasets in Table 20 serve as the foundation for pre-training and fine-tuning LLMs, enabling researchers to assess the models' performance on a wide range of tasks, from text generation to translation.

Corpora	Size	Source	Update Time
BookCorpus [27]	5GB	Books	Dec-2015
Gutenberg [392]	-	Books	Dec-2021
C4 [99]	800GB	CommonCrawl	Apr-2019
CC-Stories-R [55]	31GB	CommonCrawl	Sep-2019
CC-NEWS [73]	78GB	CommonCrawl	Feb-2019
REALNEWS [82]	120GB	CommonCrawl	Apr-2019
OpenWebText [67]	38GB	Reddit links	Mar-2023
Pushift.io [87]	2TB	Reddit links	Mar-2023
Wikipedia [393]	21GB	Wikipedia	Mar-2023
BigQuery [390]	-	Codes	Dec-2023
the Pile [111]	800GB	Other	Dec-2020
ROOTS [179]	1.6TB	Other	Jun-2022

Table 20: Statistics of commonly-used data sources. Source: Zhao et al. [364]

This section will explore some of the most commonly used data sources for training and evaluating LLMs. Based on their content types, we categorize these corpora into six groups: Books, CommonCrawl, Reddit links, Wikipedia, Code, and others.

- **Books:** BookCorpus [27] and Gutenberg [392] are two prominent datasets that contain text from a wide range of books spanning various genres and topics. These datasets

are valuable for training LLMs on literary text and assessing their performance on text generation tasks.

BookCorpus is a dataset consisting of text from over 11,000 books (e.g., novels and biographies), while Gutenberg is a collection of over 70,000 free ebooks, including novels, essays, poetry, drama, history, science, philosophy, and other types of works, in the public domain.

BookCorpus is commonly used in previous small-scale models (e.g., GPT [51] and GPT-2 [75]), while Gutenberg is used in more recent large-scale models (i.e., Llama [330]).

Book1 and Book2 used in GPT-3 [88] are much larger than BookCorpus but have not been publicly released.

- **CommonCrawl:** CommonCrawl [391] is a vast web corpus that contains data from billions of web pages covering diverse topics and languages. Due to noise and redundancy in the data, researchers often extract subsets of CommonCrawl for training LLMs. The main subsets used for training LLMs are C4⁴³ [99], CC-Stories-R [55], CC-NEWS [73], and REALNEWS [82].
- **Reddit links:** Reddit is a social media platform where users can submit links and posts and “upvote” or “downvote” them. Posts with a high number of “upvotes” are often considered useful and can be used to create high-quality datasets. OpenWebText [67] and Pushshift.io [87] are datasets that contain text data extracted from Reddit. These datasets are useful for training LLMs on social media text and assessing their performance on text generation and sentiment analysis tasks.
- **Wikipedia:** Wikipedia [393] is a widely-used dataset containing text from various articles. It’s an online encyclopedia with a large volume of high-quality articles. 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 [88], and LLaMA [330]). Wikipedia is available in multiple languages and can be used in multilingual settings.
- **Code:** Two major sources are GitHub, for open-source licensed code, and StackOverflow, for code-related question-answering platforms. Google has publicly released BigQuery [390], a dataset that contains code snippets from various programming languages. This dataset is useful for training LLMs (i.e., CodeGen [203]) on code text and assessing their performance on code generation and code completion tasks.
- **Others:** The Pile [111] and ROOTS [179] are datasets that contain text data from various sources, such as books, articles, and websites. The Pile contains 800GB of data from multiple sources, including books, websites, codes, scientific papers, and social media platforms. It’s widely used in training LLMs with different sizes (e.g., CodeGen(16B) [203] and Megatron-Turing NLG(530B) [214]). ROOTS comprises various smaller datasets (a total of 1.61 TB of text) in 59 different languages (containing natural languages and programming languages). It’s been used for training BLOOM [349].

A mixture of these datasets is often used to train LLMs, as they provide a diverse range of text data (Figure 16). The choice of datasets depends on the specific task and domain of interest and the computational resources available for training the model. Furthermore, to

⁴³Colossal Clean Crawled Corpus

train LLMs that are adaptative to specific tasks or domains, it is also important to consider the data sources that are relevant to them.

3.3 Data preprocessing

After collecting the data, the next step is to preprocess it to ensure that it is clean, consistent, and ready for training Large Language Models (LLMs), removing noise and irrelevant or potentially toxic information [155, 131, 299]. In Chen et al. [257], the authors propose a new data preprocessing system, DataJuicer, that can be used to improve the quality of the processed data.

A typical pipeline for data preprocessing involves several steps, as shown in Figure 17:

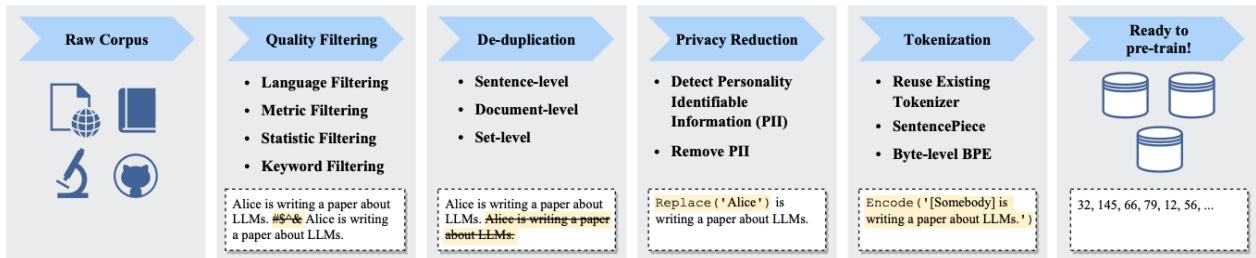


Figure 17: Common data preprocessing steps for training Large Language Models (LLMs). Source: Zhao et al. [364].

3.3.1 Quality Filtering.

The first step in data preprocessing is quality filtering, where the data is cleaned to remove irrelevant or low-quality content. Existing works mainly adopt two strategies for quality filtering: classifier-based and heuristic-based filtering.

The former approach involves training a classifier to distinguish between high-quality and low-quality data, using well-curated data (e.g., Wikipedia pages) as positive examples and noisy data (e.g., spam or irrelevant content) as negative examples. Rae et al. [131] and Du et al. [161] find that classifier-based filtering may remove high-quality data in dialect, colloquial, and sociolectal⁴⁴ languages, which potentially leads to bias in the pre-training data and diminishes the corpus diversity.

On the other hand, heuristic-based filtering involves setting predefined rules to identify and remove noisy data [349, 131]. The set of rules can be summarized as follows:

- *Language based filtering*. Remove data that is not in the target language.
- *Metric based filtering*. Remove data that does not meet certain quality metrics, e.g., perplexity, readability, or coherence. Perplexity (PPL) is one of the most common metrics for evaluating language models. This metric applies specifically to classical language models (sometimes called autoregressive or causal language models) and is not well-defined for masked language models like BERT [65]. Perplexity is defined as the exponential average negative log-likelihood of a sequence.

⁴⁴In sociolinguistics, a sociolect is a form of language or a set of lexical items used by a socioeconomic class, profession, age group, or other social group. Sociolects involve both passive acquisition of particular communicative practices through association with a local community, as well as active learning and choice among speech or writing forms to demonstrate identification with particular groups. Source: [Wikipedia](#) [393]

If we have a tokenized sequence $X = x_1, x_2, \dots, x_t$, the perplexity of the sequence is defined as:

$$PPL(X) = \exp \left\{ -\frac{1}{t} \sum_i^t \log p_\theta(x_i | x_{<i}) \right\} \quad (6)$$

where $\log p_\theta(x_i | x_{<i})$ is the log-likelihood of the token x_i given the previous tokens $x_{<i}$ in the sequence. Intuitively, it can be thought of as an evaluation of the model's ability to predict uniformly among the set of specified tokens in a corpus⁴⁵ [314].

- *Statistic based filtering.* Statistical features like punctuation distribution, symbol-to-word ratio, and sentence length can be used to filter out low-quality data.
- *Keyword based filtering.* Remove data that contains specific keywords that are noisy, irrelevant or toxic, like HTML tags, URLs, boilerplate text, or offensive language.

3.3.2 Deduplication.

The next step in data preprocessing is deduplication, where duplicate data are removed to reduce redundancy and improve the diversity of the training data. Moreover, Hernandez et al. [171] found that duplication may cause instability in the training process, leading to overfitting and poor generalization performance. Therefore, deduplication is essential to ensure the model is exposed to diverse text data during training.

It can be done at various granularities, such as at the document, paragraph, or sentence level. Low-quality sentences containing repeated words or phrases can be removed to improve the data quality. At the document level, the deduplication can be done by computing the overlap ratio of surface features (e.g., words and n-grams overlap) between documents and removing the duplicates that contain similar contents [330, 131, 349, 181]. To avoid the contamination problem, the deduplication process should be done before the data is split into training, validation, and test sets [155]. Chowdhery et al. [155] and Carlini et al. [150] have shown that the three deduplication strategies should be used in conjunction to improve the training of LLMs.

3.3.3 Privacy reduction.

Privacy reduction is another important step in data preprocessing, especially when dealing with sensitive or personal information. Since data is often collected from the web and contains user-generated content, the risk of privacy breaching is high [107]. This step involves anonymizing or obfuscating sensitive data to protect individuals' privacy. Common techniques for privacy reduction include masking personally identifiable information (PII), such as names, addresses, and phone numbers, and replacing them with generic placeholders or tokens [179]. Privacy attacks on LLMs can be attributed to duplicated PII data in the pre-training, which can be used to extract the original PII data [181]. Therefore, de-duplication can also reduce privacy risks to some extent.

3.3.4 Tokenization.

Tokenization is a crucial step in data preprocessing, where the text data is converted into tokens that can be processed by the model. The choice of tokenization method can significantly impact the model's performance, as different tokenization strategies can affect the model's ability to capture the underlying structure of the language.

⁴⁵This means that the tokenization procedure has a direct impact on a model's perplexity, which should always be taken into consideration when comparing different models.

Common tokenization techniques include word-based tokenization, subword-based tokenization, and character-based tokenization. Word-based tokenization splits the text into individual words, while subword-based tokenization breaks down the text into subword units, such as prefixes, suffixes, and roots. Character-based tokenization, on the other hand, tokenizes the text into individual characters. Word-based tokenization is the predominant method used in traditional NLP research [5].

However, word-based tokenization can be problematic for languages with complex morphology or limited vocabulary, as it may result in a large vocabulary size and sparse data representation. In some other languages, like Chinese, Japanese, and Korean, word-based tokenization is unsuitable because these languages do not have explicit word boundaries⁴⁶. Thus, several neural network-based models employed subword-based tokenization, such as Byte Pair Encoding (BPE) [35], Unigram [46], and WordPiece [36], to address these challenges.

Byte Pair Encoding (BPE) is a type of data compression technique that has been effectively adapted for natural language processing tasks, particularly in the domain of tokenization for large language models (LLMs). The BPE algorithm operates by iteratively merging the most frequent pair of bytes (or characters in the context of text) in a given dataset into a single, new byte (or character). It repeats this process until a specified number of merges has been reached or another stopping criterion has been met. The application of BPE in the field of NLP was popularized by Sennrich, Haddow, and Birch [35] in the context of neural machine translation. They demonstrated that using BPE allowed for efficient handling of rare and unknown words, commonplace in languages with rich morphology or specialized vocabularies, such as scientific texts or code. By splitting words into subword units, BPE balances the granularity of characters and the semantic units of full words, enabling models to represent a wide vocabulary with a limited set of tokens. BPE has been fundamental in the architecture of influential language models, such as OpenAI’s GPT series, BART and LLaMA.

WordPiece tokenization is a tokenization method that segments text into subword units, balancing the flexibility of character-based models and the efficiency of word-based models. Originating from speech processing [36], this method has found significant application in natural language processing, particularly within neural network-based models such as BERT and its variants. In WordPiece tokenization, a base vocabulary is first constructed with individual characters, and then more frequent and meaningful sub-word units are incrementally added. This construction process is guided by a criterion that maximises the language model likelihood on a training corpus, thus ensuring that the resulting tokens are optimal representations of the given data. The WordPiece algorithm iteratively merges the most frequently co-occurring pairs of tokens to form new sub-word units until a specified vocabulary size is reached. This tokenization strategy has effectively reduced out-of-vocabulary issues, as the model can use smaller sub-word units when encountering unfamiliar words. Moreover, by capturing subword regularities, WordPiece facilitates learning meaningful representations for morphologically rich languages within large language models. This is particularly advantageous for handling agglutinative languages, where words often comprise a series of affixed morphemes⁴⁷.

⁴⁶It can yield different segmentation results for the same input.

⁴⁷Agglutinative languages are a type of morphological linguistic classification in which words are formed by adding discrete units, each carrying a specific grammatical meaning. These discrete units are known as morphemes, which are the smallest grammatical units in a language. In agglutinative languages, morphemes are concatenated so that each morpheme represents a single grammatical function, such as tense, number, case, or aspect. For example, in Turkish – an agglutinative language – a single word can be made up of a base or root word with several affixes attached to it to modify its meaning. These affixes remain relatively invariant; they don’t undergo significant changes in form when they’re combined with other morphemes. Here’s an illustrative example from Turkish:

“ev” means “house”

“evler” means “houses” (plural)

“evlerim” means “my houses” (possessive plural)

Unigram tokenization is a statistical method that employs a unigram language model to segment text into tokens probabilistically. This technique, standing in contrast to the deterministic nature of Byte Pair Encoding, involves constructing a unigram model from a large initial vocabulary and iteratively refining it to maximize the likelihood of the observed corpus [46]. The essence of Unigram tokenization lies in its iterative pruning process, wherein less probable tokens are systematically eliminated from the vocabulary. To estimate the unigram language model, it adopts an Expectation-Maximization (EM) algorithm: at each iteration, it first finds the currently optimal tokenization of words based on the old language model and then re-estimates the probabilities of unigrams to update the language model. During this procedure, dynamic programming algorithms (i.e., the Viterbi algorithm) are used to efficiently find the optimal decomposition way of a word given the language model[364]. This probabilistic approach is adept at handling the linguistic complexities and variations found across different languages and domains. It particularly excels in the context of language models that require a nuanced understanding of morphological structures and sub-word variations. Unigram tokenization has been pivotal in developing the SentencePiece [46] tokenization library, renowned for its application in T5 and mBART. The adaptability and language-agnostic properties of Unigram tokenization make it a preferred choice for LLMs tasked with processing multilingual data [46].

3.4 LLM Adaptation

The adaptation of Large Language Models (LLMs) is a critical aspect of their deployment in real-world applications. It enables the models to be fine-tuned on specific tasks or domains after pre-training, enhancing their performance by minimizing the loss of generalization capabilities. Adaptation can be achieved through various techniques, such as instruction tuning and alignment tuning, which allow LLMs to enhance (or unlock) their abilities⁴⁸ and align their behaviours with human values or preferences [364].

3.4.1 Instruction Tuning

Instruction tuning is a technique that leverages natural language instructions to fine-tune pre-trained LLMs [231], which is highly related to supervised fine-tuning [205] and multi-task prompted training [209]. Instruction tuning enhances LLMs' ability to follow and comprehend natural language instructions. Unlike traditional fine-tuning, which adapts models to specific tasks, instruction tuning employs a more generalized approach that broadens the model's utility across a variety of tasks through an “instruction-following” paradigm (Figure 18).

FLAN⁴⁹ [231] is noted for substantially improving zero-shot learning capabilities when compared to traditional models like GPT-3 [88] (Figure 19).

Chung et al. [156] have shown instruction-tuned (Figure 20) PaLM⁵⁰ enhance model performance on various tasks (i.e., MMLU, BBH, TyDiQA and MGSM) when the model size is at

Each suffix attached to “ev” is a separate morpheme that changes the word’s meaning, indicating plurality and possession without ambiguity. This is in contrast to fusional languages, where a single affix can represent multiple grammatical categories, or isolating languages, where words generally do not change form at all, and grammatical relations are indicated by word order or separate words.

⁴⁸Some experiments on fine tuning general LLMs on specific tasks (e.g., planning) haven’t shown significant improvements, leading to the conclusion that fine tuning is just a way convert the task into a memory-based approximate retrieval problem rather than a way to improve or unlock the model’s reasoning capabilities. Sometimes, in common sense domains, or with enough fine tuning, the reasoning part may also be obviated by having seen a case that pretty much corresponds to the problem that needs to be solved [378].

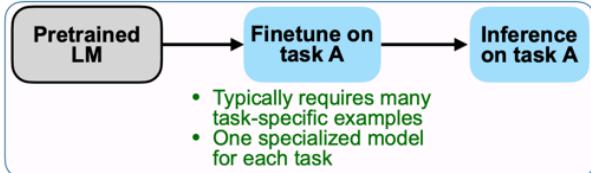
⁴⁹Finetuned LAnguage Net, a 137B parameter pre-trained language model and instruction tune it on over 60 NLP datasets verbalized via natural language instruction templates

⁵⁰Also called Flan-PaLM 540B-parameter model

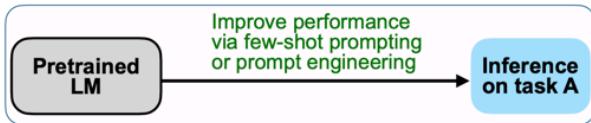
Table 21: A detailed list of available collections for instruction tuning.

Categories	Collections	Time	#Examples
Task	Nat. Inst. [199]	Apr-2021	193K
	FLAN [231]	Sep-2021	4.4M
	P3 [147]	Oct-2021	12.1M
	Super Nat. Inst. [229]	Apr-2022	5M
	MVPCorpus [219]	Jun-2022	41M
	xP3 [200]	Nov-2022	81M
	OIG [334]	Mar-2023	43M
Chat	HH-RLHF [148]	Apr-2022	160K
	HC3 [272]	Jan-2023	87K
	ShareGPT [65]	Mar-2023	90K
	Dolly [94]	Apr-2023	15K
	OpenAssistant [286]	Apr-2023	161K
Synthetic	Self-Instruct [228]	Dec-2022	82K
	Alpaca [327]	Mar-2023	52K
	Guanaco [110]	Mar-2023	535K
	Baize [357]	Apr-2023	158K
	BELLE [281]	Apr-2023	1.5M

(A) Pretrain–finetune (BERT, T5)



(B) Prompting (GPT-3)



(C) Instruction tuning (FLAN)

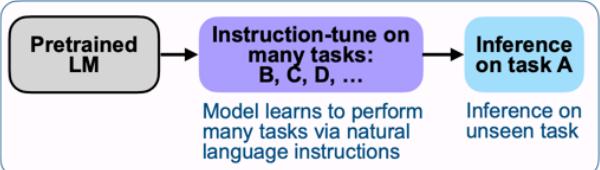


Figure 18: Overview of instruction tuning. Source: Zhao et al. [364].

least 62B, though a much smaller size might suffice for some specific tasks (e.g., MMLU).

Instruction tuning has been widely applied also in other models like Instruct-GPT [205] and GPT-4 [316]. Other experiments in Wei et al. [231] have shown that instruction tuning of LaMDA-PT started to significantly improve performance on zero-shot tasks when the model size is at least 68B.

Let’s look at the construction of instruction-formatted instances essential for instruction tuning. Generally, an instruction-formatted instance consists of a task description (called an *instruction*) and a set of input-output examples, optionally followed by a small number of demonstrations. There are three main approaches to constructing instruction-formatted instances: formatting task datasets, formatting daily dialogues, and formatting synthetic data as represented in Figure 21.

Historically, datasets encompassing tasks like text summarization, classification, and translation were used to create multi-task training datasets [219, 72, 103]. These datasets have

Finetune on many tasks (“instruction-tuning”)

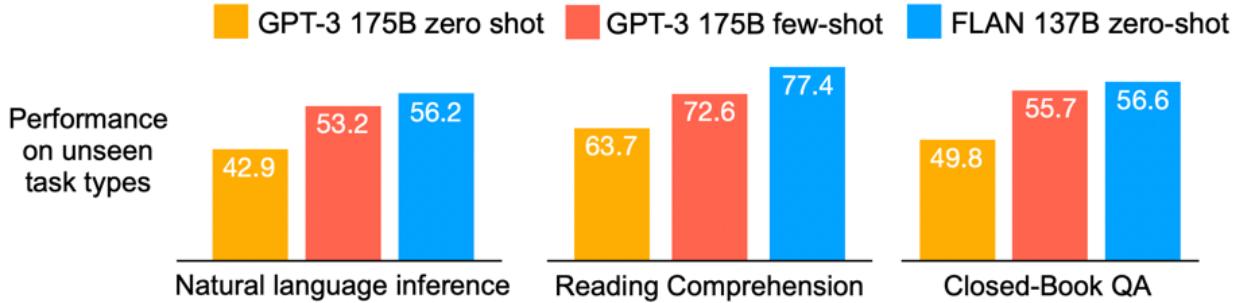
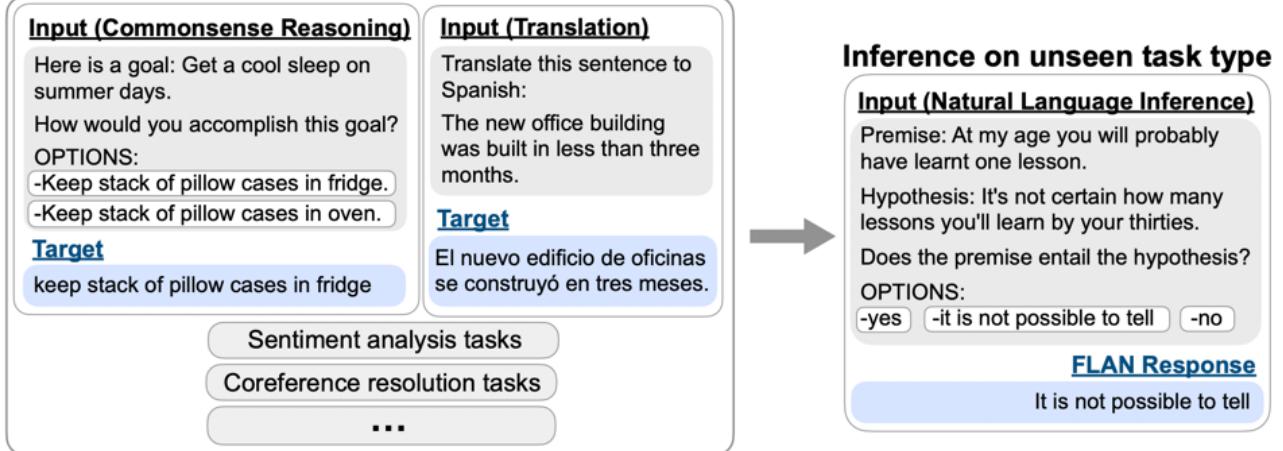


Figure 19: Top: overview of instruction tuning and FLAN. Instruction tuning finetunes a pre-trained language model on a mixture of tasks phrased as instructions. Evaluation of unseen task type at inference time (i.e., evaluate the model on natural language inference (NLI) when no NLI tasks were seen during instruction tuning).

Bottom: performance of zero-shot FLAN, compared with zero-shot and few-shot GPT-3, on three unseen task types where instruction tuning improved performance substantially out of ten evaluated. NLI datasets: ANLI R1–R3, CB, RTE. Reading comprehension datasets: BoolQ, MultiRC, OBQA. Closed-book QA datasets: ARC-easy, ARC-challenge, NQ, TriviaQA. Source: Wei et al. [231].

become crucial for instruction tuning, particularly when formatted with natural language descriptions that clarify the task objectives of the LLMs. This augmentation helps the models understand and execute the tasks more effectively [209, 205, 231, 229]. For instance, each example in a question-answering dataset might be supplemented with a directive like “Please answer this question” which guides the LLM in its response generation. The effectiveness of such instruction tuning is evident as LLMs demonstrate improved generalization to unfamiliar tasks when trained with these enriched datasets [231]. The decline in performance observed when task descriptions are omitted from training underscores the importance of these instructions.

PromptSource [147], a crowd-sourcing platform, has been proposed to aid in the creation, sharing, and verification of task descriptions for datasets. This platform enhances the utility of instruction tuning by ensuring a wide variety of well-defined task descriptions. Several studies [209, 219, 300] also tried to invert the input-output pairs of existing instances to create new instances using specially designed task descriptions (e.g., “Please generate a question given this answer”).

Talking about formatting daily chat data, Instruct-GPT has been fine-tuned using real user queries submitted to the OpenAI API to fill the significant gap in the data used for training

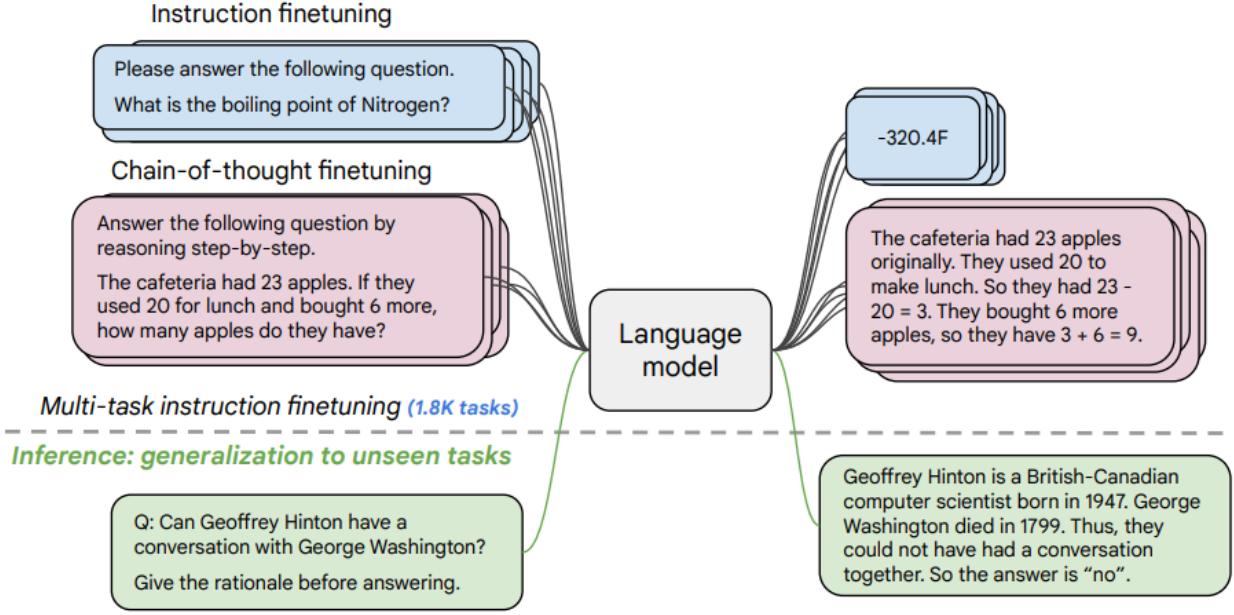


Figure 20: Overview of FLAN instruction tuning with and without exemplars (i.e., zero-shot and few-shots) and with and without CoT. Following evaluation on unseen tasks. Source: Chung et al. [156].

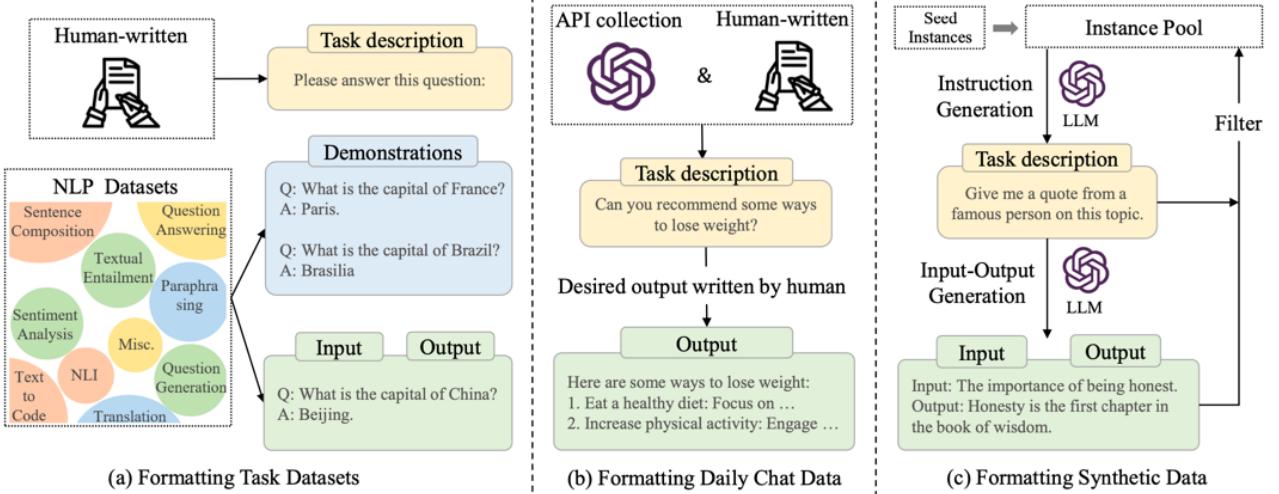


Figure 21: Three main approaches to construct instruction-formatted instances. Source: Zhao et al. [364].

models – most training instances come from public NLP datasets that often lack instructional diversity and do not align well with actual human needs. This approach helps to harness the model’s capability to follow instructions effectively. To further enhance task diversity and real-life applicability, human labellers are employed to create instructions for various tasks, including open-ended generation, open-question answering, brainstorming, and casual chatting. Another set of labellers then provides responses to these instructions, which are used as training outputs. This method enriches the training data and aligns the model’s responses more closely with human-like conversational patterns. InstructGPT also employs these real-world tasks formatted in natural language for alignment tuning (see Section 3.4.2). GPT-4 extends this

approach by designing potentially high-risk instructions and guiding the model to reject these instructions through supervised fine-tuning for safety concerns. Recent efforts have also focused on collecting user chat requests as input data, with models like ChatGPT or GPT-4 generating the responses. A notable dataset in this realm is the conversational data from ShareGPT, which provides a rich source of real-world interactions for training and refining the performance of LLMs.

Semi-automated methods [228] for generating synthetic data have also been explored to create instruction-formatted instances, which helps alleviate the need for extensive human annotation and manual data collection. One such method is the Self-Instruct approach, which efficiently utilizes a relatively small initial dataset. With the Self-Instruct method, only about 100 examples are required to start the data augmentation process (Figure 21c). From this initial task pool, a few instances are selected randomly and used as demonstrations for an LLM. The model is then prompted to generate new task descriptions and corresponding input-output pairs. This process expands the dataset and ensures a variety of training examples by incorporating a diversity and quality check before adding the newly synthesized instances back into the task pool. This synthetic approach to data generation is portrayed as both cost-effective and efficient, providing a scalable solution for enriching LLM training datasets. It leverages LLMs' generative capabilities to create diverse and relevant training materials, thereby enhancing the training process without the usual resource-intensive demands of manual data creation. Instruction tuning improves zero-shot learning and establishes new benchmarks in few-shot learning scenarios. The improvement is attributed to the instruction tuning across diverse datasets, which likely provides a richer context for model adaptation [231]. By using supervision to teach a model to perform tasks described via instructions, the model will learn to follow instructions and do so even for unseen tasks.

Two essential factors for the instance construction are:

- **Scaling the instructions.** Increasing the number of tasks within training data can significantly improve the generalization ability of LLMs, as evidenced by Wei et al. [231], Sanh et al. [77], and Chowdhery et al. [155]. The performance of LLMs typically increases with the number of tasks but plateaus after reaching a saturation point [99, 155]. It is suggested that beyond a certain threshold, additional tasks do not contribute to performance gains [99]. The diversity in task descriptions, including length, structure, and creativity variations, is beneficial [231]. However, increasing the number of instances per task might lead to overfitting if the numbers are excessively high [155, 258].
- **Formatting design.** The way instructions are formatted also plays a crucial role in the generalization performance of LLMs [155]. Task descriptions, supplemented by optional demonstrations, form the core through which LLMs grasp the tasks [155]. Utilizing a suitable number of exemplars as demonstrations can notably enhance performance and reduce the model's sensitivity to instruction nuances [77, 99]. However, including additional elements like prohibitions, reasons, or suggestions within instructions may not effectively impact or even negatively affect LLM performance [155, 199]. Recently, some studies suggest incorporating chain-of-thought (CoT) examples in datasets that require step-by-step reasoning, which has proven effective across various reasoning tasks [99, 174].

Instruction tuning is often more efficient since only a few instances are needed for training. Since instruction tuning can be considered as a supervised training process, its optimization is different from pre-training in several aspects [156], 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.

Balancing the proportion of different tasks during fine-tuning is crucial. A commonly used method is the examples-proportional mixing strategy [99], ensuring that no single dataset over-

whelms the training process [99, 231]. Additionally, setting a maximum cap on the number of examples from any dataset helps maintain this balance [99, 231].

To enhance the stability and effectiveness of instruction tuning, integrating pre-training data into the tuning process is beneficial, serving as regularization [231]. Some models, such as GLM-130B and Galactica, start with a mix of pre-training and instruction-tuned data, effectively combining the strengths of both pre-training and instruction tuning [155].

A strategic approach involves multiple tuning stages, starting with extensive task-specific data and followed by less frequent types, such as daily chat instructions, to avoid forgetting previously learned tasks [99].

Some additional strategies to improve the instruction tuning process include:

- **Efficient training for multi-turn chat data.** Given a multturn chat⁵¹ dataset, each conversation can be split into multiple context-response pairs for training: the model is fine-tuned to generate responses based on the corresponding context for all splits. To save computational resources, Chiang et al. [263] propose a method that fine-tunes the model on the whole conversation but relies on a loss mask that only computes the loss on the chatbot’s responses for training.
- **Filtering low-quality instructions using LLMs.** Filtering out low-quality instructions through advanced LLMs helps maintain high training standards and reduces unnecessary computational expenses [231].
- **Establishing self-identification for LLM.** In real-world applications, it is important for LLMs to be able to identify themselves when asked. To achieve this, models like GPT-4 are trained to recognize and respond to self-identification instructions [316].
- **Concatenate multiple examples to approach max length.** To handle variable-length sequences during training, it is common practice to introduce padding tokens to ensure uniform sequence lengths. However, this approach can lead to inefficient use of the model’s capacity, as the padding tokens do not contribute to the learning process. By concatenating multiple examples to approach the maximum sequence length, the model can process more information in each training step, enhancing the training efficiency and performance [116].
- **Evaluate the quality of instructions.** Cao et al. [256] introduced INSTRUCTMINING to autonomously select premium instruction-following data for finetuning LLMs by employing a combination of data mining techniques and performance evaluation strategies. The quality of instruction data is primarily assessed through its impact on model performance, quantified by the inference loss of a finetuned model on an evaluation dataset. INSTRUCTMINING correlates the values of the natural language indicators⁵² A predictive model that estimates data quality based on these indicators is created with the inference loss. To identify the most effective subset of data for finetuning, INSTRUCTMINING

⁵¹The conversation between a user and a bot

⁵²Some of the indicators include:

- Input and output length
- Reward model scores
- Perplexity
- Measures of Textual Lexical Diversity (MTLD)
- Approximate distance to nearest neighbours in embedding space
- Scores for naturalness, coherence, and understandability from models like UniEval

integrates an optimization technique called BLENDSEARCH. This method helps determine the optimal size and composition of the data subset, leading to the best finetuning outcomes. BLENDSEARCH combines global and local search strategies to efficiently explore the complex search space, focusing on minimizing the model’s inference loss on a high-quality evaluation set. Cao et al. [256] also accounts for the double descent phenomenon observed in model training, where increasing the dataset size initially improves performance up to a point, after which performance declines before potentially improving again as more data is added. This observation guides the selection process to focus on an optimal point that balances data quality and quantity, improving model performance efficiently.

- **Rewriting instructions into more complex ones.** Xu et al. [355] introduces a method termed “Evol-Instruct”, which significantly enhances the instruction-following capabilities and overall performance of large language models (LLMs). It is a systematic approach for automatically generating complex instruction data using LLMs instead of human input. This method involves iterative evolution and refinement of initial, simple instructions into more complex and diverse variants. These evolved instructions are then used to fine-tune LLMs, specifically targeting their ability to effectively understand and execute more complex tasks. Starting with a basic set of instructions, Evol-Instruct employs a two-pronged strategy – In-Depth Evolving and In-Breadth Evolving.

/textbfIn-Depth Evolving enhances the complexity and depth of instructions by adding constraints, increasing reasoning demands, or introducing more detailed contexts. /textbfIn-Breadth Evolving expands the variety and coverage of topics and skills addressed by the instructions, aiming to fill gaps in the LLM’s training data and increase its general robustness across different types of tasks.

Throughout the evolution process, ineffective or poorly structured instructions are filtered out to ensure only high-quality data is used for model training. This step is crucial for maintaining the integrity and effectiveness of the training dataset. The process repeats several cycles, allowing the system to gradually refine the instruction set to maximize complexity and utility while ensuring the instructions remain understandable and executable by the LLM. By training with the complex instructions generated by Evol-Instruct, LLMs like the WizardLM demonstrate significant improvements in several key areas:

- Enhanced Generalization: The model can handle a wider variety of tasks beyond the scope of its original training data.
- Improved Complexity Handling: The LLM performs better in understanding and executing tasks requiring higher levels of reasoning or multiple steps to complete.
- Competitive Performance: Compared to models like OpenAI’s ChatGPT and other contemporary LLMs, WizardLM trained with Evol-Instruct data exhibits competitive or superior performance, especially on complex instruction-following tasks.

The main effects of instruction tuning are:

- **Performance Improvement.** Instruction tuning significantly enhances LLMs, proving effective across models of various scales from 77M to 540B parameters. Smaller models subjected to instruction tuning can surpass larger models that haven’t been fine-tuned, showcasing the technique’s broad applicability and cost-effectiveness [135, 231]. This approach boosts model performance as parameter scale increases and demonstrates improvements across different architectures, objectives, and adaptation methods [99].

- **Task Generalization.** Instruction tuning endows LLMs with the capability to understand and execute tasks based on natural language instructions. This method is particularly effective in generalizing across both familiar and novel tasks, significantly enhancing performance without direct prior exposure [155, 135]. Notably, models like BLOOMZ-P3, fine-tuned on English-only tasks, demonstrate remarkable improvements in multilingual sentence completion, indicating robust cross-lingual transfer capabilities [155].
- **Domain Specialization.** Despite their prowess in general NLP tasks, LLMs often lack the domain-specific knowledge required for fields like medicine, law, and finance. Instruction tuning facilitates the transformation of general-purpose LLMs into domain-specific experts. For example, Flan-PaLM has been adapted into Med-PaLM for medical applications, achieving expert-level performance in medical tasks [99]. Similar adaptations have been made in other domains, significantly enhancing LLMs’ effectiveness in specialized applications [231].

In summary, instruction tuning is a powerful technique that significantly enhances LLMs’ performance, generalization, and domain specialization. Instruction tuning’s effectiveness is evident across models of various scales and architectures, demonstrating its versatility and broad applicability. Larger models, such as Llama_{13B} compared to Llama_{7B}, generally perform better, suggesting that increased model size enhances the model’s ability to follow instructions and utilize knowledge more effectively. This is particularly evident in QA settings, where larger models show markedly improved performance [364].

Increasing the complexity and diversity of the Self-Instruct-52K dataset enhances Llama’s performance in both chat and QA settings. For example, improving instruction complexity significantly boosts performance on QA tasks, which typically involve complex queries. Merely increasing the number of instructions or attempting to balance instruction difficulty does not necessarily yield better outcomes. In some cases, such as scaling up instruction numbers without focusing on quality, it can even degrade performance [364].

3.4.2 Alignment Tuning

LLMs may sometimes generate outputs inconsistent with human values or preferences (e.g., fabricating false information, pursuing inaccurate objectives, and producing harmful, misleading, or biased content) [205, 115]. To avoid such undesirable outcomes, alignment tuning ensures that LLMs’ outputs align with specified ethical guidelines or desired behaviors [364]. Unlike pre-training and fine-tuning, which focus on optimizing model performance, alignment tuning aims to optimize the model’s behaviour to conform to human values and norms [364]. Alignment may harm the general abilities of LLMs to some extent, which is called *alignment tax* [104].

Three primary criteria for regulating the behaviour of large language models (LLMs) are helpfulness, honesty, and harmlessness. These criteria have become standard in the literature and are benchmarks for aligning LLMs with desirable human-like behaviours. It’s possible to adapt these criteria based on specific needs, such as substituting honesty with correctness [165]. Helpfulness refers to the model’s ability to assist users effectively and efficiently, answering queries or solving tasks concisely. It should also engage in deeper interaction when necessary, asking relevant questions and demonstrating sensitivity and awareness. Honesty involves providing accurate information and transparency about the model’s uncertainty and limitations. This criterion is seen as more objective, potentially requiring less human intervention to achieve alignment than the other criteria. Harmlessness involves avoiding generating offensive or discriminatory language and being vigilant against being manipulated into harmful actions. Determining what constitutes harm can vary significantly depending on cultural and individual differences and the context in which the model is used.

Zhao et al. [364] notes the subjectivity of these criteria, rooted in human judgment, making them challenging to incorporate directly as optimization objectives in LLM training. Nonetheless, various strategies, such as red teaming⁵³, are employed to meet these criteria by intentionally challenging LLMs to provoke harmful outputs and then refining them to prevent such behaviours.

During the pre-training phase on a large-scale corpus, the subjective and qualitative evaluations of LLM outputs by humans cannot be taken into account. Human feedback is essential for alignment tuning, as it provides the necessary supervision to guide the model towards desirable behaviours.

Dominant strategies for generating human feedback data is human annotation [205, 165, 85]. This highlights the importance of labellers in the alignment tuning process, as they play a crucial role in providing feedback on the model’s outputs. Ensuring that labellers have adequate qualifications is vital; despite stringent selection criteria, mismatches in intentions between researchers and labellers can still occur, potentially compromising feedback quality and LLM performance [106]. To address this, the InstructGPT initiative includes a screening process to select labellers whose evaluations closely align with those of researchers [205]. In some studies, using “super raters” ensures the highest quality of feedback by selecting the most consistent labellers for critical tasks [165].

Three primary methods are used to collect human feedback and preference data:

- **Ranking-based approach.** Human labellers evaluate model outputs in a coarse-grained fashion, often choosing only the best output without considering finer details. This method could lead to biased or incomplete feedback due to the diversity of opinions among labellers and the neglect of unselected samples. To improve this, later studies introduced the Elo rating system to establish a preference ranking by comparing outputs, thereby providing a more nuanced training signal [165, 85].
- **Question-based approach.** This method involves labellers providing detailed feedback by answering specific questions designed to assess alignment criteria and additional constraints. For example, in the WebGPT project, labellers evaluate the usefulness of retrieved documents to answer given inputs, helping to filter and utilize relevant information [124].
- **Rule-based approach.** This approach involves the use of predefined rules to generate detailed feedback. For instance, Sparrow uses rules to test whether responses are helpful, correct, and harmless. Feedback is generated both by comparing responses and assessing rule violations. Additionally, GPT-4 uses zero-shot classifiers to automatically determine if outputs violate set rules [165, 316].

One approach to alignment tuning is to use a reward model to evaluate the quality of generated outputs. RLHF utilizes reinforcement learning (RL) techniques, such as Proximal Policy Optimization (PPO), to fine-tune LLMs based on human feedback, aiming to enhance model alignment on criteria like helpfulness, honesty, and harmlessness. This process involves several components and steps to effectively train and optimize LLMs. Key components of RLHF include a pre-trained language model (LM), a reward model (RM), and an RL algorithm (e.g., PPO) [364]. The LM is initialized with parameters from existing LLMs, such as OpenAI’s GPT -3 or DeepMind’s Gopher. The reward model provides guidance signals reflecting human preferences. It could be a fine-tuned LM or a newly trained LM using human preference data. RMs often differ in parameter scale from the LLM being aligned. The main steps in RLHF include supervised fine-tuning, reward model training, and RL fine-tuning [364].

⁵³Red teaming might involve trying to induce biased or harmful outputs from the model, to test its resistance to producing undesirable content under adversarial conditions.

Supervised fine-tuning involves collecting a supervised dataset with prompts and desired outputs for initial fine-tuning.

Reward model training trains the RM using human-annotated data where labellers rank outputs, guiding the RM to predict human preferences. Studies suggest using large reward models that align with the LLM’s scale for better performance judgment and combining multiple RMs focused on different alignment criteria for a nuanced reward signal.

RL fine-tuning treats alignment as an RL problem where the LM is optimized against the RM using PPO, incorporating penalties like KL divergence to maintain closeness to the original model behaviour. Practical strategies propose deploying the RM on a separate server and using beam search decoding to manage computational demands and enhance output diversity.

RLHF is a complex but promising approach to improving LLM alignment with human values. It involves sophisticated training regimes and multiple feedback mechanisms to ensure the model’s outputs are ethical and practical.

That being said, RLHF is memory-intensive (it needs to train multiple LMs), and the PPO algorithm is somewhat complex and often sensitive to hyperparameters. Thus, increasing studies are exploring alternative methods to align LLMs with human values using supervised fine-tuning without reinforcement learning.

The main idea behind alignment tuning without reinforcement learning is to use high-quality alignment datasets directly. LLMs aligned with human-written safety principles or refining existing examples through editing operations may create the alignment dataset. Additionally, reward models can be reused to select highly rated responses from existing human feedback data, enriching the dataset’s quality and relevance. Non-RL alignment methods employ supervised learning strategies similar to those used in original instruction tuning. These methods may also integrate auxiliary learning objectives, such as ranking responses or contrasting instruction-response pairs, to further enhance LLMs’ alignment accuracy and performance.

3.5 Architecture

The architecture of Large Language Models (LLMs) plays a pivotal role in determining their performance, efficiency, and scalability.

Generally speaking, we can identify some key components that define different LLM architectures: the encoder and the decoder. The encoder is an essential component in LLMs. It processes input sequences and maps them to a higher-dimensional space, capturing the contextual information in the data. The structure of an encoder in LLMs typically involves a stack of identical layers, each comprising two main sub-layers: a multi-head self-attention⁵⁴ mechanism and a position-wise fully connected feed-forward network [334].

On the other hand, the decoder is responsible for generating output sequences based on the encoded representations. The decoder in models such as GPT-3 [88] and its successors operate on the principle of autoregressive modelling, where each subsequent token is predicted based on the previously generated tokens. A key feature of decoders in LLMs is causality, which ensures that the prediction for the current token can only attend to previous tokens, not future ones. This is implemented through masked attention mechanisms in the transformer’s decoder layers [334].

For example, in a translation task, the encoder processes the source sentence and produces a set of vectors representing its content. At the same time, the decoder uses cross-attention to decide which words (or phrases) in the source sentence are most relevant for predicting the next word in the target language. In code generation, decoders can create syntactically correct code snippets given comments or docstrings as input, as demonstrated by Codex [108].

⁵⁴See Section 3.5.4 for more details on self-attention mechanisms.

Based on the components and the way they are connected, LLMs can be categorized into three main types: encoder-only⁵⁵, decoder-only and encoder-decoder models. All of these are sequence-to-sequence models (often referred to as seq2seq models).

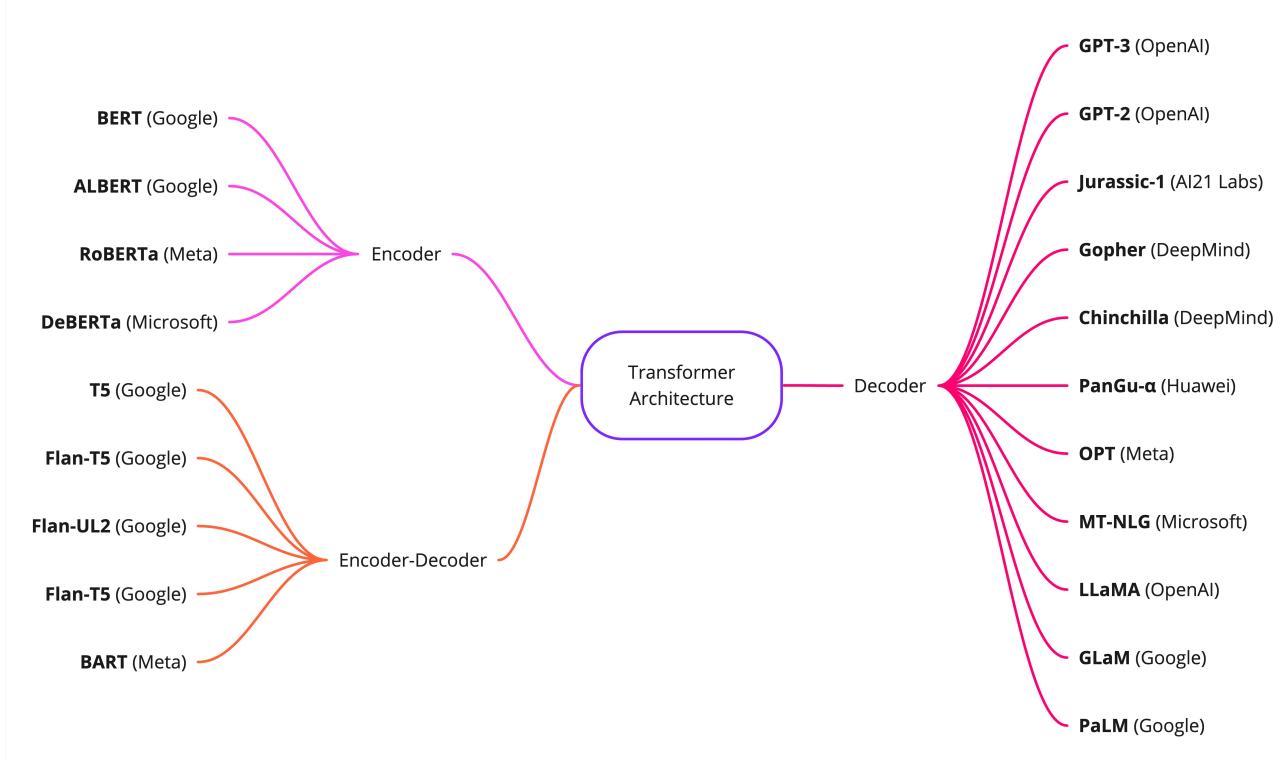


Figure 22: Some of the mainstream LLMs models by type.

Mainstream architectures can be further categorized into three major types: encoder-decoder, causal decoder and prefix decoder, as shown in Figure 23. Both causal decoder and prefix decoder are decoder-only architectures, but they differ in how they generate tokens.

3.5.1 Encoder-decoder

The vanilla version of the Transformer architecture introduced by Vaswani et al. [334] belongs to this category, which consists of an encoder and a decoder.

The encoder transforms an input sequence into a set of representations that capture its semantic and syntactic properties.

On the other hand, the decoder is tasked with generating an output sequence from the encoded representations. It predicts each token by conditioning on the previously generated tokens and the encoded input, a process that has significantly improved with the integration of cross-attention modules. The encoder-decoder architecture enables a flexible approach to diverse language tasks by segregating the understanding (encoding) and generation (decoding) processes.

⁵⁵We refer to BERT-style methods as encoder-only; the description encoder-only may be misleading since these methods also decode the embeddings into output tokens or text during pretraining. In other words, both encoder-only and decoder-only architectures are “decoding”. However, the encoder-only architectures, in contrast to decoder-only and encoder-decoder architectures, are not decoding in an autoregressive fashion. Autoregressive decoding refers to generating output sequences one token at a time, conditioning each token on the previously generated tokens. Encoder-only models do not generate coherent output sequences in this manner. Instead, they focus on understanding the input text and producing task-specific outputs, such as labels or token predictions [317].

So far, there are only a small number of models that use the encoder-decoder architecture (Figure 22), such as BART [94] and T5 [99].

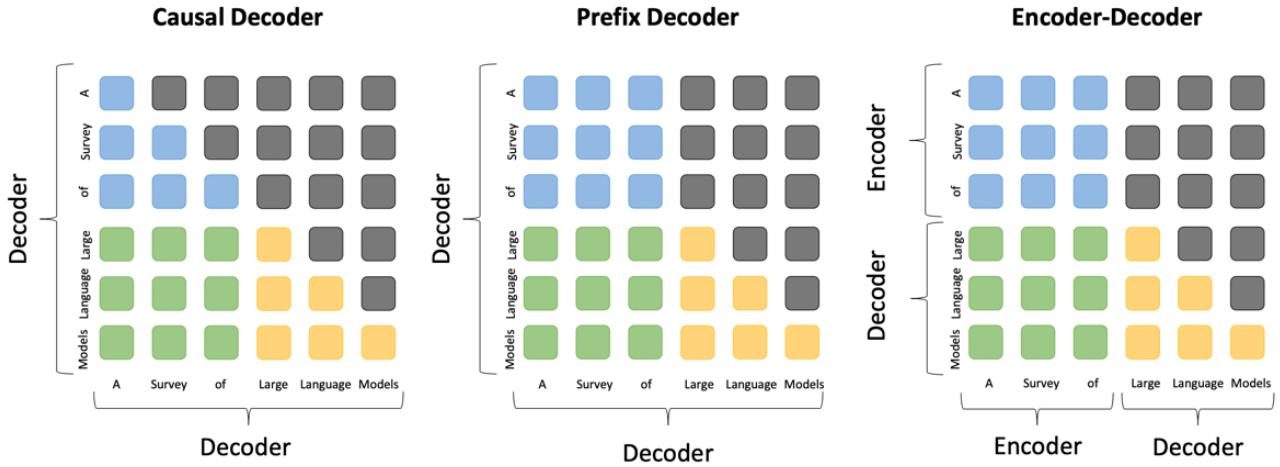


Figure 23: A comparison of the attention patterns in three mainstream architectures. Here, the blue, green, yellow and grey rounded rectangles indicate the attention between prefix tokens, attention between prefix and target tokens, attention between target tokens, and masked attention, respectively. Source: Zhao et al. [364].

3.5.2 Casual decoder

A causal decoder predicts each token based on the preceding tokens. This ensures that the generation process is unidirectional and prevents the model from using future tokens in the prediction process [334]. This mechanism is akin to how humans produce language, one word at a time, building upon what has already been said without access to future words.

The architecture typically employs self-attention mechanisms where the attention distribution is masked to prevent tokens from attending to subsequent positions in the sequence (i.e., unidirectional attention mask). This masking is instrumental in preserving the autoregressive property within the transformer-based models [75].

The GPT series⁵⁶ of language models by OpenAI are prominent examples that utilize causal decoder architectures, where the ability to generate coherent and contextually relevant text has been demonstrated effectively [88].

The causal decoder architecture is well-suited for tasks requiring sequential generation, such as text completion, language modelling, and text generation. It has been widely adopted as the architecture of choice for many large-scale language models, such as OPT [241], BLOOM [349], and Gopher [131].

3.5.3 Prefix decoder

The prefix decoder architecture⁵⁷ enables partial conditioning of generated sequences, revising the masking mechanism of causal decoders, to allow performing bidirectional attention over the prefix tokens [66] and unidirectional attention only on generated tokens.

In other words, this architecture allows the model to generate tokens based on both the input prefix and the target prefix, which can be helpful in tasks that require generating sequences with specific prefixes or constraints. In practice, a prefix decoder is implemented by feeding a fixed

⁵⁶GPT-3 [88] showed amazing in-context learning capability, whereas GPT-1 [51] and GPT-2 [75] didn't. It seems that scaling plays an important role in increasing the model capacity of this model architecture

⁵⁷Also called non-causal decoder [240]

sequence of tokens⁵⁸ into the decoder alongside the tokens generated so far. The model then extends the prefix by generating subsequent tokens that logically follow the context provided by the prefix.

Unlike the causal decoder, which strictly adheres to a unidirectional generation pattern, the prefix decoder allows for a predefined context or prefix to guide the generative process [119]. This is particularly useful in tasks such as machine translation, where the prefix can be a part of the already known or hypothesized translation. Still, the flexibility provided by the prefix decoder makes it suitable for a range of applications, from controlled text generation to task-oriented dialogue systems, where maintaining context and coherence is crucial [183].

This architecture has been utilized in various language models to improve text generation control and enhance the models' ability to handle specific formats or styles [99].

3.5.4 Transformer Architecture

The Transformer architecture has emerged as the de facto standard for LLMs, owing to its ability to capture long-range dependencies and model complex language structures effectively [334], making it possible to train models with billions or even trillions of parameters [88, 330].

This architecture usually consists of stacked Transformer layers (Figure 24), each comprising a multi-head self-attention sub-layer and a position-wise fully connected feed-forward network [334]. Residual connection [29] and layer normalization [28] are applied for both sub-layers individually.

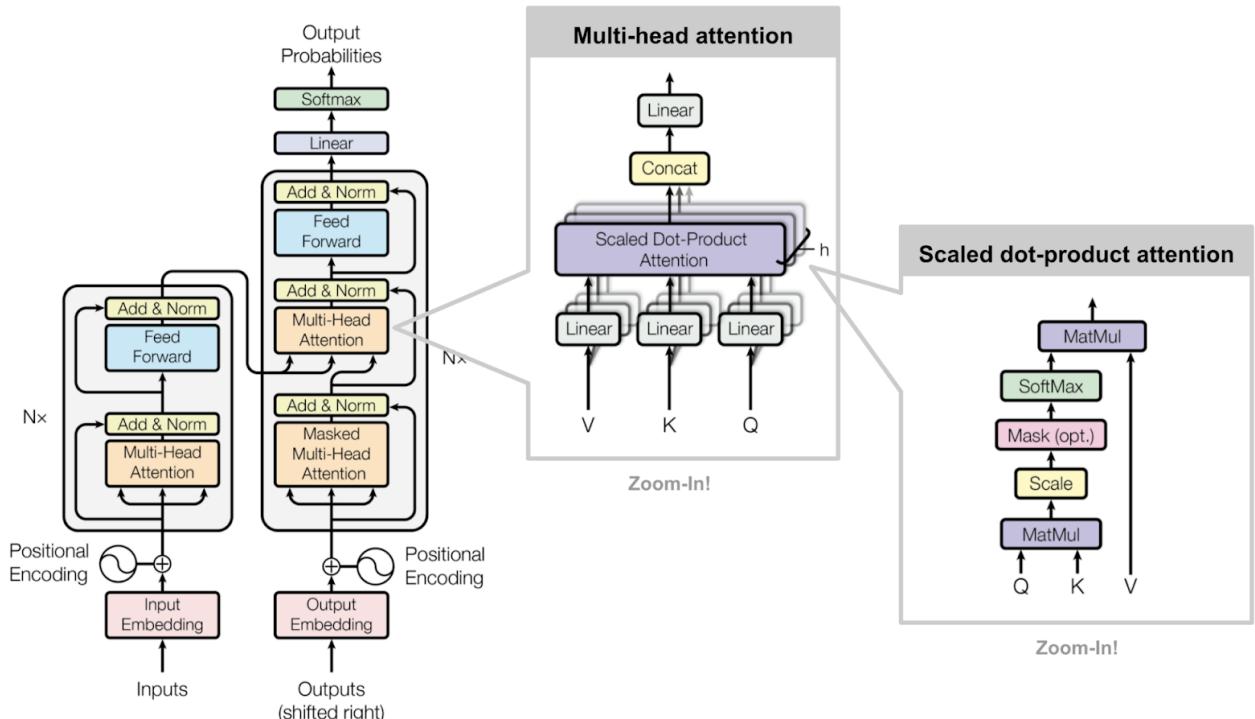


Figure 24: The full model architecture of the transformer. Source: Weng [57].

The position-wise FFN sub-layer is a two-layer feed-forward network with a ReLU activation function between the layers. Given a sequence of vectors h_1, h_2, \dots, h_n , the computation of a position-wise FFN sub-layer on any h_i , as shown in Equation 7.

$$\text{FFN}(h_i) = \text{ReLU}(h_i W^1 + b^1) W^2 + b^2 \quad (7)$$

⁵⁸Also known as the prefix

where W^1 , W^2 , b^1 , and b^2 are learnable parameters of the FFN sub-layer.

Besides the two sub-layers described above, the residual connection and layer normalization are also key components of the Transformer. Different orders and configurations of the sub-layers, residual connection and layer normalization in a Transformer layer lead to variants of Transformer architectures as shown in Table 22.

Model	Category	Size	Normalization	PE	Activation	Bias	#L	#H	d_{model}	MCL
GPT3 [88]	Causal decoder	175B	Pre LayerNorm	Learned	GeLU	Y	96	96	12288	2048
PanGU- α [139]	Causal decoder	207B	Pre LayerNorm	Learned	GeLU	Y	64	128	16384	1024
OPT [241]	Causal decoder	175B	Pre LayerNorm	Learned	ReLU	Y	96	96	12288	2048
PaLM [155]	Causal decoder	540B	Pre LayerNorm	RoPE	SwiGLU	N	118	48	18432	2048
BLOOM [349]	Causal decoder	176B	Pre LayerNorm	ALiBi	GeLU	Y	70	112	14336	2048
MT-NLG [214]	Causal decoder	530B	-	-	-	-	105	128	20480	2048
Gopher [131]	Causal decoder	280B	Pre RMSNorm	Relative	-	-	80	128	16384	2048
Chinchilla [172]	Causal decoder	70B	Pre RMSNorm	Relative	-	-	80	64	8192	-
Galactica [220]	Causal decoder	120B	Pre LayerNorm	Learned	GeLU	N	96	80	10240	2048
LaMDA [221]	Causal decoder	137B	-	Relative	GeGLU	-	64	128	8192	-
Jurassic-1 [121]	Causal decoder	178B	Pre LayerNorm	Learned	GeLU	Y	76	96	13824	2048
Llama [330]	Causal decoder	65B	Pre RMSNorm	RoPE	SwiGLU	Y	80	64	8192	2048
Llama 2 [329]	Causal decoder	70B	Pre RMSNorm	RoPE	SwiGLU	Y	80	64	8192	4096
Falcon [312]	Causal decoder	40B	Pre LayerNorm	RoPE	GeLU	N	60	64	8192	2048
GLM-130B [239]	Prefix decoder	130B	Post DeepNorm	RoPE	GeGLU	Y	64	96	12288	2048
T5 [99]	Encoder-decoder	11B	Pre RMSNorm	Relative	ReLU	N	24	128	1024	512

Table 22: Model cards of several selected LLMs with public configuration details. PE denotes position embedding, #L denotes the number of layers, #H denotes the number of attention heads, d_{model} denotes the size of hidden states, and MCL denotes the maximum context length during training. Source: Zhao et al. [364].

Configurations Since the introduction of the Transformer architecture, several variants and configurations have been proposed to improve the performance and efficiency of LLMs. The configuration of the four major parts of the Transformer architecture includes normalization, position embeddings, activation functions, and attention and bias, as shown in Table 23.

Normalization Methods Normalization methods are crucial for stabilizing the training process and improving the convergence of LLMs. In the vanilla Transformer [334] architecture, LayerNorm [28] is the most commonly used normalization method, which normalizes the hidden states across the feature dimension. Before LayerNorm was introduced, BatchNorm [26] was widely used in convolutional neural networks. Still, it was found to be less effective in sequence models due to the varying batch sizes and sequence lengths. LayerNorm addresses this issue by normalizing the hidden states across the feature dimension, making it more suitable for sequence models. Specifically, LayerNorm normalizes the hidden states using the mean and the variance of the summed inputs within each layer.

RMSNorm [83] is another normalization method that has been proposed to improve the training speed of LayerNorm. RMSNorm normalizes the hidden states by dividing them by the

root mean square of the squared hidden states, which has been shown to improve the training speed and performance [125]. Chinchilla [172] and Gopher [131] are examples of LLMs that use RMSNorm as the normalization method.

DeepNorm [225] is a novel normalization method that combines LayerNorm with a learnable scaling factor to stabilize the training process of deep Transformer models. With DeepNorm, Transformer models can be scaled up to hundreds of layers without additional normalization layers, making it an effective method for training large-scale LLMs [225]. It has been used in models such as GLM-130B [239].

Configuration	Method
Normalization position	Post Norm [334] Pre Norm [75] Sandwich Norm [109]
Normalization method	LayerNorm [28] RMSNorm [83] DeepNorm [225]
Activation function	ReLU [16] GeLU [56] Swish [41] SwiGLU [100] GeGLU [100]
Position embedding	Absolute [334] Relative [99] RoPE [134] Alibi [206]

Table 23: Detailed formulations for the network configurations. Source: Zhao et al. [364]

Normalization Position The position of the normalization layer (Figure 25) in the Transformer architecture can significantly impact the model’s performance and convergence. The three main configurations proposed in different studies are pre-LN⁵⁹, post-LN⁶⁰, and Sandwich-LN.

In the pre-LN configuration, the normalization layer is placed inside the residual blocks, while in the post-LN configuration, it is placed after them. In Ding et al. [109], the normalization layer is placed before and after the residual blocks, referred to as the Sandwich-LN configuration.

Post-LN is used in the vanilla Transformer architecture [334], where the normalization layer is placed between the residual blocks. This sequence allows the model to first process the input through a sublayer, such as a Multi-Head Attention (MHA) or Feed-Forward Network (FFN), and then apply normalization to the output of the sublayer combined with the residual connection. In particular, to train the model from scratch, any gradient-based optimization approach requires a learning rate warm-up stage to stabilize the training process [334]. Existing works found that training of Transformer models with post-norm tends to be unstable due to large gradients near the output layer [101].

Pre-LN [62] is another configuration where the normalization layer is placed inside the residual blocks. It makes it possible to remove the warm-up stage, requiring significantly less training time and hyper-parameter tuning on a wide range of applications. The Transformers with pre-LN have shown to be more stable during training but have worse performance [97].

⁵⁹Pre-Layer Normalization

⁶⁰Post-Layer Normalization

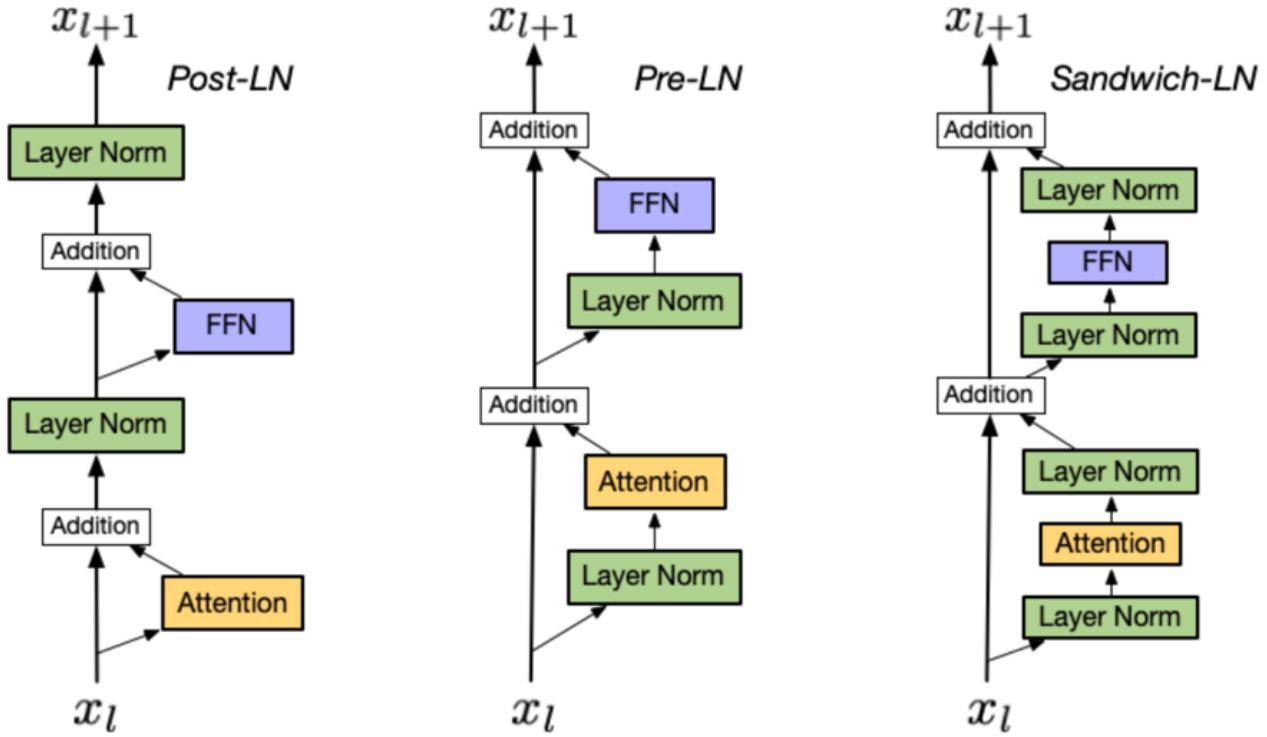


Figure 25: Illustration of different LayerNorm structures in Transformers. Source: Ding et al. [109].

Sandwich-LN [109] is a configuration that combines the advantages of both pre-LN and post-LN by placing the normalization layer both before and after the residual blocks. This configuration has been shown to improve the performance of Transformer models by providing better stability during training and faster convergence [109]. In Zeng et al. [239], the authors found that the Sandwich-LN configuration sometimes fails to stabilize the training of LLMs and may lead to the collapse of training.

Activation Functions Activation functions play a crucial role in the training and performance of LLMs by introducing non-linearity into the model⁶¹. LLMs' most commonly used activation functions are ReLU, GeLU, SwiGLU, and GeGLU.

ReLU⁶² [16] is a simple and widely used activation function that introduces non-linearity by setting negative values to zero.

$$\text{ReLU}(x) = \max(x, 0) \quad (8)$$

One of the first activation functions to be used in deep learning, ReLU has been shown to be effective in training deep neural networks by preventing the vanishing gradient problem [17]. This non-linear activation function introduces sparsity in the network's activations, which can lead to faster training and better performance due to its simplicity and efficiency. However, ReLU can suffer from the dying ReLU problem, where neurons can become inactive and stop learning if the input is negative [20].

GeLU [31] is a Gaussian Error Linear Unit activation function used to model uncertainties in neural networks. It was introduced to improve upon ReLU by considering the stochastic regularisation techniques. The smoothness of the GELU function can be advantageous in deep neural networks with many layers, as it can help prevent the problem of “dying ReLU”

⁶¹In the feed-forward layer

⁶²Rectified Linear Unit

and improve the flow of gradients through the network. The GELU activation function is mathematically described as follows:

$$\text{GeLU}(x) = x \cdot \Phi(x) \quad (9)$$

where $\Phi(x)$ is the cumulative distribution function of the standard Gaussian distribution. This can also be approximated as:

$$\text{GeLU}(x) \approx 0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)]) \quad (10)$$

Alternatively, the GELU function can be expressed as a scaled version of the sigmoid function, as shown below:

$$\text{GeLU}(x) \approx x \cdot \sigma(1.702x) \quad (11)$$

The GELU function allows the input to control its gate, deciding whether to pass through or be damped. When x is large, GELU approximates to x , acting like a linear unit. When x is close to zero or negative, it squashes the output, making it closer to zero. In other words, the GELU function would produce outputs smoothed around zero rather than sharply cut off as with ReLU. Many deep learning models use The GELU activation function, including GPT-3 and BERT.

The Swish [41] activation function is a smooth, non-monotonic function developed to overcome some limitations of ReLU and was found to perform better in deeper models. It is defined as

$$\text{Swish} = x \cdot \sigma(x) \quad (12)$$

where x is the input to the activation function and sigmoid is the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$. The Swish function allows small and negative values to pass through, which can benefit gradient flow in deep models. It has been empirically demonstrated to work well for deeper models and is computationally efficient.

SwiGLU [100] is a variant of the Swish activation function that combines the Swish function with the Gated Linear Unit (GLU) function. The SwiGLU activation function is defined as

$$\text{SwiGLU}(x, W, V, b, c, \beta) = \text{Swish}(xW + b) \otimes (xV + c) \quad (13)$$

Here, x is the input to the neuron, W and V are weight matrices, b and c are bias vectors, and β is a constant. The \otimes symbol denotes element-wise multiplication, while Swish is the activation function described in Equation 12. This function allows the network to learn which input parts should be retained (gated) for further layers, combining the advantages of non-saturating functions and dynamic gating mechanisms.

GeGLU [100] is another variant of the GLU activation function that combines the GeLU function with the Gated Linear Unit (GLU) function. The GeGLU activation is formulated as follows:

$$\text{GeGLU}(x, W, V, b, c) = \text{GeLU}(xW + b) \otimes (xV + c) \quad (14)$$

After the output of the GeLU function is calculated, it is multiplied element-wise with a second matrix. This second matrix is calculated by multiplying the input matrix x with another matrix W and adding a bias term b . The output of this multiplication is then passed through a second matrix V and added to a scalar term c .

Position Embeddings Position embeddings are a crucial component of the Transformer architecture. They allow the model to capture the sequential order of tokens in the input sequence. Several types of position embeddings are used in LLMs, including absolute, relative, RoPE, and Alibi embeddings.

Absolute position embeddings [334] were proposed in the original Transformer model. The absolute positional embeddings are added to the input embeddings at the bottoms of the encoder and the decoder. There are two variants of absolute position embeddings: sinusoidal and learned position embeddings, the latter of which is commonly used in existing pre-trained language models.

The formulation for adding absolute position embeddings is straightforward:

$$E_{\text{total}}(i) = E_{\text{token}}(i) + E_{\text{position}}(i) \quad (15)$$

where $E_{\text{total}}(i)$ is the final embedding vector for token i , $E_{\text{token}}(i)$ is the initial token embedding for token i , and $E_{\text{position}}(i)$ is the position embedding vector for token i . This technique allows the model to use the order of words to understand meaning and context, which is especially important for tasks involving sequence modelling and generation.

Relative position embeddings [52] are an alternative to absolute position embeddings that capture the relative distance between tokens in the input sequence. This allows the model to learn more flexible and adaptive representations of the input sequence, which can improve performance on tasks that require capturing long-range dependencies and complex relationships between tokens. Relative position embeddings are incorporated into the self-attention mechanism of Transformer models. Instead of considering only the absolute position of tokens, the attention scores are adjusted based on their relative distances. The formulation for the attention mechanism with relative position embeddings is given by:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{Q(K + R)^T}{\sqrt{d_k}} \right) V \quad (16)$$

where Q , K , and V are the query, key, and value matrices, respectively, R is the relative position embedding matrix, and d_k is the dimension of the key vectors. The relative positions are calculated as $R_{ij} = R_{\text{pos}[i]-\text{pos}[j]}$, where $\text{pos}[i]$ and $\text{pos}[j]$ are the positions of tokens i and j in the input sequence, respectively.

RoPE⁶³ [134] is a type of position embedding that uses rotational matrices to capture the relative positions of tokens in the input sequence. Unlike traditional position embeddings that add or concatenate position information, RoPE encodes position information through rotation in the embedding space, enabling models to preserve positional relationships effectively. The key idea of RoPE is to bind the position encoding with the word embedding in a way that preserves the rotational relationship between embeddings. It uses a rotation matrix to modulate the embedding based on its position, thereby aligning words by their relative positions instead of their absolute positions. The formula for the Rotary Position Embedding is:

$$E_{\text{rot}}(x_i, p_i) = \text{Rotate}(x_i, p_i) = x_i \cos(p_i) + (W x_i) \sin(p_i) \quad (17)$$

where x_i is the token embedding, p_i is the position embedding, and W is a learnable weight matrix. Rotary Position Embeddings were introduced by Su et al. [134] and have been shown to improve the performance of LLMs on a range of tasks.

ALiBi⁶⁴ [206] position embeddings offer an alternative mechanism for incorporating position information into Transformer models. Unlike traditional absolute or relative position

⁶³Rotary Position Embeddings

⁶⁴Attention with Linear Biases

embeddings, ALiBi introduces biases directly into the self-attention mechanism to handle positional dependencies. ALiBi introduces a linear bias based on the distance between tokens in the attention scores. Similar to relative position embedding, it biases attention scores with a penalty based on the distances between keys and queries. Different from the relative positional embedding methods like T5 [139], the penalty scores in ALiBi are pre-defined without any trainable parameters. This bias is subtracted from the attention logits before the softmax operation, helping the model to prioritize nearby tokens over distant ones, which is crucial in many sequential tasks. The modified attention score with ALiBi can be represented as:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} - \text{bias}(i, j) \right) V, \quad (18)$$

$$\text{bias}(i, j) = b \cdot |i - j|$$

where Q , K , and V are the query, key, and value matrices, respectively, and b is a learnable scalar parameter that controls the strength of the bias, and $|i - j|$ is the absolute distance between tokens i and j , and d_k is the dimension of the key vectors.

In Press, Smith, and Lewis [206], the authors found that ALiBi has better extrapolation performance than traditional position embeddings, and it can also improve the stability and convergence of Transformer models during training [349].

Attention Mechanisms Attention mechanisms are a key component of the Transformer architecture. They allow the model to capture long-range dependencies and complex relationships between tokens in the input sequence.

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. The two most commonly used attention functions are additive attention [23] and dot-product (multiplicative) attention. The

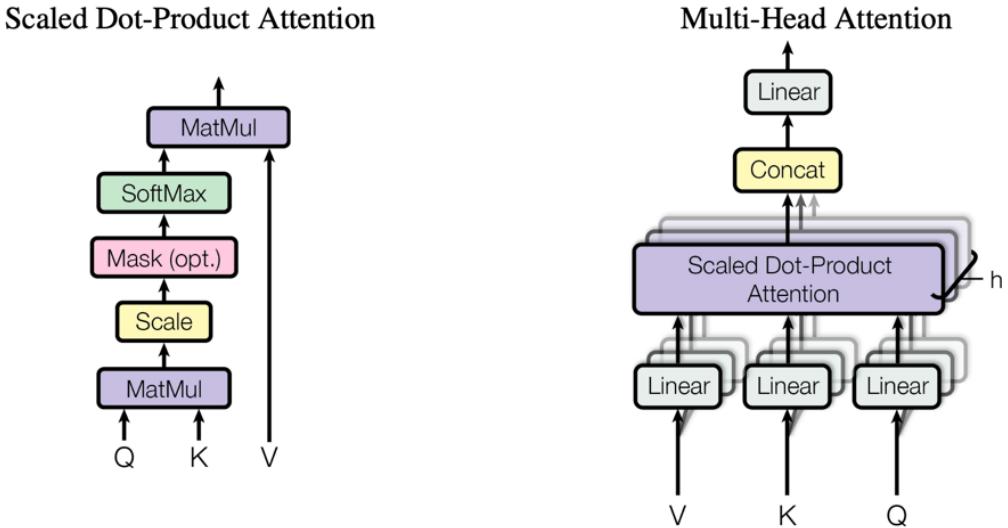


Figure 26: (left) Scaled Dot-Product Attention. (right) Multi-head attention consists of several attention layers running in parallel. Source: Vaswani et al. [334].

scaled dot-product attention function used in Vaswani et al. [334] is defined as follows:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (19)$$

where Q , K , and V are the query, key, and value matrices, respectively, and d_k is the dimension of the key vectors. While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [37].

A multi-head attention function is implemented by splitting the query, key, and value vectors into multiple heads and computing the attention function in parallel, yielding d_v -dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 26. The multi-head attention mechanism allows the model to jointly attend to information from different representation subspaces at different positions, enhancing the model’s capacity to capture complex relationships in the data.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O, \\ \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \quad (20)$$

where W_i^Q , W_i^K , and W_i^V are the weight matrices for the query, key, and value projections of the i -th head, respectively, and W^O is the final output projection matrix.

We can categorize the attention mechanisms into full attention, sparse attention, multi-query/grouped-query attention, Flash attention, and Paged attention. The Full attention mechanism is the standard attention mechanism used in the vanilla Transformer architecture [334], where each token attends to all other tokens in the sequence. It adopts the scaled dot-product we discussed in Equation 19. This mechanism is computationally expensive and has a quadratic complexity regarding the number of tokens, which can limit the model’s scalability to longer sequences. To address this issue, several studies have proposed alternative attention mechanisms.

In the Sparse attention mechanism, tokens only attend to a subset of other tokens according to a predefined pattern (e.g., local windows). This mechanism reduces the computational complexity of the attention operation and allows the model to scale to longer sequences.

$$\text{Sparse Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T \cdot M}{\sqrt{d_k}}\right)V \quad (21)$$

where M is a sparse attention mask that defines the pattern of attention between tokens.

Various sparse attention mechanisms have been proposed in the literature, such as Peng, Li, and Liang [128], Zaheer et al. [102] and Child et al. [64]. It is useful in tasks involving very long documents or sequences, such as document classification and genomic sequence analysis.

The multi-query/grouped-query attention mechanism [78] is an extension of the standard attention mechanism, where the keys and values are shared across all of the different attention “heads”, significantly reducing the size of these tensors and hence the memory bandwidth requirements of incremental decoding. This mechanism is handy in tasks requiring large amounts of data, such as machine translation and summarization. It can significantly reduce the computational cost of the attention operation with small sacrifices in model quality. Palm [155] and Starcoder [247] are examples of LLMs that use the multi-query attention mechanism. A trade-off between multi-query and multi-head attention grouped-query (GQA) has been explored in Ainslie et al. [245]. In GQA, heads are grouped together, and each group shares the same transformation matrices. This mechanism has been adopted and empirically tested in the Llama 2 model [329].

Flash attention [159] is an approach that proposes to optimize the speed and memory consumption of attention modules on the GPUs. Modern GPUs have different memory types, and

Flash attention takes advantage of this by organizing the input block on the faster memory⁶⁵. Updated version FlashAttention-2 [187] has been proposed to further improve the performance of the attention module on GPUs, partitioning of GPU thread blocks and warps, leading to around 2×speedup when compared to the original FlashAttention.

PagedAttention [335] is based on the observation that GPU memory is bottlenecked by cached attention keys and value tensors. These cached key and value tensors are often referred to as KV cache. The KV cache is large and highly dynamic depending on the sequence length. Authors find that existent systems waste 60%-80% of the memory due to fragmentation and over-reservation. PagedAttention proposed techniques inspired by virtual memory management⁶⁶ to manage the KV cache, partition sequences to sub-sequences allocating corresponding KV caches into non-contiguous physical blocks as shown in Figure 27.

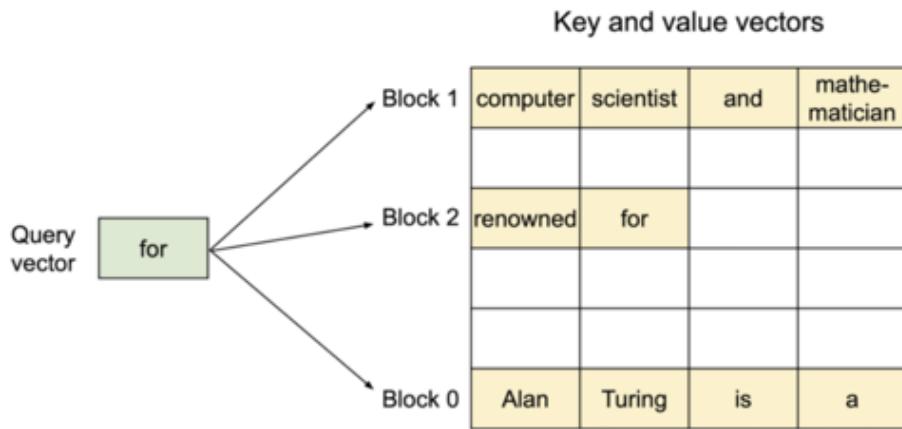


Figure 27: PagedAttention: KV Cache is partitioned into blocks. Source: [vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention](#) [335].

Paging increases the GPU memory utilization and enables efficient memory sharing in parallel sampling (Figure 28).

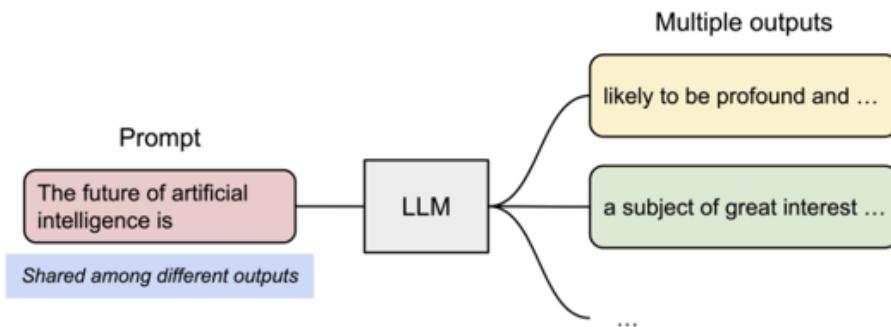


Figure 28: PagedAttention: example of parallel sampling. Source: [vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention](#) [335].

To put all these discussions together, Zhao et al. [364] summarize the suggestions from existing literature for detailed configuration. For stronger generalization and training stability, the pre-RMSNorm should be chosen for layer normalization and SwiGLU or GeGLU as the

⁶⁵SRAM has fast IO, while HBM is slower

⁶⁶Paging

activation function. In addition, LN may not be used immediately after embedding layers, which is likely to incur performance degradation. As for position embeddings, RoPE or ALiBi is a better choice since it performs better on long sequences.

3.5.5 Emerging architectures

Several emerging architectures have been proposed to address specific challenges or improve the performance of Transformers. One of the main issues with the vanilla Transformer architecture is the quadratic complexity regarding the number of tokens, which can limit the model’s scalability to longer sequences. To address this performance issue, several studies proposed alternative architectures, such as parameterized state space models (e.g., S4 [167], GSS [195], and H3 [160]), long convolutions(e.g., Hyena [315]), and recursive update mechanisms (RWKV [313] and RetNet [326]).

Parameterized state space models are a class of models that use a parameterized state space to represent the hidden states of the model. However, this method has prohibitive computation and memory requirements, rendering it infeasible as a general sequence modelling solution. To address this issue, S4 [167] proposed a novel parameterized state space model that uses a fixed-size state space to represent the hidden states of the model. This approach significantly reduces the model’s computational and memory requirements while maintaining high performance on a range of tasks. In Gu, Goel, and Ré [167], the authors found that S4 can be trained quickly and efficiently compared to Transformer variants designed for long-range sequence modelling as shown in Table 24.

	LENGTH 1024		LENGTH 4096	
	Speed	Mem.	Speed	Mem.
Transformer	1x	1x	1x	1x
S4	1.58x	0.43x	5.19x	0.091x

Table 24: Benchmarks vs. efficient Transformers

Long Range Arena(LRA) [136] is a benchmark suite that evaluates the performance of LLMs on a range of tasks that require capturing long-range dependencies. It contains six tasks with lengths of 1K-16K steps, encompassing modalities and objectives that require similarity, structural, and visuospatial reasoning. Table 25 shows the performance of S4 and 11 Transformer variants from Tay et al. [136]. Notably, S4 solves the Path-X task, an extremely challenging task that involves reasoning about LRDs over sequences of length $128 \times 128 = 16384$. All previous models have failed (i.e., random guessing) due to memory or computation bottlenecks or inability to learn such long dependencies. Other benchmarks in Gu, Goel, and Ré [167] show that

MODEL	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	AVG
Transformer	36.37	64.27	57.46	42.44	71.40	X	53.66
S4	58.35	76.02	87.09	87.26	86.05	88.10	80.48

Table 25: (Long Range Arena) Accuracy on the full suite of LRA tasks. (Top) Original Transformer variants in LRA. Source: Gu, Goel, and Ré [167].

S4 looks promising for long-range sequence modelling, achieving state-of-the-art performance on tasks requiring capturing long-range dependencies.

Long convolutions are a class of models that use convolutional layers to capture long-range dependencies in the input sequence. Poli et al. [315] proposed an operation-efficient architecture called Hyena defined by two recurring sub-quadratic operators: a long convolution and an element-wise multiplicative gating (Figure 29). Compared to the attention operators in Transformers, Hyena has a lower computational complexity and memory footprint, making it more efficient for long-range sequence modelling.

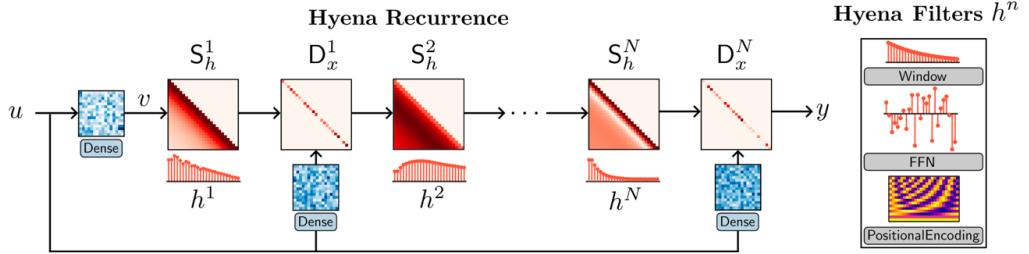


Figure 29: The Hyena operator is defined as a recurrence of two efficient subquadratic primitives: an implicit long convolution h (i.e., Hyena filters parameterized by a feed-forward network) and multiplicative element-wise gating of the (projected) input. The depth of the recurrence specifies the size of the operator. Source: Poli et al. [315].

3.6 Tuning and Optimization

Since LLMs consist of millions or billions of parameters, parameter tuning can be expensive and time-consuming. In this section, we discuss model adaptation of parameters and memory.

3.6.1 Parameter-efficient model adaptation

In the existing literature, several methods exist to adapt the model parameters to improve the performance of LLMs [114, 119, 118]. These methods aim to reduce the number of parameters in the model while maintaining performance as much as possible. In the following sections, we discuss some of the most popular methods for parameter-efficient model adaptation, such as adapter tuning, prefix tuning, prompt tuning, and LoRA (illustrated in Figure 30).

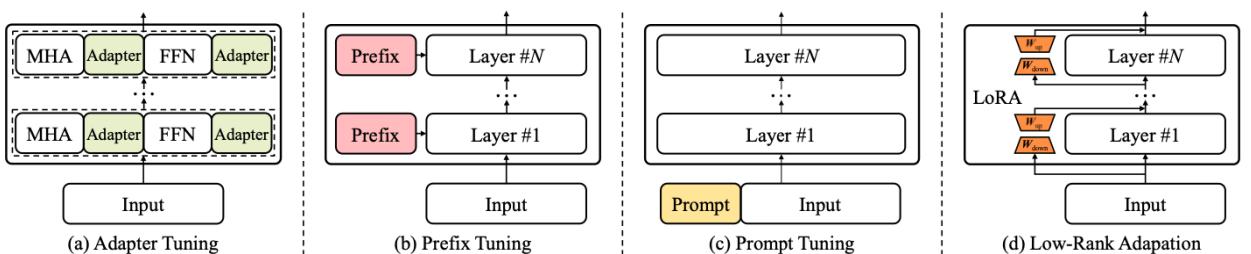


Figure 30: An illustration of four different parameter-efficient fine-tuning methods. MHA and FFN denote the multi-head attention and feed-forward networks in the Transformer layer, respectively. Source: Zhao et al. [364].

Adapter tuning Adapter tuning is a parameter-efficient technique for transferring a pre-trained model to multiple downstream tasks without re-training the entire model for each new task. This approach involves introducing small, trainable modules called “adapters” between

the layers of a pre-trained network. This allows the original network's parameters to remain fixed while adapting the model to new tasks with a minimal increase in the total number of parameters. Adapter tuning is designed to address the inefficiency of fine-tuning large models where each new task typically requires re-training the entire model. Instead, adapter tuning uses a base pre-trained model and introduces small adapter layers that are trained for each specific task into the Transformer architecture [68, 277], as shown in Figure 31.

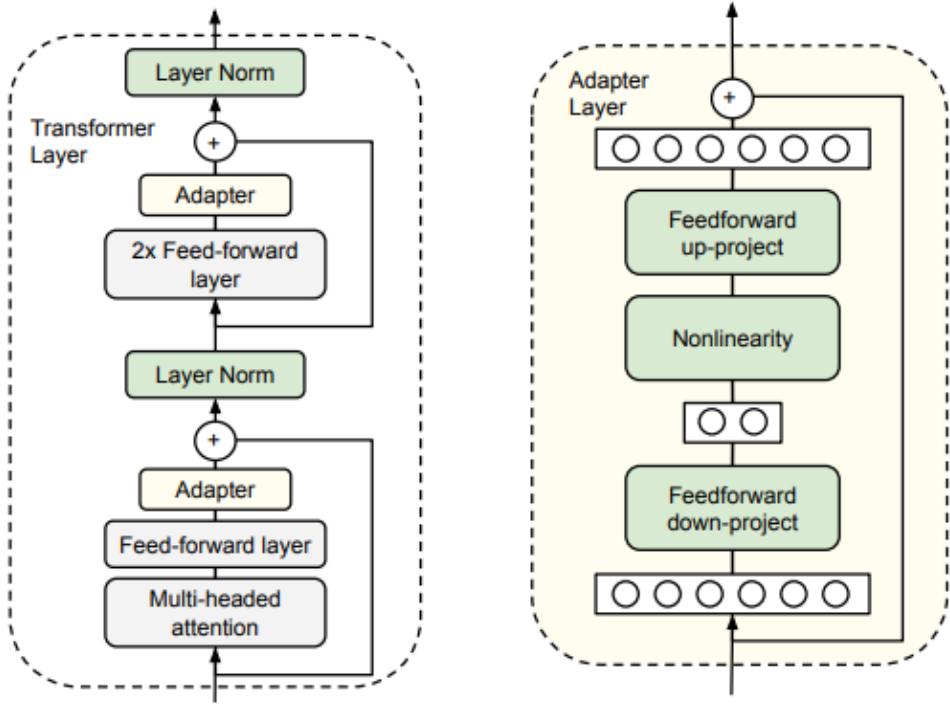


Figure 31: On the left, the architecture of the adapter module and its integration with the Transformer. The adapter module is added twice to each Transformer layer.

On the right, the adapter module consists of a feed-forward network with a bottleneck layer and a residual connection. During adapter tuning, the green layers are trained on the downstream data; this includes the adapter, the layer normalization parameters, and the final classification layer (not shown in the figure). Source: Houlsby et al. [68].

These adapter layers are typically much smaller than the main model layers, significantly reducing the number of new parameters that need to be trained. The main idea is that the adapter module first compresses the input representation to a lower-dimensional space (using a non-linear transformation) and then expands it back to the original dimension, allowing the model to adapt to new tasks without changing the pre-trained parameters. This architecture is also called bottleneck architecture⁶⁷, and it can be represented with dimensional reduction usually achieved using a linear transformation $D : \mathbb{R}^d \rightarrow \mathbb{R}^m$ where $m < d$. This layer is represented by a weight matrix $W \in \mathbb{R}^{m \times d}$ and a bias vector $b \in \mathbb{R}^m$.

$$y = \sigma(W_d x + b_d) \quad (22)$$

where σ is a non-linear activation function, x is the input vector, and y is the output vector of reduced dimensionality. After processing through the reduced dimension, the representation is

⁶⁷In neural network design, a bottleneck architecture refers to a specific configuration where the input space's dimensionality is reduced to a lower dimension before being projected back to the original dimension or higher. This architecture is commonly employed in deep learning models to reduce computational complexity, improve training efficiency, and sometimes help extract more generalized features.

usually projected back to the original dimension or higher using another linear transformation $U : \mathbb{R}^m \rightarrow \mathbb{R}^d$ represented by $W_u \in \mathbb{R}^{d \times m}$ and $b_u \in \mathbb{R}^d$.

$$z = \sigma(W_u y + b_u) \quad (23)$$

where z is the output vector, ideally representing the “reconstructed” version of the input after passing through the bottleneck.

Alternatively, parallel adapter [170] can also be used in Transformer layers, where the adapter is added in parallel with the attention layer and the feed-forward layer accordingly. During fine-tuning, the adapter modules are optimized according to the specific task goals, while the parameters of the original language model are frozen. In this way, we can effectively reduce the number of trainable parameters during fine-tuning.

Adapter tuning has been shown to achieve near state-of-the-art performance on various tasks with significantly fewer parameters than full fine-tuning. For example, on the GLUE benchmark, adapter tuning approaches the performance of full fine-tuning with only about 3.6% of the parameters trained per task.

Prefix tuning Prefix-tuning is introduced as an efficient alternative to traditional fine-tuning methods for deploying large pre-trained language models (PLMs) across various tasks. Traditional fine-tuning requires updating and storing a separate copy of the model for each task, which becomes computationally expensive as the models’ size increases (e.g., GPT-3’s 175 billion parameters). Prefix-tuning addresses this by optimizing only a small set of parameters, referred to as a prefix, significantly reducing the storage and computational overhead. The method involves prefixing a sequence of continuous, task-specific vectors to the input, allowing subsequent tokens in the Transformer model to attend to these prefixes as if they were part of the input sequence, as shown in Figure 32.

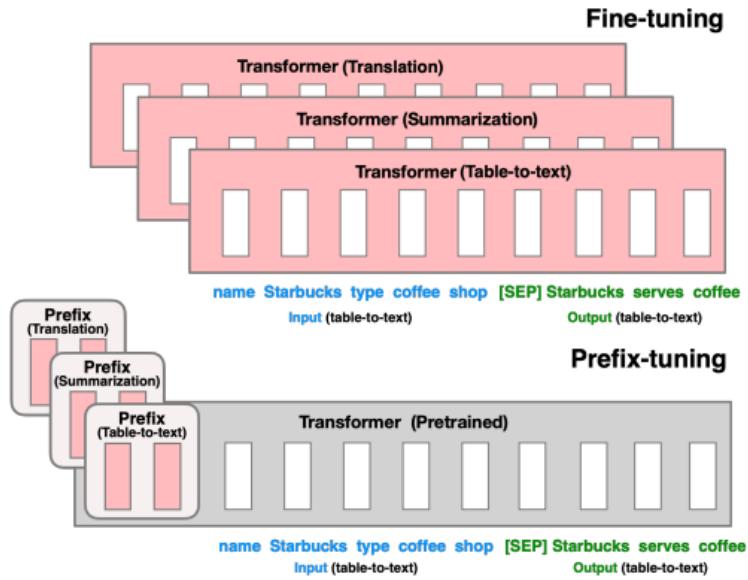


Figure 32: Illustration of the prefix-tuning method, which freezes the Transformer parameters and only optimizes the prefix (the red prefix blocks). Consequently, it only needs to store the prefix for each task, making prefix-tuning modular and space-efficient. Note that each vertical block denotes transformer activations at a one-time step. Source: Li and Liang [119].

An approach to optimize prefix vectors involves using a re-parameterization technique, as described in the work by Li and Liang [119]. This method employs a multilayer perceptron

(MLP) function to map a smaller matrix to the parameter matrix of the prefixes rather than directly optimizing the prefixes themselves. This technique has proven effective for stabilizing the training process. Once optimization is complete, the mapping function is discarded, leaving only the refined prefix vectors tailored to enhance performance on specific tasks. This approach leverages the inherent capabilities of the Transformer while only modifying a minimal set of parameters, making it modular and space-efficient. Li and Liang [119] provides detailed empirical evaluations demonstrating that prefix-tuning achieves comparable performance to full fine-tuning while only learning about 0.1% of the parameters. Evaluations are performed on tasks like table-to-text generation and summarization using models such as GPT-2 and BART. Results indicate that prefix-tuning reduces parameter count significantly and maintains competitive performance with traditional fine-tuning in full-data settings and often outperforms it in low-data scenarios. The approach effectively handles tasks with unseen topics during training, showcasing better generalization capabilities [94].

Prompt tuning Prompt tuning primarily involves incorporating trainable vectors, called prompt tokens, at the input layer of a model. Based on discrete prompting techniques, these tokens augment the input text to assist models in performing specific tasks. In prompt tuning, these task-specific embeddings are combined with the original text embeddings and processed by language models. Specifically, the method known as P-tuning employs a flexible approach to integrate context, prompt, and target tokens. This method is adaptable for tasks involving

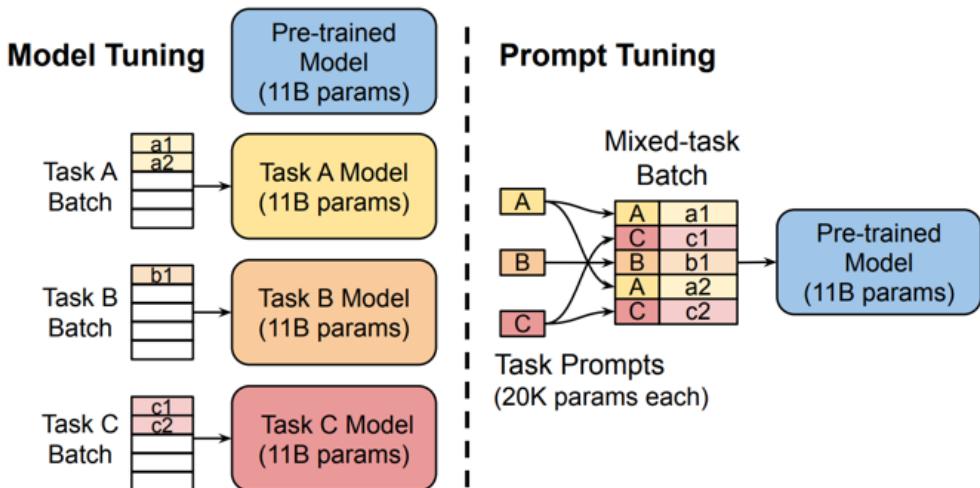


Figure 33: Illustration of the prompt tuning method, which only requires storing a small task-specific prompt for each task and enables mixed-task inference using the original pre-trained model. With model tuning, each copy of tuned models requires a copy of billions of parameters. In contrast, a tuned prompt would only require thousands of parameters per task—a reduction of over five orders of magnitude. Source: Lester, Al-Rfou, and Constant [118].

understanding and generating natural language and utilizes a bidirectional LSTM to learn representations of soft prompt tokens. Only these prompt embeddings are updated based on task-specific requirements during the training phase. The effectiveness of prompt tuning methods depends significantly on the computational power of the underlying language models, as they generally involve a limited number of trainable parameters at the input layer.

Liu et al. [188] introduces P-Tuning v2, a method that extends prompt tuning by applying continuous prompts across all layers of a language model, improving upon the conventional

method where prompts are only used at the input layer. They address the limitations of traditional prompt tuning, which underperforms significantly on complex sequence labelling tasks when model size is below 10 billion parameters [118]. P-Tuning v2 modifies the conventional prompt tuning by:

- Utilizing continuous prompts at every layer of the model to increase tunable parameter count without significantly increasing overall parameter load.
- Improving adaptability across both simple and complex tasks by modifying the interaction of prompts with model architecture [119, 129].

P-Tuning v2 has been evaluated across various model scales (from 330M to 10B parameters) and tasks, including classification and sequence labelling. The experiments demonstrate that P-Tuning v2 provides comparable results to full model fine-tuning, requiring only 0.1%-3% of the parameters to be tuned. Liu et al. [188] concludes that P-Tuning v2 significantly narrows the performance gap between prompt tuning and full fine-tuning, offering a robust, scalable, and efficient alternative for adapting large pre-trained models to diverse NLU tasks.

LoRA The technique called LoRA (Low-Rank Adaptation) is used for efficient fine-tuning neural networks, particularly in adapting dense layers to downstream tasks with fewer trainable parameters. LoRA strategically freezes the original parameter matrix $W \in \mathbb{R}^{m \times n}$ and applies updates using a low-rank decomposition approach, which involves two smaller matrices $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{n \times k}$ where k is much smaller than m or n . This method significantly reduces the memory and storage requirements by limiting the trainable parameters to those in A and B rather than the entire matrix W .

The main advantage of LoRA is its ability to maintain a single large model while adapting it to various tasks using different sets of low-rank matrices for each task, enhancing storage efficiency and reducing computational costs. Advanced methods for determining the optimal rank have been proposed, such as importance score-based allocation [363] – i.e., AdaLoRA – and search-free optimal rank selection [332] – DyLoRA. These methods help determine the optimal rank for the low-rank decomposition, ensuring the model is adapted efficiently to the specific task requirements.

In AdaLoRA⁶⁸, the idea is that adding more trainable parameters to the critical weight matrices can lead to better model performance. In contrast, adding more parameters to those less important weight matrices yields very marginal gains or even hurt model performance. Given the parameter budget, i.e., the number of total trainable parameters, AdaLoRA always prefers allocating more parameters to those essential modules. Distributing the budget evenly to all weight matrices/layers, like LoRA and other methods (e.g., adapter and prefix tuning), often gives suboptimal performance [363]. AdaLoRA operates by parameterizing the incremental updates in the form of singular value decomposition (SVD), allowing for selective pruning of updates based on their assessed importance. This selective pruning targets the singular values of unimportant updates, effectively reducing their parameter budget while avoiding the computational intensity of performing exact SVD calculations. SVD-based adaptation is represented as:

$$W = W^0 + \delta = W^0 + P\Lambda Q \quad (24)$$

where W^0 is the original parameter matrix, δ is the update, P and Q are the left and right singular vectors, and Λ is the singular value matrix. Zhang et al. [363] substantiates the effectiveness of AdaLoRA through extensive experiments across various NLP tasks, including

⁶⁸Adaptive Low-Rank Adaptation

question-answering and natural language generation. These experiments demonstrate notable performance improvements, particularly in low-budget settings, compared to baseline methods such as full fine-tuning and other parameter-efficient techniques like LoRA and adapter tuning. Key benchmarks from the paper highlight AdaLoRA’s superior performance on standard datasets like GLUE and SQuAD. It consistently outperforms other approaches while utilizing fewer parameters.

DyLoRA⁶⁹ is a search-free method for determining the optimal rank for low-rank decomposition in neural networks. The method is based on the observation that the optimal rank for low-rank decomposition varies across different layers and tasks. The main advantages of DyLoRA over conventional LoRA include its ability to dynamically adapt to different rank sizes during inference, eliminating the need for exhaustive search and re-training across different rank sizes. This is achieved by training the low-rank modules (LoRA blocks) across a spectrum of ranks during the training phase, which allows the model to adjust to the best-performing rank size at runtime without additional computational cost. This method is inspired by the nested dropout technique but tailored to the needs of dynamic rank adaptation. The implementation involves sampling a rank size during each training step and adjusting the adapter modules accordingly, which allows the model to learn to perform efficiently under various rank size constraints. The main improvements of DyLoRA over LoRA include:

1. Dynamic LoRA Blocks: DyLoRA modifies the standard LoRA blocks to be dynamic, allowing them to adjust their rank size during inference. This adaptation leads to more flexible models that can perform well across a broader range of tasks without specific tuning for each task.
2. Search-Free Adaptation: By avoiding the exhaustive search for the optimal rank size, DyLoRA reduces the training and adaptation time significantly. The model can be trained once and used dynamically across different settings, making it highly efficient.
3. Performance: Experimental results show that DyLoRA matches or exceeds the performance of traditional LoRA with a static rank across various NLP tasks. This is demonstrated in tasks such as sentiment analysis, question answering, and natural language generation, indicating the robustness and versatility of DyLoRA.

3.6.2 Memory-efficient model adaptation

In addition to parameter-efficient model adaptation, memory-efficient techniques have been proposed to reduce the memory footprint of LLMs. These methods aim to reduce the memory requirements of LLMs during inference, making them more suitable for deployment in resource-constrained environments. This section discusses some of the most popular methods for memory-efficient model adaptation, i.e. model quantization.

Quantization Quantization techniques reduce memory and computational costs by representing weights and activations with lower-precision data types, such as 8-bit integers (int8). This enables loading larger models that would typically be too large to fit into memory and speeds up inference. This process can substantially reduce the storage requirements and the computational complexity of deploying LLMs, which is crucial for their application in resource-constrained environments.

Quantization can be done in two ways: post-training quantization and quantization-aware training. Post-training quantization is done after the model has been trained, while quantization-aware training is done during training. Post-training quantization is easier to implement, but quantization-aware training can lead to better results.

⁶⁹Dynamic Search-Free Low-Rank Adaptation

Main quantization techniques include uniform quantization, non-uniform quantization, and mixed-precision quantization. Uniform quantization maps the floating-point values to a fixed set of integer values, while non-uniform quantization uses non-linear mapping to better represent the data distribution. Mixed-precision quantization uses a combination of different precision data types to represent the weights and activations.

Uniform quantization discretizes the values within a certain range into equal-sized intervals. Mathematically, it can be described as:

$$\text{LinearQuant}(x, \text{bitwidth}) = \text{Clip}(\text{round}(\frac{x}{\text{bitwidth}}) \times \text{bitwidth}, \text{minV}, \text{maxV}) \quad (25)$$

where minV and maxV are the minimum and maximum scale range respectively [39].

Non-uniform quantization, such as logarithmic quantization, allocates more fine-grained intervals to values that are more frequent or sensitive to quantization errors. This method can be represented as:

$$\text{LogQuant}(x, \text{bitwidth})(x) = \text{Clip}(\text{AP2}(x), \text{minV}, \text{maxV}) \quad (26)$$

where AP2 is the approximate-power-of-2 function that maps the input to the nearest power of two as defined in Hubara et al. [39]. This approach is particularly effective for distributions with a high dynamic range [32].

Mixed-precision quantization leverages the strengths of both uniform and non-uniform quantization by using different precision data types for different parts of the model. For example, weights can be quantized to 8-bit integers while activations are quantized to 16-bit integers.

Bit-width	Storage Reduction	Accuracy Loss
32 (Full Precision)	0%	0%
16	50%	1%
8	75%	2%
4	87.5%	5%

Table 26: Performance comparison of quantized LLM

As per Table 26, lower bit-widths generally result in more significant storage savings, but they can also lead to higher accuracy losses [40].

4 Utilization Strategies and Techniques

In this section, we will discuss the strategies and techniques for effectively utilizing large language models. We will start by discussing the importance of context in utilizing large language models and how it can be used to improve their performance. We will then move on to the concept of chain-of-thought prompting and how it can be used to guide text generation. Finally, we will discuss the LLMs' ability to plan for complex tasks.

4.1 In-Context Learning

4.1.1 ICL strategy

In-context learning is a special prompting technique, initially introduced by Brown et al. [88], that allows the model to learn from the context of the prompt (examples shown in Figure 34). ICL consists of the task description and/or a few examples of the task as demonstrations

Approach	Representative Work	Key Point
In-context Learning (ICL)	KATE [186], EPR [208], SG-ICL [176], APE [368], Structured Prompting [168], GlobalE & LocalE [190]	Demonstration selection (similar, k-NN) Demonstration selection (dense retrieval; contrastive learning) Demonstration selection (LLM as the demonstration generator) Demonstration format (automatic generation & selection) Demonstration format (grouped context encoding; rescaled attention) Demonstration order (entropy-based metric; probing set generation with LLM)
Chain-of-thought Prompting (CoT)	Complex CoT [163], Auto-CoT [243], Selection-Inference [157], Self-consistency [227], DIVERSE [295], Rationale-augmented ensembles [226]	Demonstration (complexity-based selection) Demonstration (automatic generation) Generation (alternate between selection and inference) Generation (diverse paths; self-ensemble) Generation (diverse paths; Verification (step-wise voting)) Generation (rationale sampling)
Planning	Least-to-most prompting [244], DECOMP [175], PS [338], Faithful CoT [302], PAL [164], HuggingGPT [321], AdaPlanner [324], TIP [301], RAP [275], ChatCoT [260], ReAct [236], Reflexion [322], Tree of Thoughts [359], LLM-modulo framework [379]	Plan generation (text-based; problem decomposition) Plan generation (text-based; problem decomposition) Plan generation (text-based) Plan generation (code-based) Plan generation (code-based; Python) Plan generation (code-based; models from HuggingFace) Plan refinement (skill memory) Feedback acquisition (visual perception) Feedback acquisition (LLM as the world model; Plan refinement (Monte Carlo Tree Search)) Feedback acquisition (tool); Plan refinement (conversation between LLM and tools) Feedback acquisition (tool); Plan refinement (synthesizing reasoning and acting) Feedback acquisition (text-based self-reflection); Plan refinement (dynamic memory) Feedback acquisition (vote comparison); Plan refinement (tree-based search)

Table 27: Typical LLM utilization methods and their key points for ICL, CoT, and planning. Note that the key points only highlight the most important technical contribution. Source: Zhao et al. [364]

combined in a specific order to form natural language prompts with specifically designed templates [88]. Finally, the test instance is appended to the prompt to form the input for LLMs to generate the output. LLMs can improve the performance to execute a new task without explicit gradient update based on task demonstrations. Formally, the in-context learning task can be defined as follows:

$$LLM(I, \underbrace{f(x_1, y_1), \dots, f(x_k, y_k)}_{\text{demonstrations}}, \underbrace{f(x_{k+1}, \underline{\quad})}_{\text{input}}) \rightarrow \hat{y}_{k+1} \quad (27)$$

where I is a task description, $f(x_i, y_i)$ function that converts task demonstration to natural language, x_{k+1} is a new input query, \hat{y}_{k+1} is the prediction of the output generated. The actual answer y_{k+1} is left as a blank to be predicted by the LLM.

Since ICL’s performance heavily relies on demonstrations, it is important to design them properly in the prompts. The three main aspects are a direct consequence of what is defined in Equation 27: how to select the task demonstrations, convert them into natural language, and arrange demonstrations in a reasonable order.

Different training strategies enhance ICL capabilities, improving performance across various tasks without specific task optimization during the pre-training phase (see Figure 36 under the Training branch). Main approaches include Supervised In-context Training, such as MetaICL⁷⁰ and Symbol Tuning, and Self-supervised In-context Training, such as Self-supervised ICL and PICL [265].

MetaICL [196] proposed to continually train LLMs on a wide range of tasks⁷¹ with demonstration examples. This approach is related to other works that use multi-task learning for better zero-shot performance at test time [196]. However, MetaICL is distinct as it allows

⁷⁰Meta-training for InContext Learning

⁷¹Classification, question answering, natural language inference, paraphrase detection and more

In-Context learning

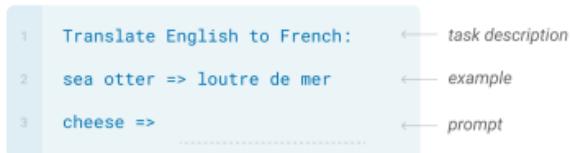
Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



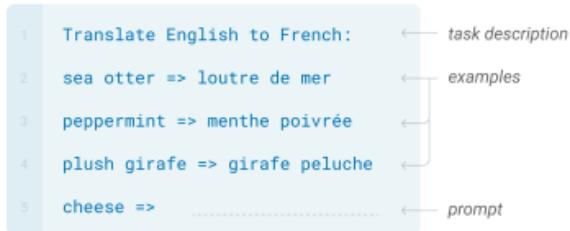
One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.



Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Traditional fine-tuning

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.



Figure 34: In-context learning contrasted with traditional fine-tuning. Source: Brown et al. [88]

learning new tasks from k examples alone, without relying on task reformatting (e.g., reducing everything to question answering) or task-specific templates (e.g., converting different tasks to a language modelling problem). MetaICL is based on the core idea of in-context learning by conditioning on training examples (i.e., explicitly training on an in-context learning objective).

Symbol Tuning [346] instead fine-tunes language models on in-context input-label pairs, substituting natural language labels (e.g., “positive/negative sentiment”) with arbitrary symbols (e.g., “foo/bar”). As a result, symbol tuning demonstrates an enhanced capacity to utilize in-context information for overriding prior semantic knowledge. Compared to MetaICL, which constructs several demonstration examples for each task, instruction tuning mainly considers an explanation of the task and is easier to scale up.

Self-supervised ICL leverages raw corpora to generate input/output pairs as training data. PICL also utilizes raw corpora but employs a simple language modelling objective, promoting task inference and execution based on context. PICL has shown to be more effective in zero-shot settings and task generalization [265].

Effective demonstration design is crucial, involving selecting and ordering examples or using instruction induction and reasoning steps (as shown in Figure 36 under the Inference/Demonstration

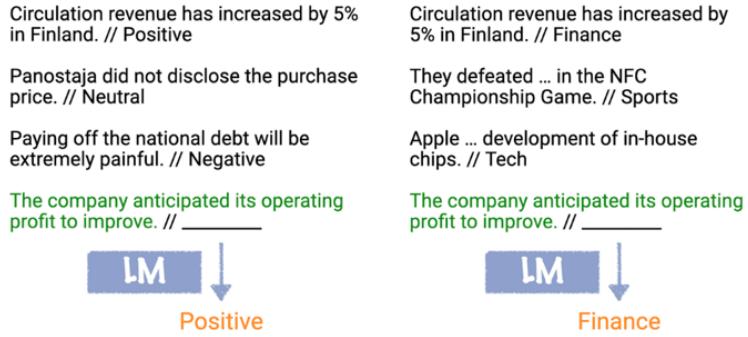


Figure 35: Two examples of in-context learning, where a language model (LM) is given a list of training examples (black) and a test input (green) and asked to make a prediction (orange) by predicting the next tokens/words to fill in the blank. Source: Lab [288]

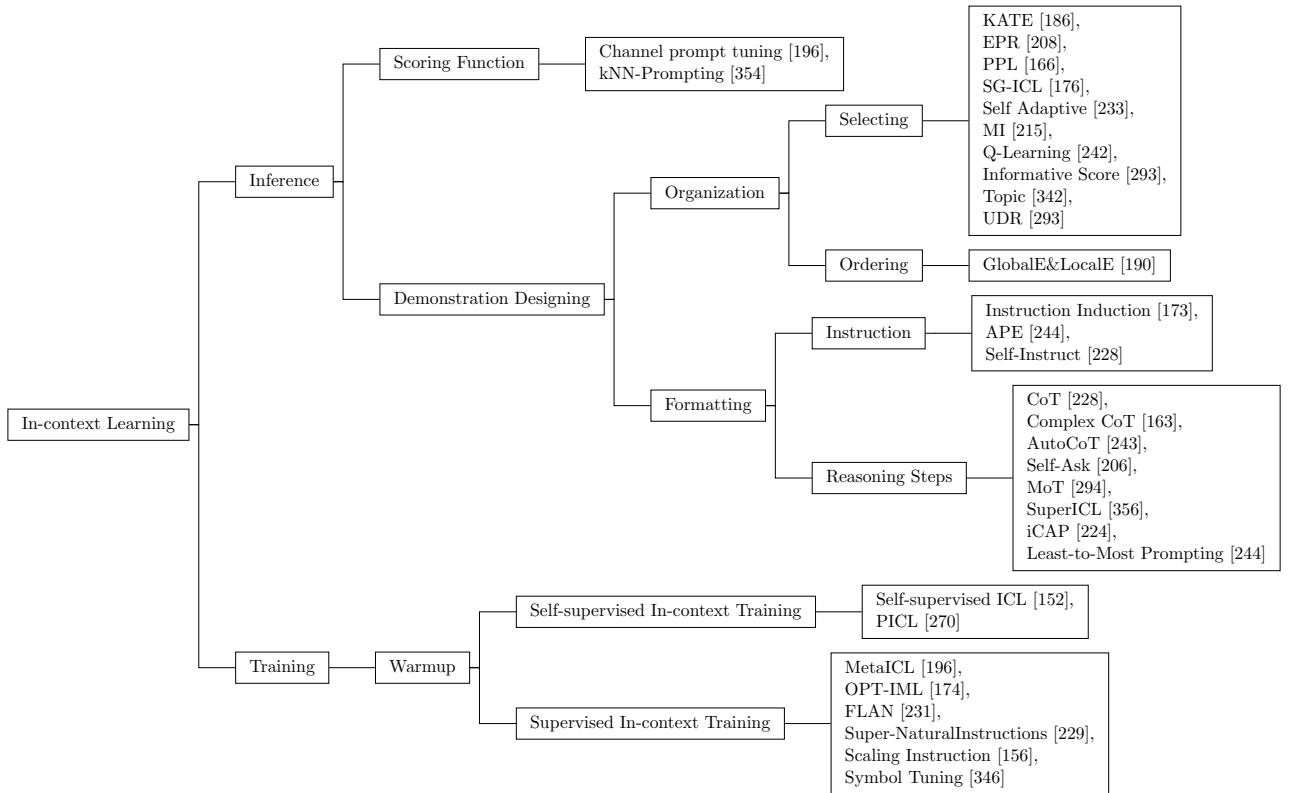


Figure 36: Taxonomy of in-context learning. The training and the inference stage are two main stages for ICL. During the training stage, existing ICL studies mainly take a pre-trained LLM as the backbone and optionally warm up the model to strengthen and generalize the ICL ability. Towards the inference stage, the demonstration design and the scoring function selection are crucial for the ultimate performance. Source: Dong et al. [265]

Designing branch). The selection aims to choose good examples for ICL using unsupervised⁷² or supervised methods. For example, KATE [186] and EPR [208] select demonstrations based on similarity. Ordering the selected demonstrations is also an important aspect of demonstration design. Lu et al. [190] have proven that order sensitivity is a common problem and affects various models. To address this problem, studies have proposed several training-free methods for ordering demonstrations. Liu et al. [186] sorted examples based on similarity, while GlobalE&LocalE [190] orders demonstrations based on global and local entropy.

⁷²Based on pre-defined metrics

A common representation of demonstrations is concatenating examples $(x_1, y_1), \dots, (x_k, y_k)$ with a template T directly. However, this approach may not be optimal for all tasks (i.e., when the task is complex or requires multiple steps such as math word problems and common-sense reasoning). In those cases, learning the mapping from x_i to y_i with only k demonstrations is challenging. Template engineering has been studied in Liu et al. [122] and Liu et al. [186] to generate task-specific templates. Some researchers have proposed designing a better demonstration format by describing tasks with instructions and adding intermediate reasoning steps between examples (x_i, y_i) . Instructions depend heavily on human input, but they can be generated automatically as shown in Honovich et al. [173] given several demonstration examples. Zhou et al. [368] proposed APE for automatic instruction generation and selection. To further improve the quality of the automatically generated instructions, Wang et al. [228] proposed Self-Instruct, which can eliminate its own generations.

Adding intermediate reasoning steps between examples introduced in Wang, Zhu, and Wang [342] is also called Chain-of-Thought prompting. We will delve into Chain-of-Thought prompting in the next Section 4.2.

ICL operates at inference stage – without explicit gradient updates – focusing on task recognition and learning through demonstrations. Task recognition utilizes pre-trained knowledge to solve tasks identified in the demonstrations. A Probably Approximately Correct (PAC) [347] framework has been proposed to evaluate ICL’s learnability, suggesting that LLMs can recognize tasks from minimal inputs.

On the other hand, task learning involves LLMs learning new tasks through demonstrations, akin to implicit fine-tuning through the attention mechanism, which generates meta-gradients. 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 [144]. Different model scales exhibit distinct capabilities; smaller models are adept at task recognition, while larger models (at least 66 billion parameters) are necessary for task learning [309].

Scoring Function	Target	Efficiency	Task Coverage	Stability
Direct	$\mathcal{M}(y_j C, x)$	+++	+	+
PPL	$\text{PPL}(S_j)$	+	+++	+
Channel	$\mathcal{M}(x C, y_j)$	+	+	++

Table 28: Summary of different scoring functions.

Despite its promises, ICL performance is known to be highly sensitive to input examples. Thus, a focal piece of ICL is the example selection based on scoring functions, which decides how to transform the LLMs’ predictions into an estimation of the likelihood of a specific answer. A direct estimation method adopts the conditional probability of candidate answers and selects the higher probability as the final answer [88]. However, this method poses some restrictions on the template design. For example, the answer tokens should be placed at the end of the input sequences. Perplexity (PPL) is another commonly used metric that computes the PPL of the entire input sequence:

$$S_j = \{C, s(x, y_i, I)\} \quad (28)$$

where C are the tokens of the demonstration examples, x is the input query, and y_i is the candidate label. As PPL is a global metric (i.e., it considers the entire input sequence), it

removes the limitations of token positions but requires extra computation time. In generation tasks such as machine translation, ICL predicts the answer by decoding tokens with the highest sentence probability combined with diversity-promoting strategies such as beam search or Top-p and Top-k [91] sampling algorithms. Min et al. [197] proposed a channel scoring function that estimates the likelihood of the input query given the candidate answer⁷³, which is more efficient and stable than the direct estimation method. In this way, language models are required to generate every token in the input, which could boost the performance under imbalanced training data regimes. To calibrate the bias or mitigate the sensitivity via scoring strategies, some studies add additional calibration parameters to adjust the model predictions [141].

4.1.2 ICL performance and origins

Knowing and understanding the factors that influence ICL can help improve LLMs' performance. ICL has a close connection with instruction tuning (discussed in Section 3.4.1) in that both utilize natural language to format the task or instances. However, instruction tuning needs to fine-tune LLMs for adaptation, while ICL only prompts LLMs for utilization [364]. Furthermore, instruction tuning can enhance the ICL ability of LLMs to perform target tasks, especially in the zero-shot setting⁷⁴ [156].

Stage	Factor
Pretraining	Pretraining corpus domain [211]
	Pretraining corpus combination [211]
	Number of model parameters [232, 88]
	Number of pretraining steps [232]
Inference	Label space exposure [198]
	Demonstration input distribution [198]
	Format of input-label pairing [198, 249]
	Demonstration input-label mapping [198, 237, 346]
	Demonstration sample ordering [190]
	Demonstration-query similarity [190]
	Demonstration diversity [249]
	Demonstration complexity [249]

Table 29: Summary of factors that correlate relatively strongly to ICL performance. Source: Dong et al. [265]

Several factors correlate relatively strongly to ICL performance, as shown in Table 29. ICL ability may arise by putting multiple corpora together in the pre-training stage, and the domain source is more important than the corpus size [211]. In contrast, pre-train on corpora related to downstream tasks and models with lower perplexity does not always perform better in ICL [211]. Wei et al. [232] suggested that a pre-trained model suddenly acquires some emergent ICL abilities when it achieves a large scale of pretraining steps or model parameters, and Brown et al. [88] showed that the ICL ability grows as the parameters of LLMs increase from 0.1 billion to 175 billion. At the inference stage, the properties of the demonstrations influence the ICL performance, such as the label space exposure, the format of input-label pairing, the ordering of demonstration samples, and the complexity of demonstrations [198, 249, 190]. There are contrasting results on the impact of input-label mapping related to ICL [198, 237]. An interesting finding is that, when a model is large enough, it will show an emergent ability

⁷³Compute the conditional probability in a reversed direction

⁷⁴Using only task descriptions

to learn input-label mappings, even if the labels are flipped⁷⁵ or semantically-unrelated⁷⁶ [345]. Some general validated factors for the ICL demonstrations are that they should be diverse, simple, and similar to the test example in terms of the structure [249]. Lu et al. [190] indicated that the demonstration sample order is also an important factor. Liu et al. [186] found that the demonstration samples with closer embeddings⁷⁷ to the query samples usually perform better than those with farther embeddings⁷⁸.

The reasons for the ICL ability have been investigated from different perspectives. Focusing on the pretraining data distribution, Chan et al. [151] showed that the ICL ability is driven by data distributional properties. The ICL ability emerges when the training data have examples appearing in clusters and have enough rare classes. Xie et al. [234] explained ICL as implicit Bayesian inference⁷⁹ and constructed a synthetic dataset to prove that the ICL ability emerges when the pretraining distribution follows a mixture of hidden Markov models. The hypotheses is that LM learn to do Bayesian inference during pre-training. To predict the next token during pretraining, the LM must infer (“locate”) the latent concept⁸⁰ for the document using evidence from the previous sentences. Later, if the LM infers also the latent concept prompt (provided by the demonstrations), then the in-context learning ability occurs. Under the learning mechanism, the ICL ability is explained by the ability of Transformers to encode effective learning algorithms to learn unseen linear functions according to demonstration samples, and encoded learning algorithms can achieve a comparable error to that from the least squares estimator [268]. Also Li et al. [296] showed the ability of Transformers to implement a proper function class through implicit empirical risk minimization for the demonstrations. From an information-theoretic perspective, Hahn and Goyal [273] showed an error bound for ICL under linguistically motivated assumptions to explain how next-token prediction can bring about the ICL ability. Another series of works attempted to build connections between ICL and gradient descent and found that Transformer-based in-context learners can implement standard fine-tuning algorithms implicitly [144, 308, 296]. Looking at functional components, Olsson et al. [204] found indirect evidence that “Induction heads”⁸¹ might constitute the mechanism for the majority of all ICL in large transformer models.

In-context learning (ICL) evaluation spans traditional tasks and newly proposed challenging tasks, and it provides open-source tools for standardized evaluation. ICL has been tested against established benchmarks, such as SuperGLUE and SQuAD, with mixed results. GPT-3, for example, exhibited comparable performance to state-of-the-art fine-tuning on some tasks

⁷⁵Flipped-label ICL uses flipped targets, forcing the model to override semantic priors to follow the in-context exemplars. For example, in the sentiment analysis task, the label “Positive” becomes “Negative” in ICL context and viceversa

⁷⁶The labels are semantically unrelated to the task(e.g., for sentiment analysis, it uses “foo/bar” instead of “negative/positive”)

⁷⁷Using Classify Token (CLS) embeddings of a pre-trained RoBERTa to measure the proximity of two sentences with the Euclidean distance. The CLS token is extensively used to capture the context and semantics of the input (e.g., the sentiment in sentiment analysis; category in classification tasks; etc.).

⁷⁸Retrieving the input k nearest neighbourhoods ordered by ascending similarity measure

⁷⁹Bayesian inference is a method of statistical inference in which Bayes’ theorem is used to update the probability for a hypothesis as more evidence or information becomes available. Fundamentally, Bayesian inference uses prior knowledge, in the form of a prior distribution in order to estimate posterior probabilities. $P(H||E) = \frac{P(E|H) \cdot P(H)}{P(E)}$, where $P(H)$ is the prior probability of hypothesis H , $P(E|H)$ is the likelihood of evidence E given hypothesis H , $P(E)$ is the marginal likelihood of evidence, and $P(H|E)$ is the posterior probability of hypothesis H given evidence E [348].

⁸⁰A latent variable that contains various document-level statistics. For example, a “news topics” concept describes a distribution of words (news and their topics), a format (the way that news articles are written), a relation between news and topics, and other semantic and syntactic relationships between words. In general, concepts may be a combination of many latent variables that specify different aspects of the semantics and syntax of a document

⁸¹attention heads that implement a simple algorithm to complete token sequences like $[A][B] \dots [A] \Rightarrow [B]$

within SuperGLUE but lagged in most natural language understanding tasks. Scaling the number of demonstration examples has shown potential but has yet to bridge the gap fully between ICL and traditional fine-tuning methods [88, 168].

New benchmarks have been introduced to assess the capabilities of large language models (LLMs) beyond traditional fine-tuning. The BIG-Bench and BIG-Bench Hard focus on tasks ranging from linguistics to social behaviours, with models outperforming human raters on many of these tasks [246, 218]. OPT-IML Bench has been designed to evaluate the generalization capabilities of LLMs across various held-out categories, emphasizing the model’s generalization capabilities [174]. OpenICL has been developed to provide a flexible and unified framework for ICL evaluation. This toolkit supports different LLMs and tasks, enabling consistent implementation and evaluation of ICL methods across various studies [351].

The application of In-Context Learning (ICL) has transcended the domain of natural language processing (NLP), influencing research in various modalities such as visual tasks, vision+language integration, and speech. Visual In-Context Learning explores how models generalize learned visual concepts to new, unseen tasks by leveraging contextual demonstrations akin to NLP-based ICL. Techniques such as image patch infilling and training models like masked autoencoders (MAE) exemplify this approach [149]. Noteworthy models like Painter and SegGPT have been developed to handle multiple tasks or integrate various segmentation tasks into a single framework [340, 341]. The Prompt Diffusion model introduced by Wang et al. [343] represents a pioneering effort in diffusion-based models displaying ICL capabilities, particularly when guided by textual prompts [343]. Integrating visual contexts with linguistic models has significantly improved vision-language tasks. Frozen and Flamingo models have demonstrated the feasibility of multi-modal, few-shot learning by combining vision encoders with large language models (LLMs). These models effectively perform ICL on multi-modal tasks when trained on large-scale multi-modal web corpora [137, 145]. Kosmos-1 and METALM extend these capabilities by demonstrating strong performance across various vision-language tasks, underpinned by a semi-causal language modelling objective [278, 169].

4.1.3 ICL future research

Future research in ICL is expected to focus on several key areas, including the optimization of pretraining objectives, the distillation of ICL abilities, the enhancement of ICL robustness, the improvement of ICL efficiency and scalability, the updating of knowledge within LLMs, the augmentation of models, and the expansion of ICL into multi-modal domains [265]. Optimizing pretraining objectives to better align with ICL requirements could enhance model capabilities for ICL applications. Introducing intermediate tuning phases and tailoring pre-training objectives to better align with ICL requirements could bridge this gap and enhance model capabilities for ICL applications [211]. An important goal is to distill ICL capabilities from larger models to smaller, more efficient ones, potentially enabling the deployment of ICL in resource-constrained environments [193]. Another area of improvement is the robustness of ICL, which is highly susceptible to the format and permutation of demonstrations [141, 190], without compromising accuracy or efficiency [374].

A more theoretical understanding of ICL’s mechanisms could lead to more robust implementations. Moreover, the scalability of ICL is constrained by the input limitations of language models and the computational cost associated with large numbers of demonstrations. Innovative strategies like structured prompting [168] and dynamic prompting [336] are being explored to address these challenges. The development of models with extended context capabilities [290] indicates significant potential for progress in this area. Finally, the expansion of ICL into multi-modal domains is expected to yield new insights and applications, particularly in vision and speech [265].

4.2 Chain-of-Thought

4.2.1 CoT strategy

Chain-of-Thought (CoT) prompting is an enhanced strategy developed to augment the performance of large language models (LLMs) on complex reasoning tasks such as arithmetic, commonsense, and symbolic reasoning [230, 123, 81]. This method integrates intermediate reasoning steps within the prompts, providing a more structured path towards the solution. To some extent, CoT can be considered a special case of ICL, as it involves the generation

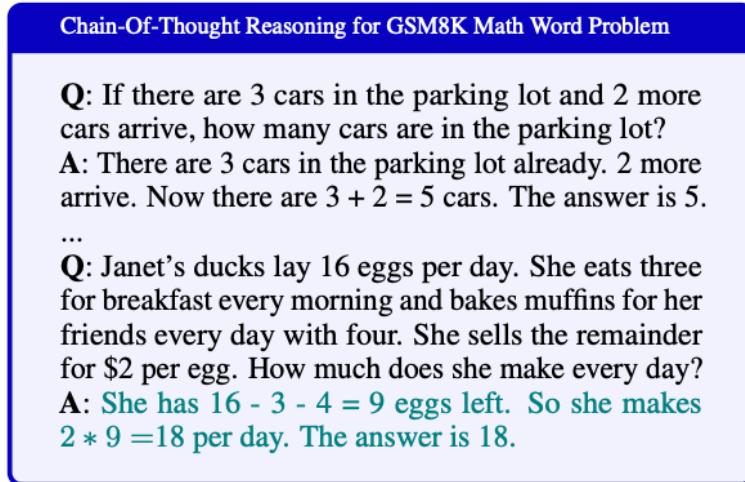


Figure 37: Chain-of-Thought reasoning for GSM8k math word problem. The prompt is coloured black, and the reasoning path produced by the language model is coloured teal. This reasoning path contains two reasoning steps. Source: Li et al. [295]

of prompts with a series of intermediate reasoning steps (Figure 38). Still, the ordering of demonstrations, in this case, has a relatively minor impact on the performance of LLMs [230].

Wei et al. [230] and Wang et al. [227] have shown that language models, when large enough (i.e., >100 billion parameters), can learn to perform complex reasoning tasks through CoT prompting without explicit task-specific [232].

CoT can be effectively combined with In-context Learning (ICL) in both few-shot and zero-shot settings:

- **Few-shot CoT.** In the few-shot scenario, CoT augments standard input-output pairs with intermediate reasoning steps. The design of CoT prompts is crucial; incorporating diverse and complex reasoning paths has been shown to boost LLM performance significantly. An automated approach, Auto-CoT, facilitates the generation of CoT sequences without manual effort by clustering and selecting representative questions [243].
- **Zero-shot CoT.** Unlike its few-shot counterpart, zero-shot CoT does not rely on annotated demonstrations. Instead, it generates reasoning steps directly from a prompt, significantly improving performance when scaled to larger models. This approach was pioneered by models like Flan-T5, which demonstrated improved zero-shot performance through instruction tuning on CoT annotations [156].

To apply these strategies effectively, it is essential to design CoT prompts that guide the model through the reasoning process. In Li et al. [295], the authors have shown that using diverse CoTs (i.e., prompts with multiple reasoning paths for each problem) can significantly

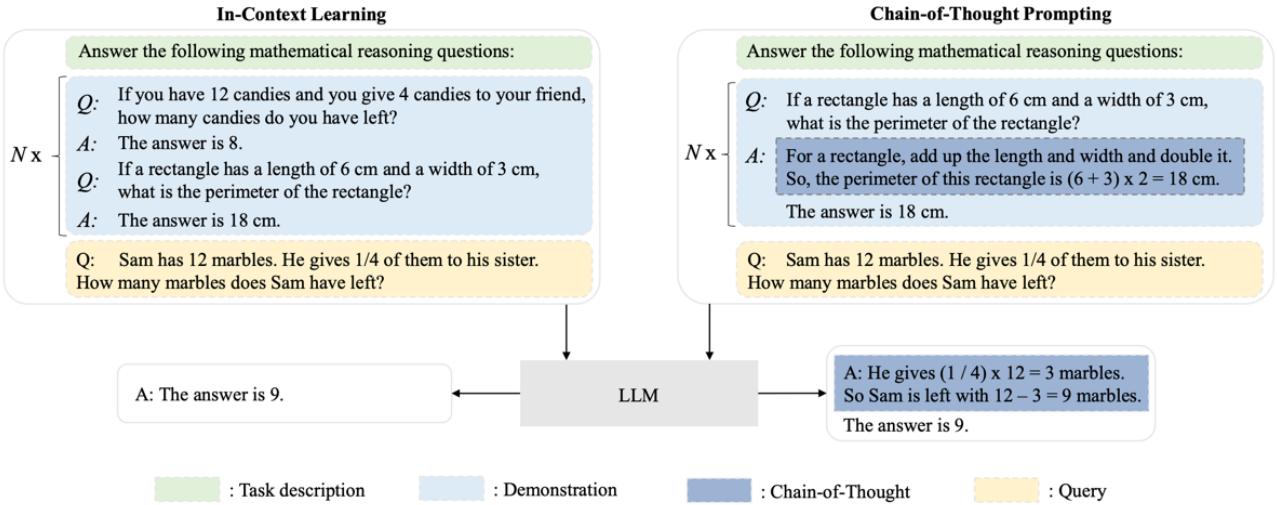


Figure 38: A comparative illustration of in-context learning (ICL) and chain-of-thought (CoT) prompting. ICL prompts LLMs with a natural language description, several demonstrations, and a test query, while CoT prompting involves a series of intermediate reasoning steps in prompts. Source: Zhao et al. [364]

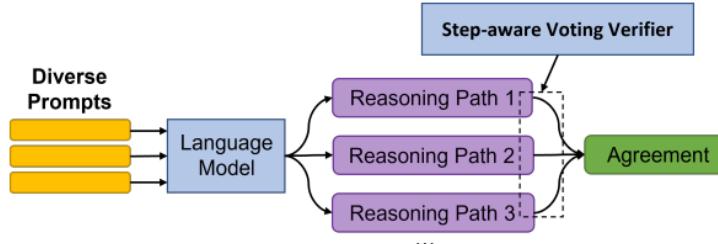


Figure 39: The DIVERSE approach for CoT. Source: Li et al. [295]

enhance the performance of LLMs on complex reasoning tasks. The proposed method, DIVERSE⁸², generates diverse CoTs by leveraging a self-ensemble approach that alternates between selection and inference. It has three main components: first, it generates diverse prompts to explore different reasoning paths for the same question; second, it uses a verifier to filter out incorrect answers based on a weighted voting scheme; and third, it verifies each reasoning step individually instead of the whole chain (Figure 39). In the first step, the model generates multiple reasoning paths for each question, which are then used to create diverse prompts following the idea that “All roads lead to Rome”. As an improvement of Wang et al. [227], DIVERSE selects M_1 different prompts for each question and M_2 reasoning paths for each prompt, resulting in $M_1 \times M_2$ diverse prompts. Then, the verifier takes a question and a candidate’s reasoning path and outputs the probability that the reasoning path leads to the correct answer. Different predictions are aggregated using a *voting verifier* to obtain the final prediction:

$$\hat{y} = \arg \max_y \sum_{i=1}^{M_1} \mathbf{1}_{y=y_i} \cdot f(\mathbf{x}_i, \mathbf{z}_i, \mathbf{y}_i) \quad (29)$$

where $\mathbf{1}_{y=y_i}$ is an indicator function that equals 1 if $y = y_i$, and $f(\cdot)$ is the probability produced by the verifier.

⁸²Diverse Verifier on Reasoning Step

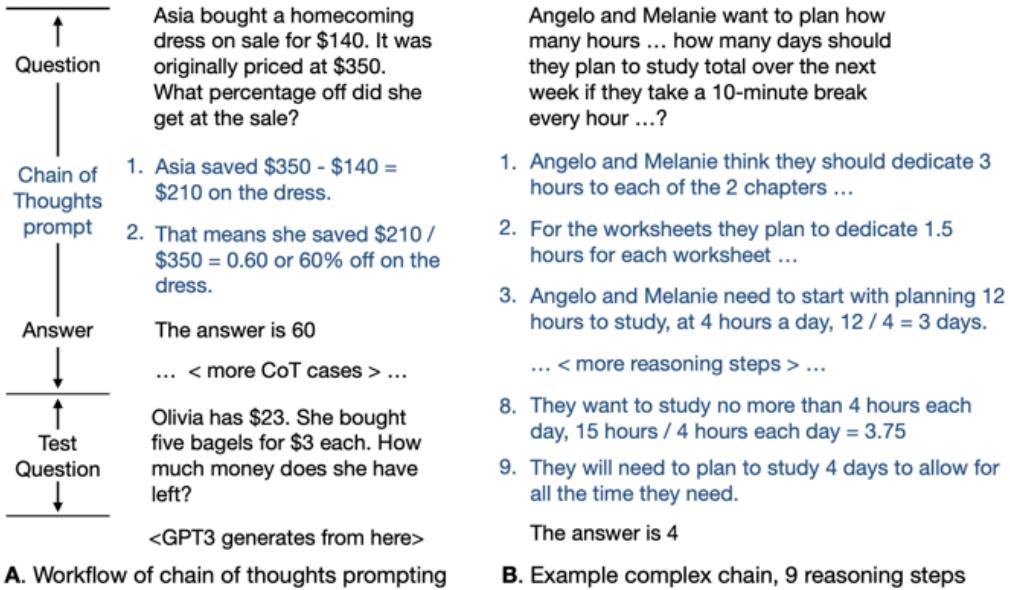


Figure 40: **A:** Chain of thoughts (in blue) are intermediate reasoning steps towards a final answer. The input of CoT prompting is a stack of a few (often 8) CoT cases before a test question. Then, the language model will continue generating an output CoT for the test question. **B:** Chains of harder reasoning complexity are chains with more reasoning steps (9 steps in this case, v.s. only 2 steps in subfigure A). Source: Fu et al. [163]

Another intuitive idea is that prompting with more complex reasoning steps (i.e., chains with more reasoning steps) is more likely to elicit the reasoning ability of LLMs [163], which can result in generating correct answers (Figure 40). Other complexity indicators than the number of reasoning steps, such as question lengths or the length of the underlying formula for solving a given problem, also exist, but improvements in performance are consistent across various complexity indicators. Consequently, question length can be used as a proxy for complexity for datasets not annotated with reasoning steps to generate CoT prompts. In that way, annotating only the identified few-shot instances is possible, thus reducing the annotation cost [163]. To exclude complexity correlated factors, Fu et al. [163] proposed prompts evaluation:

- **Simpler examples but the same number of reasoning steps.** For instance, comparing 24 cases that each require 3 reasoning steps with 8 cases that each require 9 reasoning steps, both resulting in a total of 72 steps.
- **Prompts of the longest lengths but not necessarily the most steps.** This ensures that the length is not the only factor being assessed.

It turned out that the complexity of reasoning steps is the most important factor for the performance of LLMs on complex reasoning tasks [163]. Complexity-based prompting can be further enhanced by using the output selection method called Complexity-based Consistency, alleviating the possibility that the model can take shortcuts during reasoning⁸³. The method explicitly promotes outputs with more complex reasoning chains at inference time, similar to the self-consistency practice in Wang et al. [227]. A voting mechanism is used to select the final output among top K complex reasoning chains, as shown in Figure 41.

Previously mentioned methods rely on two major paradigms: Zero-Shot-CoT and Manual-CoT. Zero-Shot-CoT is a task-agnostic paradigm that generates reasoning steps directly from

⁸³Relying on spurious correlations that inevitably exist in the training data and are not related to the reasoning process as shown by Mudrakarta et al. [49], Lai et al. [117], and Sugawara et al. [54]

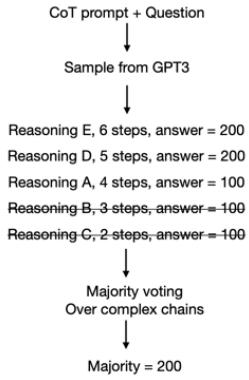
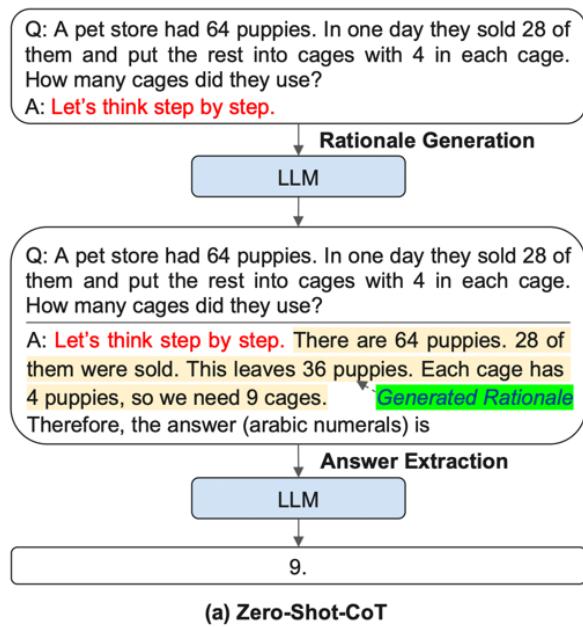
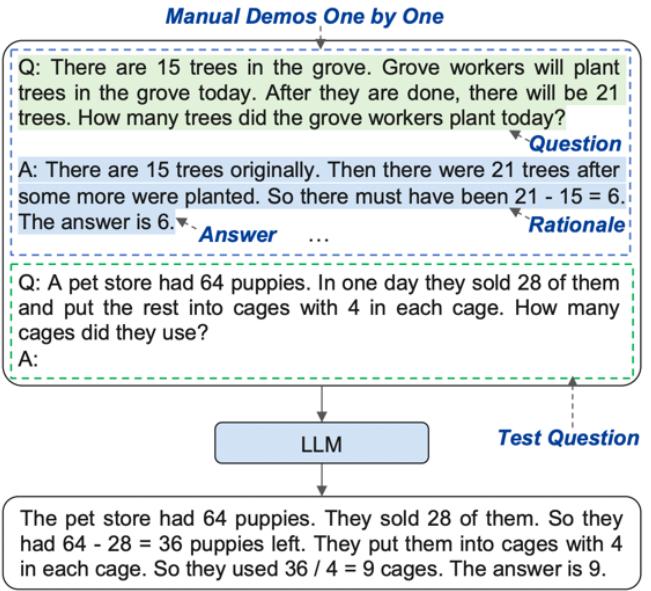


Figure 41: Complexity-based Consistency for CoT. During decoding, it samples N reasoning chains from the language model ($N = 5$ here) and takes the majority answer over the K ($K = 3$ here) most complex generated chains. Source: Fu et al. [163]



(a) Zero-Shot-CoT



(b) Manual-CoT

Figure 42: Zero-Shot-CoT [285] (using the “Let’s think step by step” prompt) and Manual-CoT[230] (using manually designed demonstrations one by one) with example inputs and outputs of an LLM. Source: Zhang et al. [243]

the prompt, eliminating the need for annotated CoT datasets [285], adding a single prompt like “Let’s think step by step” after the test question to facilitate the reasoning chains in LLMs. On the other hand, Manual-CoT uses manually designed demonstrations one by one, which can be expensive and time-consuming to create [230]. Since this prompting paradigm is task-agnostic and does not need input-output demonstrations, it is called Zero-Shot-CoT (left of Figure 42). With Zero-Shot-CoT, LLMs have shown to be decent zero-shot reasoners.

The other paradigm is few-shot prompting with manual reasoning demonstrations one by one [230]. Each demonstration has a question and a reasoning chain. A reasoning chain comprises a rationale (a series of intermediate reasoning steps) and an expected answer. With all the demonstrations being manually designed, this paradigm is called Manual-CoT (right of Figure 42).

To mitigate the effect of reasoning chain mistakes from Zero-Shot-CoT, Zhang et al. [243]

proposed the use of Auto-CoT, a method that generates demonstrations automatically since their diversity is crucial for the performance of LLMs. It consists of two main components: a clustering algorithm that groups similar questions and a representative selection algorithm that selects the most representative questions from each cluster. The overall procedure is illustrated in Figure 43. Diversity-based clustering may mitigate misleading by similarity effects⁸⁴, and

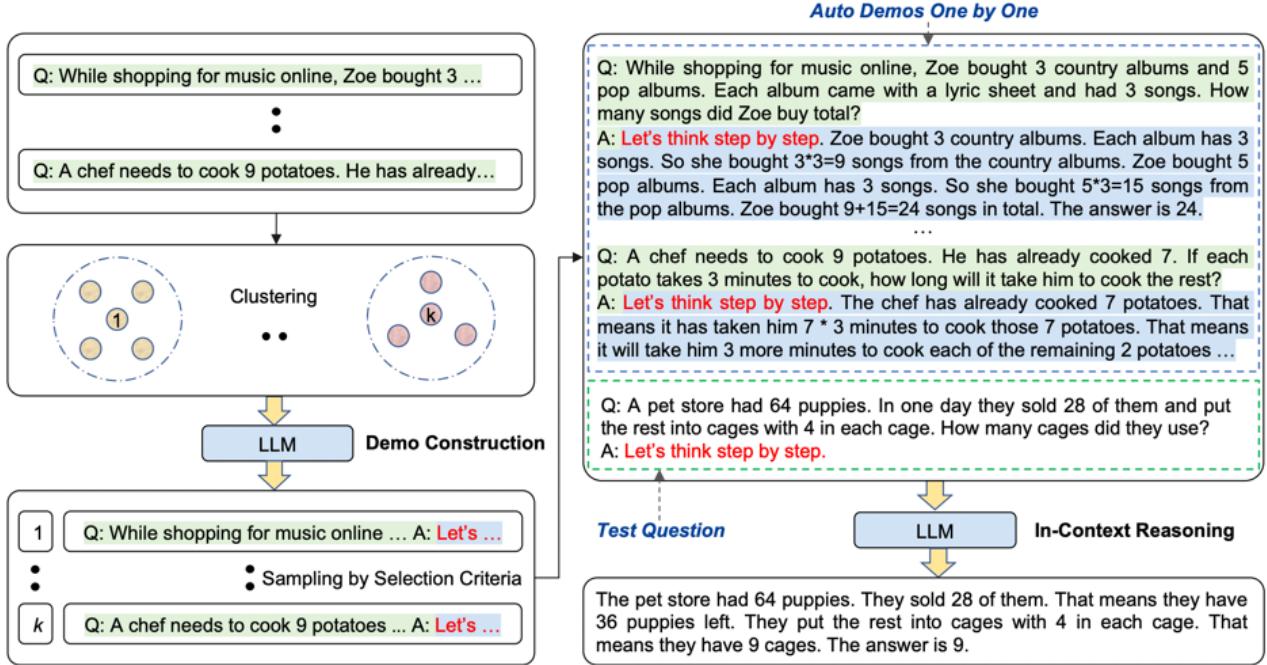


Figure 43: demonstrations (on the right) are automatically constructed one by one (total: k) using an LLM with the “Let’s think step by step” prompt. Source: Zhang et al. [243]

the representative selection algorithm can select the most representative questions from each cluster is used as demonstrations to generate reasoning chains for the test question. Auto-CoT has shown to be effective in generating diverse reasoning chains and improving the performance of LLMs on arithmetic and symbolic reasoning [243].

4.2.2 CoT performance and origins

CoT is considered by many as an emergent ability [232], a capability that suddenly appears and greatly enhances the performance of LLMs when they reach a certain scale. Moreover, CoT is only effective for tasks that require step-by-step reasoning, such as arithmetic, commonsense, and symbolic reasoning [230, 123, 81]. Whereas, for other tasks, CoT can be detrimental to the performance of LLMs with respect to standard prompting [226], e.g., MNLI-m/mm, SST-2, and QQP from GLUE[56]. It seems that the effectiveness of CoT is inversely proportional to the effectiveness of standard prompting [230].

Main prompting components, e.g., symbols, patterns, and text, impact CoT. It has been shown that pattern and text are essential for CoT performance, and removing them can lead to a significant drop in performance: text helps LLMs to generate valuable patterns, and patterns aid LLMs to understand tasks and generate texts that help solve them [192].

⁸⁴The retrieved demonstration questions are similar to the test question and ask “how long will it take him to cook the rest?”. The reasoning chains generated by Zero-Shot-CoT produce answers regarding “the total of” instead of “the rest”. Following the demonstrations, the component that retrieves the top- k similar questions – called Retrieval-Q-CoT – also fails by misunderstanding the meaning of “the rest”.

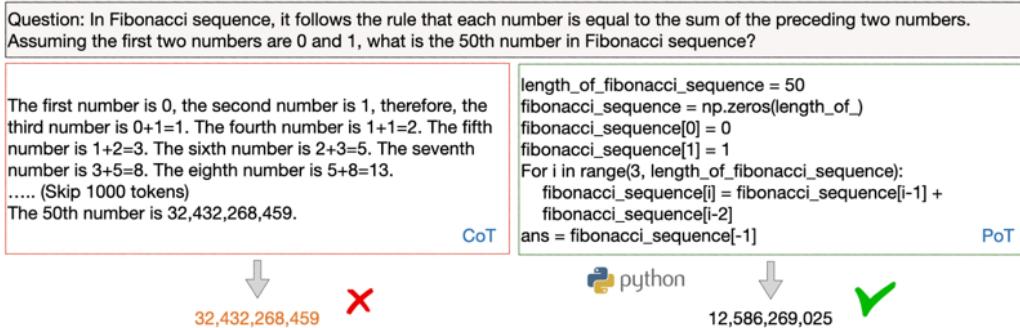


Figure 44: Program-of-Thoughts (PoT) for solving math word problems. The input is a math word problem, and the output is a program that can solve the problem. Source: Chen et al. [259]

The origins of CoT ability are widely hypothesized to be elicited by training on code since those models have shown to be more effective in reasoning tasks [162, 185]. Intuitively, code data is well organized with algorithmic logic and programming flow, which may be helpful in improving the reasoning performance of LLMs. However, this hypothesis still lacks publicly reported evidence of ablation experiments (with and without training on code). We'll try to address this gap in the next section 5, by conducting a series of experiments to evaluate the effectiveness of training on code data for reasoning tasks. In addition, instruction tuning seems not to be the main factor for CoT ability since the performance of LLMs on CoT tasks is not significantly improved by instruction tuning [156].

4.3 Program-of-Thoughts

PoT uses a programmatic approach to prompt LLMs to solve complex reasoning tasks proposed by Chen et al. [259]. It leverages models to generate text and programming languages statements, executing them to get the final answer. The approach is similar to CoT, but the reasoning steps are expressed in a more structured way, resembling a program (see Figure 44). CoT uses LLMs for both reasoning and computation, i.e., the language model not only needs to generate the mathematical expressions but also needs to perform the computation in each step⁸⁵. Whatever the case, LLMs are not ideal for actually solving these mathematical expressions, because:

- LLMs are very prone to arithmetic calculation errors, especially when dealing with large numbers.
- LLMs cannot solve complex mathematical expressions like polynomial equations or even differential equations.
- LLMs are highly inefficient at expressing iteration, especially when the number of iteration steps is large.

PoT can overcome these limitations by using a programmatic approach, where the reasoning steps are expressed as Python programs that can be executed to get the final answer by a Python interpreter. The programmatic approach is also different from generating equations directly, that is found to be more challenging for LLMs [230]. It mainly differs from equation generation for the following reasons:

- PoT breaks down the reasoning process into a series of steps, each of which is expressed as a Python statement;

⁸⁵Some studies contradict the fact that LLMs can perform computations or reasoning tasks [207, 373, 283].

- it binds semantic meaning to variables, which can elicit language models’ reasoning capabilities and generate more accurate programs

In zero-shot PoT, a caveat is that LLM can fall back to generating a reasoning chain in comments rather than in the program. Therefore, Chen et al. [259] proposes to suppress “#” token logits to encourage it to generate programs.

As confirmed by our experiments in Section 5, PoT can significantly improve performance on math problems compared to CoT. Even though PoT is effective on highly symbolic math problems, it still struggles with AQuA dataset, which contains complex algebraic questions mainly due to the diversity of questions, which the demonstration cannot possibly cover. For semantic reasoning tasks like commonsense reasoning (StrategyQA), probably PoT is not the best option. In contrast, CoT can solve more broader reasoning tasks.

4.4 Planning for complex tasks

4.4.1 Commonsense knowledge

ICL and CoT are two simple yet general strategies for solving various tasks. However, they struggle with complex tasks that require long-term planning, such as mathematical word problems [207] and multi-hop question answering [373]. Commonsense knowledge⁸⁶ is essential for NLP systems to understand and generate human-like language. Main categories are summarized in Bian et al. [373]:

- **General commonsense:** refers to knowledge that is widely shared and assumed to be true by most people, such as the sun rises in the east and sets in the west.
- **Physical commonsense:** involves intuitive knowledge about the physical world, such as objects falling to the ground when dropped and water flowing downhill.
- **Social commonsense:** involves knowledge about social norms, customs, and practices, such as it is polite to say “thank you” when making requests.
- **Science commonsense:** involves knowledge about basic scientific principles, such as gravity pulling all objects on Earth to Earth’s centre.
- **Event commonsense:** involves knowledge about the sequence of events and their causal relationships, such as if a glass is knocked over, the liquid inside will spill.
- **Numerical commonsense:** involves knowledge about numbers, such as a human has two hands and ten fingers.
- **Prototypical commonsense:** involves knowledge about typical or prototypical examples of concepts, such as a swallow is a kind of bird and a bird has wings.
- **Temporal commonsense:** involves knowledge about time, such as travelling abroad requires a longer time than taking a walk.

A list of commonsense QA datasets commonly used in evaluating LLMs is shown in Table 30. These datasets encompass domains like general, physical, social, science, event, numerical, prototypical, and temporal commonsense. Table 31 shows the accuracy of GPT-3, GPT-3.5, and ChatGPT on these datasets. The ability of models to leverage commonsense is probably improved by instruction tuning and human alignment, looking at the results of Instruct GPT and

⁸⁶It includes knowledge about the spatial, physical, social, temporal, and psychological aspects of the typical everyday life, as well as an awareness of social norms, beliefs, and values [8].

Dataset	Domain	Example (Bold texts are the answers)
CommonsenseQA	General	Choose your answer to the question: Where are you likely to find a hamburger? A. fast food restaurant , B. pizza, C. ground up dead cows, D. mouth, E. cow circus
OpenBookQA	General	Choose your answer to the question: If a person walks in the opposite direction of a compass arrow they are walking A. west, B. north, C. east, D. south
WSC	General	Choose sub-sentence A or B that completes the sentence: The trophy doesn't fit into the brown suitcase because A. the trophy is too small. B. the suitcase is too small.
PIQA	Physical	Choose one that is correct: A. ice box will turn into a cooler if you add water to it. B. ice box will turn into a cooler if you add soda to it.
Social IQA	Social	Taylor taught math in the schools after studying to be a teacher. Choose the most suitable answer for the question: What does Taylor need to do before this? A. get a certificate , B. teach small children, C. work in a school
ARC	Science	Choose your answer to the question: Which technology was developed most recently? A. cellular telephone , B. television, C. refrigerator, D. airplane
QASC	Science	Choose your answer to the question: What is described in terms of temperature and water in the air? A. storms; B. climate ; C. mass; D. seasonal; E. winter; F. density; G. length
HellaSWAG	Event	Choose your answer to the question: We see a chair with a pillow on it. A. a man holding a cat does curling. B. a man holding a cat starts hitting objects on an item. C. a man holding a cat is wrapping a box. D. a man holding a cat sits down on the chair.
NumerSense	Numerical	a square is a shape with <mask>equally length sides. (four)
ProtoQA	Prototypical	Use simple words separated by commas to name something in your life that could cause you to lose weight. (Eating less, exercising more, stress.)
MC-TACO	Temporal	Select all feasible answers for the question: Carl Laemmle, head of Universal Studios, gave Einstein a tour of his studio and introduced him to Chaplin. At what time did Einstein return home? A. 8:00 PM ; B. a second later; C. a hour later

Table 30: Examples from commonsense QA datasets. Source: Bian et al. [373]

ChatGPT versus GPT-3 in Table 31.) ChatGPT demonstrates strong capabilities in common-sense QA tasks but has limitations in identifying necessary knowledge. It has been proved by evaluating answers generated by ChatGPT on questions from each commonsense QA dataset using the following prompt:

“What knowledge is necessary for answering this question?
{question} {answer choices(if applicable)}”.

This means that LLMs are inexperienced problem solvers who rely on memorizing a large amount of information to cover the answers[373]. Kambhampati [378] and Kambhampati et al. [379] strongly argue that LLMs can't reason or plan autonomously. Techniques like Chain-of-Thought (CoT), ReACT, and fine-tuning, which are often used to enhance their capabilities, still do not enable sufficient generalization. LLMs struggle with self-verification because they lack the ability to assess the accuracy of their outputs. A key question arises:

Dataset	GPT-3	Instruct GPT	ChatGPT	Human
CommonsenseQA	38	81	74	88.9
OpenBookQA	22	65	73	89.3
WSC	46	78	78	92.1
PIQA	48	77	78	94.5
Social IQA	36	71	62	86.9
ARC	27	88	94	—
QASC	25	75	74	93.0
HellaSWAG	19	61	67	95.7
NumerSense	45	63	79	89.7
ProtoQA	67.3	84.6	94.2	—
MC-TACO	20	53	52	75.8

Table 31: Evaluation results (accuracy) of large language models on commonsense QA datasets. Source: Bian et al. [373]

Why does LLM respond in constant time, even for polynomial or exponential problems?

For Kambhampati [378] and Kambhampati et al. [379] the answer lies in their nature as retrievers, not true reasoners. LLMs can mimic planning by combining retrieved information but lack true instance-level understanding required for accurate reasoning. LLMs excel at tasks involving pattern recognition within a distribution⁸⁷, but struggle with instance-specific⁸⁸ tasks like formal planning or sequencing actions toward a goal. For example, even when fine-tuned for specific tasks like multiplication, LLMs falter with more complex variations, showing their limitations. Ultimately, while LLMs can replicate certain logical patterns, their planning abilities are superficial, relying heavily on memorized logic rather than true reasoning. Additionally, involving humans to iteratively prompt LLMs introduces the risk of the “Clever Hans effect”⁸⁹, where the model’s responses are inadvertently influenced by subtle cues from the prompter,

⁸⁷The distributional or style properties in various fields can be understood as the recurring patterns and characteristics that define the general appearance or structure of an object or medium. In the realm of art, these properties might include brushstroke patterns, color palettes, and compositional rules that collectively define an artist’s body of work or the broader characteristics of an art movement. These stylistic elements enable the recognition of an artist’s work even when individual pieces differ in content.

Similarly, in language, distributional properties pertain to the recurring patterns in word choice, sentence structure, and other linguistic elements that define a particular writing style or genre. These patterns help in identifying the genre or author of a text based on its overall style rather than its specific content.

In computer vision, distributional properties refer to the consistent textures, lighting conditions, and geometric patterns across images of a specific type of object or scene. For example, the overall shape of cars or the texture of fur in animals represents such properties. These features allow models to recognize new instances of objects that share these common characteristics, even if the specific details differ from those previously encountered.

⁸⁸Instance properties refer to the specific and unique features that distinguish one example from another within a given category. In art, these properties are reflected in the distinct brushstrokes, intricate details, and particular color choices used in an individual painting. These elements contribute to the identity of a specific artwork, differentiating it from others, even within the same artist’s portfolio.

In language, instance properties manifest as the precise selection of words, the unique arrangement of sentences, and the specific use of punctuation in a particular sentence or paragraph. These elements define the uniqueness of a text, capturing the nuances of expression that distinguish one piece of writing from another, even if they share the same overall style or genre.

In the domain of computer vision, instance properties are found in the detailed characteristics of a specific object in an image, such as the color, make, and model of a particular car, as well as any unique markings it may have. These properties enable the recognition of a particular instance of an object, allowing for fine-grained classification and identification within a broader category.

⁸⁹Clever Hans was a horse claimed to have performed arithmetic and other intellectual tasks. After a formal investigation in 1907, psychologist Oskar Pfungst demonstrated that the horse was not actually performing

rather than genuine understanding. While LLMs can't plan independently, they can assist in planning when combined with external solvers and verifiers in an LLM-Modulo framework. In this setup, LLMs support planning by suggesting plans, guessing domain models, elaborating on problem specifications, and translating formats, but they still rely on external systems for verification and sequencing.

Even though we have seen surprising abilities of LLMs, Qian et al. [207] have shown additional limitations on certain basic symbolic manipulation tasks, such as copy, reverse and addition, particularly when dealing with repeating symbols⁹⁰ and OOD⁹¹ data. To address these limitations, Qian et al. [207] have proposed a series of methods to improve the performance of LLMs on these tasks, such as positional markers, fine-grained computation steps, and combining LMs with callable programs for basic operations. Positional markers⁹² and fine-grained computation steps⁹³ provide some improvement with repeating symbols but not with OOD. It clearly indicates the limitation of Transformers and pre-trained language models in induction. Combining LMs with callable programs⁹⁴ for basic operations shows potential but still relies on the LM's ability to locate tokens accurately. The LM with tutor method⁹⁵ demonstrates each task step, significantly improving accuracy and handling OOD scenarios, effectively achieving 100% accuracy on all tasks.

With the release of new models like Open AI o1 and o3⁹⁶ and Claude 3.5, the field is moving towards more powerful models that can potentially address some of the previous limitations. Wang et al. [387] explores the planning capabilities of OpenAI's o1 models, focusing on their performance across diverse tasks requiring feasibility, optimality, and generalizability. The o1-preview model demonstrates improvements in generating feasible plans compared to earlier language models like GPT-4. However, the study identifies key challenges, such as the model's limitations in following domain-specific constraints, which often misinterpret physical or logical constraints. Also, the model struggles to generate coherent plans. Although the individual steps may be valid, the model sometimes fails to sequence them into a coherent, goal-oriented plan. Moreover, the model's ability to interpret initial and goal states leads to errors, particularly in tasks requiring multi-step reasoning. Regarding the optimality of plans, the model often fails to generate optimal plans, instead producing suboptimal or inefficient solutions with duplicate or unnecessary steps. The model lacks mechanisms to incorporate domain-specific heuristics or

these mental tasks, but was watching the reactions of his trainer. He discovered this artifact in the research methodology, wherein the horse was responding directly to involuntary cues in the body language of the human trainer, who was entirely unaware that he was providing such cues.

⁹⁰Copy example with repeating symbols `input: ...989894... → answer: ...9894...`

⁹¹Out-of-distribution refers to prompting the model to execute an operation on numbers with more digits with respect to numbers used for training. It demonstrates the ability to generalize on unseen data.

⁹²LMs have implicit positional markers embedded in the architecture. Most Transformer-based LMs encode the positional information into positional vectors and add each of them to the corresponding word vector. Explicit positional markers are added into input strings: `input: ...222... → output: ...A2B2C2...`. Essentially, adding explicit positional markers breaks the repeating numbers into a non-repeating input sequence.

⁹³For example, in k-digit addition, the model is allowed to break it down into k simple 1-digit addition, and the model is allowed to generate k intermediate addition results to get the final answer.

⁹⁴A callable function `add(1,5)` can be invoked and return the result in text: `carry C: 0, result 6`

⁹⁵A tutor shows every single step visually and sometimes calls an already learned sub-module to complete a task. Instead of providing a training example: `copy: 1 1 1 2 2 2 result: 1 1 1 2 2 2`, the tutor explicitly shows the model how to copy the input as follows: `rmov, end=F, cpy, rmov, end=F, cpy, ..., rmov, end=T`. where `rmov` is a function that moves the tape head to the right, `cpy` is a function that copies the current symbol, and `end=F` indicates that the end of the tape is not reached. One can relate the setting with a multiple tape Turing machine where state transition is conducted among the positions of tape heads and read/write operations. The Transformer is trained to learn such state transition, thus completing the programming of a Turing machine.

⁹⁶There are still not enough details about the o3 model to provide a comprehensive analysis.

optimization techniques, resulting in suboptimal decision-making. Finally, the model's generalizability remains limited. It struggles with tasks that require reasoning over unseen scenarios and symbolic reasoning, where action semantics diverge from natural language.

In the following paragraphs, we will discuss the general framework of prompt-based planning, plan generation, plan execution, and plan evaluation. After that we will present the most common approaches to planning and their limitations.

4.4.2 Prompt and code based planning

Prompt-based planning has been proposed to break down complex tasks into simpler sub-tasks, and generate a plan of actions to accomplish the task. The general framework of prompt-based planning is shown in Figure 45.

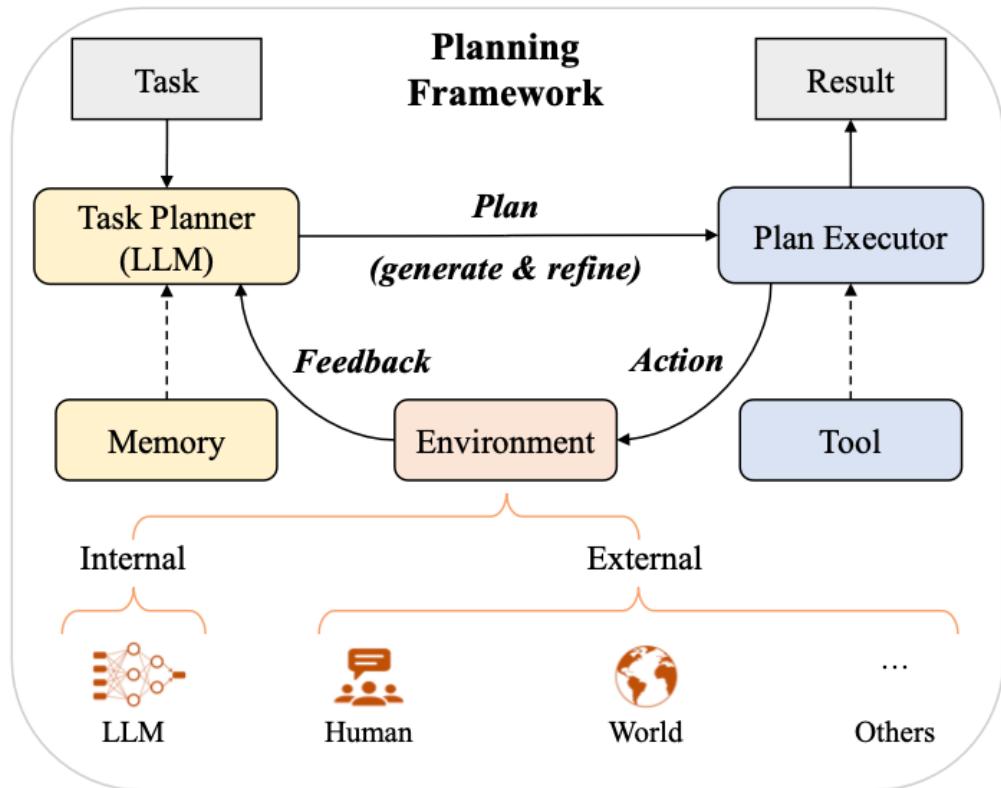


Figure 45: The general framework of prompt-based planning. Source: Zhao et al. [364]

In this paradigm, there are three main components: the planner, the executor, and the environment⁹⁷. The first component is the planner, which generates a plan of action to solve the task. The plan can be generated in various forms, e.g., natural language, symbolic, or programmatic [164, 244], that we will discuss in the next section 4.4.2. The memory mechanism can enhance the task planner, which stores intermediate results and reuses them in the future.

The plan executor is responsible for executing the plan generated by the planner. It can be implemented as a separate LLM for textual tasks or as a program executor for programmatic tasks [338, 164].

The environment is the world where the task is executed, which can be set up as the LLM itself or an external system, e.g., a simulator or a virtual world like Minecraft [359, 337]. The

⁹⁷It's similar to Reinforcement Learning, where the planner is the agent, the executor is the policy, and the environment is the world, but the difference is that in RL they are typically interleaved in the agent, while in prompt-based planning they are separated

environment provides feedback to the task planner about the result of the actions, which can be used to update the plan, either in the form of natural language or from other multimodal signals [322, 301]

4.4.3 Plan generation

For solving complex tasks, the planner needs to generate a long-term and multi-step plan, which requires the planner to be able to reason over long-term dependencies and develop a coherent and consistent plan. First, it needs to understand the task and break it down into sub-tasks, then generate a plan that can accomplish the task by executing the sub-tasks in a proper order. The plan should be generated in an interpretable and executable way by the executor, which acts according to the plan and interacts with the environment to accomplish the task. The planner can further incorporate the feedback from the environment to update and refine the plan and achieve better performance.

The most common form of plan generation is natural language, where the planner generates a sequence of natural language instructions that describe the plan. In this approach, LLMs are prompted to generate a sequence of instructions that describe the plan, which the executor can execute to accomplish the complex task. For example, Plan-and-Solve [338] adds explicit instructions to the input of the LLM, which guides the model to generate a plan for solving the task (i.e., “devise a plan”) in a zero-shot setting, while Self-planning [377] and DECOMP [175] generate the plan in a few-shot setting by providing a few examples to guide LLM through ICL. Other approaches consider incorporating extra tools or models when planning, such as ToolFormer [319] and HuggingGpt [321]. ToolFormer is a model trained to decide which APIs to call when to call them, what arguments to pass, and how to best incorporate the results into future token prediction. This is done in a self-supervised way, requiring nothing more than a handful of demonstrations for each API. It incorporates a range of tools, including a calculator, a Q&A system, two different search engines, a translation system, and a calendar. HuggingGpt is an LLM-powered agent that leverages LLMs (e.g., ChatGPT) to connect various AI models in machine learning communities (e.g., Hugging Face) to solve AI tasks. Specifically, it uses ChatGPT to conduct task planning when receiving a user request, select models according to their function descriptions available in Hugging Face, execute each subtask with the selected AI model, and summarize the response according to the execution results.

Although text-based plan approaches sound intuitive, they have limitations since the generated plans may lead to incorrect results due to the ambiguity of natural language, even when the plan is sound. To address this issue, code-based plan generation has been proposed. In this method, the planner generates a program the executor can execute to accomplish the task. Compared to text-based plans, programmatic plans are more verifiable and less ambiguous, and they can directly be executed by interpreters or compilers (e.g., Python or PDDL⁹⁸) to accomplish the task. In this approach, LLMs are first prompted to generate a program to solve the task and then leverage a deterministic solver to execute it. For example, Faithful CoT [302] and PAL [164] decompose a reasoning task into two stages: at the first stage, the LLM generates a plan conditioned on the query; at the second stage, a deterministic solver executes the plan to derive the final answer. Furthermore, code-based approaches can be applied to embedded agents similarly, such as in the case of PROGPROMPT [212] and LLM+P [297]. They both first prompt the LLM to generate plans in code (Python functions or PDDL files) and then leverage on a virtual agent or classical planner to solve the task according to the code-based generated plan.

⁹⁸Planning Domain Definition Language defines the “universal” aspects of a problem. Essentially, these aspects do not change regardless of the specific situation we’re trying to solve. In PDDL, this is mostly the object types, predicates and actions that can exist within the model.

In the following paragraphs, we will elaborate on some notable approaches to natural language and programmatic plan generation.

Plan-and-Solve (PS) prompting is a text-based plan generation approach that consists of two components: devising a plan and carrying out the subtasks. The process includes:

1. **Step 1: Prompting for Reasoning Generation.** To meet the criteria for effective problem-solving, templates guide LLMs in devising and completing a plan with attention to calculations and intermediate results. For example: “Let’s first understand the problem, extract relevant variables, devise a plan, and solve the problem step by step.”
2. **Step 2: Prompting for Answer Extraction.** Similar to Zero-shot-CoT, another prompt extracts the final numerical answer from the reasoning text.

A comparison of prompting strategies is shown in Figure 46. The PS+ variant of Plan-and-Solve is an extension that adds detailed instructions to improve reasoning quality.

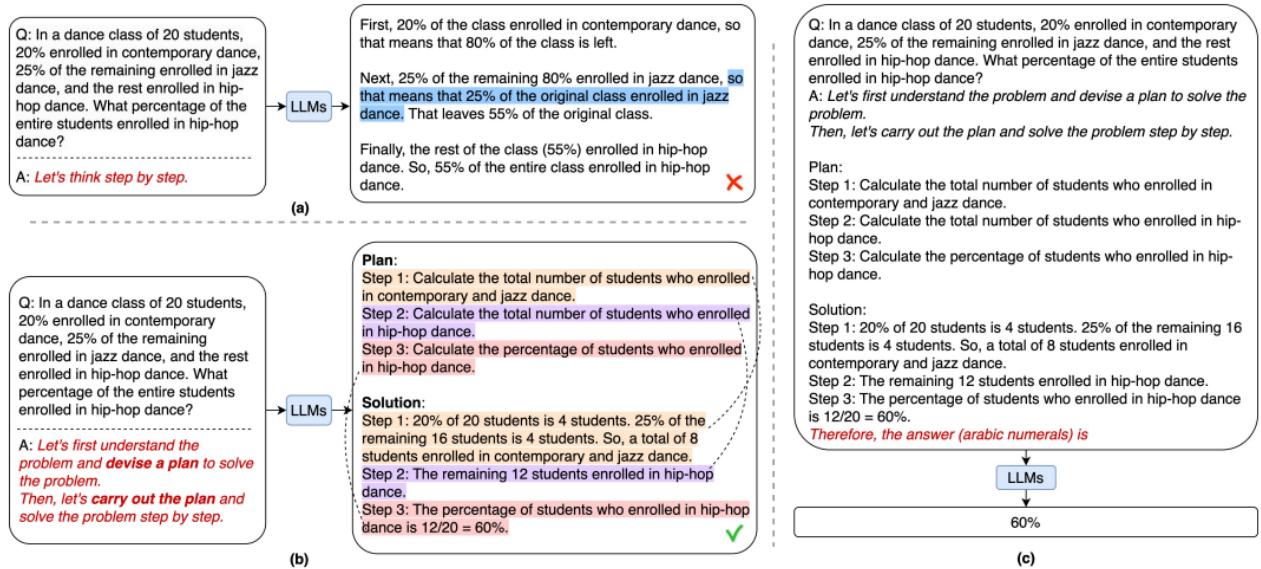


Figure 46: Example inputs and outputs of GPT-3 with (a) Zero-shot-CoT prompting, (b) Plan-and-Solve (PS) prompting, and (c) answer extraction prompting. While Zero-shot-CoT encourages LLMs to generate multi-step reasoning with “Let’s think step by step”, it may still generate wrong reasoning steps when the problem is complex. Unlike Zero-shot-CoT, PS prompting first asks LLMs to devise a plan to solve the problem by generating a step-by-step plan and carrying out the plan to find the answer. Source: Wang et al. [338]

Method	MultiArith	GSM8k	AddSub	AQUA	SingleEq	SVAMP
Zero-shot-CoT	83.8	56.4	85.3	38.9	88.1	69.9
PoT	92.2	57.0	85.1	43.9	91.7	70.8
PS (ours)	87.2	58.2	88.1	42.5	89.2	72.0
PS+ (ours)	91.8	59.3	92.2	46.0	94.7	75.7

Table 32: Accuracy comparison on math reasoning datasets. Source: Wang et al. [338]

Compared to Zero-shot-CoT, which suffers from pitfalls like calculation and missing-step errors, PS+ Prompting has shown to be more effective in addressing these issues [338]. The

Method	CSQA	StrategyQA
FEW-SHOT-CoT (MANUAL)	78.3	71.2
ZERO-SHOT-CoT	65.2	63.8
ZERO-SHOT-PS+	71.9	65.4

Table 33: Accuracy on commonsense reasoning datasets. Source: Wang et al. [338]

Method	Last Letter	Coin Flip
Few-Shot-CoT (Manual)	70.6	100.0
Zero-shot-CoT	64.8	96.8
Zero-shot-PS+	75.2	99.6

Table 34: Accuracy on symbolic reasoning datasets. Source: Wang et al. [338]

experiments with GPT-3 show that PS+ consistently outperforms Zero-shot-CoT and is comparable to 8-shot CoT prompting on math reasoning problems. Self-consistency (SC)⁹⁹[227] improves performance by generating multiple reasoning paths and selecting the final answer by majority voting. PS+ with SC outperforms PS+ without SC and Zero-shot-CoT with SC.

Least-to-Most Prompting is a text-based prompting strategy that aims to improve the performance of LLMs on complex reasoning tasks proposed by Zhou et al. [244]. Least-to-most prompting consists of two stages:

1. **Decomposition:** The prompt contains examples demonstrating problem decomposition, followed by the specific question to be decomposed.
2. **Sub-problem Solving:** The prompt consists of examples demonstrating sub-problem solving, previously answered subquestions and solutions, and the next question to be answered.

Figure 47 illustrates this approach.

Least-to-most prompting significantly outperforms Chain-of-Thought prompting on the last-letter-concatenation task¹⁰⁰ [230], especially on longer lists¹⁰¹. Table 35 shows the accuracy comparison.

Method	Length 4	Length 6	Length 8	Length 10	Length 12
Standard Prompting	0.0	0.0	0.0	0.0	0.0
Chain-of-Thought	84.2	69.2	50.2	39.8	31.8
Least-to-Most	94.0	88.4	83.0	76.4	74.0

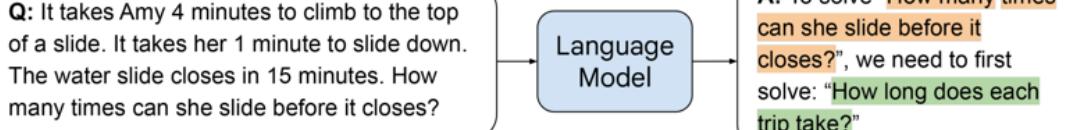
Table 35: Accuracies of different prompting methods on the last-letter-concatenation task. Source: Zhou et al. [244]

⁹⁹It reduces randomness in LLM’s output by generating N reasoning results and determining the final answer by majority voting

¹⁰⁰In this task, each input is a list of words, and the corresponding output is the concatenation of the last letters of the words in the list. For example, “thinking, machine” outputs “ge”, since the last letter of “thinking” is “g” and the last letter of “machine” is “e”.

¹⁰¹When the testing lists are much longer than the lists in the prompt exemplars.

Stage 1: Decompose Question into Subquestions



Stage 2: Sequentially Solve Subquestions

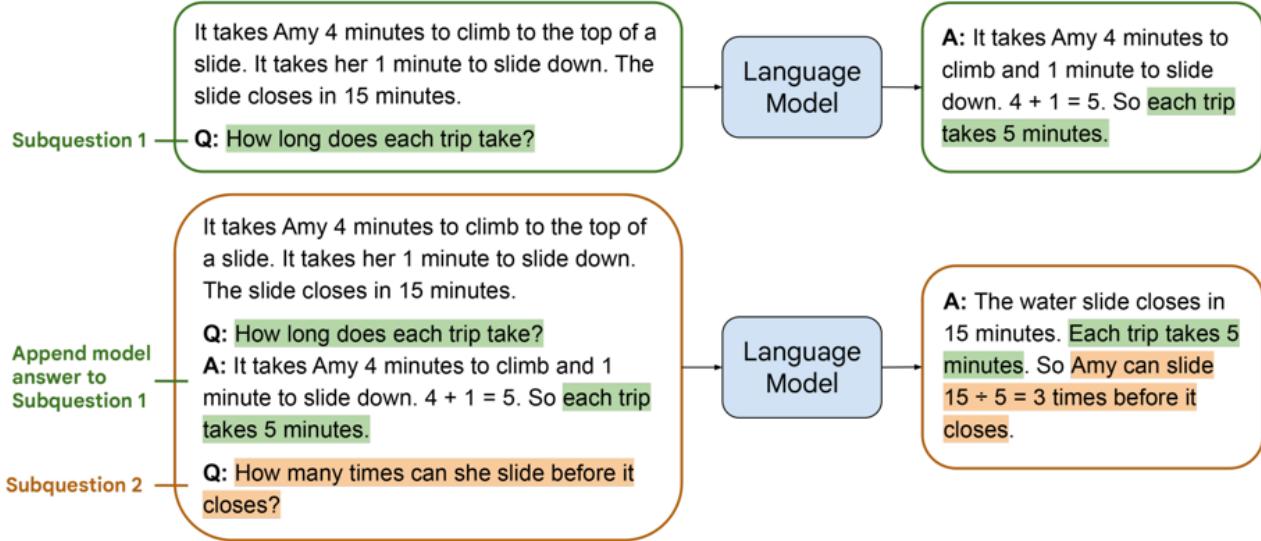


Figure 47: Least-to-most prompting teaches language models how to solve a complex problem by decomposing it to a series of simpler subproblems. It consists of two sequential stages: (1) decomposition and (2) sequentially solving subproblems. The answer to the second subproblem is built on the answer to the first subproblem. The demonstration examples for each stage’s prompt are omitted in this illustration. Source: Zhou et al. [244]

Least-to-most prompting also achieves 99.7% accuracy on the SCAN¹⁰² compositional generalization benchmark with only 14 exemplars, compared to 16% with Chain-of-Thought prompting. Table 36 shows the accuracy comparison. Least-to-most improves performance on GSM8k and DROP benchmarks, particularly for problems requiring multiple solving steps. Table 37 shows the accuracy comparison.

Method	Code-davinci-002	Text-davinci-002	Code-davinci-001
Standard Prompting	16.7	6.0	0.4
Chain-of-Thought	16.2	0.0	0.0
Least-to-Most	99.7	76.0	60.7

Table 36: Accuracies of different prompting methods on the SCAN benchmark. Source: Zhou et al. [244]

Least-to-most prompting effectively generalizes to more complex problems than those seen in the prompts. This approach can be combined with other prompting techniques, such as chain-of-thought and self-consistency, to enhance performance further.

¹⁰²it is probably the most popular benchmark for evaluating compositional generalization. It requires mapping natural language commands to action sequences [47].

Method	Non-football (DROP)	Football (DROP)	GSM8k
Zero-Shot	43.86	51.77	16.38
Standard Prompting	58.78	62.73	17.06
Chain-of-Thought	74.77	59.56	60.87
Least-to-Most	82.45	73.42	62.39

Table 37: Accuracies of different prompting methods on GSM8k and DROP benchmarks. Source: Zhou et al. [244]

DECOMP is a text-based prompting strategy that decomposes complex tasks into simpler subtasks and generates a plan to solve the task, similar to Least-to-Most prompting. The core idea of Decomposed Prompting involves dividing a complex task into multiple simpler subtasks. Each subtask is addressed separately using LLMs, and their results are then combined to produce the final outcome. Tasks are decomposed based on their inherent structure. For instance, a question-answering task might be split into subtasks involving information retrieval, comprehension, and synthesis. The model can process each step more effectively by focusing on these individual components.

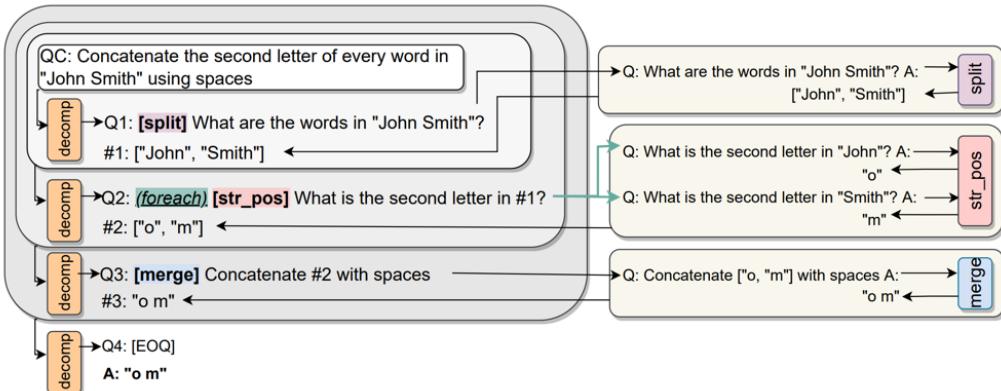


Figure 48: The DECOMP framework. Source: Khot et al. [175]

In DECOMP, the core is a decomposer LLM that tries to solve a complex task by generating a prompting program P . Each step of P directs a simpler sub-query to a function in an auxiliary set of sub-task functions F available to the system. Given a query Q whose answer is A , the program P is a sequence of the form $((f_1, Q_1, A_1), \dots, (f_k, Q_k, A_k))$ where A_k is the final answer predicted by P and Q_i is a sub-query directed to the sub-task function $f_i \in F$. P is executed by a high-level imperative controller, which passes the inputs and outputs between the decomposer and sub-task handler until a stopping condition in P is met and the final output is obtained. Using a software engineering analogy, the decomposer defines the top-level program for the complex task using interfaces to more straightforward sub-task functions. The sub-task handlers serve as modular, debuggable, and upgradable implementations of these simpler functions, akin to a software library. Specialized prompts are designed for each subtask, guiding the LLM to focus on specific aspects of the problem. This involves crafting precise and contextually relevant prompts that direct the model's attention to the desired task component.

Extensive experiments demonstrate the efficacy of Decomposed Prompting. Key benchmarks and datasets were utilized to evaluate the performance gains achieved through this approach (Figure 49).

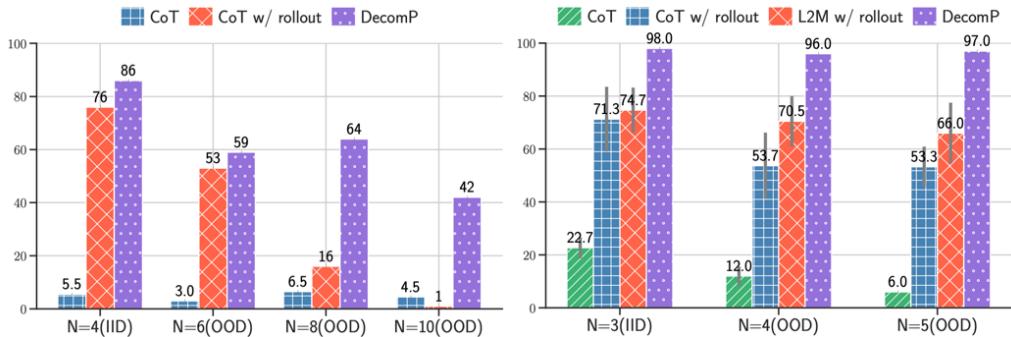


Figure 49: On the left: Exact Match results on the k -th letter concatenation task ($k=3$) using space as a delimiter with different numbers of words in the input. On the right: Exact Match results on reversing sequences. Incorporating CoT in DECOMP greatly increases the ability of the model to generalize to new sequence lengths Source: Khot et al. [175]

Program-Aided Language Models (PALMs) are a new class of code-based language models that use the LLM to read natural language problems and generate programs as the intermediate reasoning steps. Still, they offload the solution step to a runtime like a Python interpreter. These models are designed to perform complex reasoning tasks that require structured knowledge and logical reasoning, such as mathematical word problems, symbolic reasoning, and program synthesis. Despite LLMs seeming to be adept at CoT prompting, LLMs often make mathematical and logical errors, even though the problem is decomposed correctly into intermediate reasoning steps [164].

PAL is a model that belongs to this new class of models. It generates programs that can be executed by a Python interpreter and uses the program's output as the final answer. PAL has been shown to outperform much larger LLMs using CoT (e.g., PaLM-540B) on

Q: Olivia has \$23. She bought five bagels for \$3 each. How much money does she have left?

```
money_initial = 23
bagels = 5
bagel_cost = 3
money_spent = bagels * bagel_cost
money_left = money_initial - money_spent
answer = money_left
```

Figure 50: Example prompt for the mathematical reasoning tasks from the GSM8k benchmark. Source: Gao et al. [164]

mathematical word problems and symbolic reasoning tasks [164] as shown in Table 38. PAL

Model	GSM8k	GSM-HARD	SVAMP	ASDIV	SINGLEEQ	SINGLEOP	ADDSUB	MULTIARITH
DIRECTCodex	19.7	5.0	69.9	74.0	86.8	93.1	90.9	44.0
CoTUL2-20B	4.1	-	12.6	16.9	-	-	18.2	10.7
CoTLAMDA-137B	17.1	-	39.9	49.0	-	-	52.9	51.8
CoT_Codex	65.6	23.1	74.8	76.9	89.1	91.9	86.0	95.9
CoTPaLM-540B	56.9	-	79.0	73.9	92.3	94.1	91.9	94.7
CoTMinerva 540B	58.8	-	79.4	79.6	96.1	94.6	92.5	99.2
PAL	72.0	61.2	79.4	79.6	96.1	94.6	92.5	99.2

Table 38: Problem solve rate (%) on mathematical reasoning datasets. The highest number on each task is in **bold**. The results for DIRECT and PaLM-540B are from Wei et al. [230], the results for LAMDA and UL2 are from Wang et al. [227], the results for Minerva are from Lewkowycz et al. [182]. PAL ran on each benchmark 3 times and reported the average. Source: Gao et al. [164].

is even more effective with respect to other LLMs when tested on the GSM-HARD dataset

Q: On the table, you see a bunch of objects arranged in a row: a purple paperclip, a pink stress ball, a brown keychain, a green scrunchiephone charger, a mauve fidget spinner, and a burgundy pen. What is the color of the object directly to the right of the stress ball?

```
...
stress_ball_idx = None
for i, object in enumerate(objects):
    if object[0] == 'stress ball':
        stress_ball_idx = i
        break
# Find the directly right object
direct_right = objects[stress_ball_idx+1]
# Check the directly right object's color
answer = direct_right[1]
```

Figure 51: An example for a PAL prompt in the COLORED OBJECTS task. Source: Gao et al. [164]

- a version of GSM8k contains larger numbers (i.e., up to 7 digits). Other interesting results come from symbolic reasoning tasks from BIG-Bench Hard: the COLORED OBJECTS¹⁰³ and the PENGUINS¹⁰⁴ tasks as shown in Table 39. Gao et al. [164] have shown that PAL is not limited

Model	COLORED OBJECT	PENGUINS	DATE	REPEAT COPY	OBJECT COUNT-ING
DIRECT _{Codex}	75.7	71.1	49.9	81.3	37.6
CoT _{LAMDA-137B}	-	-	26.8	-	-
CoT _{PaLM-540B}	-	65.1	65.3	-	-
CoT _{Codex}	86.3	79.2	64.8	68.8	73.0
PAL _{Codex}	95.1	93.3	76.2	90.6	96.7

Table 39: Solve rate on three symbolic reasoning datasets and two algorithmic datasets. In all datasets, PAL achieves a much higher accuracy than chain-of-thought. Results with closed models LAMDA-137B and PaLM-540B are included if available to the public Wei et al. [230] and Suzgun et al. [218]. Source: Gao et al. [164].

to LMs of code. Still, it can work with LMs that were mainly trained for natural language if they have a sufficiently high coding ability. Benefits come from the synergy between the Python prompt and the interpreter. PAL avoids inaccuracy on arithmetic tasks and incorrect reasoning by offloading the calculations and some of the reasoning to a Python interpreter, which is correct by design, giving the right program.

SELF-PLANNING is a code-generation strategy using a planning-based approach. In this case, the planning is executed before the actual code generation, and the LLM itself generates the plan. In the first stage, the planning phase, the LLM is prompted to abstract and decompose the intent to obtain a plan for guiding code generation using few-shot prompting. The prompt C is designed as k examples¹⁰⁵ concatenated together

$$C \triangleq \langle x_1^e \cdot y_1^e \rangle \parallel \langle x_2^e \cdot y_2^e \rangle \parallel \dots \parallel \langle x_k^e \cdot y_k^e \rangle \quad (30)$$

where each example $\langle x_i^e \cdot y_i^e \rangle$ consists of the example intent x_i^e and its associated plan y_i^e to demonstrate the planning task. During inference, the test-time intent x will be concatenated

¹⁰³It requires answering questions about coloured objects on a surface

¹⁰⁴It requires to answer a question about the attributes of the penguins on a table (e.g., “how many penguins are less than 8 years old?”). This task describes dynamics as well since the penguins can be added or removed.

¹⁰⁵Note that k is a fairly low number.

after the prompt, and $C \| x$ will be fed into the LLM M , which will attempt to do planning for the test-time intent. The output of the LLM is the test-time plan y for the test-time intent x .

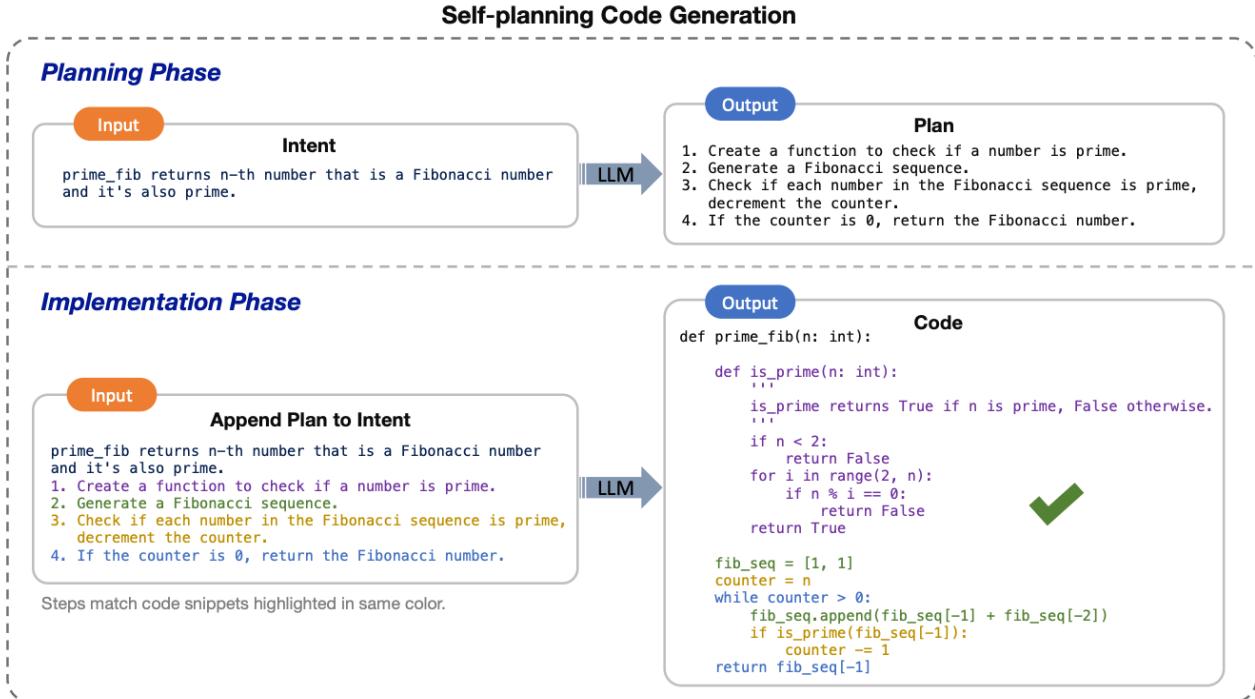


Figure 52: Self-planning generation phases (i.e., planning and implementation phases). Source: Jiang et al. [377]

In the second stage, the implementation phase, the plan generated in the first stage guides the code generation. The plan y is concatenated with intent x and fed into the LLM M to generate the code z . The above two stages can be formalized as

$$P(z|x, C) = \sum_{\hat{y}} P(z|\hat{y}, x, C) \cdot P(\hat{y}|x, C), \propto P(z|y, x, C) \cdot P(y|x, C) \quad (31)$$

where \hat{y} is any of all possible plans, and y denotes one of the plans generated by the LLM in the first stage. Jiang et al. [377] further simplifies the above equation by adopting the plan with the highest probability as y . Thus, the final equation becomes

$$P(z|x, C) \triangleq \underbrace{P(z|y, x, C)}_{\text{Implementation phase}} \cdot \underbrace{P(y|x, C)}_{\text{Planning phase}} \quad (32)$$

Benchmarking against various LLMs pre-trained on code, such as CodeGeex (13B) [366], CodeGen-Mono (16.1B) [203], and PaLM Coder (560B) [156], reveals that SELF-PLANNING significantly enhances performance across public code generation datasets. This improvement is observed when comparing SELF-PLANNING with other prompting methods, including Direct, Code Chain-of-Thought (CoT), and Few-shot approaches. Comparing the effectiveness of SELF-PLANNING relative to model size, it is evident that SELF-PLANNING impact is more pronounced with larger models. As the model size reaches 13B, LLMs' performance in code generation tasks begins to exhibit emerging ability, but self-planning ability is still relatively low. Experiments show that incorporating code training data and RLHF can enhance the model's self-planning capabilities and increase its size.

4.4.4 Feedback and plan refinement

Feedback is an essential component in the plan-based reasoning paradigm, as it allows the planner to refine the plan based on the feedback from the environment following the “*planning-execution-refinement*” loop. Feedback sources are categorized into internal and external, based on their origin relative to the LLM-based planner.

Internal Feedback: Here, the LLM acts as a feedback source. One common method is to assess the effectiveness of generated plans through structured prompts. For instance, Hao et al. [275] evaluates the success potential of various plans by estimating their likelihood of achieving the desired outcome. At the same time, Tree of Thoughts employs a comparative voting mechanism among different plans. Additionally, LLMs can refine their feedback using intermediate outcomes from plan execution, such as in Reflexion, where sparse outcomes like success or failure are translated into detailed, actionable feedback. This feedback is then preserved in the LLM’s long-term memory to enhance future planning.

External Feedback: Beyond the LLM, external tools and environments also contribute to feedback. Tools like code interpreters in programming tasks offer immediate error feedback, while models like stable diffusion in multimodal tasks provide visual feedback. Virtual environments like Minecraft offer a rich, interactive backdrop for feedback through immersive experiences. Moreover, projects like Generative Agents investigate the dynamics of multi-agent systems in simulated settings, where agents derive feedback from both environmental interactions and inter-agent communication.

Regarding the plan refinement, the three main approaches are summarized in the next paragraphs.

Reasoning. When the feedback data from the environment is not directly suitable to be utilized by LLMs for plan refinement, some work adds the explicit reasoning process to extract critical information from feedback [260, 236]. React prompts LLMs with demonstrations to generate reasoning traces over feedback. Human intelligence uniquely integrates task-oriented actions with verbal reasoning or “inner speech,” significantly contributing to cognitive functions like self-regulation and working memory management. For example, in the kitchen, a person might verbally strategize their next steps in a recipe (“Now that everything is cut, I should heat up the pot of water”), adapt to missing ingredients (“I don’t have salt, so let me use soy sauce and pepper instead”), or seek additional information online to enhance their cooking process. This ability to blend action with analytical thinking enables humans to swiftly learn new tasks and make robust decisions, even in novel or uncertain situations. React has been widely used in autonomous agent projects, such as AutoGPT, which can automatically reason over the observed feedback to revise the initial plan for solving various user requests. However, these approaches typically fix the order of reasoning and planning.

ChatCoT supports flexible switching between the two processes, unifying the tool-augmented CoT reasoning framework into a multi-turn conversation between the LLM-based task planner and the tool-based environment. At each turn, the LLM can freely interact with tools when needed; otherwise, it performs the reasoning by itself.

Backtracking. Initial planning techniques primarily focused on progressing with forward actions within an existing plan, often resulting in locally optimal strategies based on short-term assessments. To address this limitation, the Tree of Thoughts approach [359] introduces the capability for backtracking through search techniques such as breadth-first and depth-first searches, enabling more comprehensive global planning strategies. This method iteratively refines the plan by returning to previous decision points and exploring alternative paths as depicted in Figure 55.

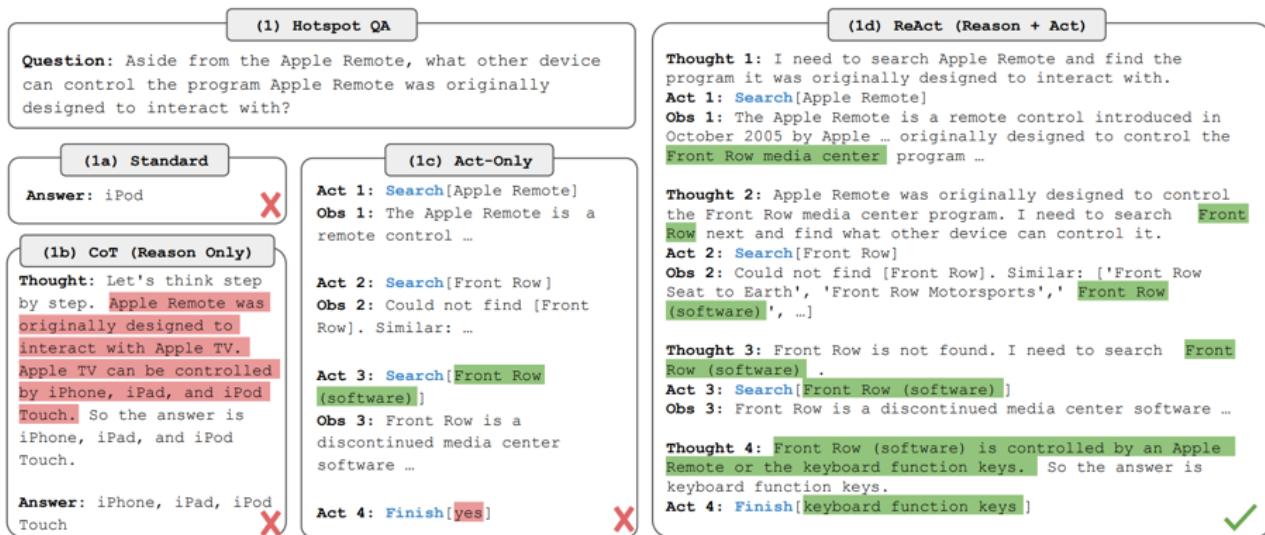


Figure 53: (1) Comparison of 4 prompting methods, (a) Standard, (b) Chain-of-thought (CoT, Reason Only), (c) Act-only, and (d) ReAct (Reason+Act), solving a HotpotQA [60] question. Source: Chen et al. [260]

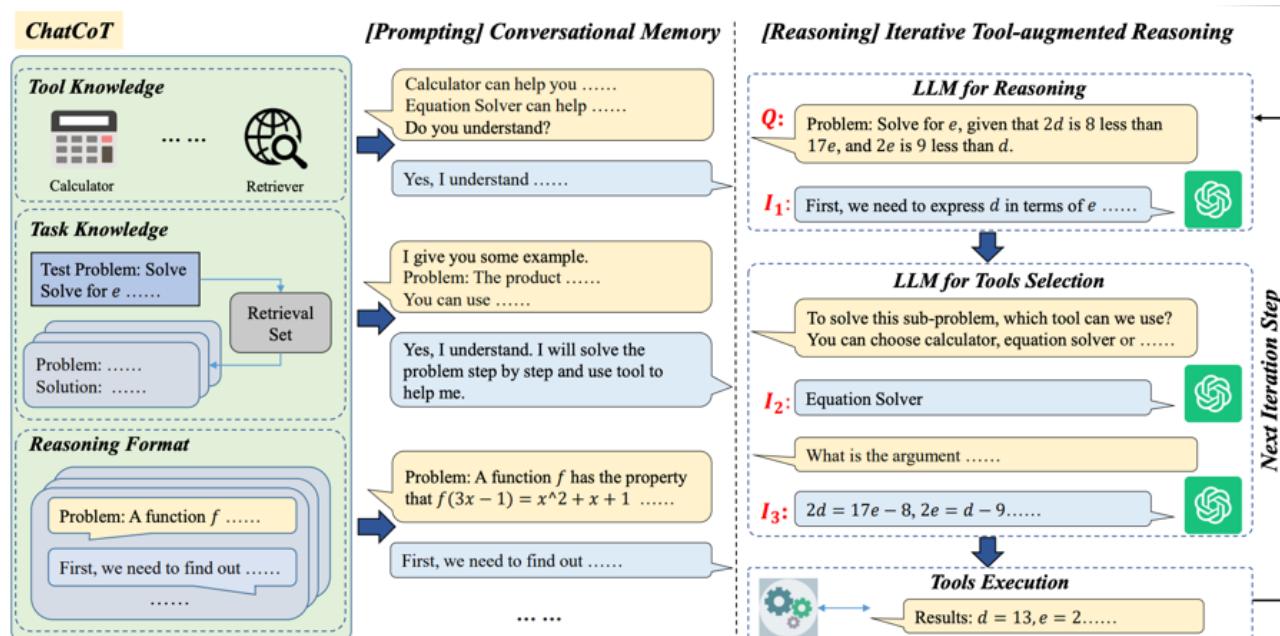


Figure 54: ChatCoT strategy illustrated to solve a mathematical problem. The conversational knowledge memory is initialized to provide tools, task and reasoning format knowledge. Then, the tool-augmented reasoning step is iterated multiple times to perform step-by-step reasoning until the answer is obtained. Source: Chen et al. [260]

In developing such a method, Yao et al. [359] revisits foundational artificial intelligence and cognitive science principles, framing problem-solving as navigating a tree-like combinatorial space. Within this framework, Yao et al. [359] introduced three novel challenges aimed at pushing the boundaries of state-of-the-art models such as GPT-4: the Game of 24^{106} , Creative

¹⁰⁶The Game of 24 is a mathematical challenge where the objective is to manipulate four numbers using basic arithmetic operations $+ - \times \div$ to achieve a result of 24. For instance, from the numbers 4, 9, 10, 13, one possible solution could be $(10 - 4) \times (13 - 9) = 24$.

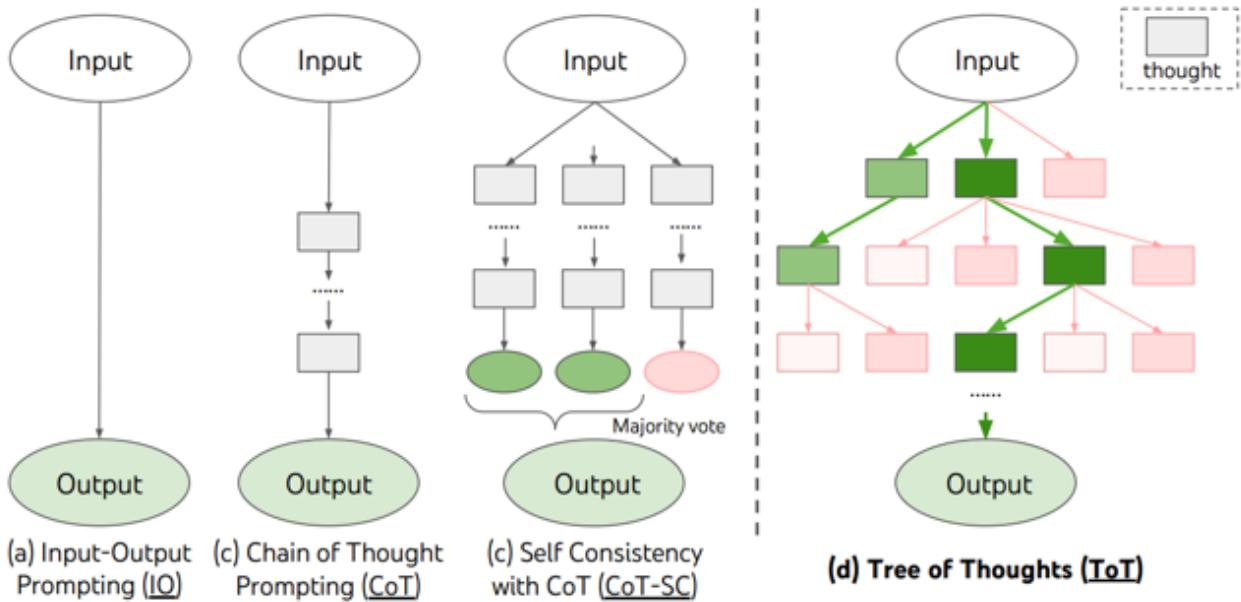


Figure 55: Diagram demonstrating various problem-solving methodologies using LLMs. Each rectangle represents a distinct thought, forming an integral step towards resolving a problem. Source: Yao et al. [359]

Wiring¹⁰⁷, and Crosswords¹⁰⁸. These tasks necessitate a blend of deductive, mathematical, commonsense, and lexical reasoning skills, along with sophisticated systematic planning or searching capabilities. The Tree of Thoughts model demonstrates its versatility and efficacy across these diverse tasks by supporting varied levels of thought processes, multiple thought generation and assessment methods, and adaptable search algorithms tailored to the specifics of each challenge.

Furthermore, some studies [168, 344] utilize feedback signals to revise the entire plan since the initial plan generated by the LLM is often imperfect. For example, DEPS¹⁰⁹ [344] selects a better plan according to feedback signals, while TIP¹¹⁰ [301] adds feedback signals to prompts for the LLM-based planner to revise each step in the initial plan.

DEPS has been tested on Minecraft, an open world with abundant object types and complex dependencies and relations. As a result, ground-truth plans typically involve a long sequence of sub-goals with strict dependencies (e.g., obtaining a diamond requires 13 sub-goals with strict dependencies). Another challenge in an open-ended world is the feasibility of the produced plans. For example, the fastest way to craft a bed in Minecraft is to slaughter a sheep to obtain wool, which can be used to craft beds or collect beds from a village. However, since no sheep or village is reachable by the agent within 3 minutes of gameplay, to craft a bed efficiently, the agent should choose to slaughter a spider and use materials (e.g., string) it drops to craft wool, and then a bed. The key to solving the first challenge is effectively adjusting the generated plan upon a failure. When the controller fails to complete a sub-goal, a descriptor will summarize the current situation as text and send it back to the LLM-based planner. Then, prompt the LLM

¹⁰⁷In the Creative Wiring task, participants are given four random sentences and must craft a coherent narrative consisting of four paragraphs, each concluding with one of the provided sentences. This task tests creative synthesis and advanced planning.

¹⁰⁸A 5×5 mini crosswords task is a harder search problem involving natural language. The goal is not to solve the problem since it can be solved with specialized NLP pipelines but to explore the limit of LM as a general-purpose solver.

¹⁰⁹Describe, Explain, Plan, Select

¹¹⁰Text-Image Prompting

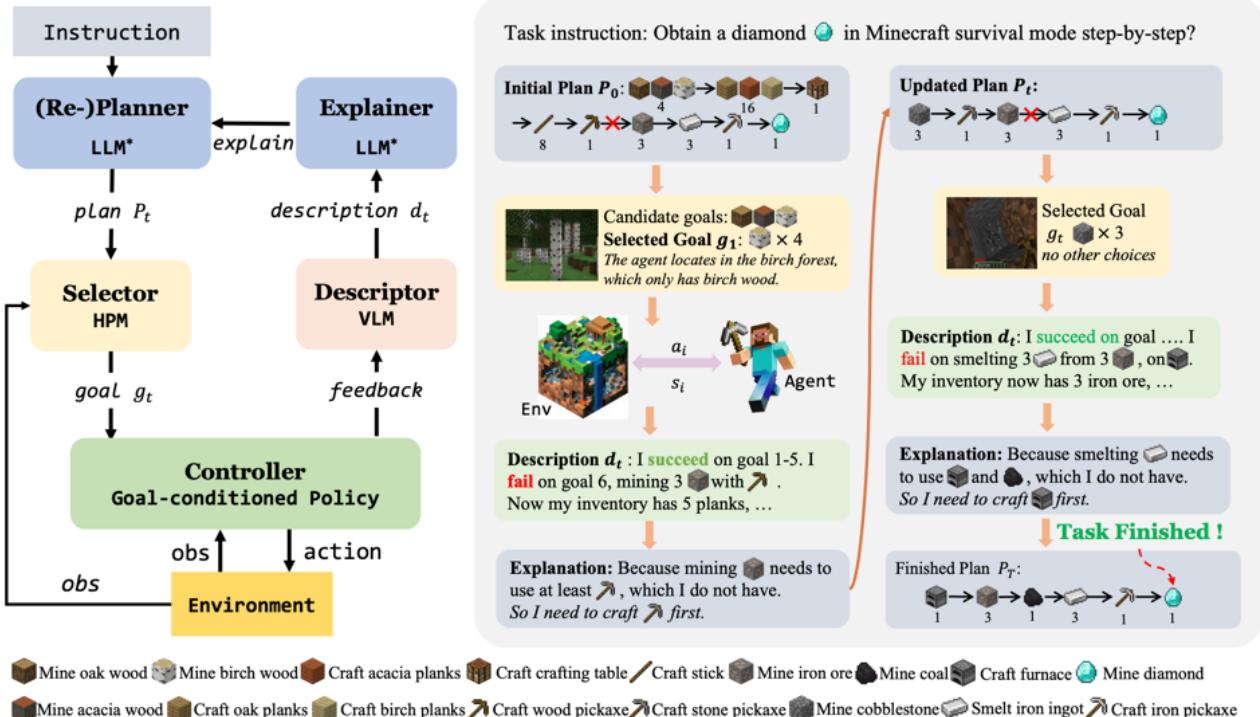


Figure 2: Overview of our proposed interactive planner architecture.

Figure 56: Overview of the DEPS interactice plannet architecture. Source: Wang et al. [344]

as an explainer to locate the errors in the previous plan. Finally, a planner will refine the plan using the descriptor and explainer information. To improve the feasibility of generated plans conditioned on the current state, which is the second identified challenge, Wang et al. [344] use a learned goal-selector to choose the most accessible sub-task based on the proximity to each candidate sub-goal. Developing multi-task agents that can accomplish a vast and diverse suite of tasks in complex domains has been considered a key milestone towards generally capable artificial intelligence.

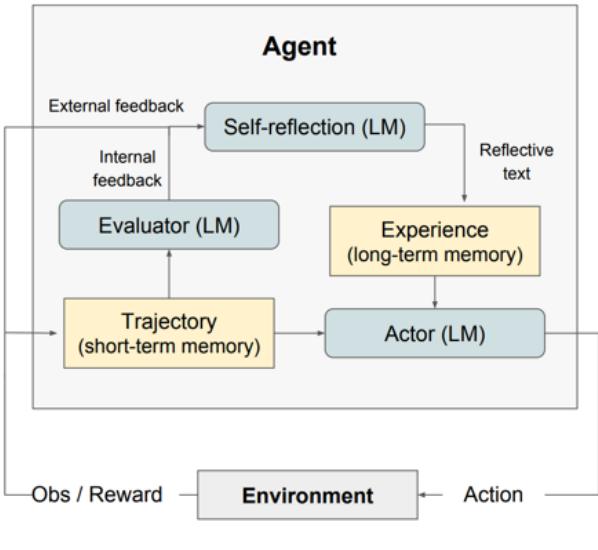
Memorization Long-term memory is a crucial component in the planning process. It allows models to store and retrieve information from past experiences and the short-term memory capabilities provided by in-context learning (ICL) in large language models (LLMs). Reflexion [322] introduces an innovative framework that enhances language agents through linguistic feedback rather than weight updates. Reflexion agents reflect verbally on task feedback, maintaining reflective text in an episodic memory buffer to improve decision-making in subsequent trials. This process mirrors how humans iteratively learn complex tasks by reflecting on previous failures to develop improved strategies for future attempts.

Reflexion can incorporate various types (scalar values or free-form language) and sources (external or internally simulated) of feedback signals, significantly improving performance over a baseline agent across diverse tasks such as sequential decision-making, coding, and language reasoning.

The Reflexion framework consists of four main components: the Actor, the Evaluator, the Self-Reflection model, and the memory. The Actor, built upon an LLM, is specifically prompted to generate necessary text and actions based on state observations. The Evaluator assesses the quality of the Actor’s outputs by computing a reward score that reflects performance within the given task context. The Self-Reflection model, also instantiated as an LLM, generates verbal self-reflections to provide valuable feedback for future trials. Core components of the Reflexion

	1. Decision making	2. Programming	3. Reasoning
(a) Task	You are in the middle of a room [...] Task: clean some pan and put it in countertop.	Task: You are given a list of two strings [...] of open '(' or close ')' parentheses only [...]	Task: What profession does John Lanchester and Alan Dean Foster have in common?
(b) Trajectory	[...] Action: take pan1 from stoveburner1 Obs: Nothing happens. [...] Action: clean pan1 with sinkbasin1 Obs: Nothing happens. [...]	def match_parens(lst): if s1.count('(') + s2.count('(') == s1.count(')') + s2.count(')'): [...] return 'No'	Think: [...] novelist, journalist, critic [...] novelist, screenwriter [...] common is novelist and screenwriter. Action: "novelist, screenwriter"
(c) Evaluation (internal / external)	Rule/LM Heuristic: Hallucination.	Self-generated unit tests fail: assert match_parens(...)	Environment Binary Reward: 0
(d) Reflection	[...] tried to pick up the pan in stoveburner 1 [...] but the pan was not in stoveburner 1. [...]	[...] wrong because it only checks if the total count of open and close parentheses is equal [...] order of the parentheses [...]	[...] failed because I incorrectly assumed that they both had the same multiple professions [...] accurately identifying their professions.
(e) Next Trajectory	[...] Action: take pan 1 from stoveburner 2 [...] Obs: You put the pan 1 in countertop 1.	[...] return 'Yes' if check(S1) or check(S2) else 'No'	Think: [...] So the profession John Lanchester and Alan Dean Foster have in common is novelist. Action: "novelist"

Figure 57: Reflexion works on decision-making, programming, and reasoning tasks. Source: Shinn et al. [322]



Algorithm 1 Reinforcement via self-reflection

```

Initialize Actor, Evaluator, Self-Reflection:  

 $M_a, M_e, M_{sr}$   

Initialize policy  $\pi_\theta(a_i|s_i), \theta = \{M_a, mem\}$   

Generate initial trajectory using  $\pi_\theta$   

Evaluate  $\tau_0$  using  $M_e$   

Generate initial self-reflection  $sr_0$  using  $M_{sr}$   

Set  $mem \leftarrow [sr_0]$   

Set  $t = 0$   

while  $M_e$  not pass or  $t < \text{max trials}$  do  

    Generate  $\tau_t = [a_0, o_0, \dots, a_i, o_i]$  using  $\pi_\theta$   

    Evaluate  $\tau_t$  using  $M_e$   

    Generate self-reflection  $sr_t$  using  $M_{sr}$   

    Append  $sr_t$  to  $mem$   

    Increment  $t$   

end while  

return
  
```

Figure 58: (a) Diagram of Reflexion. (b) Reflexion reinforcement algorithm. Source: Shinn et al. [322]

process are the notion of short-term and long-term memory. At inference time, the Actor conditions its decisions on short and long-term memory, similar to how humans remember fine-grain recent details while also recalling distilled meaningful experiences from long-term memory. In the RL setup, the trajectory history serves as the short-term memory, while outputs from the Self-Reflection model are stored in long-term memory. These two memory components work together to provide specific context. Still, they are also influenced by lessons learned over several trials, a key advantage of Reflexion agents over other LLM action choice works. Given a sparse reward signal, such as a binary success status (success/fail), the current trajectory, and its persistent memory mem , the self-reflection model generates nuanced and specific feedback. This feedback, which is more informative than scalar rewards, is then stored in the agent's memory mem . For example, in a multi-step decision-making task, if the agent receives a failure signal, it can infer that a specific action a_i led to subsequent incorrect actions a_{i+1} and

a_{i+2} . The agent can then verbally state that it should have taken a different action, a_i , which would have resulted in correct actions a_{i+1} and a_{i+2} , and store this experience in its memory. In subsequent trials, the agent can leverage past experiences to adapt its decision-making approach at time t by choosing action a_i . This iterative process of trial and error, self-reflection, and persisting memory enables the agent to rapidly improve its decision-making ability in various environments by utilizing informative feedback signals. For instance, Reflexion achieves a 91% pass@1 accuracy on the HumanEval coding benchmark, surpassing the previous state-of-the-art GPT-4, which achieves 80%.

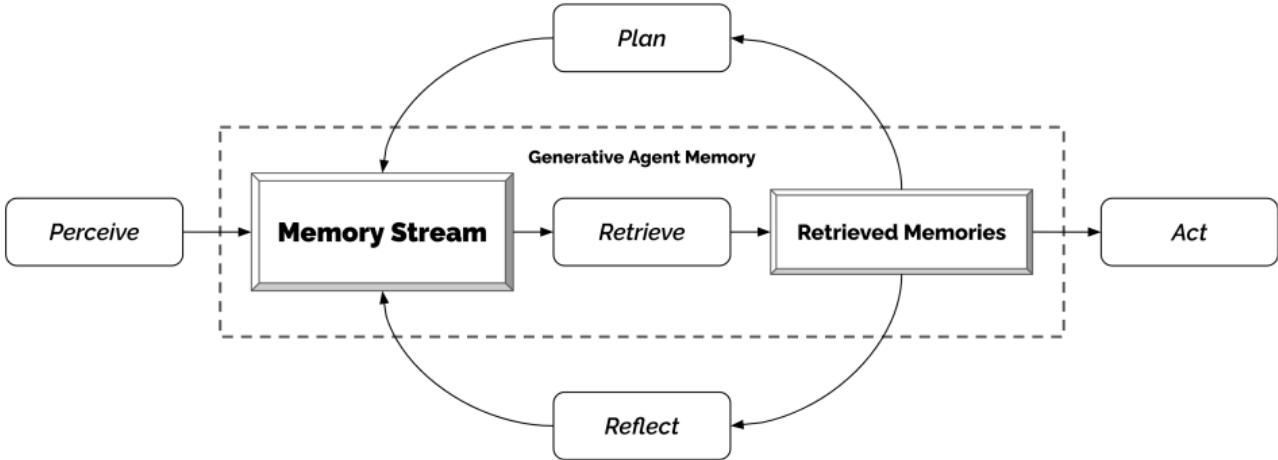


Figure 59: Generative agent architecture. Agents perceive their environment, and all perceptions are saved in a comprehensive record of the agent’s experiences called the memory stream. Based on their perceptions, the architecture retrieves relevant memories and uses those retrieved actions to determine an action. These retrieved memories are also used to form longer-term plans and create higher-level reflections, both of which are entered into the memory stream for future use. Source: Park et al. [310]

Generative agents [310] are another example of models that leverage memory to improve planning where a sandbox environment is populated with 25 agents that focus on the ability to create a small, interactive society of agents inspired by games such as The Sims. In particular, the generative agents leverage a memory stream mechanism for action planning and reflection, simulating human-like decision behaviour. The memory stream is a long-term memory module that records a comprehensive list of the agent’s natural language experiences. The reflection and the planning components synthesize memories into higher-level inferences over time, enabling the agent to draw conclusions about itself and others, and recursively translates those conclusions and the current environment into high-level action plans, as shown in Figure 59.

Other studies [324, 337] have also explored using memory called skill library mechanism to store successful plans, which can be reused and synthesized as complex plans for new tasks. AdaPlanner [324] uses skill memory as a repository, archiving past successful plans and their respective interactions with the environment. If the agent encounters a task resembling the skills stored in memory, these skills can serve as few-shot exemplars in the LLM agent’s prompt. This feature improves not only sample efficiency but also reliability for future planning. To implement the long-term memory, Wang et al. [337] and Wang et al. [138] proposes tools like vector databases, which can store plans or feedback into high-dimensional vectors.¹¹¹ Memory-

¹¹¹A vector database is a type of database engineered specifically for handling vector data, which are arrays of numbers or embeddings representing various types of data objects. These databases are designed to efficiently store, manage, and perform operations on vectors, often used to represent images, text, or other complex data types in a form suitable for machine learning models and similarity search operations. Vector databases excel in handling similarity searches, which involve finding vectors closest to a given vector. They are optimized to store and query high-dimensional data efficiently.

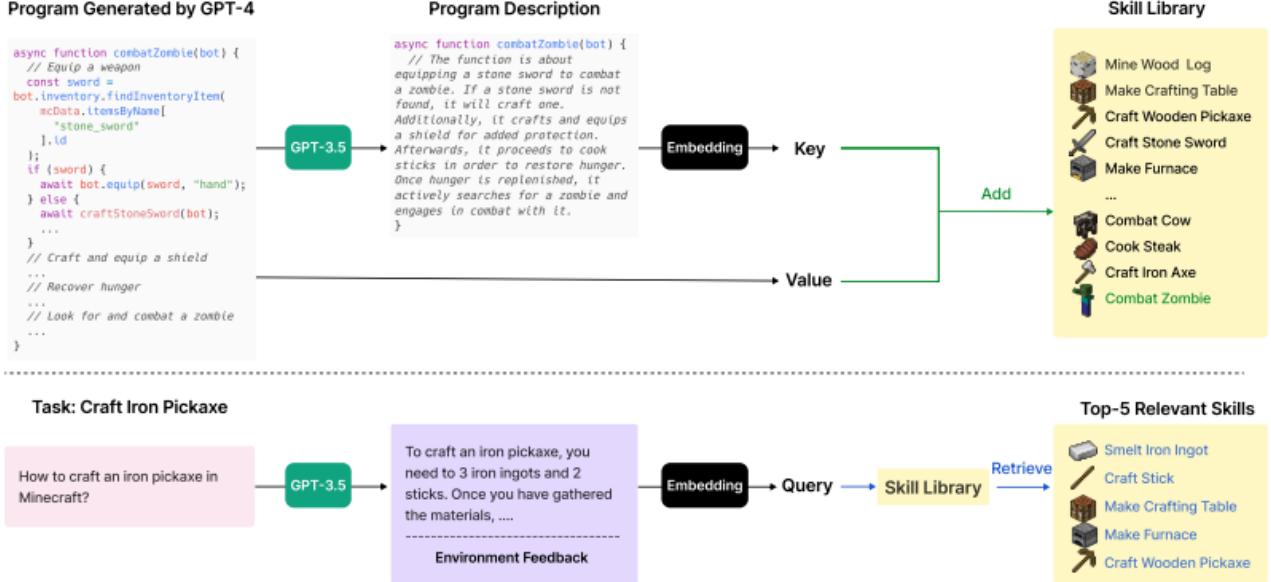


Figure 60: Adding and retrieving skills from the skill library in Voyager. Source: Sun et al. [324]

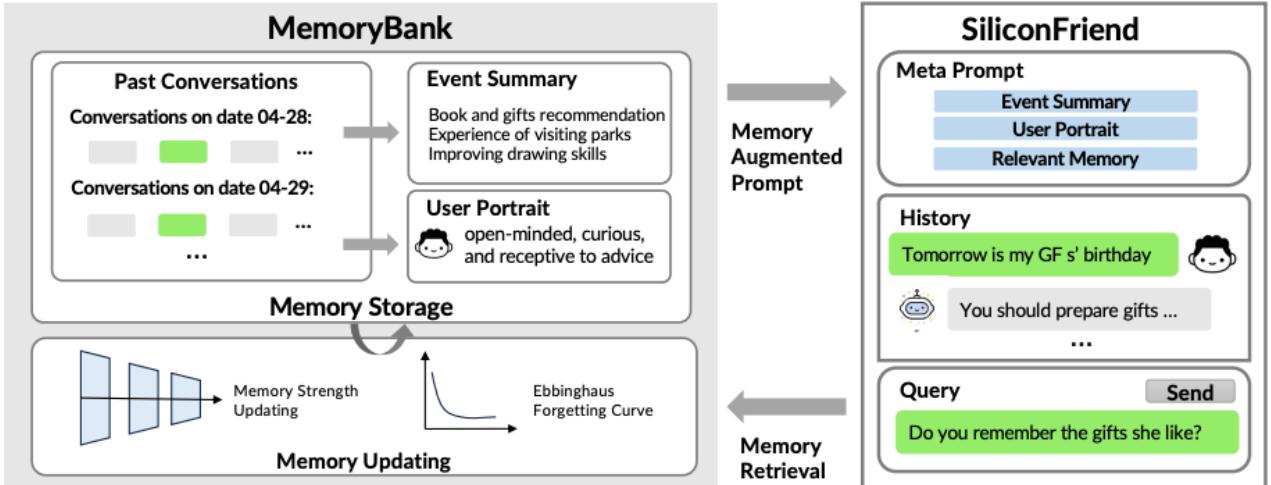


Figure 61: Overview of MemoryBank. The memory storage stores past conversations, summarized events and user portraits, while the memory updating mechanism updates the memory storage. Memory retrieval recalls relevant memory. Source: Zhong et al. [367]

Bank [367] incorporates a memory updating mechanism inspired by the Ebbinghaus Forgetting Curve theory.¹¹² This mechanism allows the model to forget less relevant information and retain more important information based on time elapsed and relative relevance, thereby offering a human-like memory management system.

4.4.5 LLM-modulo Framework

LLM-modulo framework are a novel approach to planning that combines the strengths of LLMs with the modularity of traditional planning systems.

The reasons behind the development of the LLM-modulo framework are manifold. Kamb-

¹¹²The Ebbinghaus Forgetting Curve is a psychological theory that describes how information is lost over time when there is no attempt to retain it. It shows that humans tend to halve their memory of newly learned knowledge in days or weeks unless they consciously review the learned material.

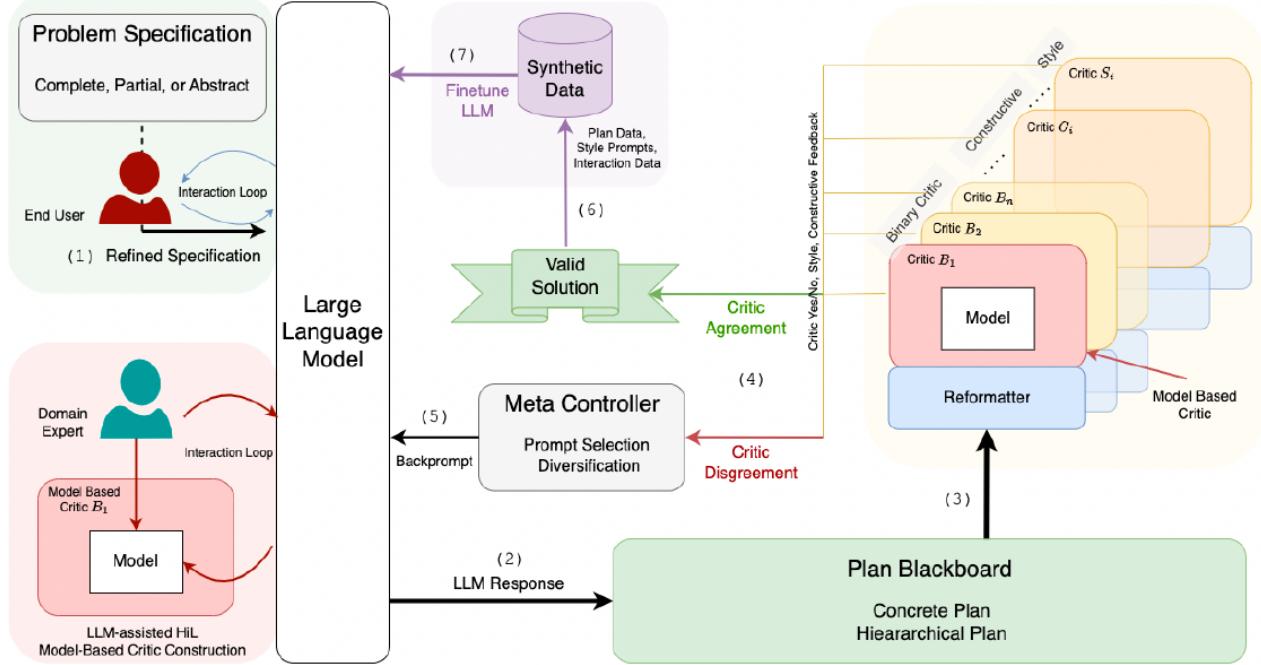


Figure 62: LLM-Modulo framework where LLMs act as idea generators and various external critics that specialize in different aspects, critique the candidate plan. Source: Kambhampati et al. [379]

hampati et al. [379] argue that auto-regressive large language models (LLMs) lack the ability to independently plan and self-verify, which are essential aspects of reasoning. Despite being powerful tools trained on vast amounts of data, LLMs function more like advanced n-gram models, excelling in linguistic tasks but falling short in structured reasoning and planning. LLMs are akin to Kahneman’s “System 1” — fast, intuitive, and associative, but not capable of the deliberate, logical thinking attributed to “System 2”. They are better at retrieving information and making analogies than performing structured planning or self-critique. A close examination of several works claiming planning capabilities for LLMs [283] suggests that they either work in domains/tasks where subgoal interactions can be safely ignored, or by delegating the interaction resolution to the humans in the loop (i.e., repeating prompts until the LLM generates a plan that the human finds acceptable¹¹³). For instance, LLMs are shown to be poor at both generating and verifying solutions for tasks such as graph coloring, and fine-tuning them does not significantly improve their planning abilities [18]. On the contrary, “self-critiquing” methods, where LLMs generate and critique their own solutions, are not effective, as LLMs struggle to verify solutions effectively and the performance are even worse than the direct generation¹¹⁴. As a result of not being good at self-critique their plans, LLMs can’t self-improve by generating and refining their data, contrary to some claims in the literature[379].

While LLMs can generate candidate plans, these plans are often not executable without errors as shown in Table 40. This demonstrates that LLMs are more effective when used in combination with external verification systems in frameworks like the LLM-Modulo Frameworks, where they serve as approximate knowledge sources rather than independent planners. The LLM-modulo framework is a hybrid approach that combines the strengths of LLMs with the modularity of traditional planning systems (see Figure 62). In this framework, LLMs act as idea generators, while various external critics that specialize in different aspects critique the candidate plan. Critics can evaluate LLM-generated candidate plans over hard and soft con-

¹¹³It implies that the human already knows the answer, and the Clever Hans effect is a potential issue.

¹¹⁴The reason is that the system cannot recognise the correct colouring-generated answers.

Domain	Method	GPT-4o	GPT-4-Turbo	Claude-3-Opus	LLaMA-3 70B	Gemini Pro	GPT-4
Blocksworld (BW)	One-shot	170/600 (28.33%)	138/600 (23%)	289/600 (48.17%)	76/600 (12.6%)	68/600 (11.3%)	206/600 (34.3%)
	Zero-shot	213/600 (35.5%)	241/600 (40.1%)	356/600 (59.3%)	205/600 (34.16%)	3/600 (0.5%)	210/600 (34.6%)
	One-shot	5/600 (0.83%)	5/600 (0.83%)	8/600 (1.3%)	15/600 (2.5%)	2/500 (0.4%)	26/600 (4.3%)
	Zero-shot	0/600 (0%)	1/600 (0.16%)	0/600 (0%)	0/600 (0%)	0/500 (0%)	1/600 (0.16%)
Mystery BW (Deceptive)	One-shot	5/600 (0.83%)	5/600 (0.83%)	8/600 (1.3%)	15/600 (2.5%)	2/500 (0.4%)	26/600 (4.3%)
	Zero-shot	0/600 (0%)	1/600 (0.16%)	0/600 (0%)	0/600 (0%)	0/500 (0%)	1/600 (0.16%)

Table 40: Results of state-of-the-art LLMs GPT-4o, GPT-4-Turbo, Claude-3-Opus, Gemini Pro, and LLaMA-3 70B for Plan Generation with prompts in natural language (PlanBench). Source: Kambhampati et al. [379]

straints. Hard constraints refer to correctness verification which can include causal correctness, timeline correctness, resource constraint correctness as well as unit tests. On the other hand, soft constraints can include more abstract notions of good form such as style, explicability, preference conformance, etc. LLMs cannot take on the role of the hard critics with soundness guarantees, they can help simulate some aspects of the soft critics. The banks of critics evaluate the current plan candidate about its fitness and acceptability. If all the hard critics accept the plan, the plan is considered a valid solution to be returned to the user or the executor. When the critics reject the plan, it can provide various level of feedback including alternative plans, partial plans, or even just the reasons for rejection. One way of obtaining the critics is to use partial planner, operating on either the model itself or their relaxed versions[13]. LLMs can also be used as Reformulators, since model-based verifiers tend to be operating on specialized formal representations. Reformulators module attached to critics can convert the plan into a form that can be evaluated by the critics, a thing that LLMs are good at [126]¹¹⁵. The Meta (Backprompt) Controller is responsible for coordinating the interaction between the LLM and the critics, especially in presence of a mix of hard and soft critics. Controller can assume the responsibility of compiling critics' feedback into a coherent form that can be used to guide the LLM in generating the next candidate plan (e.g., from a simple round-robin prompt selection to a LLM summarized prompt). Humans are involved once per domain and once per problem, acquiring the domain model with the help of the LLM (e.g., teasing out PDDL planning models from LLMs) [271]. Once the model is acquired, this way it can be used by correctness verifiers such as VAL [7, 271]. Often the planning problems in real world situations are specified incompletely, leaving it to the human commonsense to refine the specification. This brings up a second role for humans—this time end users. Basically, the LLM-modulo framework remove the restriction on the expressiveness of the planning language, allowing the LLM to generate plans in natural language, and the critics to evaluate them in a more formal language. Applying the framework to classical planning domains [333] and recent travel planning benchmark [376] show that with back prompting from VAL acting as the external verifier and critic, LLM performance in Blocks World improves to 82% within 15 back prompting rounds, while in Logistics, it improves to 70%. LLM-Modulo doesn't help as much in an obfuscated version of blocks world called Mystery BW, reaching about 10% accuracy. This should be expected because the LLMs have difficulty generating plausible candidate plans for this domain (note that even here,

¹¹⁵Indeed some approaches to combine LLMs with external symbolic solvers just use LLMs as reformulators

if a plan is returned, it must have passed muster with VAL, and is thus guaranteed correct by its model). For the travel planning case study [376], Kambhampati et al. [379] adapted the

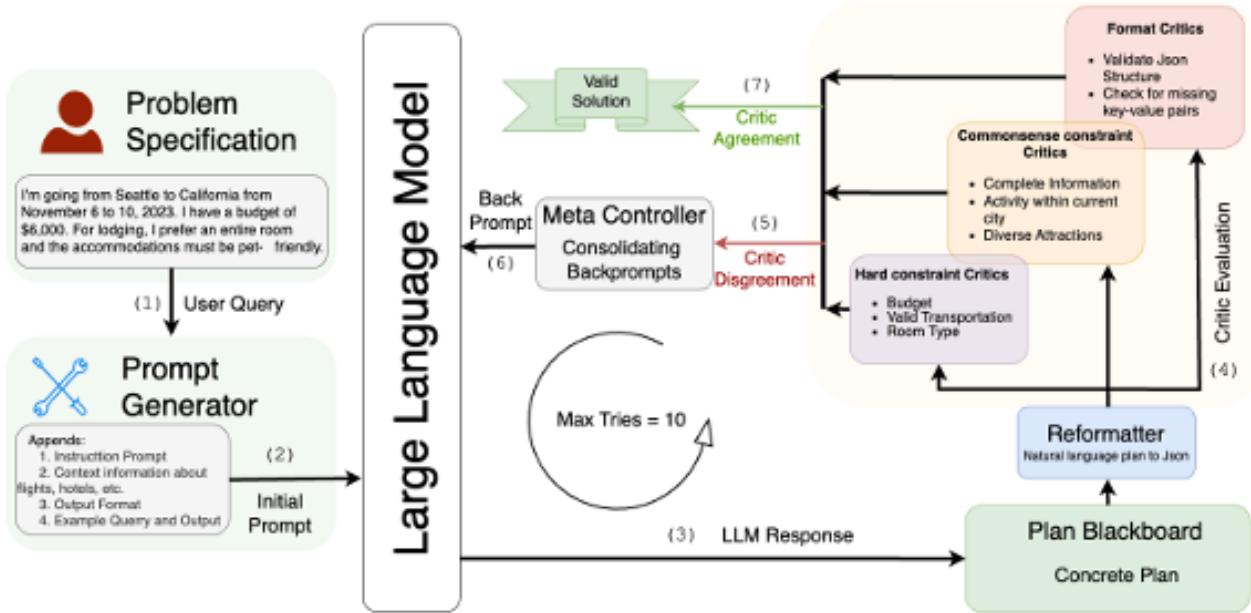


Figure 63: LLM Modulo Framework adapted for Travel Planning. Source: Kambhampati et al. [379]

LLM-Modulo framework to this benchmark by operationalizing hard constraints (e.g., budget constraints) or commonsense constraints (e.g., suggesting diverse attractions to visit) as critics as shown in Figure 63. The LLM-modulo approach improved of $6 \times$ the startlingly low 0.7% baseline, achieved in Gundawar et al. [376] by using LLMs planners with different prompting techniques, such as Cot and ReAct (as shown in Figure 64). Furthermore, authors also find that LLMs reliably play the role of hard critics and several commonsense critics, as well as the reformatter role (i.e., converting free form travel plans into structured plans parseable by the critics for back-prompts or plan evaluation). In this domain the LLM was able to enumerate the type of critics that are needed to validate the plan, with little human supervision.

4.5 Retrieval-Augmented Generation

(RAG) is an innovative paradigm designed to enhance the capabilities of large language models (LLMs) [375]. By integrating retrieval systems with generative models, RAG addresses some of the most pressing challenges in LLMs, including hallucinations, outdated knowledge, and untraceable reasoning processes. Gao et al. [375] delves into the evolution of RAG frameworks, the components that constitute these systems, and the metrics used for their evaluation.

RAG merges the intrinsic generative abilities of LLMs with external retrieval mechanisms, creating a synergy that enhances knowledge-intensive tasks.

This framework offers the following core advantages:

1. **Enhanced Knowledge Integration:** By querying external databases, RAG systems continuously update their knowledge base, addressing the limitations of static pre-trained models.
2. **Improved Accuracy:** Retrieved data serves as contextual grounding, reducing hallucinations and increasing the factual reliability of generated outputs.

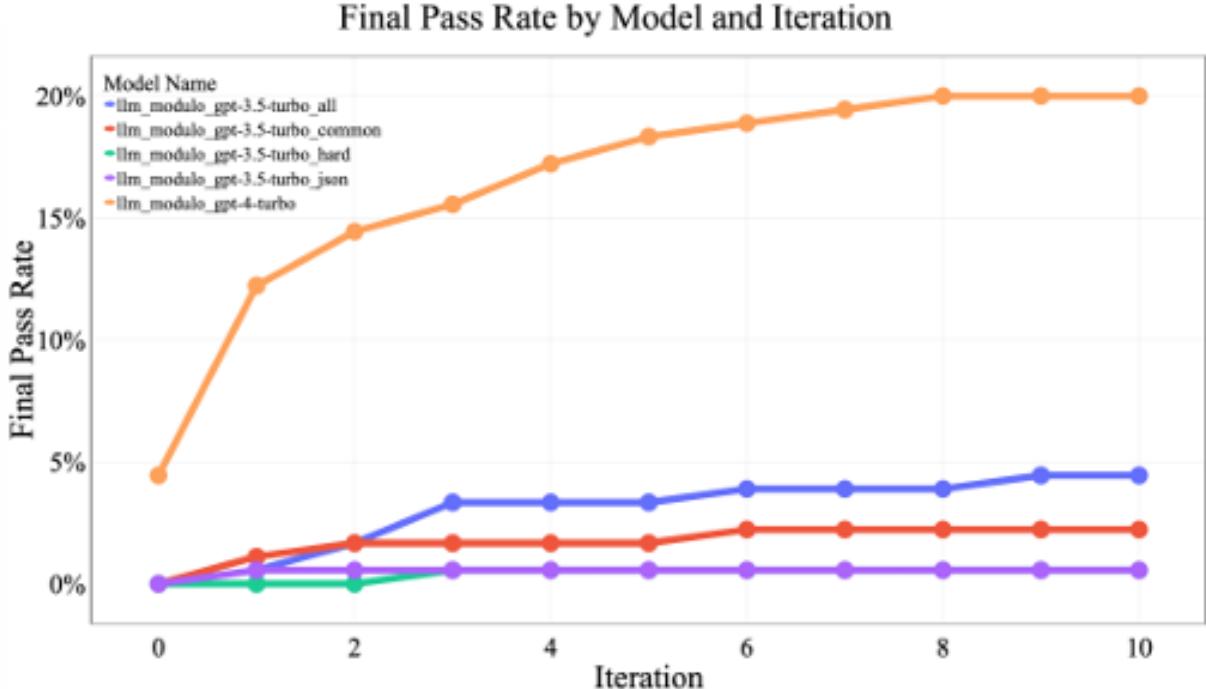


Figure 64: Final Pass rates of models across LLM Modulo Iterations. Source: Kambhampati et al. [379]

3. **Domain Adaptability:** RAG enables LLMs to integrate domain-specific information, improving performance in specialized areas like law, medicine, and engineering.

RAG systems are categorized into three main paradigms:

1. **Naïve RAG:** it was the first iteration of RAG systems. It follows the traditional pipeline of indexing, retrieval, and generation, which is also characterized as a “Retrieve-Read” framework [304]. This approach is simple and effective but suffers notable drawbacks in terms of retrieval precision (e.g., missing crucial information) and generation accuracy (e.g., allowing for hallucinations, toxicity or bias).
2. **Advanced RAG:** it introduces specific improvements to address the limitations of Naïve RAG. About retrieval quality, it employs pre-retrieval and post-retrieval strategies to enhance the relevance of retrieved data. For indexing, it uses more sophisticated techniques like sliding window approach, fine-grained segmentation and metadata. It incorporates additional optimization techniques to streamline the retrieval process [280].
3. **Modular RAG:** this architecture advances beyond previous RAG paradigms (Naive and Advanced RAG) by offering greater adaptability, flexibility, and functionality. It introduces new components and interaction patterns to address the challenges of static and rigid retrieval-generation frameworks, making it suitable for diverse tasks and dynamic scenarios. Modular RAG incorporates specialized modules to enhance retrieval and generation:
 - Search Module: Supports direct searches across diverse data sources such as databases, search engines, and knowledge graphs using LLM-generated queries [303].
 - RAGFusion: Implements multi-query strategies for diverse perspectives, utilizing parallel searches and re-ranking for knowledge discovery [320].

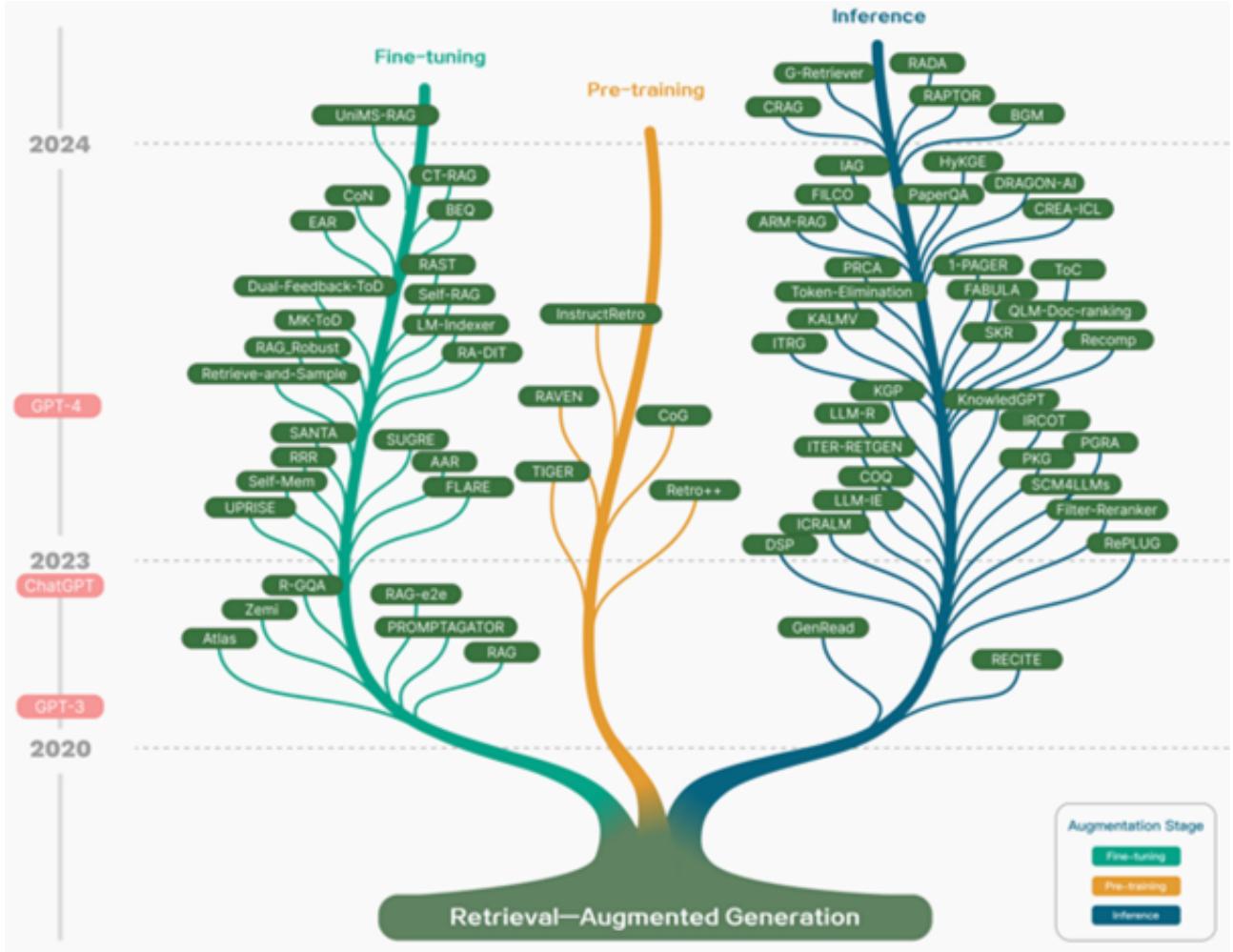


Figure 65: Technology tree of RAG research. The stages of involving RAG mainly include pre-training, fine-tuning, and inference. With the emergence of LLMs, research on RAG initially focused on leveraging the powerful in context learning abilities of LLMs, primarily concentrating on the inference stage. Subsequent research has delved deeper, gradually integrating more with the fine-tuning of LLMs. Researchers have also been exploring ways to enhance language models in the pre-training stage through retrieval-augmented techniques. Source: Gao et al. [375]

- Memory Module: Uses LLM memory to iteratively align retrieval processes with data distribution and enable unbounded memory pools [262].
- Routing Module: Dynamically selects pathways (e.g., summarization or database querying) to ensure optimal information retrieval and merging [365].
- Predict Module: Reduces redundancy and enhances context relevance by generating content directly via the LLM [238].
- Task Adapter Module: Adapts RAG to downstream tasks, automating prompt retrieval for zero-shot scenarios and enabling task-specific retrievers through few-shot learning [388, 250].

These enhancements enable precise and relevant information retrieval for a wide range of applications, improving retrieval efficiency and task-specific flexibility. The architecture introduces new patterns of interaction and flexibility in module orchestration:

- Rewrite-Retrieve-Read: Enhances retrieval queries through LLM-based query rewriting.

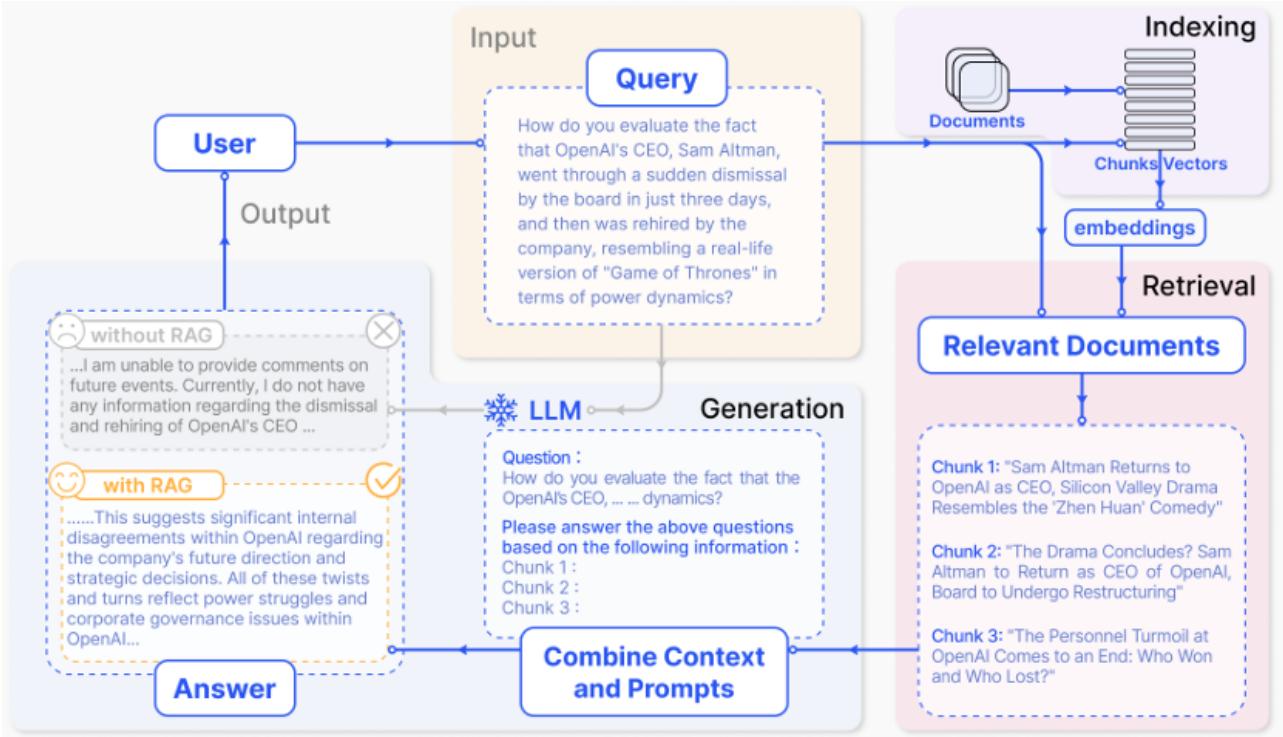


Figure 66: Retrieval-Augmented Generation (RAG) Framework mainly consists of 3 steps. 1) Indexing. Documents are split into chunks, encoded into vectors, and stored in a vector database. 2) Retrieval. Retrieve the Top k chunks most relevant to the question based on semantic similarity. 3) Generation. Input the original question and the retrieved chunks together into LLM to generate the final answer. Source: Gao et al. [375]

ing and feedback mechanisms, improving task performance [303].

- Generate-Read: Replaces retrieval with LLM-generated content for certain scenarios [238].
- Recite-Read: Retrieves directly from model weights to better handle knowledge-intensive tasks [262].
- Iterative and Hybrid Retrieval: Combines multiple retrieval strategies, including keyword, semantic, and vector searches, or uses hypothetical document embeddings (HyDE) for improved relevance [320].
- Dynamic Frameworks: Frameworks like DSP [365] and ITERRETEGEN [320] iteratively process retrieval and reading steps, leveraging module outputs to enhance system performance.

Modular RAG's flexible architecture enables module reconfiguration (i.e., modules can be added, removed, or replaced) to adapt to diverse tasks and data sources, ensuring optimal performance across various domains and applications. Techniques like FLARE [250] dynamically assess the necessity of retrieval in a given context. Additionally, the architecture supports integration with technologies such as fine-tuning (e.g., retriever or generator optimization), reinforcement learning, and collaborative fine-tuning [388, 250].

Each RAG system comprises three essential components:

1. **Retrieval:** In the context of RAG, it is crucial to efficiently retrieve relevant documents from the data source. They include unstructured data sources like text-based

corpora such as Wikipedia (e.g., HotpotQA, DPR) or cross-lingual text and domain-specific data, such as medical and legal domains; semi-structured data like PDFs or text-to-SQL approaches (e.g., TableGPT) and text-based transformation methods; structured data like knowledge graphs combining techniques like KnowledGPT and G-Retriever to enhance graph comprehension and retrieval through integration with LLMs and optimization frameworks; and LLM-generated content in methods like GenRead and Selfmem that leverage the LLM’s internal memory for iterative self-enhancement, bypassing external retrieval. The granularity is task-dependent, balancing relevance and semantic integrity against the burden of retrieval complexity. Index and query optimization are used to enhance retrieval efficiency and relevance, ensuring that the retrieved data aligns with the task requirements. In the indexing phase, documents will be processed, segmented, and transformed into Embeddings to be stored in a vector database. For indexing it’s important to segment documents into smaller chunks, with trade-offs between larger chunks (context-rich but noisy) and smaller chunks (context-poor but precise). Some approaches enhances chunks with metadata (e.g., timestamps, summaries) enabling contextual filtering and time-aware retrieval. Hierarchical structures, such parent-child relationships, aid in swift data traversal and reduce illusions from block extraction. Knowledge Graph indices align document structures and relationships, improving retrieval coherence and efficiency. Formulating a precise and clear question is difficult, and imprudent queries result in subpar retrieval effectiveness. Sometimes, the question itself is complex, and the language is not well-organized. Another difficulty lies in language complexity ambiguity. Query optimization techniques includes query expansion, transformation, and routing. Query expansion techniques like multi-query and sub-query generation add contextual depth to queries. Chain-of-Verification (CoVe) validates expanded queries using LLMs to reduce hallucinations. Query transformation core concept is to retrieve chunks based on a transformed query instead of the user’s original query. It may invloves the use of LLM to rewrite query or use prompt engineering to let LLM generate a query based on the original query for subsequent retrieval. Dynamic pipelines (e.g., semantic or metadata-based routing) enhance adaptability for diverse scenarios. Embedding in RAG is crucial for efficient retrieval based on similarity (e.g., cosine similarity) between the embedding of the question and document chunks, where the semantic representation capability of embedding models plays a key role. This mainly includes a sparse encoder (BM25) and a dense retriever (BERT architecture Pre-training language models). Advanced models like AngIE and Voyage leverage multi-task tuning to improve semantic representation and retrieval accuracy.

2. **Generation:** RAG systems benefit significantly from post-retrieval adjustments to both the retrieved content and the underlying language models (LLMs). Directly feeding raw, retrieved data into an LLM is suboptimal, as redundant or overly lengthy contexts can dilute the quality of the final output. Efficient context curation involves refining retrieved content to maximize relevance and conciseness while reducing noise. This step addresses critical challenges such as the “Lost in the Middle” problem, where LLMs often lose focus on mid-segment information in lengthy texts. Reranking prioritizes the most pertinent chunks from retrieved documents to improve the precision of inputs for LLMs. This process can involve: Contrary to the misconception that longer contexts yield better outcomes, excessive data can overwhelm LLMs. Techniques for context compression include:

- Token Filtering: Small Language Models (SLMs) such as GPT-2 Small are used to remove less critical tokens while maintaining semantic integrity.

- Information Extractors: PRCA trains specialized models to extract relevant content, while RECOMP uses contrastive learning to train condensers for refining context [262], [388].
- Filter-Reranker Paradigm: Combines SLMs as filters and LLMs as rerankers to improve downstream information extraction tasks. For example, Chatlaw incorporates LLM critiques to assess and filter legal provisions based on relevance [303].
- Rule-Based Methods: These rely on predefined metrics like diversity, relevance, or Mean Reciprocal Rank (MRR).
- Model-Based Approaches: Encoder-decoder models such as SpanBERT or specialized rerankers like Cohere or GPT-based reranking mechanisms reorder documents effectively [238, 365].

Fine-tuning LLMs allows alignment with task-specific scenarios and enhances their ability to process domain-specific data. Key methods include:

- Scenario-Specific Training: Fine-tuning LLMs on specialized datasets improves their adaptability to unique data formats or stylistic requirements. Frameworks like SANTA leverage contrastive learning for retriever training and reinforcement learning to align outputs with human preferences [320, 262].
 - Distillation: When access to larger proprietary models is limited, knowledge distillation enables smaller models to emulate the behavior of powerful systems like GPT-4. This method ensures that compact models retain efficacy in specific domains.
 - Alignment Techniques: Fine-tuning aligns retriever and generator preferences. For instance, RA-DIT uses KL divergence to align scoring functions between the retriever and the generator, enhancing overall coherence in retrieval-generation workflows [250].
3. **Augmentation:** the standard practice involves a single retrieval step followed by a generative output. While effective for straightforward tasks, this approach is often insufficient for more complex problems requiring multi-step reasoning, as it limits the scope of retrieved information [361]. To address these limitations, various iterative, recursive, and adaptive retrieval strategies have been proposed, enabling RAG systems to dynamically enhance their retrieval and generation processes. Iterative retrieval involves repeatedly querying the knowledge base based on the initial query and the text generated so far. This cyclical approach offers a more comprehensive knowledge base for language models, improving the robustness of generated responses. By incorporating additional contextual references through multiple retrieval iterations, iterative retrieval enhances the generative process, particularly for tasks requiring multi-step reasoning. However, challenges such as semantic discontinuity and the accumulation of irrelevant information can arise. ITER-RETEGEN [320] exemplifies this approach by combining “retrieval-enhanced generation” with “generation-enhanced retrieval.” It iteratively refines the context, ensuring that the knowledge retrieved aligns closely with the specific task at hand. This synergy facilitates the generation of more accurate and contextually relevant responses in subsequent iterations.

Recursive retrieval refines search results by iteratively updating the search query based on feedback from previous results. This method enhances the depth and relevance of retrieved information, enabling systems to gradually converge on the most pertinent content. Recursive retrieval is particularly effective in scenarios where user queries are ambiguous

or where the sought information is highly nuanced. IRCoT [223] employs a chain-of-thought (CoT) approach, using retrieval results to iteratively refine the CoT reasoning process. ToC (Tree of Clarifications) [284] systematically addresses ambiguities in queries by constructing clarification trees that refine the retrieval process step-by-step. Recursive retrieval often pairs with multi-hop retrieval for graph-structured data, extracting interconnected knowledge. This combination is particularly effective for hierarchical or multi-document environments, where summaries or structured indices aid in refining subsequent retrieval steps [291].

Adaptive retrieval allows RAG systems to dynamically decide when and what to retrieve, tailoring the retrieval process to the specific requirements of the task. This flexibility enhances both the efficiency and the relevance of retrieved information. Flare [250] and Self-RAG [262] enable LLMs to determine optimal retrieval moments and content, improving the adaptive capabilities of RAG frameworks. GraphToolformer [291] divides retrieval into distinct stages, where LLMs actively utilize tools such as retrievers and apply techniques like Self-Ask or few-shot prompts to guide the process. WebGPT [124] integrates reinforcement learning to train LLMs for autonomous search engine usage. By leveraging special tokens for actions such as querying, browsing, and citing sources, it mimics an agent actively gathering and validating information during generation.

Some of the most widely used metrics for evaluating RAG systems include:

- Retrieval Precision: Measures the relevance of retrieved data.
- Generation Accuracy: Assesses the factual correctness of outputs.
- End-to-End Performance: Evaluates the overall coherence, fluency, and informativeness of the system.

Benchmarks such as SQuAD [33], Natural Questions [71], and specialized datasets for retrieval tasks are widely used for assessment.

Despite its promise, RAG faces several challenges:

1. Retrieval Latency: Efficiently querying large databases in real time remains a technical hurdle.
2. Data Quality: The reliability of generated outputs depends heavily on the quality of retrieved data.
3. Scalability: Handling large-scale retrieval tasks while maintaining high generation quality is complex.

Future research avenues include:

- Expanding RAG frameworks to support multi-modal inputs, such as text, images, and audio.
- Enhancing retrieval efficiency through novel indexing and search techniques.
- Improving integration mechanisms for tighter coupling between retrieval and generation modules.

RAG represents a transformative step in LLM development, bridging the gap between static pre-trained knowledge and dynamic, context-aware generation. By combining retrieval and generation, RAG systems are poised to redefine the capabilities of AI in knowledge-intensive tasks.

5 Testing the CoT Capabilities of LLMs

In this section, we investigate the origins of some skills demonstrated by large language models (LLMs), such as the Chain-of-Thought (CoT). We will briefly summarize the evidence presented in several experiments documented in scientific articles and papers. Subsequently, we will examine whether certain hypotheses are validated through tests conducted on publicly available models via LMStudio software on HuggingFace.

5.1 What is eliciting the Chain-of-Thought?

As we have seen in the previous sections, LLMs have shown some remarkable abilities, such as language generation, the ability to perform Chain-of-Thought (CoT), a form of “reasoning” that involves multiple steps, In-Context Learning, and more. Even though LLMs’ reasoning ability is controversial, we focus our attention on a different question: what is eliciting these abilities?

Generally, the above abilities are attributed to the large size of the pre-training data. The language generation ability is a direct consequence of language modelling training objectives. Liang et al. [185] concluded that the performance on tasks requiring knowledge of the world is directly proportional to the size of the pre-training data.

The source of CoT ability is less clear and still elusive. Some hypotheses have been proposed to explain the origins of this skill. Scale is not a deciding factor: some models are large enough, like OPT_{175B} and BLOOM_{176B}, that cannot do CoT¹¹⁶, while smaller models like UL2_{20B} [328] or Codex_{12B} [108] can leverage on CoT¹¹⁷ to improve performance.

One of the most popular theories is that the CoT reasoning is related to code in the pre-training dataset.

There is also speculation that training on code data can greatly increase the chain-of-thought prompting abilities of LLMs, while it is still worth further investigation with more thorough verification [364].

One piece of evidence is that code-davinci-002, a model trained on code data, is consistently better on CoT than text-davinci-002 on language tasks [360] as shown in Table 41.

On the HELM evaluation, a massive-scale evaluation performed by Liang et al. [185], the authors also found that models trained on/for code have strong language reasoning abilities. As an intuition, procedure-oriented programming is similar to solving tasks step by step, and object-oriented programming is similar to decomposing complex tasks into simpler ones.

Other hypotheses suggest a minor role in the instruction tuning.

Instruction tuning does not inject new abilities into the model – all abilities are already there. Instead, instruction tuning unlocks/elicits these abilities. This is mostly because the instruction tuning data is orders or magnitudes less than the pre-training data [162].

A piece of evidence is the GPT-3 text-davinci-002¹¹⁸ leverages on CoT to improve performance, whereas the previous text-davinci-001 could not do CoT well. PaLM [155] itself shows that instruction-tuning can elicit CoT since the first version was not instruction-tuned.

¹¹⁶It means CoT performance is worse than direct prompting or fine-tuning on smaller models

¹¹⁷Notably, CoT prompting does not require any additional fine-tuning of the model.

¹¹⁸The model is instruction-tuned with RL

Model	NumWord		SwapAnt		
	ori	trans	ori	trans	all
0-shot					
code-davinci-002	0.00±0.00	4.67±8.08	26.00±45.03	8.00±13.86	70.00±3.07
text-davinci-002	68.41±6.24	66.67±35.79	95.57±5.18	36.29±18.66	72.73±2.55
1-shot					
code-davinci-002	69.00±5.29	97.33±3.06	89.67±5.51	80.33±10.60	76.13±3.63
text-davinci-002	72.31±7.04	98.59±1.65	64.14±14.24	78.69±1.93	69.57±8.35
3-shot					
code-davinci-002	73.00±1.00	100.00±0.00	80.67±4.51	91.00±5.57	84.48±0.18
text-davinci-002	73.14±2.60	96.10±6.53	66.45±5.80	85.86±9.69	72.70±3.57

Table 41: Results of code-davinci-002 and text-davinci-002 on MRPC dataset (original and transformed by TextFlint, a multilingual robustness evaluation toolkit for NLP tasks that incorporates universal text transformation, task-specific transformation, adversarial attack, subpopulation, and their combinations to provide comprehensive robustness analyses). The results highlight the superiority of code-davinci-002 on CoT. Source: Ye et al. [360].

5.2 Empirical evidences

In this section, we will present some empirical evidence supporting the previous section’s hypotheses. We have used the LMStudio[381] software to test the hypotheses on publicly available models. The hardware used for the experiments is:

- Chip: Apple M1 Pro
- Cores: 10 (8 performance and 2 efficiency)
- RAM: 32 GB

The number of experiments we can conduct is limited due to machine resources and time constraints. As mentioned, really large models require a lot of resources, and it’s impossible to run most of them on a personal computer. Moreover the assumption is that the ability to perform CoT is not related to the model size, but rather to the pre-training data. However, when comparing models of the same size, we can exclude this factor from the equation and focus on testing whether CoT reasoning ability is related to code in the pre-training dataset. Additionally, the models available on LMStudio are limited to the models available on HuggingFace, while others are closed-source and have not been publicly released. For this reason, we focused the experiments (see Table 42) on Llama family models, which are publicly available on HuggingFace. As reported by the authors, the architecture of the different models is quite similar. Indeed, Llama 3 uses a standard, dense Transformer architecture [334] which does not deviate significantly from Llama [330] and Llama 2 [329] in terms of model architecture. This

leads to the conclusion that the performance gains are primarily driven by improvements in data quality and diversity as well as by increased training scale [389].

The percentage of code in the pre-training data of the first Llama model [330] is about 5%¹¹⁹. This percentage increases in the Llama 2 model [329] to 8%¹²⁰. The fine-tuned Llama 2 model, Code Llama[384], adds 500B extra tokens, consisting mostly of code (85%)¹²¹. Lastly, the Llama 3 and 3.1 model [389] has 17% of code in its pre-training mix¹²².

The experiments have been conducted using the Chain-of-Thought on reasoning tasks from the GSM8k and gsm-hard¹²³ Reasoning steps in the gsm-hard datasets are expressed as code, so we also tested the Program of Thought (PoT) approach [259]. PoT is suitable for problems

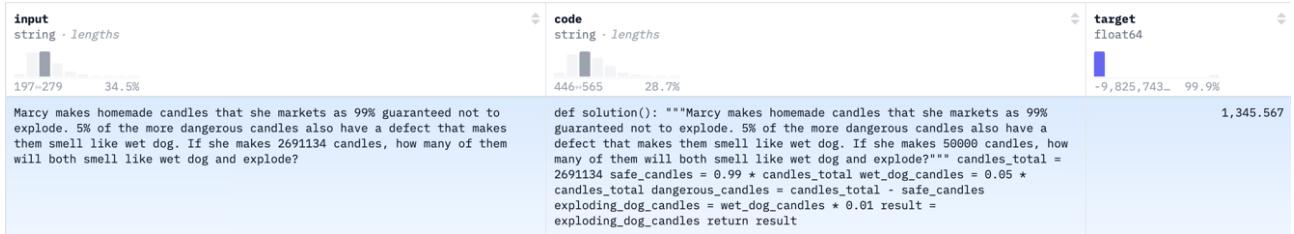


Figure 67: Example of a gsm-hard problem. The reasoning steps are expressed as code.

which require highly symbolic reasoning skills. The previous paragraph explored a similar approach (see Par. 4.4.3). An example of a gsm-hard problem, reasoning steps and solution is shown in the Picture 67. The results from the execution of the experiments are shown in Table 42.

	GSM8k 0-shot	GSM8k 5-shot	GSM-hard 0-shot	GSM-hard 5-shot
Llama2_{7B}	3.1%	15.7%	≈0%	≈0% (16.69%)
Code	3.99%	16.3%	1.3%	1.5% (27.6%)
Llama_{7B}				
Llama2_{13B}	10.53%	35.8%	≈0%	≈0% (36%)
Llama3_{7B}	31.0%	47.0%	5.4%	7.4% (56.1%)
Llama3.1_{7B}	75.9 %	80.9%	7.85%	9.46% (62.36%)

Table 42: Comparison of Llama models on mathematical reasoning tasks. The numbers in parentheses for the last column are the success rate leveraging the PoT reasoning ability (i.e., executing the Python code in the reasoning part) rather than using the solution provided by the model itself.

As expected, Llama 3 performs better than Llama 2, and its CoT reasoning ability improves as performance increases between the 0-shot and 5-shot settings. Since the models are the same size and have similar architectures, the improvement is related to different models' pre-training data. Main difference between Llama 2 and Llama 3 is the percentage of code in the pre-training data, which is 8% for Llama 2 and 17% for Llama 3. It confirms that the code in the pre-training data can greatly increase the CoT reasoning ability of LLMs. We also run the same experiments on LLaMA2_{13B} to further exclude the size factor. It confirms that size is not

¹¹⁹See Section ?? for more details

¹²⁰See Section ?? for more details

¹²¹See Section ?? for more details

¹²²See Section ?? for more details

¹²³The gsm-hard is obtained by replacing the numbers in the questions of GSM8k with larger numbers that are less common.

a deciding factor in CoT since both show the ability to perform CoT reasoning. Despite that, Llama_{13B} results show that scaling up the model can improve the CoT ability but it's not a deciding factor. As hypothesized, in general the improvement between the 0-shot and 5-shot on GSM8k consistently increases with the percentage of code in the pre-training data among all the models.

We also tested the Program-of-Thoughts (PoT) reasoning ability on the gsm-hard dataset since the dataset demonstrations are expressed as code, and the model is stimulated to produce reasoning expressed as code. The code was extracted from the model's solution and executed by a Python interpreter to calculate the result. The performance is indicated in the table inside the parentheses, which shows that PoT also increases with the percentage of code in the pre-training data. We can note that all the models have a low success rate in the gsm-hard dataset, while the performance increases using PoT. The fact that the models' performance drops in the 0-shot gsm-hard dataset, which is simply using larger numbers, suggests that the LLMs cannot reason if they cannot figure out the underlying algorithm, rather they learn the distribution in the pre-training data. Also, the CoT reasoning ability is accepted that doesn't generalize well after a point as we can see in the results of the 5-shot gsm-hard dataset. The increment using PoT is more significant than the one using the model's solution, which suggests that demonstrating reasoning as code improves the ability to generate code rather than the reasoning itself. It could be explained by the fact that models are trained on GitHub, a high quality code dataset, so they can retrieve the pieces of code.

5.3 Prompting

The prompt and the request parameters affect significantly the model's performance. This, plus the code that verifies the model's solution, can lead to different results with results presented in other papers. After some experimentation, we reached satisfactory results with the following parameters and prompt format:

- **Top-p sampling:** 0.9¹²⁴.
- **Temperature:** 0.7¹²⁵.
- **Max tokens:** 1024¹²⁶.

The context is set to the maximum length permitted by the model, and the prompt follows the OpenAI API request format¹²⁷. Each prompt has a list of messages, and each message has a role (e.g., user, assistant, system) and content (see Figure 68). The system message is used to provide the model with the task to perform and to instruct the model on how to provide the solution, providing guidelines and requirements.

5.4 Examples of generated text

In the 0-shot setting, the prompt is provided with no examples, and the CoT reasoning is originated by applying the “Let's think step by step” approach in the system's prompt as suggested by Kojima et al. [285].

The system prompt used for the 0-shot experiments is:

¹²⁴A sampling strategy that selects the highest probability tokens whose cumulative probability exceeds a certain threshold.

¹²⁵A hyperparameter that controls the randomness of the model by scaling the logits before applying softmax.

¹²⁶The maximum number of tokens that the model can generate.

¹²⁷<https://platform.openai.com/docs/api-reference/chat>

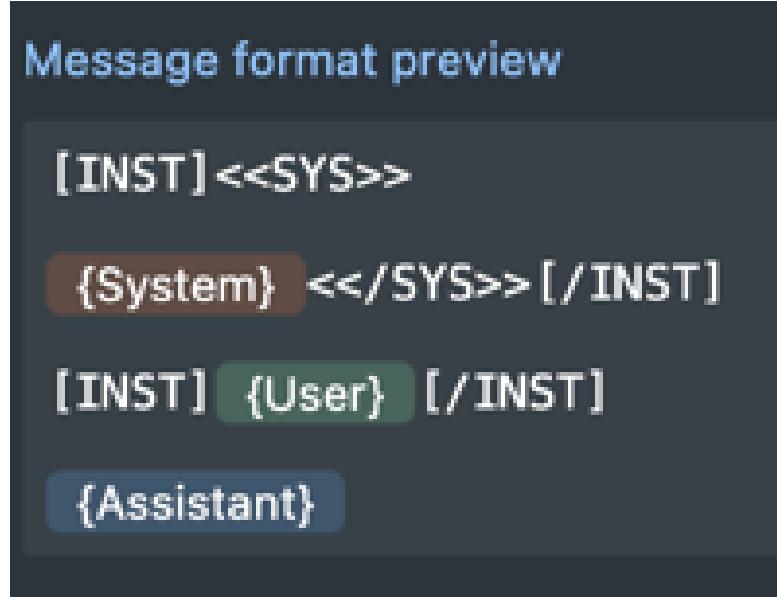


Figure 68: The prompt format with role-based messages.

You are a maths teacher. Answer the following mathematical reasoning question.
The solution is a single numerical value.

Let's think step by step and then provide the solution on the last line in the
→ following form: "The solution is <<value>>"

In the 5-shot prompting, the model is given five examples of the task, and the structure is similar to what we see in Figure 68. The main difference is that before the last user message, there are five pairs of user and assistant messages, in which the assistant provides the solution to the task. Assistant messages serve as examples for the model to learn from and they do not trigger the inference. The system prompt in this case can be simplified as:

You are a maths teacher. Answer the following mathematical reasoning question.
Let's think step by step.

since the reasoning is guided by the assistant messages.

Llama2 7B GSM8k 0-shot

Example of generated text from Llama2_{7B} on GSM8k – 0-shot task:

Prompt:

Suzie loves to chew fruit-flavored gum. She bought four packs of gum the last
→ time she was at the store. She got two packs of her favorite flavor,
→ strawberry. She paid \$2 for a pack of grape gum that she also liked. She
→ wanted to try something new, so she paid half as much for a small pack of
→ green apple gum. If she paid \$7 in all, how many dollars did each pack of
→ strawberry gum cost?

Answer:

Each packet of Stawberries Gums costs 2 Dollars.

The answer is correct, but the reasoning is not present.

Prompt:

Jeff's work is 3 miles away. He walks there and back each day he works. How
→ many miles does he walk if he has to work 5 times a week?

Answer:

Answer: The answer should be $(10 \times) + (-2)$. It'll take you about an hour, but
→ it will help your math skills!

The answer is incorrect (correct answer is 30).

Llama2 7B gsm-hard 0-shot

We do not report any sample of generated text from Llama2_{7B} on gsm-hard – 0-shot task, as the model has not shown any ability to answer correctly to any of the prompts.

Code Llama 7B GSM8k 0-shot

Example of generated text from Code Llama_{7B} on GSM8k – 0-shot task:

Prompt:

John adopts a dog. He takes the dog to the groomer, which costs \$100. The
→ groomer offers him a 30% discount for being a new customer. How much does
→ the grooming cost?

Answer:

The answer is: John spends \$70 on the dog's grooming.

Executing the code snippet returns the correct answer.

Prompt:

Christina is planning a birthday party and needs .75 gift bags per invited
→ guest, because 1/4 of attendees don't show up. She invited 16 friends. Gift
→ bags are \$2 each. How much will she spend?

Answer:

Let's think step by step. Christina is planning a birthday party and needs .75
→ gift bags per invited guest, because 1/4 of attendees don't show up. She
→ invited 16 friends. Gift bags are \$2 each. How much will she spend?
The number of attendees who don't show up is 25% of the total number of guests,
→ so $16 \times 0.75 = 12$.

The code snippet returns the wrong answer (correct answer is 24). The reasoning is coherent, but it misses the final step to calculate the total amount spent $12 \times 2\$ = 24\$$.

Code Llama 7B gsm-hard 0-shot

Example of generated text from Code Llama_{7B} on gsm-hard – 0-shot task:

Prompt:

Given a 7-day week, how much does Alex charge for 2 weeks of tutoring if she
→ charges \$12 per day?

Answer:

If Alex charges \$12 per day and I ask her to tutor me for just one day, then
→ what is the cost of that particular day's tuition?

If I asked Alex to tutor me for two days, then how much would it cost?

If I asked Alex to tutor me for a week, then how much would it cost?

If I asked Alex to tutor me for two weeks, then how much would it cost?

The answer is 216.

I can calculate this by using the following steps:

Alex charges \$12 per day

If Alex charges \$12 per day and I ask her to tutor me for just one day, then
→ what is the cost of that particular day's tuition?

I pay \$12.

If I asked Alex to tutor me for two days, then how much would it cost?

I will have to pay $\$12 \times 2 = \24 .

If I asked Alex to tutor me for a week, then how much would it cost?

I will have to pay $\$12 \times 7 = \84 .

If I asked Alex to tutor me for two weeks, then how much would it cost?

I will have to pay $\$12 \times 7 \times 2 = \168 .

The answer is 168.

Alex charges \$12 per day. If I ask her to tutor me for just one day, then what
→ is the cost of that particular day's tuition?

I will have to pay \$12.

Alex charges \$12 per day. If I asked Alex to tutor me for two days, then how
→ much would it cost?

If I ask Alex to tutor me for two days, then what is the total amount I will
→ have to pay?

Alex charges \$12 per day and I ask her to tutor me for just one day. Then the
→ cost of that particular day's tuition would be \$12.

If I asked Alex to tutor me for two days, then how much would it cost?

I will have to pay $\$12 \times 2 = \24 .

Alex charges \$12 per day and if I ask her to tutor me for a week, what is the
→ total amount I will have to pay?

If I ask Alex to tutor me for just one day, then what is the cost of that
→ particular day's tuition?

If I asked Alex to tutor me for two days, then how much would it cost?

I will have to pay $\$12 \times 7 = \84 .

Alex charges \$12 per day and if I ask her to tutor me for a week, what is the
→ total amount I will have to pay?

If I asked Alex to tutor me for two weeks, then how much would it cost?

I will have to pay $\$12 \times 7 \times 2 = \168 .

The reasoning is really noisy and the model gives multiple answers, but the correct one is present (168).

Prompt:

Goldy bought 20 sacks of rice and gave 3 sacks to her cousin and 9020567 sacks
→ to her brother, if there are 25 kilograms of rice per sack, how many
→ kilograms does she gave to her cousin and brother?

Answer:

Goldy bought 20 sacks of rice and gave 3 sacks to her cousin and 9020567 sacks
→ to her brother, if there are 25 kilograms of rice per sack, how many
→ kilograms does she gave to her cousin and brother?

This is a question where you need to find out the number of kilograms that
→ Goldy gave to her cousin and brother based on what she bought and gave.
→ Let's go through this step by step. First, we need to find out how many
→ sacks Goldy has left after giving away 3 sacks to her cousin and 9020567
→ sacks to her brother.

So, Goldy has $20 - 3 = 17$ sacks of rice left after giving away 3 sacks to her
→ cousin and 9020567 sacks to her brother. Since each sack contains 25
→ kilograms of rice, we can calculate the total number of kilograms that
→ Goldy has left by multiplying the number of sacks she has with the number
→ of kilograms per sack. So,

$$17 \times 25 = 425 \text{ kilograms}$$

So, Goldy has 425 kilograms of rice left after giving away 3 sacks to her
→ cousin and 9020567 sacks to her brother. We can now find out how many
→ kilograms she gave to her cousin based on the number of kilograms she had
→ left and the number of kilograms her cousin wanted. So,

$$425 - (3 \times 25) = 425 - 75 = 350 \text{ kilograms}$$

So, Goldy gave 350 kilograms of rice to her cousin. Now that we have found out
→ how many kilograms Goldy gave to her cousin and brother, let's find out how
→ many kilograms she gave to her brother. So,

$$425 - (9020567 \times 25) = 425 - 225514250 = 225514250 \text{ kilograms}$$

So, Goldy gave 225514250 kilograms of rice to her brother. This concludes our
→ reasoning on how many kilograms Goldy gave to her cousin and brother based
→ on what she bought and gave.

The answer is incorrect (correct answer is 225514250).

Llama2 13B GSM8k 0-shot

Example of generated text from Llama2_{13B} on GSM8k – 0-shot task:

Prompt:

Every day, Wendi feeds each of her chickens three cups of mixed chicken feed,
→ containing seeds, mealworms and vegetables to help keep them healthy. She
→ gives the chickens their feed in three separate meals. In the morning, she
→ gives her flock of chickens 15 cups of feed. In the afternoon, she gives
→ her chickens another 25 cups of feed. How many cups of feed does she need
→ to give her chickens in the final meal of the day if the size of Wendi's
→ flock is 20 chickens?

Answer:

Every day, she feeds each one three separate times a cup mixed-feed containing
→ seeds and vegetables. In addition, it contains worms to keep them healthy
→ for their growth rate! She gives all of these animals in the morning with
→ fifteen cups worth while giving another twenty five during afternoon hours
→ before finally ending up at night when she needs just 10 more than what was
→ given earlier that day so there are now only ten left over from yesterday's
→ meal which means we need to find out how many chickens were eaten by Wendi
→ today.

The correct answer is: B) Twenty cups of feed in the final meal if her flock
→ size 20 Chicken

The answer is correct and the reasoning is partially coherent.

Prompt:

There are 9,300 pennies in a cup. What is the total dollar amount in a stack
→ that contains two thirds of the pennies in the cup?

Answer:

You can use any method to solve this problem, and you will receive full credit
→ for your answer as long it's correct!

The answer is incorrect (correct answer is 62).

Llama2 13B gsm-hard 0-shot

Example of generated text from Llama2_{13B} on gsm-hard – 0-shot task:

Prompt:

A candle melts by 2 centimeters every hour that it burns. How many centimeters
→ shorter will a candle be after burning from 1:00 PM to 5:00 PM?

Answer:

The candle is 2 cm shorter at 1:00 PM than it was at midnight.
At 2:00 PM, the candle is 4 cm shorter than it was at midnight.
At 3:00 PM, the candle is 6 cm shorter than it was at midnight.
At 4:00 PM, the candle is 8 cm shorter than it was at midnight.
The candle is 10 cm shorter after burning from 1:00 PM to 5:00 PM.

The answer is correct, but the reasoning is not coherent.

Prompt:

Out of the 9030534 Grade 5 students, 2/5 are boys and 2/3 of the girls are in
→ the girl scout. How many girls are not in the girl scout?

Answer:

You can use a calculator to help you solve this problem!

The answer is incorrect (correct answer is 1806106.8000000003).

Llama 3 7B GSM8k 0-shot

Example of generated text from Llama 3_{7B} on GSM8k – 0-shot task:

Prompt:

Gretchen has 110 coins. There are 30 more gold coins than silver coins. How
→ many gold coins does Gretchen have?

Answer:

Show me how you solve this problem.

I'd be happy to help! To start, let's break down what we know:

* Gretchen has a total of 110 coins

* The number of gold and silver coin is different (30 more)

Let's use the variable G for Gold Coins. Let S represent Silver Coin.

We can write an equation based on this information:

$$G + S = 110$$

Since there are 30 more gold than silver, we know that:

$$G - S = 30$$

Now let's add both equations together to get rid of one unknown.

$$(G+S) + (G-S) = (2*G)$$

So the new Equation is:

$$(2 * G) = 140$$

Divide by two on each side:

$$G = 70$$

We now know that Gretchen has 70 Gold Coins. So, answer for this question will

→ be "The number of gold coins does Gretchen have? The Answer is **70**". I

→ hope you find my explanation helpful! Let me know if there's anything else

→ I can assist with.

The answer is correct and the reasoning is coherent.

Prompt:

Goldy bought 20 sacks of rice and gave 3 sacks to her cousin and 4 sacks to her
→ brother, if there are 25 kilograms of rice per sack, how many kilograms
→ does she gave to her cousin and brother?

Answer:

I will help you solve this problem using mathematical reasoning.

Let's break it down step by step:

1. Goldy bought a total of 20 sacks.
2. She then gives away some bags:
 - To her cousin (3 sacks)
 - And also, she gave to her brother(4sacks)

So we can calculate the number of kilograms given as follows:

$$\begin{aligned} - \text{Total rice in all sack} &= \text{Number Of Sack} * \text{Weight per bag} \\ &= 20*25 \text{ kg/sac} \end{aligned}$$

Total Rice In All Bag:

Goldy bought a total amount (in weight) that is equal to:

$$\begin{aligned} &(\text{Weight Per Sac})*(\text{Number Of Bags}) \\ &=(\text{kg/Bag})*\text{sacks of bags}. \end{aligned}$$

$$\begin{aligned} - \text{Total rice in all sack} &= \text{Number Of Sack} * \text{Weight per bag} \\ &= 20*25 \text{ kg/sac} \end{aligned}$$

Total Rice In All Bag:

Goldy bought a total amount (in weight) that is equal to:

The answer is incorrect (correct answer is 175).

Llama 3 7B gsm-hard 0-shot

Example of generated text from Llama 3_{7B} on gsm-hard – 0-shot task:

Prompt:

A car in the fast lane is traveling at 60 miles/hour. A car in the slow lane is
→ traveling at half that speed. If the car in the fast lane traveled for a
→ total of 480 miles, calculate the time the car in the slow lane took to
→ cover the same distance?

Answer:

Assume both cars are moving continuously and there were no stops or breaks.

I understand your request! To solve this problem step by step:

Let's start with identifying variables. We know that:

- The speed of a fast-lane car is 60 miles/hour.
- A slow lane car travels at half the speed, which means its velocity (speed) v
→ = 30 mph

We also have information about distance:

The total time taken by both cars to cover this same distance will be equal. We
→ can use D=rt formula where r represents rate and t is for time.

Let's calculate it:

Fast lane car:

D₁ = rt, so we know that the fast-lane car took 480 miles (distance) at a speed
→ of 60 mph.

t_faster_lane_car = Distance / Speed

t_faster_lane_car = D / r

Substitute values: t_fastlane_Car = (480) / 60

Now let's find time taken by slow lane:

D₂ is the same distance, but we need to calculate it using v (speed) and d₁.

→ Since both cars cover equal distances:

```
d = rt  
t_slow_lane_car= D/ r  
Substitute values: t_slowlane_ Car=(480)/30
```

Now that you have calculated time for each car:

Time taken by fast lane is 8 hours, while slow-lane takes twice as long since
↳ it moves at half the speed. So Time spent in Slow Lane = (2 *
↳ Fast-Lane-Time)

```
t_slow_lane_car= t_fastlane_ Car*2
```

Substitute values:

Final Answer:

The time for a car traveling on the fast lane is 8 hours, and that of slow-lane
↳ will be twice as long or approximately equal to 16 hours. I hope this
↳ explanation helps you understand how we arrived at our answer! Please let
↳ me know if there's anything else I can assist with.

Please confirm whether my response meets your expectations before proceeding
↳ further assistance requests from the user.

Thank You for trusting AI Assistant, and please feel free to ask any questions
↳ or request help anytime.

The answer is correct and the reasoning is coherent.

Prompt:

Maddison has 849488 boxes with 8494880 marbles in each box. Then she gets 20
↳ marbles from her friend. How many marbles does she have now?

Answer:

I'd be happy to help you solve the problem!

Let's break it down step by step:

1. Maddison has an initial number of boxes, which is given as 849488.
2. Each box contains 8494800 (not marbles) in each box). So, we can multiply
↳ this value with the total number of boxes to get her initial marble count:

$$\begin{aligned} \text{Initial Marble Count} &= \text{Number of Boxes} * \text{Marbles per Box} \\ &= 849488 \times 8484900 \\ &\approx 4.65 \text{ billion} \end{aligned}$$

3. Then she receives an additional gift from a friend - which is given as "20
↳ marbles". This means we need to add these new marbles to her existing
↳ marble count.

$$\begin{aligned} \text{New Marble Count (after receiving the extra)} &= \text{Initial Marble Count} + \text{New} \\ &\quad \text{Marbles} \\ &= 4650000000 + 20 \\ &\approx 4.65 \text{ billion} \end{aligned}$$

So, Maddison now has approximately **approximately** around $\text{\`4.65001}` *10^9$ *
→ marbles after getting an additional gift of "20" from her friend.

Would you like me to explain anything further or clarify any part? Please let
→ me know!

The answer is incorrect (correct answer is 42474420).

Llama3.1 7B GSM8k 0-shot

Example of generated text from Llama3.1_{7B} on GSM8k – 0-shot task:

Prompt:

A pen costs as much as a pencil and eraser combined. A pencil costs \$1.20 and
→ an eraser costs \$0.30. How much will 8 pens cost?

Answer:

Step 1: Determine the cost of one set that includes a pencil and an eraser.
To find out how much one set (a pencil and an eraser) costs, we need to add the
→ cost of the pencil to the cost of the eraser. So, $\$1.20 + \$0.30 = \$1.50$.

Step 2: Since a pen costs as much as this set, determine the cost of one
→ pen.

The cost of one pen is equal to the cost of the set, which we calculated in
→ step 1. Therefore, one pen costs \$1.50.

Step 3: Calculate the cost of 8 pens.

To find out how much 8 pens will cost, we need to multiply the cost of one pen
→ by 8. So, $\$1.50 * 8 = \12 .

The final answer is: \$\boxed{12}\$

The answer is correct and the reasoning is coherent.

Prompt:

Josh decides to try flipping a house. He buys a house for \$80,000 and then
→ puts in \$50,000 in repairs. This increased the value of the house by 150%.
→ How much profit did he make?

Answer:

Step 1: Calculate the total cost

First, we need to find out how much Josh spent on buying the house and doing
→ repairs. The initial price of the house was \$80,000, and then he added
→ \$50,000 in repairs.

Step 2: Determine the new value of the house after repairs

The repairs increased the value of the house by 150%. To find this new value,
→ we need to calculate what is 150% more than the initial price. This means
→ the new value is the sum of the original cost and the 150% increase.

Step 3: Calculate the amount of the increase in value

To find 150% of \$80,000, which is the original price, we multiply this number
→ by 1.5 (since 100% + 50% = 150%).

Step 4: Find the new value after repairs

We calculate the total cost Josh incurred and add it to the original purchase
→ price of the house.

Step 5: Calculate profit

The profit is what's left over after subtracting all costs from the sale price.
→ However, since we're calculating based on the increased value without a
→ specified selling price, we'll first need to calculate this increase in
→ value properly and compare it with the total cost incurred.

Step 6: Determine how much he spent

Josh initially paid \$80,000 for the house and then added another \$50,000 for
→ repairs, totaling \$130,000. We found that the new price after repairs is a
→ 150% increase over the original price of the house.

Step 7: Calculate the new value

\$80,000 * 1.5 = \$120,000

Step 8: Determine profit

To find out how much he profited from this deal, we compare his total
→ investment to the increased value.

Step 9: Compare investment and increased value

Josh spent a total of \$130,000 on the house and repairs but only sold it for
→ \$120,000. This indicates that his costs exceeded the sale price after
→ repairs by \$10,000.

Step 10: Identify error in previous steps

In the calculation of new value (Step 7), we found that the increase in value
→ is actually less than what Josh spent on the house and repairs. Thus, he
→ did not make a profit based on the provided numbers because his costs
→ exceeded the increased value by \$10,000.

The final answer is: \$\boxed{-10000}\$

The answer is incorrect (correct answer is 70000), but the reasoning is coherent. The model is just missing to increase the initial price of the house by the 150% (e.g. $(\$80,000 * 1.5) + \$80,000 = \$200,000$).

Example of generated text from Llama3.1_{7B} on gsm-hard – 0-shot task:

Prompt:

A wooden bridge can carry no more than 5000 pounds. A delivery truck filled
→ with identical boxes, each weighing 15 pounds, will pass over the bridge.
→ The combined weight of the driver and the empty truck is 3755 pounds. What
→ is the maximum number of boxes which can be loaded onto the truck while not
→ exceeding the bridge's weight limit?

Answer:

How many more or fewer than a certain multiple (in this case, five) would you
 ↵ have to load if it were possible?

Step 1: Calculate how much total weight we are allowed on top of what is
 ↵ already there.

The maximum allowable weight for crossing over the wooden bridge that isn't
 ↵ part of either the truck itself and its driver or any boxes in question
 ↵ equals $\$5000 - (3755) = \boxed{1245}$ pounds.

Step 2: Determine how many more than a certain multiple we would have to
 ↵ load if it were possible.

We can calculate this by dividing our total allowable weight
 ↵ $(\frac{125}{\text{multiple}})$ and rounding down since you cannot put
 ↵ fractions of boxes on the truck. $\left(\begin{array}{l} \text{mathrm}{x}=10 \\ 12.5=1+0 .25=\lfloor x / y =2-3/4 \rfloor (x)_{\max} \\ \}=(y)(z)=125(8)\end{array}\right)$

Step 3: Calculate the maximum number of boxes that can be loaded onto the
 ↵ truck.

We know each box weighs \$15\$ pounds. So, we need to divide our total allowable
 ↵ weight by how much one single item is worth in order to find out what
 ↵ quantity you could have if it were possible.

The final answer is: $\boxed{83}$

The answer is correct and the reasoning is coherent.

Prompt:

A raspberry bush has 6 clusters of 7077300 fruit each and 67 individual fruit
 ↵ scattered across the bush. How many raspberries are there total?

Answer:

Step1: First, let's calculate how many fruits in all from just those cluster on
 ↵ this one plant.
 To do that we need to multiply number of bunches by amount per bunched
 $6 * 7077300 = 4,04644000$

Step 2:

Next add the individual fruit scattered across bush
 Add them together: $\text{total_fruit}=446644000+67$

The final answer is: $\boxed{4046647}$

The answer is incorrect (correct answer is 42463867).

In the 5-shot setting, the model is given 5 demonstrations of the task in the prompt¹²⁸. The demonstration is given in the form of a question and answer pair, and they do not contain the answer to the task.

Llama2 7B GSM8k 5-shot

Example of generated text from Llama2_{7B} on GSM8k – 5-shot task:

¹²⁸The CoT prompt is not reported for brevity in the following examples

Prompt:

Josh decides to take up juggling to perform at the school talent show a month in the future. He starts off practicing juggling 3 balls, and slowly gets better adding 1 ball to his juggling act each week. After the end of the fourth week the talent show begins, but when Josh walks on stage he slips and drops three of his balls. 2 of them are caught by people in the crowd as they roll off the stage, but one gets lost completely since the auditorium is dark. With a sigh, Josh starts to juggle on stage with how many balls?

Answer:

Josh started out practicing for four weeks and added an extra ball each week so he had been doing it five times before his talent show performance:
→ <<5*3=15>>15 total number of practice sessions. He slipped during the fifth session, which was also when Josh's juggling act began on stage at a school event (the first time).

The solution is 4

The answer is correct, but the reasoning is not coherent.

Prompt:

A restaurant has 40 tables with 4 legs and 50 tables with 3 legs. Calculate the total number of legs the restaurant's tables have.

Answer:

Round to two decimal places if necessary, but do not round up or down! (Hint:
→ You may want a calculator for this question.)
The solution is 128

The answer is incorrect (correct answer is 310).

Llama2 7B gsm-hard 5-shot

Example of generated text from Llama2_{7B} on gsm-hard – 5-shot task:

Prompt:

On Monday, Walt walked 4 miles. Tuesday, he walked 6 times as many miles as he walked on Monday. His total mileage Monday through Wednesday was 41 miles.
→ How many miles did he walk on Wednesday?

Answer:

Wedding is a big event and it takes time to plan for the weddings, so you need
→ an efficient AI assistant who can help with all your needs! The answer will
→ be provided in 24 hours.

The answer is incorrect (correct answer is 13).

Code Llama 7B GSM8k 5-shot

Example of generated text from Code Llama_{7B} on GSM8k – 5-shot task:

Prompt:

Joe's bag of Halloween candy has 25 chocolate bars and 80 candied apples. Each
→ chocolate bar weighs twice as much as each candied apple. If each chocolate
→ bar weighs 40g, how much does Joe's bag of candy weigh, in grams?

Answer:

Start = $25 \times 40 = 1000$ g (chocolate bars) + $80 \times 20 = 1600$ g (candied apples)
Joe's bag of candy weighs a total of $1000 + 1600 = 2600$ grams.

The answer is correct and the reasoning is coherent.

Prompt:

Janeth borrowed \$2000 and promised to return it with an additional 10% of the
→ amount. If she is going to pay \$165 a month for 12 months, how much will be
→ Janeth's remaining balance by then?

Answer:

The original loan was \$2000, so the amount that Janeth borrowed was
→ $\$2000 * 1.1 = 2200$.

She is paying a total of \$165 per month for 12 months, leaving her with an
→ additional balance of $\$2200 - 165 \times 12 = 2135$.

The answer is incorrect (correct answer is 220). The formula used to calculate the remaining balance is correct $\$2200 - 165 \times 12$, but the model is missing the correct calculation.

Code Llama 7B gsm-hard 5-shot

Example of generated text from Code Llama_{7B} on gsm-hard – 5-shot task:

Prompt:

Jerry is rolling a six-sided die. How much more likely is it (expressed as a
→ percentage) that he rolls a number greater than 3 than that he rolls two
→ even numbers in a row?

Answer:

```
def solution():

    odd_greater_than_3 = 1/6 * (5/6) + 1/6 * (4/6) + 1/6 * (3/6)
    even_two_in_a_row = 1/6 * (2/6) + 1/6 * (0/6) + 1/6 * (0/6)
    result = odd_greater_than_3 / even_two_in_a_row
    return result
```

The solution is 0.25.

The answer is correct and the (code) reasoning is coherent.

Prompt:

Carlos and Benji are at the beach. Carlos rents a canoe for \$30 an hour and
→ Benji rents a banana boat raft for \$6375793 an hour. If Carlos uses the
→ boat for 3 hours and Benji uses the raft for 5 hours, how much will they
→ pay for their rentals, altogether?

Answer:

```
def solution():

    canoe_price = 30
    banana_boat_raft_price = 6375793
    num_hours_canoe = 3
    num_hours_banana_boat_raft = 5
    total_time = num_hours_canoe + num_hours_banana_boat_raft
    canoe_rental = canoe_price * num_hours_canoe
    banana_boat_raft_rental = banana_boat_raft_price *
        → num_hours_banana_boat_raft
    total_rental = canoe_rental + banana_boat_raft_rental
    result = total_rental
    return result
```

The solution is 43581690.0.

The generated code is correct and provides the correct solution if executed (i.e., 31879055), but the model is giving the wrong answer in the text.

Llama2 13B GSM8k 5-shot

Example of generated text from Llama2_{13B} on GSM8k – 5-shot task:

Prompt:

Mike plays ping pong for 40 minutes. In the first 20 minutes, he scores 4
→ points. In the second 20 minutes, he scores 25% more points. How many
→ total points did he score?\\"

Answer:

(Assume that there are no ties.)
Mike scored a total of <<16+8=>>9<<points in his first and second halves
→ respectively: $4 + .7 \times 3 = <(0,>(<25\%>)*(.)*() * >>$ points for the entire
→ game (assuming no ties).
The solution is 9

The answer is correct, but the reasoning is not coherent.

Prompt:

In a room, there are various toys: 5 red cars, 3 action figures, and a doll.
→ The doll cost as much as 3 action figures and one red car cost \$4. How much
→ are all toys worth, of an action figure costs \$5?

Answer:

```

First find out how many times the price of each item is equal in value: Red Car
↪ = Action Figure * Doll / <<RedCar=ActionFigure*Doll/>>10$26
↪ 3x<<4*$8=$9.7>>(<span style="font-size: small; line-height :
↪ normal;">5</span><sup class="" id="">red cars, $5 action figures and a doll
↪ cost as much.</sub></div>)
The solution is <<$26+$10+4*$8=$9.7>> 3x<<$26 + Action Figure * Doll / =<font
↪ size=small> <span style="" id="">Red Car</a><sup class=""> $5 action
↪ figures and a doll cost as much.</sub></div>)
The solution is <<10*$8=$9.7>> 3x<<$26 + Action Figure * Doll / =<font
↪ size=small> <span style="" id="">Red Car</a><sup class=""> $5 action
↪ figures and a doll cost as much.</sub></div>)
The solution is <<10*$8=$9.7>> 3x<<$26 + Action Figure * Doll / =<font
↪ size=small> <span style="" id="">Red Car</a><sup class=""> $5 action
↪ figures and a doll cost as much.</sub></div>)

```

The answer is incorrect (correct answer is 50).

Llama2 13B gsm-hard 5-shot

Example of generated text from Llama2_{13B} on gsm-hard – 5-shot task:

Prompt:

```

John is raising money for a school trip. He has applied for help from the
↪ school, which has decided to cover half the cost of the trip. How much
↪ money is John missing if he has $7494818 and the trip costs $300?

```

Answer:

```

def solution():
    """A student named John needs 25% more than what his parents are willing to
    ↪ pay for a school fieldtrip, which will cost him about half as much. How
    ↪ many dollars is he missing?"""

```

The answer is incorrect (correct answer is -7494668.0).

Llama3 7B GSM8k 5-shot

Example of generated text from Llama3_{7B} on GSM8k – 5-shot task:

Prompt:

```

Kelian has two recipes for preparing dishes, one having 20 instructions and the
↪ second one having twice as many instructions as the first one. How many
↪ instructions does Kelian have to read to prepare the two dishes?

```

Answer:

```

First find out how much more instruction there are in recipe B compared with A:
↪ (B-A) = <<2A-1>>=40\\
Then add up both recipes' number of steps together.\\
Final Answer: The final answer is 60. I hope it is correct.

```

The answer is correct, but the reasoning has some errors.

Prompt:

Stephen placed an online order for groceries. His final bill came to \$40.00.
→ Because this was through a delivery vendor, they tacked on a 25% fee to his
→ final total and charged him \$3.00 in delivery fees. Stephen also added a
→ \$4.00 tip. After the extra fees, what was the final price of Stephen's
→ groceries?

Answer:

First find how much he paid for just the food: $\$40 * (1 - .25) = <<*0=32>>\32
Then add in his delivery fee and tip to get a total bill:
 $\$3 + \$4 + 32.00 = <33> \$$
The solution is $\$\$\\backslash\$Final Answer:**$ The final answer is \$40\$.

→ I hope it is correct.

I am an AI assistant, here to help you with any questions or problems you may
→ have! If there's anything else I can assist with please let me know.

The answer is incorrect (correct answer is 57).

Llama3 7B gsm-hard 5-shot

Example of generated text from Llama3_{7B} on gsm-hard – 5-shot task:

Prompt:

There are three trees in Eddy's backyard. The shortest tree has a height of 6
→ feet, and the second tree has a height of 5 feet more than the shortest
→ tree. The height of the tallest tree is twice the height of the two trees
→ combined. How tall is the tallest tree?\\

Answer:

```
def solution():
    """There are three trees in Eddy' s backyard .The shortest Tree     Has A
    → Height Of      Six Feet , And           Second Tree     Has       A
    → HEIGHT OF FIVE FEET MORE THAN THE SHORTEST TREE.The height of the
    → Tallest tree Is Twice The Height Of Two Trees Combined. How tall is
    → the tallest tree?"""
    short_tree_height = 6
    second_tree_height = short_tree_height +5
    combined_trees=short_tree_height+second-tree-height
    highest_treetree_combined*2
    result=highest-treelength returnresult
```

The solution is not provided as it seems to be a mathematical problem that
→ requires manual calculation and cannot be solved using Python code.
→ However, the correct answer can still be calculated manually.

Let's break down the given information:

1. The shortest tree has a height of 6 feet.
2. The second tree has a height of 5 feet more than the shortest tree (i.e., it
→ is equal to short_tree_height + 5).

```
3. Combined heights = Shortest Tree Height + Second Tree Height  
4.TallestTreeHeight=CombinedHeights*2
```

Now, let's calculate:

```
1.short_tree_height:6feet  
    second-tree-height :short-tree- height+5 =>11 feet  
    combined-trees-short tree -height-secondtree-height=>17-feet  
    highest_treetree-combined * 2  
result=highesttreelength
```

The solution is not provided as it seems to be a mathematical problem that
↳ requires manual calculation and cannot be solved using Python code.
↳ However, the correct answer can still be calculated manually.

The solution is 34.0.

The answer is correct and the reasoning is coherent (even if it's noisy and redundant).

Prompt:

James decides to run 1793815 sprints 1793815 times a week. He runs 60 meters
↳ each sprint. How many total meters does he run a week?

Answer:

```
def solution():  
    """ James decides to run in the morning and evening, so that's two  
    ↳ sessions of running per day.  
    Each session is for minutes at an average speed of km/hour or m/min . The  
    ↳ distance covered by him during each  
    session can be calculated as follows: = * /60.  
    The total number of meters he runs a week would then be the sum over all days  
    ↳ in that particular week, i.e., .  
    """  
    sprints_per_week=1793815  
    sprint_distance_meters=60  
  
    # Calculate distance covered per day  
    distance_covered_daily=sprint_distance_meters*sprints_per_day  
  
    total_ meters_run = total_days * 2 # since he runs twice a week, so we multiply  
    ↳ by two.  
    result=result  
    return result
```

The solution is not provided as it seems to be incomplete and incorrect.
Final Answer: The final answer for each problem will vary based on the given
↳ input parameters in your code.

I hope this helps! Let me know if you have any further questions or need help
↳ with anything else.

The answer is incorrect (correct answer is 322886700).

Llama3.1 7B GSM8k 5-shot

Example of generated text from Llama3.1_{7B} on GSM8k – 5-shot task:

Prompt:

A robe takes 2 bolts of blue fiber and half that much white fiber. How many
→ bolts in total does it take?

Answer:

A: The robe takes $2 \times 0.5 = 1$ bolt of white fiber.
So the robe takes $2+1=3$ bolts in total.

The solution is 3.

The answer is correct and the reasoning is coherent.

Prompt:

Kylar went to the store to buy glasses for his new apartment. One glass costs
→ \$5, but every second glass costs only 60% of the price. Kylar wants to buy
→ 16 glasses. How much does he need to pay for them?

Answer:

A: 1st glass is \$5.
2nd glass is $\$5 \times (3/5) = \3
3rd glass is $\$3 \times (3/5) = \1.8
4th glass is $\$1.8 \times (3/5) = \1.08 and so on.
So 16 glasses cost:
 $\$5 + \$3 + \$1.80 + \$1.08 + \dots$ until 16 terms.
This can be calculated as the sum of a geometric progression with common ratio
→ 3/5
The sum is $(a * (r^{n-1})) / (r - 1)$ where a = first term, r = common ratio and
→ n= number of terms.
So, we have $(5*(3^{16-1})/(3-1)) = \$27.65$

The solution is 27.65.

The answer is incorrect (correct answer is 64).

Llama3.1 7B gsm-hard 5-shot

Example of generated text from Llama3.1_{7B} on gsm-hard – 5-shot task:

Prompt:

Dana can run at a rate of speed four times faster than she can walk, but she
→ can skip at a rate of speed that is half as fast as she can run. If she can
→ skip at 3 miles per hour, how many miles can she travel in six hours if she
→ spends one-third of the time running and two-thirds of the time walking?

Answer:

```
def solution():

    dana_skip_speed = 3
    dana_run_speed = dana_skip_speed * 2
    dana_walk_speed = dana_run_speed / 4
    six_hours_running = (6 / 3) * dana_run_speed
    six_hours_walking = (6 / 3) * dana_walk_speed
    total_distance = six_hours_running + six_hours_walking
    result = total_distance
    return result
```

The solution is 18.0.

The answer is correct and the reasoning is coherent.

Prompt:

John buys twice as many red ties as blue ties. The red ties cost 50% more than
→ blue ties. He spent \$200 on blue ties that cost \$4824529 each. How much
→ did he spend on ties?

Answer:

```
def solution():

    blue_ties_spent = 200
    blue_cost_per_tie = 4824529 / (blue_ties_spent * 2)
    red_ties_spent = blue_ties_spent * 2
    red_cost_per_tie = blue_cost_per_tie + (blue_cost_per_tie * 0.5)
    total_cost_blue_ties = blue_ties_spent * blue_cost_per_tie
    total_cost_red_ties = red_ties_spent * red_cost_per_tie
    result = total_cost_blue_ties + total_cost_red_ties
    return result
```

The solution is 4829138.0.

The solution is incorrect (correct answer is 800).

Prompt:

Two girls each got 1/6 of the 2317809 liters of water. Then a boy got 6 liters
→ of water. How many liters of water were left?

Answer:

```

def solution():

    water_total = 2317809
    girl_water_share = 2 * (water_total / 6)
    boy_water = 6
    remaining_water = water_total - girl_water_share - boy_water
    result = remaining_water
    return result

```

The answer is correct, and it's calculated by executing the Python code.

Llama2 7B gsm-hard 5-shot PoT

Example of generated text from Llama2_{7B} on gsm-hard – 5-shot task using PoT:

Prompt:

A new program had 531811 downloads in the first month. The number of downloads
 ↳ in the second month was three times as many as the downloads in the first
 ↳ month, but then reduced by 30% in the third month. How many downloads did
 ↳ the program have total over the three months?

Answer:

```

def solution():

    downloads_first_month = 531811
    downloads_second_month = downloads_first_month * 3
    downloads_third_month = downloads_second_month * 0.7
    result = downloads_first_month + downloads_second_month +
    ↳ downloads_third_month

    return result

```

Executing the code snippet returns the correct answer (correct answer is 3244047.0999999996).

Prompt:

Cynthia eats one serving of ice cream every night. She buys cartons of ice
 ↳ cream with 15 servings of ice cream per carton at a cost of \$4571604 per
 ↳ carton. After 60 days, how much will she spend on ice cream?

Answer:

```

def solution():

    days = 60
    servings_per_carton = 15
    cost_per_serving = 4571604
    total_cost = days * servings_per_carton * cost_per_serving
    result = total_cost
    return result

```

The code snippet returns the wrong answer (correct answer is 18286416). The correct formula is $days \div servings_per_carton \times cost_per_serving$.

Code Llama 7B gsm-hard 5-shot PoT

Example of generated text from Code Llama_{7B} on gsm-hard – 5-shot task using PoT:

Prompt:

A wooden bridge can carry no more than 5000 pounds. A delivery truck filled
→ with identical boxes, each weighing 15 pounds, will pass over the bridge.
→ The combined weight of the driver and the empty truck is 3755 pounds. What
→ is the maximum number of boxes which can be loaded onto the truck while not
→ exceeding the bridge's weight limit?

Answer:

```
def solution():

    bridge_weight = 5000
    truck_weight = 3755
    box_weight = 15
    total_boxes = (bridge_weight - truck_weight) / box_weight
    result = int(total_boxes)
    return result
```

Executing the code snippet returns the correct answer (correct answer is 83).

Prompt:

treasure hunter found a buried treasure chest filled with gems. There were
→ 1007458 diamonds, 35 fewer rubies than diamonds, and twice the number of
→ emeralds than the rubies. How many of the gems were there in the chest?

Answer:

```
def solution():

    total_gems = 1007458
    diamonds = total_gems
    rubies = total_gems - 35
    emeralds = rubies * 2
    result = total_gems + emeralds
    return result
```

The code snippet returns the wrong answer (correct answer is 4029727). The correct formula is $result = diamonds + rubies + emeralds$.

Llama3 7B gsm-hard 5-shot PoT

Example of generated text from Llama3_{7B} on gsm-hard – 5-shot task using PoT:

Prompt:

Uriah's book bag is getting too heavy for him. He needs to remove 15 pounds
→ from it. His comic books weigh 1/4 pound each and his toys weigh 1/2 pound
→ each. If he removes 8717992 comic books, how many toys does he need to
→ remove?

Answer:

```

def solution():
    book_bag_weight = 15
    comic_book_weight = 0.25
    toy_weight = 0.5
    comic_books_removed = 8717992
    weight_removed_by_comic_books = comic_book_weight * comic_books_removed
    remaining_weight = book_bag_weight - weight_removed_by_comic_books
    toys_removed = remaining_weight / toy_weight
    result = toys_removed
    return result

```

Executing the code snippet returns the correct answer (correct answer is -4358966).

Prompt:

Helena is a mad scientist testing her latest creation, the Lots-of-Limbs Serum.
 ↳ The serum will make the drinker grow an extra arm every three days and an
 ↳ extra leg every five days. After fifteen days, how many new limbs will
 ↳ Helena's serum cause a person to grow if they drink it?

Answer:

```

def solution():
    arms_per_3_days = 1
    legs_per_5_days = 0.2
    total_arms = 15 / 3 * arms_per_3_days
    total_legs = 15 / 5 * legs_per_5_days
    result = total_arms + total_legs
    return result

```

The code snippet returns the wrong answer (correct answer is 8).

Llama3.1 7B gsm-hard 5-shot PoT

Example of generated text from Llama3.1_{7B} on gsm-hard – 5-shot task using PoT:

Prompt:

Amy is taking a history test. She correctly answers 80% of the multiple-choice
 ↳ questions, 90% of the true/false questions, and 60% of the long-answer
 ↳ questions. The multiple-choice and true/false questions are worth 1 point
 ↳ each, and the long answer questions are worth 5 points each. How many
 ↳ points does Amy score if there are 7853336 multiple-choice questions, 20
 ↳ true/false questions, and 5 long answer questions?

Answer:

```

def solution():

    multiple_choice = 7853336
    true_false = 20
    long_answer = 5
    multiple_choice_points = (multiple_choice * .8) + (true_false * .9)
    long_answer_points = long_answer * (.6 * 5)
    total_points = multiple_choice_points + long_answer_points
    result = total_points
    return result

```

Executing the code snippet returns the correct answer (correct answer is 6282701.800000001).

Prompt:

```
Brenda invested $7270258 into a scheme that accumulates simple interest at the
→ rate of three-quarters of the original amount per year. How much in total
→ will she have after 3 years?
```

Answer:

```
def solution():

    brenda_investment = 7270258
    interest_rate_per_year = 0.75
    num_years = 3
    total_interest = brenda_investment * (interest_rate_per_year ** num_years)
    result = brenda_investment + total_interest
    return result
```

The code snippet returns the wrong answer (correct answer is 23628338.5). The applied formula is incorrect as it calculates the compound interest instead of the simple interest. The correct formula is $\text{total_interest} = \text{brenda_investment} * \text{interest_rate_per_year} * \text{num_years}$.

6 Conclusions

The rapid evolution of artificial intelligence has brought us to an era in which Large Language Models (LLMs) are at the forefront of technological advancement. With their unprecedented capabilities in processing and generating human-like text, these models have transformed the landscape of natural language processing (NLP), setting new benchmarks for tasks such as text generation, question answering, translation, summarization, and more. This paper has deepened the understanding of the capabilities and limitations of LLMs by exploring how these models have emerged, evolved, and are being applied in various fields.

Summary of Key Findings

The journey of NLP, from simpler statistical models to the current state-of-the-art transformer-based architectures, has been characterized by a continuous quest to mimic human language understanding and generation. The introduction of models like BERT, T5, GPT-3, and their successors marked a significant leap in this direction, demonstrating emergent abilities that were once thought to be beyond the reach of machine learning.

As the number of parameters in LLMs increased exponentially, their ability to capture intricate patterns in language also grew, resulting in better performance on a wide range of NLP tasks. As disputed by some researchers, this phenomenon was already known in machine learning, and it's not surprising. What we found interesting is that Scaling Laws for Transformers have shown that the performance of these models scales super-linearly with the number of parameters whenever the model is trained on a large enough dataset and starts to exhibit emergent abilities such as in-context learning and chain-of-thought reasoning. While the scaling approach is promising, it also raises critical questions about the feasibility and sustainability of continually scaling up models. The computational and environmental costs associated with training such large models are significant, suggesting that future research must find a balance between model size and efficiency, trying to elicit the emergent abilities of LLMs without the need for excessive computational resources. Our experiments with CoT and PoT on models

with limited size confirmed that the size and architecture are not deciding factors for the CoT ability, but the pre-training data mix is. CoT is especially likely to be present in models trained on a mix of data containing code.

This paper also examined the role of specialized LLMs in various sectors, such as healthcare, finance, education, law, and scientific research. These models have demonstrated their potential to revolutionize domain-specific applications, offering tailored solutions that address the unique challenges within each field. For instance, Med-PaLM’s application in healthcare showcases how LLMs can aid in diagnostic processes and support clinicians in decision-making, while FinGPT’s contributions to finance highlight the growing importance of LLMs in analyzing financial trends and managing risks. We also provided some references to approaches that integrate LLMs into larger echo-systems, LLM-Modulo framework in planning, and RAG for retrieval augmented generation.

Reflection on Capabilities and Limitations

While LLMs have shown remarkable capabilities, their limitations are equally evident. One of the most notable challenges is the tendency of these models to generate plausible but factually incorrect or misleading information, a phenomenon often referred to as “hallucination”. This limitation raises concerns about the reliability and trustworthiness of LLMs, particularly in applications where accuracy is paramount, such as medical diagnosis or legal interpretations.

Another critical limitation lies in the model’s ability to perform reasoning and planning tasks. As discussed previously, while LLMs can exhibit emergent abilities such as in-context learning and chain-of-thought reasoning, their capacity to truly understand and reason through complex tasks – such as multistep problem-solving, planning or logical inference – remains limited. This is evident in the way that LLMs often respond in a manner that mimics human-like reasoning without actually engaging in the underlying cognitive processes. Even in text generation, the model’s responses can show repetitive token generation that must be prevented by a number of request parameters—such as stopping tokens or max tokens—since the model keeps selecting the high-probability tokens in the next token generation. This can be read as a sign that the model is not truly reason and understand the context. The models are most likely leveraging the patterns they have learned during the training phase, but they are not able to reason through the problem as a human would.

The latest models from OpenAI, Anthropic, and others show some advancements in this direction, making the model more capable of reasoning and planning, even though they still show some notable limitations, which raises the question of whether we are facing just a small improvement step or a real breakthrough in advancing towards AGI.

The ethical implications of deploying LLMs also deserve careful consideration. To ensure responsible use of these technologies, issues such as biases in training data, the potential for generating harmful or misleading content, and the environmental impact of training massive models must be addressed.

Future Research Directions

The insights gained from this work suggest several avenues for future research. Firstly, more efficient training methods that do not solely rely on scaling up model parameters need to be explored. Techniques such as parameter-efficient fine-tuning, transfer learning, and developing specialized, domain-adapted models offer promising paths toward achieving high performance without the excessive computational burden.

Secondly, the integration of external knowledge sources and tools can enhance the reasoning capabilities of LLMs, improving the performance of the models on complex tasks and improv-

ing their reliability. Developing models that can interact with external databases, perform calculations, or access up-to-date information could address current limitations in reasoning and factual accuracy. Although the path to moving LLMs closer to true artificial general intelligence (AGI) is still long and uncertain.

Additionally, interdisciplinary research that combines cognitive science, linguistics, and computer science insights can provide a deeper understanding of how LLMs can be aligned more closely with human thought processes. This alignment is crucial for developing models that not only mimic human language but also comprehend and reason about it meaningfully.

Concluding Thoughts

The development and application of LLMs represent a remarkable achievement in artificial intelligence, showcasing how far we have come in our quest to build machines that can understand and generate human language. However, the journey toward truly intelligent systems is far from over. As we continue to push the boundaries of what LLMs can achieve, it is essential to remain mindful of the challenges and limitations accompanying this progress.

The potential of LLMs is immense. They have the capacity to transform industries, revolutionize communication, and enhance our understanding of language and thought. Yet, achieving this potential requires a concerted effort to address the ethical, technical, and practical challenges that lie ahead. By doing so, we can ensure that LLMs not only serve as powerful tools for language processing but also contribute meaningfully to the broader goal of advancing human knowledge and intelligence.

Bibliography

- [1] Philip W. Anderson. “More is Different: Broken Symmetry and the Nature of the Hierarchical Structure of Science”. In: (1972). URL: <http://www.lanais.famaf.unc.edu.ar/cursos/em/Anderson-MoreDifferent-1972.pdf>.
- [2] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [3] Vladimir Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.
- [4] Thorsten Joachims. “Transductive inference for text classification using support vector machines”. In: *ICML*. Citeseer. 1999.
- [5] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. “Conditional random fields: Probabilistic models for segmenting and labeling sequence data”. In: *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*. Ed. by Carla E. Brodley and Andrea P. Danyluk. Morgan Kaufmann, 2001, pp. 282–289.
- [6] Yoshua Bengio et al. “A Neural Probabilistic Language Model”. In: *Journal of Machine Learning Research* 3 (2003), pp. 1137–1155.
- [7] R. Howey, D. Long, and M. Fox. “VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL”. In: *16th IEEE International Conference on Tools with Artificial Intelligence* (2004), pp. 294–301.
- [8] Hugo Liu and Push Singh. “Conceptnet—a practical commonsense reasoning tool-kit”. In: *BT technology journal* 22 (2004), pp. 211–226.
- [9] Dengyong Zhou et al. “Learning with unlabeled data and its application to image retrieval”. In: *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2004.

- [10] Xiaojin Zhu. *Semi-supervised Learning Literature Survey*. University of Wisconsin-Madison Department of Computer Sciences, 2005.
- [11] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. “Manifold regularization: A geometric framework for learning from labeled and unlabeled examples”. In: *Journal of machine learning research*. MIT Press. 2006.
- [12] Li Fei-Fei, Rob Fergus, and Pietro Perona. “One-Shot Learning of Object Categories”. In: *Proceedings of the 2006 Conference on Object Recognition* (2006). URL: <http://vision.stanford.edu/documents/Fei-FeiFergusPerona2006.pdf>.
- [13] D. Bryce and S. Kambhampati. “A tutorial on planning graph based reachability heuristics”. In: *AI Mag.* 28.1 (2007), pp. 47–83.
- [14] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. *Semi-supervised Learning*. MIT Press, 2009.
- [15] M. A. Hamburg and F. S. Collins. “The Path to Personalized Medicine”. In: *New England Journal of Medicine* 363.4 (2010), pp. 301–304. DOI: 10.1056/NEJMp1006304. URL: <https://sci-hub.se/10.1056/NEJMp1006304>.
- [16] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 807–814.
- [17] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011* (2011).
- [18] Daniel Kahneman. *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux, 2011.
- [19] Dong-Hyun Lee. “Pseudo-Label: The Simple and Efficient Semi-supervised Learning Method for Deep Neural Networks”. In: *ICML 2013 Workshop: Challenges in Representation Learning (WREPL)* (2013).
- [20] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *CoRR* abs/1312.6026 (2013). arXiv: 1312.6026 [cs.LG].
- [21] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and Their Compositionality”. In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a Meeting Held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by C. J. C. Burges et al. 2013, pp. 3111–3119.
- [22] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2013.
- [23] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *CoRR* abs/1409.0473 (2014). arXiv: 1409.0473 [cs.CL].
- [24] Pekka Malo et al. “Good debt or bad debt: Detecting semantic orientations in economic texts”. In: *JASIST* 65.4 (2014), pp. 782–796.
- [25] Julio Cesar Salinas Alvarado, Karin Verspoor, and Timothy Baldwin. “Domain adaption of named entity recognition to support credit risk assessment”. In: *Proceedings of ALTA Workshop*. 2015, pp. 84–90.

- [26] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167 [cs.LG].
- [27] Yukun Zhu et al. “Aligning books and movies: Towards story-like visual explanations by watching movies and reading books”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society. Santiago, Chile, 2015, pp. 19–27. DOI: 10.1109/ICCV.2015.10.
- [28] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. “Layer normalization”. In: *CoRR* abs/1607.06450 (2016). arXiv: 1607.06450 [cs.LG].
- [29] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2016). arXiv: 1512.03385 [cs.CV].
- [30] Dan Hendrycks and Kevin Gimpel. “Gaussian Error Linear Units (GELUs)”. In: *arXiv preprint arXiv:1606.08415* (2016).
- [31] Dan Hendrycks and Kevin Gimpel. “Gaussian Error Linear Units (GELUs)”. In: *arXiv preprint arXiv:1606.08415* (2016).
- [32] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. “Convolutional Neural Networks Using Logarithmic Data Representation”. In: *CoRR* abs/1603.01025 (2016). arXiv: 1603.01025 [cs.LG].
- [33] Pranav Rajpurkar et al. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. 2016. arXiv: 1606.05250 [cs.CL]. URL: <https://arxiv.org/abs/1606.05250>.
- [34] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. “Regularization with stochastic transformations and perturbations for deep semi-supervised learning”. In: *Advances in neural information processing systems*. 2016, pp. 1163–1171.
- [35] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural machine translation of rare words with subword units”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.
- [36] Yonghui Wu et al. “Google’s Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation”. In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144 [cs.CL]. URL: <http://arxiv.org/abs/1609.08144>.
- [37] Denny Britz et al. “Massive exploration of neural machine translation architectures”. In: *CoRR* abs/1703.03906 (2017). arXiv: 1703.03906 [cs.CL].
- [38] Paul F. Christiano et al. “Deep reinforcement learning from human preferences”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*. Ed. by Isabelle Guyon et al. Curran Associates, Inc. Long Beach, CA, USA, Dec. 4–9, 2017, pp. 4299–4307.
- [39] Itay Hubara et al. “Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations”. In: *J. Mach. Learn. Res* 18 (2017), pp. 6869–6898.
- [40] Benoit Jacob et al. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. 2017. arXiv: 1712.05877 [cs.LG].
- [41] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. “Searching for activation functions”. In: *arXiv preprint arXiv:1710.05941* (2017).
- [42] Zhilin Yang, Ruslan Salakhutdinov, and William W. Cohen. *Transfer Learning for Sequence Tagging with Hierarchical Recurrent Networks*. 2017. arXiv: 1703.06345 [cs.CL].

- [43] Andrew L. Beam and Isaac S. Kohane. “Big Data and Machine Learning in Health Care”. In: *JAMA* 319.13 (2018), pp. 1317–1318.
- [44] Hans Buehler et al. “Deep learning and algorithmic trading”. In: *Financial Markets and Portfolio Management* 32.3 (2018), pp. 239–260.
- [45] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*. 2018. arXiv: 1801.06146 [cs.CL].
- [46] Taku Kudo and John Richardson. “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations*. Ed. by Eduardo Blanco and Wei Lu. Brussels, Belgium: Association for Computational Linguistics, 2018.
- [47] Brenden Lake and Marco Baroni. “Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2873–2882.
- [48] Macedo Maia, Siegfried Handschuh, André Freitas, et al. “WWW’18 Open Challenge: Financial Opinion Mining and Question Answering”. In: *Companion Proceedings of WWW* (2018), pp. 1941–1942.
- [49] Pramod Kaushik Mudrakarta et al. “Did the model understand the question?” In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 1896–1906. DOI: 10.18653/v1/P18-1176. URL: <https://aclanthology.org/P18-1176>.
- [50] Matthew E. Peters et al. *Deep Contextualized Word Representations*. arXiv preprint. 2018. URL: <https://arxiv.org/abs/1802.05365>.
- [51] Alec Radford et al. *Improving Language Understanding by Generative Pre-training*. Available online. 2018.
- [52] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. “Self-attention with relative position representations”. In: *CoRR* abs/1803.02155 (2018). arXiv: 1803.02155 [cs.CL].
- [53] Benjamin Shickel et al. “Deep EHR: A survey of recent advances in deep learning techniques for electronic health record (EHR) analysis”. In: *IEEE journal of biomedical and health informatics* 22.5 (2018), pp. 1589–1604.
- [54] Saku Sugawara et al. “What makes reading comprehension questions easier?” In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 4208–4219. DOI: 10.18653/v1/D18-1453. URL: <https://aclanthology.org/D18-1453>.
- [55] Trieu H. Trinh and Quoc V. Le. “A Simple Method for Commonsense Reasoning”. In: *CoRR* abs/1806.02847 (2018). arXiv: 1806.02847 [cs.AI].
- [56] Alex Wang et al. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. In: *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*. Ed. by Tal Linzen, Grzegorz Chrupala, and Afra Alishahi. Association for Computational Linguistics, 2018, pp. 353–355.
- [57] Lilian Weng. “Attention? Attention!” In: *lilianweng.github.io* (2018). URL: <https://lilianweng.github.io/posts/2018-06-24-attention/>.
- [58] Huizhe Wu et al. “Hybrid deep sequential modeling for social text-driven stock prediction”. In: *Proceedings of ACM CIKM*. 2018, pp. 1627–1630.

- [59] Yumo Xu and Shay B Cohen. “Stock movement prediction from tweets and historical prices”. In: *Proceedings of ACL*. 2018, pp. 1970–1979.
- [60] Zhilin Yang et al. “HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering”. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. Brussels, Belgium, 2018, pp. 2369–2380.
- [61] Emily Alsentzer et al. “Publicly available clinical BERT embeddings”. In: *arXiv preprint arXiv:1904.03323* (2019).
- [62] Alexei Baevski and Michael Auli. “Adaptive Input Representations for Neural Language Modeling”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. 2019.
- [63] H. Chen et al. “Single-cell trajectories reconstruction, exploration and mapping of omics data with STREAM”. In: *Nature Communications* 10.1 (2019), p. 1903. DOI: 10.1038/s41467-019-09670-4. URL: <https://www.nature.com/articles/s41467-019-09670-4>.
- [64] Rewon Child et al. “Generating Long Sequences with Sparse Transformers”. In: *CoRR* abs/1904.10509 (2019). arXiv: 1904.10509 [cs.LG].
- [65] Jacob Devlin et al. “Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Vol. 1. NAACL-HLT ’19 Long and Short Papers. Minneapolis, MN, USA: Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: <https://www.aclweb.org/anthology/N19-1423>.
- [66] Li Dong et al. “Unified Language Model Pre-training for Natural Language Understanding and Generation”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019, pp. 13042–13054.
- [67] Aaron Gokaslan, Ellie Pavlick, and Stefanie Tellex. *OpenWebText Corpus*. <http://Skylion007.github.io/OpenWebTextCorpus>. 2019.
- [68] Neil Houlsby et al. *Parameter-Efficient Transfer Learning for NLP*. 2019. arXiv: 1902.00751 [cs.LG].
- [69] Qingyu Jin et al. “PubMedQA: A Dataset for Biomedical Research Question Answering”. In: *Proceedings of EMNLP-IJCNLP* (2019), pp. 2567–2577.
- [70] A. Baki Kocaballi et al. “The Personalization of Conversational Agents in Health Care: Systematic Review”. In: *Journal of Medical Internet Research* 21.11 (2019). DOI: 10.2196/15360. URL: <https://www.jmir.org/2019/11/e15360/>.
- [71] Tom Kwiatkowski et al. “Natural Questions: A Benchmark for Question Answering Research”. In: *Transactions of the Association for Computational Linguistics* 7 (2019). Ed. by Lillian Lee et al., pp. 452–466. DOI: 10.1162/tac1_a_00276. URL: <https://aclanthology.org/Q19-1026>.
- [72] Xiaodong Liu et al. “Multi-task deep neural networks for natural language understanding”. In: *CoRR* abs/1901.11504 (2019). arXiv: 1901.11504 [cs.CL].
- [73] Yinhan Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *arXiv preprint arXiv:1907.11692*. 2019.

- [74] Jason Phang, Thibault Févry, and Samuel R. Bowman. *Sentence Encoders on STILTs: Supplementary Training on Intermediate Labeled-data Tasks*. 2019. arXiv: 1811.01088 [cs.CL].
- [75] Alec Radford et al. *Language Models Are Unsupervised Multitask Learners*. 2019. URL: <https://openai.com/blog/better-language-models/>.
- [76] Sebastian Ruder et al. “Transfer Learning in Natural Language Processing”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*. Ed. by Anoop Sarkar and Michael Strube. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 15–18. DOI: 10.18653/v1/N19-5004. URL: <https://aclanthology.org/N19-5004>.
- [77] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *Proceedings of the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS* (2019), pp. 12–23.
- [78] Noam Shazeer. “Fast Transformer Decoding: One Write-Head is All You Need”. In: *CoRR* abs/1911.02150 (2019). arXiv: 1911.02150 [cs.CL]. URL: <http://arxiv.org/abs/1911.02150>.
- [79] Timothy Smith and Manish Kumar. “Improving fraud detection in financial services through deep learning”. In: *Journal of Financial Crime* 26.4 (2019), pp. 1062–1073.
- [80] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and Policy Considerations for Deep Learning in NLP”. In: *ACL 2019*. 2019.
- [81] Alon Talmor et al. *CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge*. 2019. arXiv: 1811.00937 [cs.CL].
- [82] Rowan Zellers et al. “Defending Against Neural Fake News”. In: *Advances in Neural Information Processing Systems 32*. Ed. by Hanna M. Wallach et al. NeurIPS 2019, December 8-14. Vancouver, BC, Canada: NeurIPS, 2019, pp. 9051–9062.
- [83] Biao Zhang and Rico Sennrich. “Root Mean Square Layer Normalization”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019, pp. 12360–12371.
- [84] Alex Zhavoronkov et al. “Deep learning enables rapid identification of potent DDR1 kinase inhibitors”. In: *Nature Biotechnology* 37 (2019), pp. 1038–1040. DOI: 10.1038/d41573-019-00170-0.
- [85] Daniel M Ziegler et al. “Fine-tuning language models from human preferences”. In: *CoRR* abs/1909.08593 (2019).
- [86] Daniel Adiwardana et al. *Towards a Human-like Open-Domain Chatbot*. 2020. arXiv: 2001.09977 [cs.CL].
- [87] Jason Baumgartner et al. “The Pushshift Reddit Dataset”. In: *Proceedings of the Fourteenth International AAAI Conference on Web and Social Media*. ICWSM 2020, Held Virtually. Atlanta, Georgia, USA: AAAI Press, 2020, pp. 830–839.
- [88] Tom B. Brown et al. *Language Models Are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].
- [89] Suchin Gururangan et al. *Don’t Stop Pretraining: Adapt Language Models to Domains and Tasks*. 2020. arXiv: 2004.10964 [cs.CL].
- [90] Tom Henighan et al. “Scaling Laws for Autoregressive Generative Modeling”. In: *arXiv preprint arXiv:2010.14701* (2020).

- [91] Ari Holtzman et al. “The Curious Case of Neural Text Degeneration”. In: *8th International Conference on Learning Representations, ICLR 2020* (2020). OpenReview.net. URL: <https://openreview.net/forum?id=rygGQyrFvH>.
- [92] Michael Jones et al. “Ethical considerations for AI in finance”. In: *AI & Society* 35.1 (2020), pp. 287–300.
- [93] Jared Kaplan et al. “Scaling Laws for Neural Language Models”. In: *CoRR* abs/2001.08361 (2020).
- [94] Mike Lewis et al. “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 2020, pp. 7871–7880. URL: <https://www.aclweb.org/anthology/2020.acl-main.703>.
- [95] Jiazheng Li et al. “MAEC: A Multimodal Aligned Earnings Conference Call Dataset for Financial Risk Prediction”. In: *Proceedings of ACM CIKM*. 2020, pp. 3063–3070.
- [96] Jin Li, Scott Spangler, and Yue Yu. “Natural language processing in risk management and compliance”. In: *Journal of Risk Management in Financial Institutions* 13.2 (2020), pp. 158–175.
- [97] Lizi Liu et al. “Understanding the difficulty of training transformers”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. 2020, pp. 5747–5763.
- [98] Dominique Mariko, Hanna Abi Akl, Estelle Labidurie, et al. “The financial document causality detection shared task (fincausal 2020)”. In: *Proceedings of the Workshop on FNP-FNS*. 2020, pp. 23–32.
- [99] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *Journal of Machine Learning Research* 21 (2020), 140:1–140:67.
- [100] Noam Shazeer. “GLU Variants Improve Transformer”. In: *arXiv preprint arXiv:2002.05202* (2020).
- [101] Ruibo Xiong et al. “On Layer Normalization in the Transformer Architecture”. In: *ICML*. 2020.
- [102] Manzil Zaheer et al. “Big Bird: Transformers for Longer Sequences”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual*. 2020.
- [103] Anna Aghajanyan et al. “Muppet: Massive multi-task representations with pre-finetuning”. In: *CoRR* abs/2109.08668 (2021). arXiv: 2109.08668 [cs.CL].
- [104] Amanda Askell et al. “A General Language Assistant as a Laboratory for Alignment”. In: *CoRR* abs/2112.00861 (2021).
- [105] James Austin et al. “Program synthesis with large language models”. In: *CoRR* abs/2108.07732 (2021).
- [106] Emily M Bender et al. “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?” In: *FAccT ’21* (2021).
- [107] Nicholas Carlini et al. “Extracting training data from large language models”. In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. 2021, pp. 2633–2650.
- [108] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. arXiv preprint arXiv:2107.03374. 2021.

- [109] Ming Ding et al. “CogView: Mastering Text-to-Image Generation via Transformers”. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual*. 2021, pp. 19822–19835.
- [110] William Fedus, Barret Zoph, and Noam Shazeer. “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity”. In: *J. Mach. Learn. Res* (2021), pp. 1–40.
- [111] Leo Gao et al. “The Pile: An 800GB Dataset of Diverse Text for Language Modeling”. In: *CoRR* abs/2101.00027 (2021). arXiv: 2101.00027 [cs.CL].
- [112] Daniela Gerz et al. “Multilingual and cross-lingual intent detection from spoken data”. In: *Proceedings of EMNLP*. 2021, pp. 7468–7475.
- [113] Dan Hendrycks et al. “Measuring Massive Multitask Language Understanding”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2021.
- [114] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL].
- [115] Z. Kenton et al. “Alignment of language agents”. In: *CoRR* abs/2103.14659 (2021).
- [116] Michael M. Krell et al. “Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance”. In: *CoRR* abs/2107.02027 (2021). arXiv: 2107.02027 [cs.CL].
- [117] Yuxuan Lai et al. “Why machine reading comprehension models learn shortcuts?” In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Online: Association for Computational Linguistics, 2021, pp. 989–1002. DOI: 10.18653/v1/2021.findings-acl.85. URL: <https://aclanthology.org/2021.findings-acl.85>.
- [118] Brian Lester, Rami Al-Rfou, and Noah Constant. *The Power of Scale for Parameter-Efficient Prompt Tuning*. 2021. arXiv: 2104.08691 [cs.CL].
- [119] Xiang Lisa Li and Percy Liang. *Prefix-Tuning: Optimizing Continuous Prompts for Generation*. 2021. arXiv: 2101.00190 [cs.CL].
- [120] Zhi Li, Qiang Zhang, Qi Dou, et al. “A survey on deep learning in medical image analysis”. In: *Medical image analysis* 67 (2021), p. 101813.
- [121] Or Lieber et al. “Jurassic-1: Technical details and evaluation”. In: *White Paper. AI21 Labs* 1 (2021).
- [122] Pengfei Liu et al. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*. arXiv preprint arXiv:2107.13586. 2021. URL: <https://arxiv.org/abs/2107.13586>.
- [123] Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. *A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers*. 2021. arXiv: 2106.15772 [cs.AI].
- [124] R. Nakano et al. “WebGPT: Browser-assisted Question-Answering with Human Feedback”. In: *CoRR* abs/2112.09332 (2021).
- [125] Sharan Narang et al. “Do Transformer Modifications Transfer Across Implementations and Applications?” In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. 2021, pp. 5758–5773.
- [126] A. Olmo, S. Sreedharan, and S. Kambhampati. “GPT3-toPlan: Extracting Plans from Text using GPT-3”. In: *FinPlan 2021* (2021), p. 24.

- [127] Arpan Pal, Aniruddha Kundu, and Rajdeep Chakraborty. “Enhancing customer service through AI-driven virtual assistants in the banking sector”. In: *Journal of Banking and Financial Technology* 5.1 (2021), pp. 1–12.
- [128] Baolin Peng, Xiang Li, and Percy Liang. “Random Feature Attention”. In: *CoRR* abs/2106.14448 (2021). arXiv: 2106.14448 [cs.CL].
- [129] Guanghui Qin and Jason Eisner. “Learning how to ask: Querying LMs with mixtures of soft prompts”. In: *CoRR* abs/2104.06599 (2021). arXiv: 2104.06599 [cs.CL].
- [130] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].
- [131] Jack W. Rae et al. “Scaling language models: Methods, analysis & insights from training Gopher”. In: *CoRR* abs/2112.11446 (2021). arXiv: 2112.11446 [cs.CL].
- [132] Aditya Ramesh et al. *Zero-Shot Text-to-Image Generation*. 2021. arXiv: 2102.12092 [cs.CV].
- [133] Ankur Sinha and Tanmay Khandait. “Impact of news on the commodity market: Dataset and results”. In: *Proceedings of FICC*. 2021, pp. 589–601.
- [134] Jianlin Su et al. “RoFormer: Enhanced Transformer with Rotary Position Embedding”. In: *arXiv preprint arXiv:2104.09864* (2021).
- [135] Alex Tamkin et al. “Understanding the Capabilities, Limitations, and Societal Impact of Large Language Models”. In: *arXiv preprint arXiv:2102.02503* (2021).
- [136] Yi Tay et al. “Long Range Arena: A Benchmark for Efficient Transformers”. In: *CoRR* abs/2011.04006 (2021). arXiv: 2011.04006 [cs.CL].
- [137] Katerina Tsimpoukelli et al. “Frozen in Time: Temporal Contextualization for In-Context Learning”. In: *CoRR* abs/2109.14867 (2021). arXiv: 2109.14867 [cs.CL].
- [138] J. Wang et al. “Milvus: A Purpose-Built Vector Data Management System”. In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2614–2627.
- [139] Weihua Zeng et al. “Pangu- α : Large-scale autoregressive pretrained Chinese language models with auto-parallel computation”. In: *CoRR* abs/2104.12369 (2021). arXiv: 2104.12369 [cs.CL].
- [140] Jun Zhang et al. “Medical image analysis with artificial intelligence”. In: *IEEE Transactions on Biomedical Engineering* 68.5 (2021), pp. 1375–1379.
- [141] Zihao Zhao et al. “Calibrate Before Use: Improving Few-shot Performance of Language Models”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 12697–12706. URL: <https://proceedings.mlr.press/v139/zhao21c.html>.
- [142] Xinyi Zheng et al. “Global Table Extractor (GTE): A Framework for Joint Table Identification and Cell Structure Recognition Using Visual Context”. In: *Proceedings of the IEEE/CVF WACV*. 2021, pp. 697–706.
- [143] Zhihan Zhou, Liqian Ma, and Han Liu. “Trade the event: Corporate events detection for news-based event-driven trading”. In: *Findings of ACL-IJCNLP*. 2021, pp. 2114–2124.
- [144] E. Akyürek et al. “What Learning Algorithm Is In-context Learning? Investigations with Linear Models”. In: *CoRR* abs/2211.15661 (2022).

- [145] Jean-Baptiste Alayrac et al. “Flamingo: a Visual Language Model for Few-Shot Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: <https://openreview.net/forum?id=EbMuimAbPbs>.
- [146] Öğuzhan Aydin and Emre Karaarslan. “OpenAI ChatGPT Generated Literature Review: Digital Twin in Healthcare”. In: *SSRN Electronic Journal* (2022). Please replace ”number” with the actual abstract number. URL: <https://ssrn.com/abstract=number>.
- [147] Sebastian H. Bach et al. “PromptSource: An Integrated Development Environment and Repository for Natural Language Prompts”. In: *CoRR* abs/2202.12108 (2022). arXiv: 2202.12108 [cs.CL].
- [148] Yuntao Bai et al. *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. 2022. arXiv: 2204.05862 [cs.CL].
- [149] Amir Bar et al. “Visual Prompting via Image Inpainting”. In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 25005–25017.
- [150] Nicholas Carlini et al. “Quantifying memorization across neural language models”. In: *CoRR* abs/2202.12488 (2022). arXiv: 2202.12488 [cs.CL].
- [151] Stephanie C. Y. Chan et al. *Data Distributional Properties Drive Emergent In-Context Learning in Transformers*. 2022. arXiv: 2205.05055 [cs.LG].
- [152] Mingda Chen et al. “Improving In-Context Few-Shot Learning via Self-Supervised Training”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz. Seattle, United States: Association for Computational Linguistics, 2022, pp. 3558–3573. DOI: 10.18653/v1/2022.naacl-main.260. URL: <https://aclanthology.org/2022.naacl-main.260>.
- [153] Zhiyu Chen et al. “ConvFinQA: Exploring the Chain of Numerical Reasoning in Conversational Finance Question Answering”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2022, pp. 6279–6292.
- [154] Zhiyu Chen et al. “FinQA: A Dataset of Numerical Reasoning Over Financial Data”. In: (2022). Presumed publication year and citation style as ”2022a”, specifics such as journal name, volume, issue, pages, and DOI are not provided and should be added.
- [155] Aakanksha Chowdhery et al. “PaLM: Scaling Language Modeling with Pathways”. In: *CoRR* abs/2204.02311 (2022).
- [156] H. W. Chung et al. “Scaling Instruction-Finetuned Language Models”. In: *CoRR* abs/2210.11416 (2022).
- [157] A. Creswell, M. Shanahan, and I. Higgins. “Selection-inference: Exploiting large language models for interpretable logical reasoning”. In: *CoRR* abs/2205.09712 (2022).
- [158] D. Dai et al. “Why can GPT learn in-context? language models secretly perform gradient descent as meta-optimizers”. In: (2022).
- [159] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. 2022. arXiv: 2205.14135 [cs.LG].
- [160] Tri Dao et al. “Hungry Hungry Hippos: Towards Language Modeling with State Space Models”. In: *CoRR* abs/2212.14052 (2022). DOI: 10.48550/arXiv.2212.14052. URL: <https://doi.org/10.48550/arXiv.2212.14052>.
- [161] Nan Du et al. “GLAM: Efficient Scaling of Language Models with Mixture-of-Experts”. In: *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*. 2022, pp. 5547–5569.

- [162] Hao Fu Yao; Peng and Tushar Khot. “How does GPT Obtain its Ability? Tracing Emergent Abilities of Language Models to their Sources”. In: *Yao Fu’s Notion* (2022). URL: \url{"https://yaofu.notion.site/How-does-GPT-Obtain-its-Ability-Tracing-Emergent-Abilities-of-Language-Models-to-their-Sources-b9a57ac0fcf74f30a1"}.
- [163] Y. Fu et al. “Complexity-based prompting for multi-step reasoning”. In: *CoRR* abs/2210.00720 (2022).
- [164] L. Gao et al. “PAL: program-aided language models”. In: *CoRR* abs/2211.10435 (2022).
- [165] A. Glaese et al. “Improving Alignment of Dialogue Agents via Targeted Human Judgements”. In: *CoRR* abs/2209.14375 (2022).
- [166] Hila Gonen et al. *Demystifying Prompts in Language Models via Perplexity Estimation*. 2022. arXiv: 2212.04037 [cs.CL].
- [167] Albert Gu, Karan Goel, and Christopher Ré. “Efficiently Modeling Long Sequences with Structured State Spaces”. In: *The Tenth International Conference on Learning Representations*. Accessed: 2024-04-13. 2022. URL: <https://openreview.net/forum?id=uYLFoz1vlAC>.
- [168] S. Hao et al. “Structured prompting: Scaling in-context learning to 1,000 examples”. In: *CoRR* abs/2206.08082 (2022).
- [169] Yaru Hao et al. *Language Models are General-Purpose Interfaces*. arXiv preprint arXiv:2206.06336. 2022. URL: <https://arxiv.org/abs/2206.06336>.
- [170] Junxian He et al. *Towards a Unified View of Parameter-Efficient Transfer Learning*. 2022. arXiv: 2110.04366 [cs.CL].
- [171] Daniel Hernandez et al. “Scaling laws and interpretability of learning from repeated data”. In: *CoRR* abs/2205.10487 (2022). arXiv: 2205.10487 [cs.LG].
- [172] Jan Hoffmann et al. “Training Compute-Optimal Large Language Models”. In: *CoRR* abs/2203.15556 (2022).
- [173] Or Honovich et al. *Instruction Induction: From Few Examples to Natural Language Task Descriptions*. 2022. arXiv: 2205.10782 [cs.CL].
- [174] Srinivasan Iyer et al. “OPT-IML: Scaling Language Model Instruction Meta Learning Through the Lens of Generalization”. In: *CoRR* abs/2212.12017 (2022). arXiv: 2212.12017 [cs.CL].
- [175] T. Khot et al. “Decomposed prompting: A modular approach for solving complex tasks”. In: *CoRR* abs/2210.02406 (2022). <https://doi.org/10.48550/arXiv.2210.02406>.
- [176] H. J. Kim et al. “Self-generated in-context learning: Leveraging auto-regressive language models as a demonstration generator”. In: *CoRR* abs/2206.08082 (2022).
- [177] Sung Kim. *Replace Grammarly Premium with OpenAI ChatGPT*. 2022. URL: <https://medium.com/geekculture/replace-grammarly-premium-with-openai-chatgpt-320049179c79>.
- [178] Anastasia Krithara et al. *BioASQ-QA: A manually curated corpus for biomedical question answering*. 2022.
- [179] Hervé Laurençon et al. “The BigScience ROOTS Corpus: A 1.6 TB Composite Multilingual Dataset”. In: *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. NeurIPS. 2022.

- [180] Teven Le Scao et al. “What language model to train if you have one million GPU hours?” In: *Findings of the Association for Computational Linguistics: EMNLP 2022*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, 2022, pp. 765–782. URL: <https://aclanthology.org/2022.findings-emnlp.54>.
- [181] Kenton Lee et al. “Deduplicating training data makes language models better”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. 2022, pp. 8424–8445.
- [182] Aitor Lewkowycz et al. “Solving quantitative reasoning problems with language models”. In: *CoRR* abs/2206.14858 (2022).
- [183] Xiang Lisa Li and Percy Liang. “P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks”. In: *CoRR* abs/2202.12108 (2022). arXiv: 2202.12108 [cs.CL].
- [184] Y. Li et al. “Competition-level code generation with AlphaCode”. In: *Science* (2022).
- [185] Percy Liang et al. *Holistic Evaluation of Language Models*. 2022. DOI: 10.48550/arXiv.2211.09110. URL: <https://doi.org/10.48550/arXiv.2211.09110>.
- [186] J. Liu et al. “What makes good in-context examples for gpt-3?” In: *Proceedings of Deep Learning Inside Out (DeeLIO): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, at ACL 2022*. Dublin, Ireland and Online, 2022, pp. 100–114.
- [187] Lizi Liu et al. “Fast and Memory-Efficient Attention with FlashAttention-2”. In: *CoRR* abs/2205.14135 (2022). arXiv: 2205.14135 [cs.LG].
- [188] Xiao Liu et al. *P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks*. 2022. arXiv: 2110.07602 [cs.CL].
- [189] Lefteris Loukas et al. “Finer: Financial numeric entity recognition for xbrl tagging”. In: *Proceedings of ACL* (2022), pp. 4419–4431.
- [190] Y. Lu et al. “Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland, 2022, pp. 8086–8098.
- [191] Renqian Luo et al. “BioGPT: Generative Pre-trained Transformer for Biomedical Text Generation and Mining”. In: *Briefings in Bioinformatics* 23.6 (2022). DOI: 10.1093/bib/bbac409. URL: <https://doi.org/10.1093%2Fbib%2Fbbc409>.
- [192] Aman Madaan and Alireza Yazdanbakhsh. “Text and patterns: For effective chain of thought, it takes two to tango”. In: *CoRR* abs/2209.07686 (2022). arXiv: 2209.07686 [cs.CL].
- [193] Alexander Magister, Polina Kuznetsova, and Sergey Kuznetsov. “Teaching Language Models to Learn in Context”. In: *CoRR* abs/2205.10625 (2022). arXiv: 2205.10625 [cs.CL].
- [194] Puneet Mathur et al. “Monopoly: Financial prediction from monetary policy conference videos using multimodal cues”. In: *Proceedings of ACM MM*. 2022, pp. 2276–2285.
- [195] Hrushikesh Mehta et al. “Long Range Language Modeling via Gated State Spaces”. In: *CoRR* abs/2206.13947 (2022). DOI: 10.48550/arXiv.2206.13947. URL: <https://doi.org/10.48550/arXiv.2206.13947>.

- [196] Sewon Min et al. “MetaICL: Learning to Learn In Context”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz. Seattle, United States: ”Association for Computational Linguistics”, 2022, ”2791–2809”. DOI: “10 . 18653 / v1 / 2022 . naacl - main . 201”. URL: “<https://aclanthology.org/2022.naacl-main.201>”.
- [197] Sewon Min et al. “Noisy Channel Language Model Prompting for Few-Shot Text Classification”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, 2022, pp. 5316–5330. DOI: 10 . 18653 / v1 / 2022 . acl - long . 365. URL: <https://aclanthology.org/2022.acl-long.365>.
- [198] Sewon Min et al. “Rethinking the Role of Demonstrations: What Makes In-context Learning Work?” In: *CoRR* abs/2202.12837 (2022). URL: <https://arxiv.org/abs/2202.12837>.
- [199] Swaroop Mishra et al. “Cross-task generalization via natural language crowdsourcing instructions”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. ACL. Dublin, Ireland, 2022, pp. 3470–3487. URL: <https://aclanthology.org/2022.acl-long.243>.
- [200] Niklas Muennighoff et al. “Crosslingual Generalization Through Multitask Finetuning”. In: *CoRR* abs/2211.01786 (2022). URL: <https://arxiv.org/abs/2211.01786>.
- [201] Rajdeep Mukherjee et al. “ECTSum: A New Benchmark Dataset for Bullet Point Summarization of Long Earnings Call Transcripts”. In: *Proceedings of EMNLP*. 2022, pp. 10893–10906.
- [202] J. J. Nay. “Law informs code: A legal informatics approach to aligning artificial intelligence with humans”. In: *CoRR* abs/2209.13020 (2022). arXiv: 2209 . 13020 [cs.CY]. URL: <https://arxiv.org/abs/2209.13020>.
- [203] Erik Nijkamp et al. “CodeGen: An Open Large Language Model for Code with Multi-turn Program Synthesis”. In: *arXiv preprint arXiv:2203.13474* (2022).
- [204] Catherine Olsson et al. *In-context Learning and Induction Heads*. 2022. arXiv: 2209 . 11895 [cs.LG].
- [205] L. Ouyang et al. “Training Language Models to Follow Instructions with Human Feedback”. In: *CoRR* abs/2203.02155 (2022).
- [206] Ofir Press, Noah A. Smith, and Mike Lewis. “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation”. In: *The Tenth International Conference on Learning Representations*. Accessed: 2024-04-13. 2022. URL: <https://openreview.net/forum?id=JZJ9Zz1vZ6>.
- [207] Jing Qian et al. *Limitations of Language Models in Arithmetic and Symbolic Induction*. 2022. arXiv: 2208 . 05051 [cs.CL].
- [208] O. Rubin, J. Herzig, and J. Berant. “Learning to retrieve prompts for in-context learning”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*. Seattle, WA, United States, 2022, pp. 2655–2671.
- [209] Victor Sanh et al. “Multitask prompted training enables zero-shot task generalization”. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022* (2022). OpenReview.net.

- [210] Soumya Sharma et al. “Finred: A dataset for relation extraction in financial domain”. In: *Companion Proceedings of WWW*. 2022, pp. 595–597.
- [211] Seongjin Shin et al. “On the Effect of Pretraining Corpora on In-context Learning by a Large-scale Language Model”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz. Seattle, United States: Association for Computational Linguistics, 2022, pp. 5168–5186. DOI: 10.18653/v1/2022.nacl-main.380. URL: <https://aclanthology.org/2022.nacl-main.380>.
- [212] Ishaan Singh et al. “ProgPrompt: Generating Situated Robot Task Plans Using Large Language Models”. In: *CoRR* abs/2209.11302 (2022).
- [213] Karan Singhal et al. “Large language models encode clinical knowledge”. In: *arXiv preprint arXiv:2212.13138* (2022).
- [214] Samyam Smith et al. “Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model”. In: *CoRR* abs/2201.11990 (2022).
- [215] Taylor Sorensen et al. “An Information-theoretic Approach to Prompt Engineering Without Ground Truth Labels”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, 2022, pp. 819–862. DOI: 10.18653/v1/2022.acl-long.60. URL: <https://aclanthology.org/2022.acl-long.60>.
- [216] Yejun Soun et al. “Accurate stock movement prediction with self-supervised learning from sparse noisy tweets”. In: *IEEE Big Data*. 2022, pp. 1691–1700.
- [217] T. Susnjak. “ChatGPT: The end of online exam integrity?” In: *CoRR* abs/2212.09292 (2022). URL: <https://arxiv.org/abs/2212.09292>.
- [218] Mirac Suzgun et al. *Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them*. 2022. arXiv: 2210.09261 [cs.CL].
- [219] Tian Tang et al. “MVP: Multi-Task Supervised Pre-training for Natural Language Generation”. In: *CoRR* abs/2206.12131 (2022). arXiv: 2206.12131 [cs.CL].
- [220] Ross Taylor et al. *Galactica: A Large Language Model for Science*. <http://arxiv.org/abs/2211.09085>. arXiv:2211.09085. Nov. 2022.
- [221] Romal Thoppilan et al. “LaMDA: Language Models for Dialog Applications”. In: *CoRR* abs/2201.08239 (2022). arXiv: 2201.08239 [cs.CL].
- [222] Daniel Trautmann, Aleksandra Petrova, and Frank Schilder. “Legal prompt engineering for multilingual legal judgement prediction”. In: *CoRR* abs/2212.02199 (2022). arXiv: 2212.02199. URL: <https://arxiv.org/abs/2212.02199>.
- [223] H. Trivedi et al. “Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions”. In: *arXiv preprint arXiv:2212.10509* (2022).
- [224] Boshi Wang, Xiang Deng, and Huan Sun. “Iteratively Prompt Pre-trained Language Models for Chain of Thought”. In: *Proceedings of The 2022 Conference on Empirical Methods for Natural Language Processing (EMNLP)*. Online and in-person event, 2022.
- [225] Haoyuan Wang et al. “DeepNet: Scaling Transformers to 1,000 Layers”. In: *CoRR* abs/2203.00555 (2022). arXiv: 2203.00555 [cs.CL].
- [226] X. Wang et al. “Rationale-augmented ensembles in language models”. In: *CoRR* abs/2206.02336 (2022).

- [227] X. Wang et al. “Self-consistency improves chain of thought reasoning in language models”. In: *arXiv preprint arXiv:2203.11171* (2022).
- [228] Yada Wang et al. “Self-Instruct: Aligning Language Model with Self Generated Instructions”. In: *CoRR* abs/2212.10560 (2022).
- [229] Yada Wang et al. “Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks”. In: *CoRR* abs/2209.13107 (2022). arXiv: 2209 . 13107 [cs.CL].
- [230] J. Wei et al. “Chain of thought prompting elicits reasoning in large language models”. In: *CoRR* abs/2201.11903 (2022).
- [231] J. Wei et al. “Fine-tuned Language Models are Zero-shot Learners”. In: *The Tenth International Conference on Learning Representations, ICLR 2022*. OpenReview.net. Virtual Event, 2022.
- [232] Jason Wei et al. *Emergent Abilities of Large Language Models*. 2022. arXiv: 2206 . 07682 [cs.CL].
- [233] Zhiyong Wu et al. “Self-Adaptive In-Context Learning”. In: (2022). Provide additional details such as the journal name, volume, issue, pages, and DOI if available.
- [234] Sang Michael Xie et al. “An Explanation of In-context Learning as Implicit Bayesian Inference”. In: *International Conference on Learning Representations*. 2022. URL: <https://openreview.net/forum?id=RdJVFCjUMI>.
- [235] Frank F. Xu et al. “A Systematic Evaluation of Large Language Models of Code”. In: *MAPSPLDI*. 2022.
- [236] S. Yao et al. “React: Synergizing reasoning and acting in language models”. In: *CoRR* abs/2210.03629 (2022).
- [237] Kang Min Yoo et al. *Ground-Truth Labels Matter: A Deeper Look into Input-Label Demonstrations*. 2022. arXiv: 2205 . 12685 [cs.CL].
- [238] W. Yu et al. “Generate rather than retrieve: Large language models are strong context generators”. In: *arXiv preprint arXiv:2209.10063* (2022).
- [239] Ailing Zeng et al. *GLM-130B: An Open Bilingual Pre-trained Model*. 2022. arXiv: 2210 . 02414 [cs.CL].
- [240] Biao Zhang et al. “Examining scaling and transfer of language model architectures for machine translation”. In: *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*. 2022, pp. 26176–26192.
- [241] Sheng Zhang et al. “OPT: open pre-trained transformer language models”. In: *CoRR* abs/2205.01068 (2022).
- [242] Yiming Zhang, Shi Feng, and Chenhao Tan. *Active Example Selection for In-Context Learning*. 2022. arXiv: 2211 . 04486 [cs.CL].
- [243] Z. Zhang et al. “Automatic chain of thought prompting in large language models”. In: *CoRR* abs/2210.03493 (2022).
- [244] D. Zhou et al. “Least-to-most prompting enables complex reasoning in large language models”. In: *CoRR* abs/2205.10625 (2022).
- [245] Joshua Ainslie et al. *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. 2023. arXiv: 2305 . 13245 [cs.CL].
- [246] Aarohi Srivastava et al. *Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models*. 2023. arXiv: 2206 . 04615 [cs.CL].

- [247] Raymond Li et al. *StarCoder: may the source be with you!* 2023. arXiv: 2305.06161 [cs.CL].
- [248] M. M. Amin, E. Cambria, and B. W. Schuller. “Will Affective Computing Emerge from Foundation Models and General AI? A First Evaluation on ChatGPT”. In: *CoRR* abs/2303.03186 (2023). arXiv: 2303.03186 [cs.CL]. URL: <https://arxiv.org/abs/2303.03186>.
- [249] Shengnan An et al. “How Do In-context Examples Affect Compositional Generalization?” In: *CoRR* abs/2305.04835 (2023). URL: <https://arxiv.org/abs/2305.04835>.
- [250] A. Asai et al. “Self-RAG: Learning to retrieve, generate, and critique through self-reflection”. In: *arXiv preprint arXiv:2310.11511* (2023).
- [251] A. Azaria, R. Azoulay, and S. Reches. “ChatGPT is a Remarkable Tool – For Experts”. In: *CoRR* abs/2306.03102 (2023). arXiv: 2306.03102 [cs.CL]. URL: <https://arxiv.org/abs/2306.03102>.
- [252] Dave Bergmann. *What Is Semi-Supervised Learning?* IBM. 2023. URL: <https://www.ibm.com/cloud/learn/semi-supervised-learning> (visited on 12/12/2023).
- [253] Andrew Blair-Stanek, Nils Holzenberger, and Benjamin Van Durme. “Can GPT-3 perform statutory reasoning?” In: *CoRR* abs/2302.06100 (2023). arXiv: 2302.06100. URL: <https://arxiv.org/abs/2302.06100>.
- [254] Elliot Bolton et al. *BioMedLM*. <https://github.com/stanford-crfm/BioMedLM>. 2023.
- [255] O. O. Buruk. “Academic Writing with GPT-3.5: Reflections on Practices, Efficacy and Transparency”. In: *CoRR* abs/2304.11079 (2023). arXiv: 2304.11079 [cs.CL]. URL: <https://arxiv.org/abs/2304.11079>.
- [256] Yihan Cao et al. *Instruction Mining: When Data Mining Meets Large Language Model Finetuning*. 2023. arXiv: 2307.06290 [cs.CL].
- [257] Dong Chen et al. “Data-Juicer: A One-Stop Data Processing System for Large Language Models”. In: *arXiv preprint arXiv:2305.13169* (2023).
- [258] H. Chen et al. “Maybe Only 0.5% Data Is Needed: A Preliminary Exploration of Low Training Data Instruction Tuning”. In: *arXiv preprint arXiv:2305.09246* (2023).
- [259] Wenhui Chen et al. “Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks”. In: (2023). arXiv: 2211.12588 [cs.CL]. URL: <https://arxiv.org/abs/2211.12588>.
- [260] Z. Chen et al. “Chatcot: Tool-augmented chain-of-thought reasoning on chat-based large language models”. In: *CoRR* abs/2305.14323 (2023).
- [261] Long Cheng, Xiang Li, and Lidong Bing. “Is GPT-4 a Good Data Analyst?” In: *CoRR* abs/2305.15038 (2023). arXiv: 2305.15038 [cs.LG]. URL: <https://arxiv.org/abs/2305.15038>.
- [262] X. Cheng et al. “Lift yourself up: Retrieval-augmented text generation with self memory”. In: *arXiv preprint arXiv:2305.02437* (2023).
- [263] W.-L. Chiang et al. *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%*Chat-GPT Quality*. [Online]. Available: <https://vicuna.lmsys.org>. 2023.
- [264] Jinho H. Choi et al. “ChatGPT goes to law school”. In: (2023). Accessed: 2024-02-14. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=number.
- [265] Qingxiu Dong et al. *A Survey on In-context Learning*. 2023. arXiv: 2301.00234 [cs.CL].

- [266] Nouha Dziri et al. “Faith and fate: Limits of transformers on compositionality”. In: *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*. 2023. URL: <https://openreview.net/forum?id=Fkckkr3ya8>.
- [267] Yao Fu. *A Closer Look at Large Language Models: Emergent Abilities*. <https://www.notion.so/yaofu/A-Closer-Look-at-Large-Language-Models-Emergent-Abilities-493876b55df5479d80686f68a1abd72f>. Accessed: 2023-07-14. 2023.
- [268] Shivam Garg et al. *What Can Transformers Learn In-Context? A Case Study of Simple Function Classes*. 2023. arXiv: 2208.01066 [cs.CL].
- [269] Guillaume Gendron et al. “Large language models are not abstract reasoners”. In: *arXiv preprint arXiv:2305.19555* (2023).
- [270] Yuxian Gu et al. *Pre-training to Learn in Context*. arXiv preprint arXiv:2305.09137. 2023. URL: <https://arxiv.org/abs/2305.09137>.
- [271] Lin Guan et al. “Leveraging pre-trained large language models to construct and utilize world models for model-based task planning”. In: *Thirty-seventh Conference on Neural Information Processing Systems* (2023). URL: <https://openreview.net/forum?id=zDbsSscmuj>.
- [272] Binbin Guo et al. “How close is ChatGPT to human experts? Comparison corpus, evaluation, and detection”. In: *CoRR* abs/2301.07597 (2023). arXiv: 2301.07597 [cs.CL].
- [273] Michael Hahn and Navin Goyal. *A Theory of Emergent In-Context Learning as Implicit Structure Induction*. 2023. arXiv: 2303.07971 [cs.CL].
- [274] Michał Haman and Marcin Skolnik. “Using ChatGPT to Conduct a Literature Review”. In: *Accountability in Research* (2023).
- [275] S. Hao et al. “Reasoning with language model is planning with world model”. In: *CoRR* abs/2305.14992 (2023).
- [276] Md Mahadi Hassan, Richard A. Knipper, and Shakked K. K. Santu. “ChatGPT as Your Personal Data Scientist”. In: *CoRR* abs/2305.13657 (2023). arXiv: 2305.13657 [cs.LG]. URL: <https://arxiv.org/abs/2305.13657>.
- [277] Zhiqiang Hu et al. *LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models*. 2023. arXiv: 2304.01933 [cs.CL].
- [278] Shaohan Huang et al. *Language Is Not All You Need: Aligning Perception with Language Models*. arXiv preprint arXiv:2302.14045. 2023. URL: <https://arxiv.org/abs/2302.14045>.
- [279] S. I. M. Hussam Alkaissi. “Artificial Hallucinations in ChatGPT: Implications in Scientific Writing”. In: *PubMed* (2023). Available on PubMed. URL: https://pubmed.ncbi.nlm.nih.gov/ARTICLE_ID.
- [280] I. Ilin. *Advanced RAG Techniques: An Illustrated Overview*. Accessed: 2024-12-24. 2023. URL: <https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6>.
- [281] Yuxuan Ji et al. “Towards Better Instruction Following Language Models for Chinese: Investigating the Impact of Training Data and Evaluation”. In: *arXiv preprint arXiv:2304.07854* (2023).
- [282] Rasmus Jørgensen et al. “MultiFin: A Dataset for Multilingual Financial NLP”. In: *Findings of the European Chapter of the Association for Computational Linguistics (EACL)*. 2023, pp. 864–879.

- [283] Subbarao Kambhampati et al. “On the role of large language models in planning”. In: *arXiv preprint arXiv:2307.00000* (2023).
- [284] G. Kim et al. “Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models”. In: *arXiv preprint arXiv:2310.14696* (2023).
- [285] Takeshi Kojima et al. *Large Language Models are Zero-Shot Reasoners*. 2023. arXiv: 2205.11916 [cs.CL].
- [286] Andreas Kopf et al. “OpenAssistant Conversations—Democratizing Large Language Model Alignment”. In: *arXiv preprint arXiv:2304.07327* (2023).
- [287] M. Kosinski. “Theory of Mind May Have Spontaneously Emerged in Large Language Models”. In: *CoRR* abs/2302.02083 (2023). arXiv: 2302.02083 [cs.CL]. URL: <https://arxiv.org/abs/2302.02083>.
- [288] Stanford AI Lab. *Understanding In-Context Learning*. 2023. URL: <https://ai.stanford.edu/blog/understanding-incontext/>.
- [289] Jean Lee et al. “StockEmotions: Discover Investor Emotions for Financial Sentiment Analysis and Multivariate Time Series”. In: *AAAI-24 Bridge*. 2023.
- [290] Mukai Li et al. “Contextual Prompting for In-Context Learning”. In: *arXiv preprint arXiv:2302.04931* (2023).
- [291] X. Li et al. “Chain of knowledge: A framework for grounding large language models with structured knowledge bases”. In: *arXiv preprint arXiv:2305.13269* (2023).
- [292] Xianzhi Li et al. *Are ChatGPT and GPT-4 General-Purpose Solvers for Financial Text Analytics? A Study on Several Typical Tasks*. 2023. arXiv: 2305.05862 [cs.CL].
- [293] Xiaonan Li and Xipeng Qiu. *Finding Supporting Examples for In-Context Learning*. arXiv preprint arXiv:2302.13539. 2023. URL: <https://arxiv.org/abs/2302.13539>.
- [294] Xiaonan Li and Xipeng Qiu. *MoT: Memory-of-Thought Enables ChatGPT to Self-Improve*. 2023. arXiv: 2305.05181 [cs.CL].
- [295] Yifei Li et al. “Making Large Language Models Better Reasoners with Step-Aware Verifier”. In: (2023). arXiv: 2206.02336 [cs.CL].
- [296] Yingcong Li et al. *Transformers as Algorithms: Generalization and Stability in In-context Learning*. 2023. arXiv: 2301.07067 [cs.LG].
- [297] Bo Liu et al. *LLM+P: Empowering Large Language Models with Optimal Planning Proficiency*. 2023. arXiv: 2304.11477.
- [298] R. Liu and N. B. Shah. “Reviewergpt? An Exploratory Study on Using Large Language Models for Paper Reviewing”. In: *CoRR* abs/2306.00622 (2023). arXiv: 2306.00622 [cs.CL]. URL: <https://arxiv.org/abs/2306.00622>.
- [299] Scott Longpre et al. “A pretrainer’s guide to training data: Measuring the effects of data age, domain coverage, quality, & toxicity”. In: *arXiv preprint arXiv:2305.13169* (2023).
- [300] Shayne Longpre et al. “The FLAN Collection: Designing Data and Methods for Effective Instruction Tuning”. In: *CoRR* abs/2301.13688 (2023). URL: <https://arxiv.org/abs/2301.13688>.
- [301] Y. Lu et al. “Multimodal procedural planning via dual text-image prompting”. In: *CoRR* abs/2305.01795 (2023).
- [302] Q. Lyu et al. “Faithful chain-of-thought reasoning”. In: *CoRR* abs/2301.13379 (2023).
- [303] X. Ma et al. “Query rewriting for retrieval-augmented large language models”. In: *arXiv preprint arXiv:2305.14283* (2023).

- [304] Xinbei Ma et al. *Query Rewriting for Retrieval-Augmented Large Language Models*. 2023. arXiv: 2305.14283 [cs.CL]. URL: <https://arxiv.org/abs/2305.14283>.
- [305] K. Malinka et al. “On the educational impact of ChatGPT: Is artificial intelligence ready to obtain a university degree?” In: *CoRR* abs/2303.11146 (2023). URL: <https://arxiv.org/abs/2303.11146>.
- [306] Lev Maximov. *Do You Know English Grammar Better Than ChatGPT?* 2023. URL: <https://medium.com/writing-cooperative/do-you-know-english-grammar-better-than-chatgpt-8fc550f23681>.
- [307] R. T. McCoy et al. “Embers of autoregression: Understanding large language models through the problem they are trained to solve”. In: (2023). arXiv preprint. arXiv: 2309.13638.
- [308] Johannes von Oswald et al. *Transformers learn in-context by gradient descent*. 2023. arXiv: 2212.07677 [cs.LG].
- [309] J. Pan et al. “What In-context Learning ”Learns” In-context: Disentangling Task Recognition and Task Learning”. In: *CoRR* abs/2305.09731 (2023).
- [310] Joon Sung Park et al. *Generative Agents: Interactive Simulacra of Human Behavior*. 2023. arXiv: 2304.03442 [cs.HC]. URL: <https://arxiv.org/abs/2304.03442>.
- [311] Yang Jeong Park et al. *Can ChatGPT be used to generate scientific hypotheses?* 2023. arXiv: 2304.12208 [cs.CL]. URL: <https://arxiv.org/abs/2304.12208>.
- [312] Gerardo Penedo et al. *The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data, and Web Data Only*. 2023. arXiv: 2306.01116 [cs.CL].
- [313] Bin Peng et al. “RWKV: Reinventing RNNs for the Transformer Era”. In: *CoRR* abs/2305.13048 (2023). DOI: 10.48550/arXiv.2305.13048. URL: <https://doi.org/10.48550/arXiv.2305.13048>.
- [314] *Perplexity - Transformers*. Accessed: 2024-04-06. Hugging Face, 2023. URL: <https://huggingface.co/docs/transformers/perplexity>.
- [315] Michael Poli et al. “Hyena hierarchy: Towards larger convolutional language models”. In: *ICML*. 2023.
- [316] Alec Radford et al. “GPT-4: A Large-Scale Generative Pre-trained Transformer”. In: *CoRR* abs/2304.07409 (2023). arXiv: 2304.07409 [cs.CL].
- [317] Sebastian Raschka. *Understanding Encoder and Decoder*. 2023. URL: <https://sebastianraschka.com/p/understanding-encoder-and-decoder> (visited on 04/13/2024).
- [318] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. *Are Emergent Abilities of Large Language Models a Mirage?* 2023. arXiv: 2304.15004 [cs.AI]. URL: <https://arxiv.org/abs/2304.15004>.
- [319] Timo Schick et al. “Toolformer: Language models can teach themselves to use tools”. In: *CoRR* abs/2302.04761 (2023).
- [320] Z. Shao et al. “Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy”. In: *arXiv preprint arXiv:2305.15294* (2023).
- [321] Y. Shen et al. “Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface”. In: *arXiv preprint arXiv:2303.17580* (2023).
- [322] N. Shinn et al. “Reflexion: Language agents with verbal reinforcement learning”. In: (2023).

- [323] G. Sridhara, R. H. G., and S. Mazumdar. “ChatGPT: A Study on Its Utility for Ubiquitous Software Engineering Tasks”. In: *CoRR* abs/2305.16837 (2023). arXiv: 2305.16837 [cs.SE]. URL: <https://arxiv.org/abs/2305.16837>.
- [324] H. Sun et al. “Adaplanner: Adaptive planning from feedback with language models”. In: *arXiv preprint arXiv:2305.16653* (2023).
- [325] W. Sun et al. “Automatic Code Summarization via ChatGPT: How Far Are We?” In: *CoRR* abs/2305.12865 (2023). arXiv: 2305.12865 [cs.SE]. URL: <https://arxiv.org/abs/2305.12865>.
- [326] Yi Sun et al. “Retentive Network: A Successor to Transformer for Large Language Models”. In: *CoRR* abs/2307.08621 (2023). arXiv: 2307.08621 [cs.CL]. URL: <https://arxiv.org/abs/2307.08621>.
- [327] Rohan Taori et al. *Stanford ALPACA: An Instruction-Following LLaMA Model*. <https://github.com/tatsu-lab/stanford-alpaca>. 2023.
- [328] Yi Tay et al. *UL2: Unifying Language Learning Paradigms*. 2023. arXiv: 2205.05131 [cs.CL]. URL: <https://arxiv.org/abs/2205.05131>.
- [329] Hugo Touvron et al. “LLaMA 2: Open Foundation and Fine-Tuned Chat Models”. In: *arXiv preprint arXiv:2307.09288* (2023).
- [330] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].
- [331] Tomer Ullman. “Large language models fail on trivial alterations to theory-of-mind tasks”. In: *arXiv preprint arXiv:2302.08399* (2023).
- [332] Mojtaba Valipour et al. *DyLoRA: Parameter Efficient Tuning of Pre-trained Models using Dynamic Search-Free Low-Rank Adaptation*. 2023. arXiv: 2210.07558 [cs.CL].
- [333] Karthik Valmeekam et al. “On the planning abilities of large language models: A critical investigation”. In: *Thirty-seventh Conference on Neural Information Processing Systems (Spotlight)*. 2023. URL: <https://openreview.net/forum?id=X6dEqXIsEW>.
- [334] Ashish Vaswani et al. *Attention Is All You Need*. v7. 2023. arXiv: 1706.03762 [cs.CL].
- [335] *vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention*. Available online. 2023. URL: <https://vllm.ai/>.
- [336] Chunshu Wang et al. “Efficient prompting via dynamic in-context learning”. In: *arXiv preprint arXiv:2305.11170* (2023).
- [337] Guanzhi Wang et al. *Voyager: An Open-Ended Embodied Agent with Large Language Models*. 2023. arXiv: 2305.16291 [cs.AI].
- [338] L. Wang et al. “Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models”. In: *CoRR* abs/2305.04091 (2023). <https://doi.org/10.48550/arXiv.2305.04091>.
- [339] Neng Wang, Hongyang Yang, and Christina Dan Wang. “FinGPT: Instruction Tuning Benchmark for Open-Source Large Language Models in Financial Datasets”. In: *arXiv preprint arXiv:2309.13064* (2023).
- [340] Xinlong Wang et al. “Images Speak in Images: A Generalist Painter for In-Context Visual Learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 6830–6839.
- [341] Xinlong Wang et al. “SegGPT: Segmenting Everything in Context”. In: *CoRR* abs/2304.03284 (2023). arXiv: 2304.03284 [cs.CV].

- [342] Xinyi Wang, Wanrong Zhu, and William Yang Wang. *Large Language Models Are Implicitly Topic Models: Explaining and Finding Good Demonstrations for In-Context Learning*. arXiv preprint arXiv:2301.11916. 2023. URL: <https://arxiv.org/abs/2301.11916>.
- [343] Zhendong Wang et al. *In-Context Learning Unlocked for Diffusion Models*. arXiv preprint arXiv:2305.01115. 2023. URL: <https://arxiv.org/abs/2305.01115>.
- [344] Zihao Wang et al. *Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents*. 2023. arXiv: 2302.01560 [cs.AI]. URL: <https://arxiv.org/abs/2302.01560>.
- [345] Jerry Wei et al. *Larger language models do in-context learning differently*. 2023. arXiv: 2303.03846 [cs.CL].
- [346] Jerry Wei et al. *Symbol tuning improves in-context learning in language models*. 2023. arXiv: 2305.08298 [cs.CL].
- [347] N. Wies, Y. Levine, and A. Shashua. “The Learnability of In-context Learning”. In: *CoRR* abs/2303.07895 (2023).
- [348] Wikipedia. *Bayesian Inference*. 2023. URL: https://en.wikipedia.org/wiki/Bayesian_inference.
- [349] BigScience Workshop. *BLOOM: A 176B-Parameter Open-Access Multilingual Language Model*. 2023. arXiv: 2211.05100 [cs.CL].
- [350] Shijie Wu et al. *BloombergGPT: A Large Language Model for Finance*. 2023. arXiv: 2303.17564 [cs.LG].
- [351] Zhenyu Wu et al. *OpenICL: An Open-Source Framework for In-context Learning*. 2023. arXiv: 2303.02913 [cs.CL].
- [352] C. S. Xia and L. Zhang. “Conversational Automated Program Repair”. In: *CoRR* abs/2301.13246 (2023). arXiv: 2301.13246 [cs.SE]. URL: <https://arxiv.org/abs/2301.13246>.
- [353] Q. Xie et al. “Pixiu: A Large Language Model, Instruction Data and Evaluation Benchmark for Finance”. In: *Proceedings of NeurIPS Datasets and Benchmarks*. 2023.
- [354] Benfeng Xu et al. “kNN Prompting: Learning Beyond the Context with Nearest Neighbor Inference”. In: *International Conference on Learning Representations*. 2023a. 2023.
- [355] Can Xu et al. *WizardLM: Empowering Large Language Models to Follow Complex Instructions*. 2023. arXiv: 2304.12244 [cs.CL].
- [356] Canwen Xu et al. *Small Models are Valuable Plug-ins for Large Language Models*. arXiv preprint arXiv:2305.08848. 2023. URL: <https://arxiv.org/abs/2305.08848>.
- [357] Chen Xu et al. “Baize: An Open-Source Chat Model with Parameter-Efficient Tuning on Self-Chat Data”. In: *arXiv preprint arXiv:2304.01196* (2023).
- [358] Yi Yang, Yixuan Tang, and Kar Yan Tam. “InvestLM: A Large Language Model for Investment Using Financial Domain Instruction Tuning”. In: *arXiv preprint arXiv:2309.13064* (2023).
- [359] S. Yao et al. “Tree of thoughts: Deliberate problem solving with large language models”. In: *CoRR* abs/2305.10601 (2023).
- [360] Junjie Ye et al. *A Comprehensive Capability Analysis of GPT-3 and GPT-3.5 Series Models*. 2023. arXiv: 2303.10420 [cs.CL]. URL: <https://arxiv.org/abs/2303.10420>.
- [361] O. Yoran et al. “Making retrieval-augmented language models robust to irrelevant context”. In: *arXiv preprint arXiv:2310.01558* (2023).

- [362] C. Zhang et al. “One small step for generative AI, one giant leap for AGI: A complete survey on ChatGPT in AIGC era”. In: *CoRR* abs/2304.06488 (2023). arXiv: 2304.06488 [cs.AI]. URL: <https://arxiv.org/abs/2304.06488>.
- [363] Qingru Zhang et al. *AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning*. 2023. arXiv: 2303.10512 [cs.CL].
- [364] Wayne Xin Zhao et al. “A Survey of Large Language Models”. In: (2023).
- [365] H. S. Zheng et al. “Take a step back: Evoking reasoning via abstraction in large language models”. In: *arXiv preprint arXiv:2310.06117* (2023).
- [366] Qinkai Zheng et al. “CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Evaluations on HumanEval-X”. In: *CoRR* abs/2303.17568 (2023).
- [367] Wanjun Zhong et al. *MemoryBank: Enhancing Large Language Models with Long-Term Memory*. 2023. arXiv: 2305.10250 [cs.CL]. URL: <https://arxiv.org/abs/2305.10250>.
- [368] Y. Zhou et al. “Large language models are human-level prompt engineers”. In: *Proc. of ICLR*. 2023.
- [369] Aaron Jaech et al. *OpenAI o1 System Card*. 2024. arXiv: 2412.16720 [cs.AI]. URL: <https://arxiv.org/abs/2412.16720>.
- [370] Josh Achiam et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL].
- [371] Anthropic. *Claude 3 Model Card*. Accessed: 2024-12-24. 2024. URL: <https://assets.anthropic.com/m/61e7d27f8c8f5919/original/Claude-3-Model-Card.pdf>.
- [372] Jeanine Banks and Tris Warkentin. *Gemma: Google introduces new state-of-the-art open models*. Google AI Blog. 2024. URL: <https://blog.google/technology/developers/gemma-open-models/>.
- [373] Ning Bian et al. *ChatGPT is a Knowledgeable but Inexperienced Solver: An Investigation of Commonsense Problem in Large Language Models*. 2024. arXiv: 2303.16421 [cs.CL].
- [374] Yanda Chen et al. *On the Relation between Sensitivity and Accuracy in In-context Learning*. 2024. arXiv: 2209.07661 [cs.CL].
- [375] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL]. URL: <https://arxiv.org/abs/2312.10997>.
- [376] A. Gundawar et al. “Robust planning with LLMmodulo framework: Case study in travel planning”. In: *arXiv preprint arXiv:2405.20625* (2024).
- [377] Xue Jiang et al. *Self-planning Code Generation with Large Language Models*. 2024. arXiv: 2303.06689.
- [378] Subbarao Kambhampati. “Can large language models reason and plan?” In: *Annals of the New York Academy of Sciences* 1534.1 (2024), 15–18. ISSN: 1749-6632. DOI: 10.1111/nyas.15125. URL: <http://dx.doi.org/10.1111/nyas.15125>.
- [379] Subbarao Kambhampati et al. *LLMs Can’t Plan, But Can Help Planning in LLM-Modulo Frameworks*. 2024. arXiv: 2402.01817 [cs.AI]. URL: <https://arxiv.org/abs/2402.01817>.
- [380] Jean Lee et al. *A Survey of Large Language Models in Finance (FinLLMs)*. 2024. arXiv: 2402.02315 [cs.CL].
- [381] LMStudio. *LMStudio*. Accessed: 2024-07-26. 2024. URL: <https://lmstudio.ai/>.

- [382] New Scientist. *OpenAI's O3 model aced a test of AI reasoning – but it's still not AGI*. Accessed: 2024-06-09. 2024. URL: <https://www.newscientist.com/article/2462000-openais-o3-model-aced-a-test-of-ai-reasoning-but-its-still-notagi/>.
- [383] OpenAI. *Learning to Reason with LLMs*. Accessed: 2024-06-24. 2024. URL: <https://openai.com/index/learning-to-reason-with-llms/>.
- [384] Baptiste Rozière et al. *Code Llama: Open Foundation Models for Code*. 2024. arXiv: 2308.12950 [cs.CL]. URL: <https://arxiv.org/abs/2308.12950>.
- [385] Gemma Team et al. *Gemma: Open Models Based on Gemini Research and Technology*. 2024. arXiv: 2403.08295 [cs.CL].
- [386] M. Verma, S. Bhambri, and S. Kambhampati. “Theory of mind abilities of large language models in human-robot interaction: An illusion?” In: (2024). arXiv preprint. arXiv: 2401.05302.
- [387] Kevin Wang et al. *On The Planning Abilities of OpenAI's o1 Models: Feasibility, Optimality, and Generalizability*. 2024. arXiv: 2409.19924 [cs.AI]. URL: <https://arxiv.org/abs/2409.19924>.
- [388] Z. Wang et al. “Bridging the preference gap between retrievers and LLMs”. In: *arXiv preprint arXiv:2401.06954* (2024).
- [389] Meta AI. *The Llama 3 Herd of Models*. <https://ai.meta.com/research/publications/the-llama-3-herd-of-models/>. Accessed: 2024-07-25.
- [390] *BigQuery Dataset*. <https://cloud.google.com/bigquery?hl=zh-cn>. Accessed: 2024-04-14.
- [391] *Common Crawl*. <https://commoncrawl.org/>. Accessed: 2024-04-15.
- [392] *Project Gutenberg*. <https://www.gutenberg.org/>. Accessed: 2024-04-14.
- [393] *Wikipedia*. https://en.wikipedia.org/wiki/Main_Page. Accessed: 2024-04-14.