SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

An Evolutionary Framework for Instructing Large Language Models

Oguz Gültepe

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

An Evolutionary Framework for Instructing Large Language Models

Ein Evolutionärer Rahmen für die Anleitung Großer Sprachmodelle

Author:	Oguz Gültepe
Supervisor:	Prof. Dr. Georg Groh
Advisors:	Prof. Dr. Hinrich Schütze
	Lütfi Kerem Senel, M.Sc.
Submission Date:	15th November 2023

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15th November 2023 Oguz Gültepe

Acknowledgments

Throughout my studies, I have received support from so many lovely people that it is not feasible for me to list each and every one of them here. So I dedicate this work to all my loved ones who have listened to my endless ramblings and stood by me as I worked day and night.

Abstract

Pre-trained Large Language Model (LLM)s have demonstrated impressive capabilities such as reasoning and tool use. However, unleashing these capabilities requires us to instruct the LLMs in order to achieve the desired behaviour. This is commonly accomplished via hand written prompts where minor changes in the prompt structure may lead to significant changes in model behaviour. This prevents the fully automated utilization of LLM capabilities since achieving the best performance requires manually crafting prompts for each task. In this work, we propose a task agnostic approach that leverages evolutionary selection to produce high-performing prompts . Our framework learns instruction scores through evolutionary methods, and samples instructions based on these scores in order to improve few-shot performance. In our experiments, we demonstrate that our approach is able to discern high quality instructions and progressively improve model performance. Our experiments also reveal that the learned instruction scores are likely specific to the model and the task distribution.

Contents

1. Introduction

LLMs have taken the world by storm in the last year and interest in Artificial Intelligence (AI) driven by LLMs continues to grow. (Figure 1.1) The near-future implications of rapidly advancing AI is now a common topic of discourse, where the predictions range from total destruction of human kind to an AI enabled utopia where all work is automated.

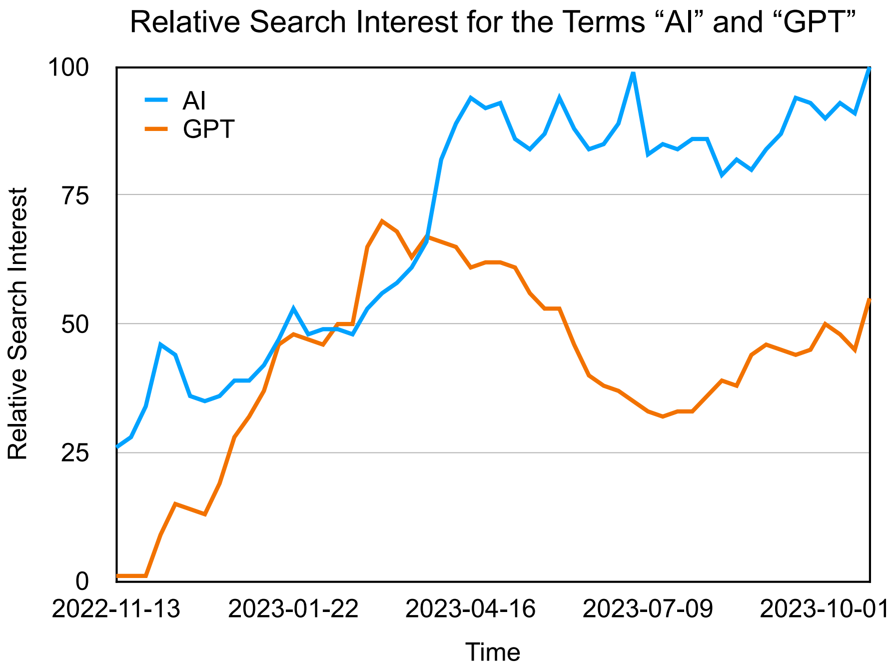

Figure 1.1.: Within the last year, search interest for the term *AI* increased 400% while the search interest for the term *GPT* increased 5000%. Data Source: Google Trends

While there is a large amount of hype and exaggeration surrounding AI and LLMs, State-of-the-Art (SOTA) models display truly impressive capabilities such as reasoning and tool use. These capabilities are unleashed through the natural language descriptions of the desired model behaviours. This is a relatively new phenomenon; earlier transfer learning approaches required task specific fine-tuning. LLMs, on the other hand, can be conditioned via textual input without updating the model parameters. This is enabled by LLM's auto-regressive property: LLMs are trained to predict the next token based on the preceding text. This property can be leveraged to adapt a LLM to any task that can be described in natural language. We just need to provide an input to the LLM that naturally continues with the desired output. For example, conditioning a LLM with the input "`[SENTENCE]` translated to French is " will lead the LLM to generate the French translation of `[SENTENCE]`.

Adapting a LLM to a new task with a task description is called the *zero-shot* setting. If a few task demonstrations are provided as well, then it is called the *few-shot* setting. While the zero-shot/few-shot abilities are a natural byproduct of training a generative language model, contemporary LLMs are additionally fine-tuned to enhance instruction following abilities. Today, both zero-shot and the few-shot task adaptation is commonly accomplished via hand written prompts. As minor changes in the prompt structure may lead to significant changes in model behaviour, achieving the best performance requires manually crafting prompts for each task which prevents the fully automated utilization of LLM capabilities. Improving instruction adherence and curating high quality instructions remain open research problems.

In this work, we propose a task agnostic approach to automatically curate few-shot prompts through evolutionary selection. We leverage LLMs to evolve instruction pairs consisting of task descriptions and the corresponding model outputs. When a new task is encountered, we construct a few-shot prompt by sampling previously generated instructions. We sample instructions based on task similarity and past few-shot performance, thereby exerting evolutionary pressure on instructions. If the model generates a correct output, the task/output pair is added to the set of instructions and the performance measures of the instructions in the few-shot prompt are updated.

The intuition behind our approach is that over time, the instructions that consistently yield correct outputs will end up with higher scores and will therefore be sampled more frequently, leading to better few-shot performance. Our approach can be seen as a case of *genetical programming*; the application of evolutionary techniques to optimize code or functions. Our experiments show that sampling instructions based on past performance improves few-shot performance over sampling random or similar instructions, but that an instruction's influence is specific to each model and task distribution. In other words, an instruction's performance on a model does not translate to performance on another model, and an instruction's performance over a given dataset does not translate to performance over another dataset.

In the following sections, we will first explore the historical trends in Natural Language Processing (NLP) research that have culminated in the contemporary LLMs and then briefly mention some emergent LLM capabilities. Next, we will explain our evolutionary framework in detail and demonstrate how it works over an example task. Afterwards, we will go through our experiments and analyze our framework under various conditions to better understand how instruction quality effects model behaviour. Finally, we conclude by mentioning the limitations of our approach and potential directions for future work.

2. Theoretical Background

In order to keep this work focused and concise, the readers are assumed to be familiar with the basics of Deep Learning (DL) as well as NLP. For those who wish to learn more about these topics, we refer to Goodfellow et al. 2016 for DL and Jurafsky and Martin 2009 for NLP.

2.1. Emergence of Large Language Models

2.1.1. Deep Natural Language Processing

Pre-2010s NLP had been dominated by non-neural machine learning models that relied on extensive feature engineering (Figure 2.1) as well as rule-based methods such as regular expressions. (Jurafsky and Martin 2009)

Var	Definition
x_1	count(positive lexicon words $\in$ doc)
x_2	count(negative lexicon words $\in$ doc)
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	ln(word count of doc)

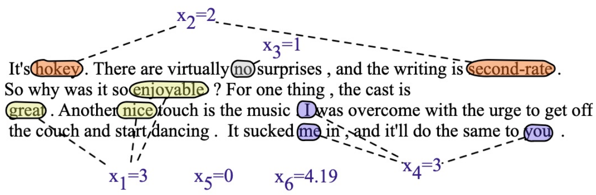

Figure 2.1.: A hand crafted feature template for sentiment classification applied to an example paragraph. Source: Jurafsky and Martin 2009

By the early 2010s, Neural Network (NN)s were already being applied to NLP tasks with promising results. Deep NLP using NNs posed a contrast to what some called the "traditional" NLP as NNs required minimal feature engineering. Instead, words could be fed into a NN as one-hot encoded vectors which were then projected to learned dense vector representations called *word embeddings*. (Goldberg 2016). These learned representations would come to play a central role in early transfer learning for NLP. (Subsection 2.1.3)

While many different neural architectures have been proposed to deal with sequence processing tasks (Lipton 2015), the SOTA LLMs are all enabled by the Transformer architecture. (Bommasani et al. 2021) (Bubeck et al. 2023)

2.1.2. The Transformer Architecture

The Transformer architecture (Vaswani et al. 2017) follows an encoder-decoder approach that was introduced by Cho et al. 2014 and popularized by the Sequence to Sequence (seq2seq) model. (Sutskever et al. 2014).

The original seq2seq model uses a stacked Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) network to encode an input sequence as a finite dimensional vector, and then decodes the output sequence from this vector using another stacked LSTM network. The Transformer architecture replaces the LSTM with attention mechanisms and feed-forward networks as the core building blocks of the encoder/decoder stacks.

In most general terms, an attention mechanism is a function that maps a query vector and pairs of key/value vectors to an output vector. The mapping is facilitated as a sum of values weighted by a *compatibility function* of the query with the associated keys.

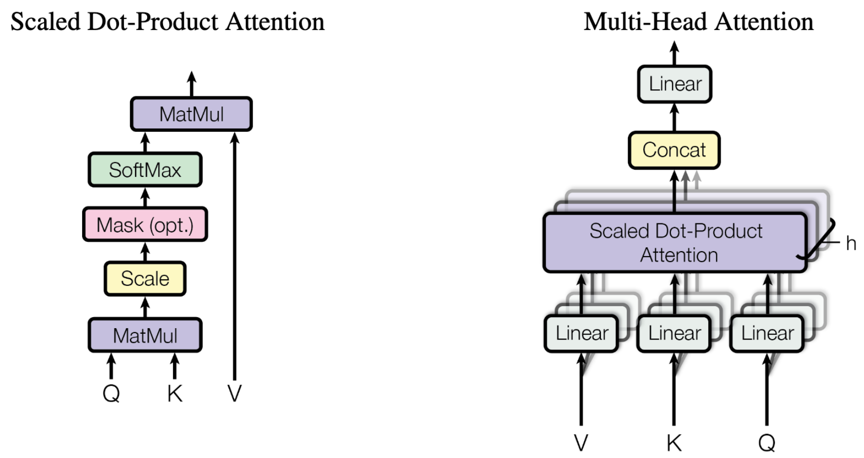

Figure 2.2.: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. Source: Vaswani et al. 2017

The particular variant of attention that the Transformer architecture utilizes is called *Scaled Dot-Product Attention*. (Figure 2.2) The weights of the values are obtained by applying a softmax function to the scaled[1] dot-products of the query and the keys. The attention function is computed in practice for many queries in parallel via matrix multiplication.

[1]The dot-products are scaled by a factor of $\frac{1}{\sqrt{d_k}}$ where d_k is the dimensionality of the query and the key vectors. The scaling is intended to prevent the dot-products from growing large in magnitude which could lead to a vanishing gradient problem (Hochreiter, Bengio, et al. 2001) with the softmax function.

The Transformer architecture also makes use of *multi-head attention*. (Figure 2.2) Key, value and query vectors are all projected to equally many lower dimensional vectors via learned linear projections. The attention function is then applied to these lower dimensional vectors in parallel. The outputs are concatenated and once again linearly projected. The original Transformer model utilizes a total of 8 attention heads over 64 dimensional vectors each.

The attention mechanism has a few advantages over the recurrent architectures such as LSTM that it replaces. For one, the attention function can be computed in parallel, which makes it easy to accelerate training with additional compute. Another major advantage is that the attention mechanism allows each position to directly attend to any other position, which makes it easier to learn long-range dependencies. (Hochreiter, Bengio, et al. 2001) In practice, however, the Transformer models have limited context length due to the quadratic computational complexity of each position attending to all other positions.

The encoder and the decoder of the original Transformer architecture share many characteristics. (Figure 2.3) Both networks consist of 6 stacked layers that are all composed of multiple sublayers. Each sublayer in each network employs residual connection (K. He et al. 2016) and layer normalization (Ba et al. 2016). The encoder and the decoder slightly differ in layer composition: An encoder layer is composed of two sublayers: First contains a multi-head attention mechanism and the second contains a feed-forward network. A decoder layer on the other hand is composed of three sublayers: First two containing multi-head attention mechanisms and the third one containing a feed-forward network.

The Transformer architecture employs three different multi-head attention mechanisms:

- *Self-attention* is found in the encoder. It operates over identical query, key and value vectors. Each position attends to all other positions.
- *Masked self-attention* is found in the decoder. It also operates over identical query, key and value vectors. However, each position only attends to itself or positions that came before, hence the name masked. This is intended to prevent leftward information flow and preserve the auto-regressive property.
- *Encoder-decoder attention* is found in the decoder. It operates over queries from the decoder and key/value pairs from the encoder.[2]

[2]The output of the entire encoder stack is used as identical key/value pairs for encoder-decoder attention. This stands in contrast to self-attention and masked self-attention which both exclusively operate over the outputs of the preceding layers.

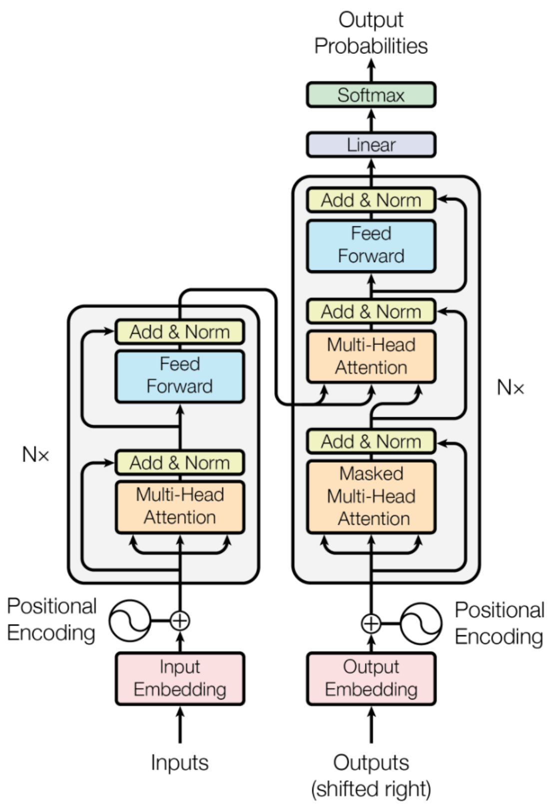

Figure 2.3.: The Transformer architecture consist of an encoder (left) and a decoder (right). Source: Vaswani et al. 2017

The input sequence is fed to the encoder. At each step, the previously generated output sequence is fed to the decoder along with the encoder output. A probability distribution for the next token is obtained by applying softmax function to a learned linear projection of the decoder stack output. This learned projection shares the same weight matrix with the two embedding layers at the base of the encoder/decoder stacks. The only difference is that in the embedding layers, the weights are multiplied by the square root of the vector dimensionality. This is done in order to ensure desirable distributional properties of the internal network states.

Recurrent architectures such as LSTM implicitly incorporate positional information. Attention mechanism on the other hand contains no recurrence. To compensate, the Trasformer architecture explicitly injects positional information by adding *positional encodings* to the word embeddings before feeding them into the encoder/decoder stacks.

2.1.3. Early Transfer Learning

Transfer learning refers to the practice of utilizing commonly available data in order to improve the performance of machine learning systems in data-scarce settings. (Weiss et al. 2016) In early 2010s, popular transfer learning approaches for NLP involved the use of unsupervised methods to induce word representations. The induced representations could then be plugged into existing machine learning systems to improve performance. (Turian et al. 2010)

Over time, NNs would become the dominant models for inducing word representations. (Gutiérrez and Keith 2019) Neural word embeddings were surprisingly good at capturing syntactic and semantic regularities. It was shown that algebraic operations on the vector representations could approximate semantic relations. For example, the vector representations for the words *King*, *Queen*, *Man* and *Woman* were such that *King - Man + Woman* would result in a vector very close to *Queen*. (Mikolov, Yih, et al. 2013) Figure 2.4 demonstrates another interesting example of a semantic relation implicitly captured by the induced vector representations.

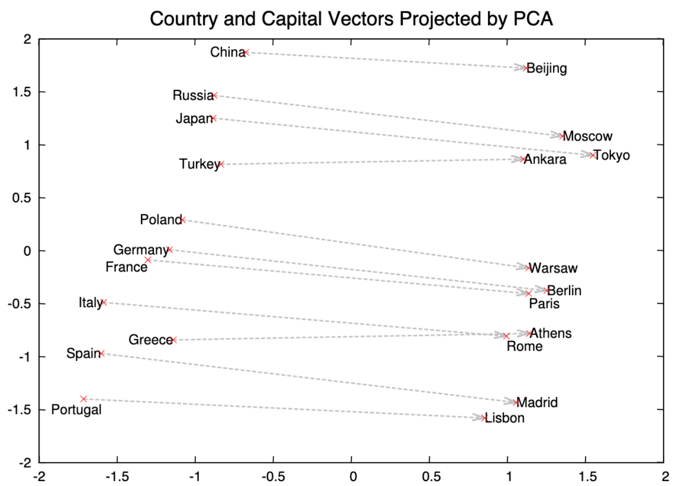

Figure 2.4.: Two-dimensional PCA projection of 1000-dimensional word embeddings of countries and their capital cities. Country/capital relation is implicitly captured by the embeddings without any explicit supervised information during training. Source: Mikolov, Sutskever, et al. 2013

Some well-known examples of pre-trained neural word embeddings include word2vec (Mikolov, K. Chen, et al. 2013), GloVe (Pennington et al. 2014) and fastText (Mikolov, Grave, et al. 2018). We do not delve further into the technical details of these embeddings as their relevance to our topic is more contextual than technical.

2.1.4. Pre-Trained Language Models

In the second half of the 2010s, the focus of transfer learning in NLP would shift from pre-training vector representations to pre-training language models. Dai and Le 2015 were the first to propose an unsupervised pre-training approach to improve NN performance on NLP taks, as well as the first to propose auto-regressive language modeling as a pre-training objective. Their approach would achieve SOTA on 4 text classification benchmarks.

In 2018, 4 pre-trained language models would be released within a single year. The first 2 of these models employed LSTM based recurrent architectures. (Peters et al. 2018) (Howard and Ruder 2018) These models achieved SOTA performance on several benchmarks and garnered the attention of the NLP community.

A few months later, the first pre-trained Transformer model Generative Pre-Trained Transformer (GPT) was released. (Radford, Narasimhan, et al. 2018) GPT is a decoder-only Transformer model, consisting of 12 decoder layers. Each layer is composed of two sublayers. The first sublayer contains a masked self-attention mechanism with 12 attention heads and the second sublayer contains a feed-forward network. GPT fine-tuned on downstream tasks achieved SOTA results on 9 benchmarks covering wide range of tasks such as commonsense reasoning, question answering and textual entailment.

Shortly after, Bidirectional Encoder Representations from Transformers (BERT) was introduced. (Devlin et al. 2019) BERT is an encoder-only Transformer model that is pre-trained with masked language modelling[3] and next sentence prediction objectives. BERT is released in two sizes: $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$. $\text{BERT}_{\text{BASE}}$ is designed to be a comparison point against GPT and consists of 12 encoder layers. Each layer is composed of two sublayers. The first sublayer contains a self-attention mechanism with 12 attention heads and the second sublayer contains a feed-forward network. $\text{BERT}_{\text{LARGE}}$ on the other hand consists of 24 layers that follow the same composition as $\text{BERT}_{\text{BASE}}$ with the expection of utilizing 16 attention head per attention mechanism. Fine-tuned $\text{BERT}_{\text{BASE}}$ models achieve SOTA performance on 11 benchmarks covering diverse array of tasks, which are all further improved upon by the fine-tuned $\text{BERT}_{\text{LARGE}}$ models.

Following the massive success of BERT, utilizing pre-trained language models became the norm in NLP research. (Bommasani et al. 2021) BERT was indeed so influential that it spawned more than 150 follow up studies and numerous variants within a little over a year. (Rogers et al. 2020) Naturally, understanding the factors that contribute to pre-trained model performance became crucial to build better models.

[3] Also known as *cloze* task, masked language modelling refers to randomly masking input tokens and predicting the masked tokens based on their context.

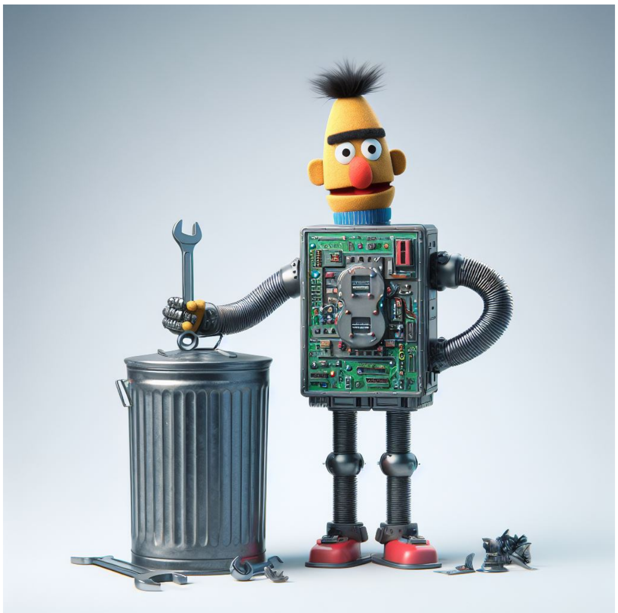

Figure 2.5.: BERT is the second language model to be named after a Sesame Street character following ELMo. Source: Bing Image Creator, October 2023.

2.1.5. Large Language Models

When BERT was released, the idea that the model performance would improve with the model size was not unheard of. Early work had already demonstrated power-law scalings[4] between model performance and dataset size. (Banko and Brill 2001) (Goodman 2001) Later on, power-law scaling between model size and dataset size was also established. (Hestness, Narang, et al. 2017) (Hestness, Ardalani, et al. 2019) However, BERT was the first work to convincingly demonstrate the benefits of scaling to what the authors dubbed "extreme model size". Unknown to the authors at the time, BERT would soon be dwarfed by the models that followed.

The first of these models was GPT-2 (Radford, J. Wu, et al. 2019), a decoder-only Transformer that was architecturally identical to GPT except a few minor modifications. GPT-2, was released in 4 sizes: The smallest variant consisted of ~110 million parameters and was equivalent to GPT/$\text{BERT}_{\text{BASE}}$ in size. The second smallest variant was equivalent to $\text{BERT}_{\text{LARGE}}$ in size and consisted of ~340 million parameters. The two larger variants consisted of ~760 million and ~1.5 billion parameters. The largest model achieved SOTA performance on many NLP benchmarks in zero-shot setting, meaning

[4]Power-law scaling refers to a relation between two variables x, y where $\alpha \cdot x^{\beta} \approx y$.

that the model did not require any fine-tuning to perform downstream tasks. Instead, the model could be conditioned via textual input to generate the desired output. For example, conditioning the model on an article followed by `TL;DR`[5] would produce a summary of the article.

The next release was Text-to-Text Transfer Transformer (T5) (Raffel et al. 2020), an encoder-decoder Transformer model that came in 5 sizes ranging from ~60 million to ~11 billion parameters. T5 framed each task as text generation conditioned on input text, hence the name text-to-text. While similar to GPT-2 in this aspect, T5 was conceived as a transfer learning model and did not report zero-shot performance. Fine-tuned T5 models achieved SOTA on many NLP benchmarks.

Later on, Kaplan et al. 2020 presented *scaling laws for neural language models* which showed that model performance scaled as power-law with model size, dataset size and the amount of compute used for training. Based on these scaling laws, the authors introduced methods to calculate optimal model/dataset size for a fixed compute budget.[6] They showed that training very large models and stopping significantly before convergence was preferable over training a smaller model to convergence since larger models were more sample efficient. (Figure 2.6)

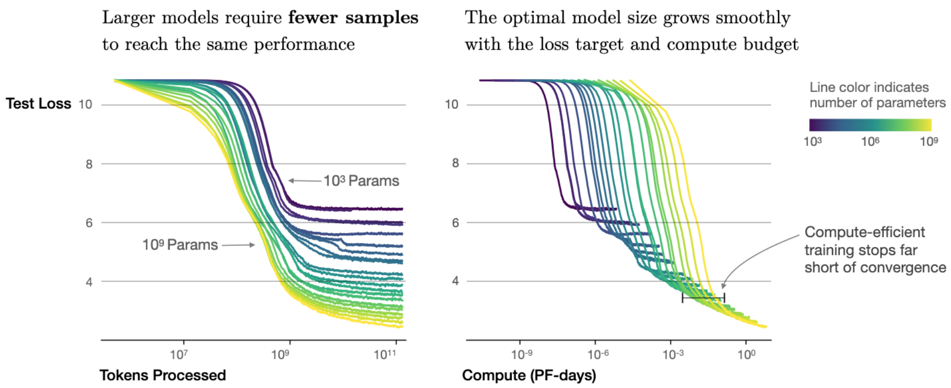

Figure 2.6.: A series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings). Source: Kaplan et al. 2020

GPT-3 was introduced only a few months later. (Brown et al. 2020) Following the previously mentioned optimal model/dataset size calculations, GPT-3 models were much larger and were trained on many fewer tokens than what was typical at the time.

[5]Short for "Too Long; Didn't Read".

[6]The compute optimal model/dataset size calculations were later revised by Hoffmann et al. 2022, which showed that dataset size should be scaled equally with model size and not sublinearly as previously understood.

The model architecture was mostly the same as GPT-2, with the exception that GPT-3 utilized alternating dense and sparse attention (Child et al. 2019) patterns in the layers of the transformer. 8 models of varying sizes were trained, largest of which had ~175 billion parameters.

In addition to achieving impressive zero-shot performance, GPT-3 popularized the few-shot setting where the model is provided a few demonstrations along with the task description. However, arguably the most striking capability of GPT-3 was that it could be conditioned on a title and a subtitle to produce realistic news articles. In fact, their experiments showed that humans could tell apart GPT-3 produced articles from human-written ones with only 52% accuracy; barely above random choice.

Citing the competitive AI landscape and potential misuse in mass disinformation campaigns, GPT-3 was never publicly released and could only be accessed through an Application Programming Interface (API).

GPT-3 was listed as one of the 10 breakthrough technologies of the year by MIT Technology Review. (Heaven 2021). GPT-3 is considered to be the first model with emergent in-context learning ability. (Wei, Tay, et al. 2022) (W. X. Zhao et al. 2023)

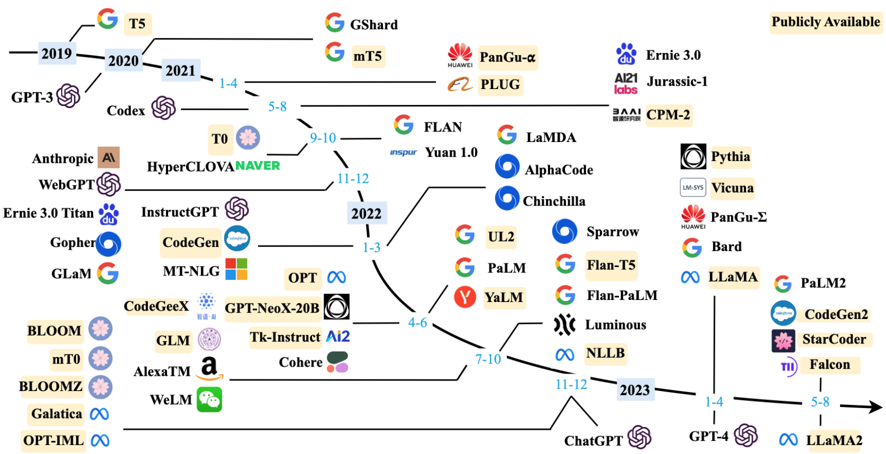

Figure 2.7.: A timeline of LLMs with more than 10 billion parameters. Source: W. X. Zhao et al. 2023

In the following years, LLMs would proliferate (Figure 2.7) and continue to grow in size, the largest LLM to date reaching 1.2 trillion parameters. (Du et al. 2022)

2.1.6. Alignment

It became clear over time that scaling LLMs would not be enough to produce models capable of tackling complex tasks. LLMs were able generate realistic text, but conditioning a model to achieve desired behaviour remained challenging. LLMs were trained to mimic natural language, which did not make them inherently better at following user instructions. (S. Zhang et al. 2023)

Ziegler et al. 2019 proposed aligning model behaviour with user intent via Reinforcement Learning from Human Feedback (RLHF). RLHF (Christiano et al. 2017) starts with a model and a set of inputs. For each input, the model generates multiple outputs which are then ranked by human participants. A reward function is learned based on these rankings and this reward function is in turn used to train the model via reinforcement learning. In language modeling context, human participants rank machine generated text based on desirable qualities such as positive sentiment and descriptive language. The reward function is a pre-trained language model fine-tuned to predict human rankings.

Later, on *instruction-tuning* was proposed to bridge the gap between the pre-training objective and the desired model behaviour. (Wei, Bosma, et al. 2022) Instruction-tuning involved fine-tuning a model on NLP tasks verbalized via natural language instruction templates and was shown to substantially improve model performance on unseen tasks.

InstructGPT was the first model to combine instruction-tuning and RLHF. (Ouyang et al. 2022) InstructGPT was based on GPT-3, yet the outputs from the 1.3B parameter InstructGPT model were shown to be superior to the outputs from the 175B GPT-3.

Despite the substantial benefits of instruction-tuning, curating human-written instructions remained expensive. This limited the quantity, diversity, and creativity of the available instructions.

Self-instruct was proposed in response. (Y. Wang et al. 2023) Self-Instruct was a framework for improving the instruction-following capabilities of pre-trained language models by bootstrapping off their own generations. A pre-trained language model is utilized to generate instructions, input, and output samples, which are then filtered before being used to fine-tune the original model. Instruction-tuning GPT-3 via self-instruct yielded a model of comparable quality to InstructGPT without the costly RLHF step.

It was shown later on that an existing instruction-tuned model could be leveraged via self-instruct to instruction-tune another pre-trained language model for the fraction of the cost. (Taori et al. 2023) In the following years, more methods to leverage LLMs to generate instructions have been proposed. (C. Xu et al. 2023) (Z. Sun et al. 2023) (X. Li, P. Yu, et al. 2023)

Even though instruction-tuning allows for a more controllable and predictable model behavior compared to standard LLMs, there are a few drawbacks. First, while instruction-tuning is computationally efficient, crafting instruction datasets to faciliate domain adaptation remain challenging. Furthermore, it has been argued that instruction-tuning does little to improve model generalization ability beyond what is heavily supported in the instruction dataset, and that the models merely learn to imitate the superficial patterns such as the style and the input/output formats. (Kung and Peng 2023) (Gudibande et al. 2023) It has also been suggested, that almost all knowledge in LLMs is learned during pre-training, and only limited instruction tuning data is necessary to teach models to produce high quality output. (C. Zhou et al. 2023)

Improving instruction adherence and curating high quality instructions remain open research problems, which is a primary driver of this work.

2.2. Large Language Model Capabilities

Scaling up language models has lead to predictable improvements, yet LLMs also display unpredictable emergent abilities that are not present in smaller models. (Wei, Tay, et al. 2022) Due to the novelty of this phenomenon, the full extent of these capabilities are not well understood. LLMs have been called anything from mere stochastic parrots (Bender et al. 2021) to an early yet incomplete version of Artificial General Intelligence (AGI). (Bubeck et al. 2023) Truth likely lies somewhere in between. In this section, we explore some LLM capabilities that are relevant to our use-case. For reasoning and tool use capabilities, we differentiate between prompting-based approaches that makes use of a few-shot context vs. fine-tuning based approaches.

2.2.1. Reasoning

Reasoning is the process of logical and systematic thinking that makes use of evidence and past experiences. Reasoning plays a crucial role in activities such as problem solving, decision making, and critical thinking, and is considered to be a fundamental aspect of human intelligence. (Huang and Chang 2023) Despite not being explicitly trained for it, LLMs show emergent reasoning capabilities. While the extent of these abilities are not fully understood, Dziri et al. 2023 draw parallels between complex reasoning tasks and computation graphs to argue that LLM performance will rapidly decay with increased task complexity.

via Prompting

Nye et al. 2022 was the first work to show that producing intermediate computational steps improved few-shot model performance on complex tasks. Later on, Wei, X. Wang, et al. 2022 observed that producing intermediate reasoning steps significantly improves the complex reasoning capabilities of LLMs. (Figure 2.8)

Many more prompting strategies that make use of intermediate steps to improve complex reasoning ability have been proposed since. (D. Zhou et al. 2023) (Kojima et al.

2022) (Press et al. 2023) (Yao, D. Yu, et al. 2023)

B. Wang et al. 2023 explores which aspects of the demonstrated intermediate computational steps impact model reasoning performance. Their work shows that even invalid demonstrations can achieve high performance, and that the order of the intermediate steps as well as their relevance to the query have a much higher impact on the reasoning performance.

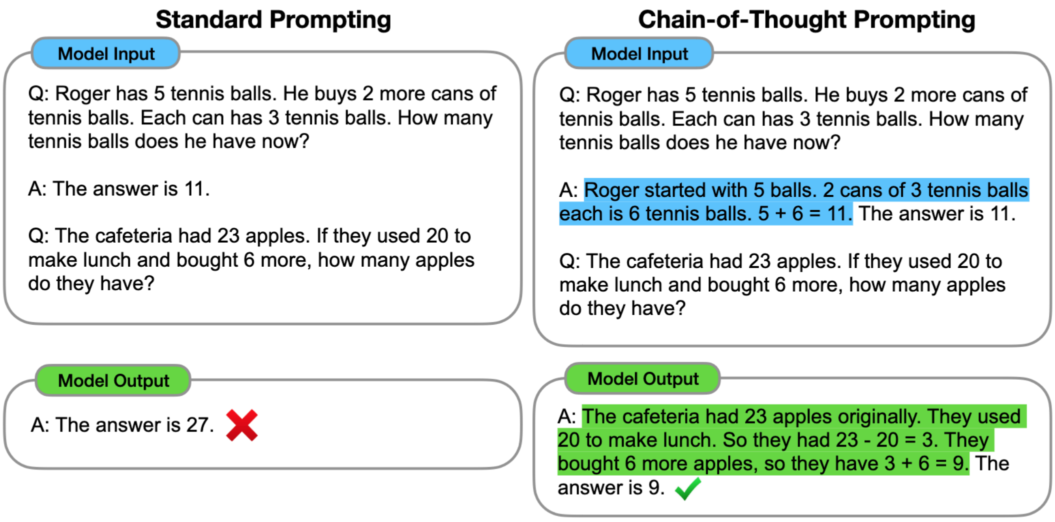

Figure 2.8.: Chain-of-though prompting is common way to boost few-shot reasoning performance by inducing intermediate reasoning steps. Source: Wei, X. Wang, et al. 2022

via Fine-Tuning

Rajani et al. 2019 show that pre-trained language models fine-tuned on rationales that explain model behaviour perform better on commonsense question answering tasks. Talmor, Herzig, et al. 2019 and Talmor, Tafjord, et al. 2020 fine-tune pre-trained language models to reason over implicit pre-trained knowledge and explicit free-text statements.

Hendrycks et al. 2021 fine-tune a pre-trained language model to solve mathematics problems by generating step-by-step solutions. Similarly, Nye et al. 2022 fine-tune a pre-trained language models to facilitate multi-step reasoning for program synthesis/execution by generating intermediate computational steps.

More recently, Zelikman et al. 2022 propose a framework to iteratively improve LLM reasoning ability by fine-tuning the model on intermediate reasoning steps generated by itself. Some other works boost reasoning capabilities of smaller pre-trained language models by fine-tuning the models on intermediate reasoning steps generated by LLMs (S. Li et al. 2022) (Magister et al. 2023) (Fu et al. 2023) (Mukherjee et al. 2023) (Yue et al. 2023) or by rule-based approaches. (Liu and Low 2023)

2.2.2. Tool Use

Tool use is a common characteristic of many intelligent species. (Shumaker et al. 2011) Sufficiently large language models demonstrate emergent tool use capabilities as well. These capabilities pose an opportunity to leverage external tools in order to augment LLMs in areas where they fall short.

via Prompting

W. Chen et al. 2023 and Gao et al. 2023 demonstrate that LLM reasoning capacity may be augmented by expressing the reasoning process as a program which is then executed in an external code interpreter. Davis and Aaronson 2023 show that a SOTA LLM can use a code interpreter and Wolfram Alpha[7] to solve high school and college level science and math problems.

Cai et al. 2023 propose a framework that differentiates between tool making and tool using. They argue that tool making, which in this context refers to generating a Python utility function for a given task, requires more sophisticated capabilities than applying the tools for problem-solving. They show that this distinction may be leveraged to increase computational efficiency by utilizing more capable and expensive models for tool making and less capable and cheap models for tool using.

via Fine-Tuning

Schick et al. 2023, Patil et al. 2023 and Qin et al. 2023 fine-tune LLMs to use increasingly larger and more diverse pools of APIs. The most recent work makes use of 16000+ real world APIs.

2.2.3. Evolution

Evolutionary Algorithm (EA)s are biologically-inspired techniques that utilize evolutionary principles to explore problem-spaces. Leveraging a form of synthetic digital

[7]Wolfram Alpha is an answer engine capable of evaluating advanced mathematical and scientific queries.

evolution, EAs are able to produce novel solutions to complex problems across various domains. Synthetic digital evolution is a dynamic mechanism that apply replication, variation and selection over a population of entities that represent potential solutions. (Sloss and Gustafson 2020)

- Replication refers to the formation of new entities. This can happen on a generational basis where the entire population is replaced, or on an individual basis within the same population.
- Variation refers to diversification of the population and is mainly facilitated in two forms: Recombination combines parts of existing entities to produce new entities. Mutation, on the other hand, randomly alters specific features of the entities in a population.
- Selection refers to the application of evolutionary pressure, where only the *fittest* individuals are used to generate new entities.

Lehman et al. 2024 propose utilizing code LLMs as mutation operators to evolve functional Python programs. Later on, Meyerson et al. 2023 show that LLMs can be applied as variation operators to various structures represented as text, such as binary bit-strings, sentences, equations, text-to-image prompts, and code. A. Chen et al. 2023 utilizes LLMs to evolve novel neural architectures that outperform SOTA architectures on algorithmic reasoning tasks. C. Xu et al. 2023 evolves novel complex instructions via LLMs to generate an instruction-tuning dataset. Finally, C. Yang et al. 2023 propose a prompting framework that leverages LLMs as optimizers and utilizes this framework, among other tasks, to evolve high-performing LLM prompts.

2.2.4. Agency

Agency refers to the capability of taking purposeful action in order to accomplish goals. LLMs' emergent in-context learning, reasoning and tool-use capabilities pose a unique opportunity to build autonomous agents that can tackle complex task. In the following we explore some LLM powered agents.

Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is one of the most basic agent configurations. Originally proposed by Lewis et al. 2020, a RAG agent interacts with an external knowledge base to retrieve contextually relevant information which is then incorporated into the generated text. RAG may be leveraged to improve the factuality of LLM generated text, incorporate up-to-date information, adapt LLMs to new domains or to simply interact with existing knowledge bases. (Lazaridou et al. 2023) (Menick et al. 2022) (Khattab et al. 2023) (H. He et al. 2022) (X. Li, R. Zhao, et al. 2023)

Operational

Kim et al. 2023 propose a prompting scheme that enables LLMs to solve computer tasks by interacting with a digital environment. Shen et al. 2023 utilize a LLM to break down complex AI tasks into subtasks and leverage open-source models to complete the subtasks. X. Zhou et al. 2023 shows that LLMs can be effectively utilized as database administrators. Guo et al. 2023 fine-tune a LLM to automatize IT operations. Hong et al. 2023 introduce a multi-agent framework that simulates an entire software company.

Task Agnostic

ReAct (Yao, J. Zhao, et al. 2022) is a prompting framework that allows LLMs to combine reasoning and tool use in order to solve arbitrary tasks. At each time step, a ReAct agent generates a thought, takes an action and observes the outcome. (Figure 2.9)

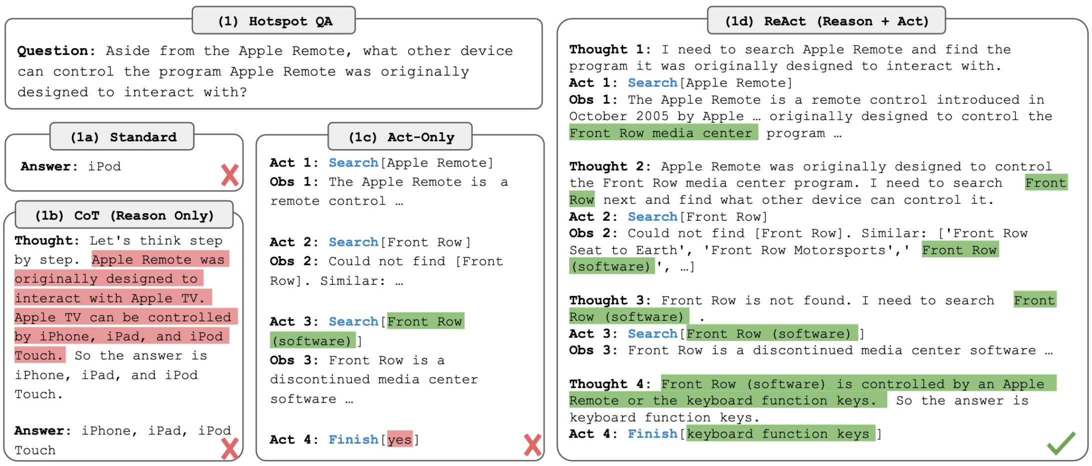

Figure 2.9.: Comparison of 4 prompting methods; (a) Standard, (b) Reason Only (Chain-of-thought), (c) Act-only, and (d) ReAct (Reason+Act) Source: Yao, J. Zhao, et al. 2022

Paranjape et al. 2023 propose an automated prompt building framework to incorporate relevant task demonstrations. They present two approaches for determining task relevancy: The first strategy involves manually dividing a diverse set of tasks into 5 clusters based on the skills required to complete the tasks. When a new task is encountered, example prompts from each cluster is tested on a few instances. Demonstrations are then sampled from the cluster with the highest performance. The second strategy is to prompt a LLM to determine whether two tasks are similar based on task descriptions and then sampling demonstrations from the most similar tasks.

Frameworks such as ReAct stop text generation when an action is taken, append the observation to the generated text, and then generate a follow up. As the number of steps grow, the computational complexity grows quadratically since the LLM needs to process all preceding text at each step. In order to alleviate this computational overhead, B. Xu et al. 2023 propose the Planner-Worker-Solver (PWS) framework that decouples reasoning from acting by offloading the tasks of planning, acting and integrating the observations to different agents.

3. Methodology

3.1. Evolutionary Instruction Selection

In this section, we explain our evolutionary selection framework for automatically generating high quality few-shot prompts. Our approach can be considered a case of *Genetical Progamming*, which refers to the application of evolutionary techniques to optimize code or functions. (Sloss and Gustafson 2020) We treat successful utilized instructions as the population. We replicate the instructions by recombining them with a LLM. We apply evolutionary selection through the use of *instruction scores*, derived based on an instruction's past performance. The open-endedness of our approach makes it adaptable to many different domains and use-cases. In the following, we explain the concrete details of our framework.

3.1.1. Training

We generate new instructions at training time based on a task described in natural language. (Figure 3.1) First, we leverage a neural embedding model to produce a dense vector representation of the task. We use this embedding representation to calculate the semantic similarity between the task and the previously encountered tasks. We measure semantic similarity between two task descriptions as the cosine similarity between their vector representations. This formulation allows us to efficiently calculate similarity between a novel task and all previously encountered tasks. We select the instructions for the top K most similar tasks and then sample from this pool N instructions. We calculate the sample weights as a function of the instruction scores and the cosine similarity values. We combine the sampled instructions into a few-shot prompt and automatically adapt the prompt preamble with information about the tools used in the instructions. If the few-shot prompt leads to success, we increment the score of each instruction in the few-shot prompt by 1. The generated instruction is then added to the population with a default instruction score of 1.

As the scores of all instructions in the few-shot prompt are collectively incremented, it is natural to consider whether an instruction that has not contributed to the success of the prompt might end up being assigned undeserved credit. While this is possible, the iterative nature of evolutionary optimization means that instructions that do not

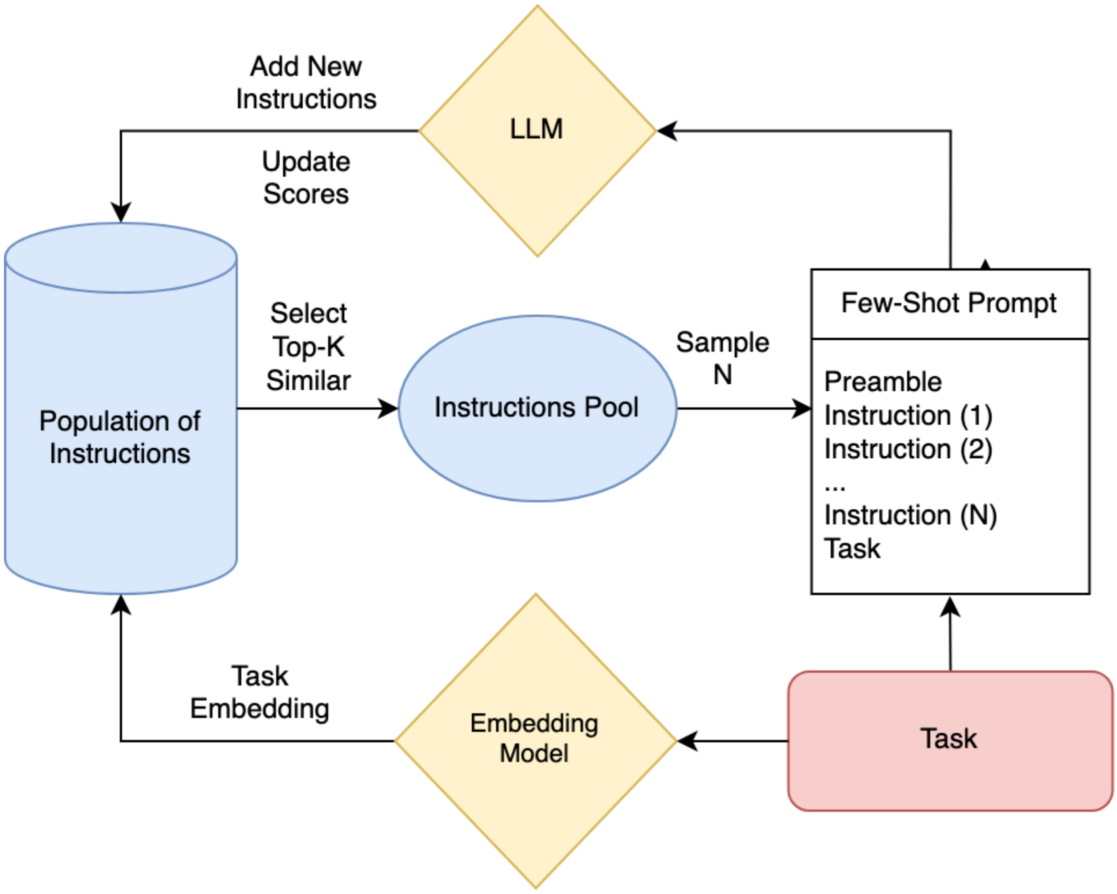

Figure 3.1.: An overview of our evolutionary framework for automatically generating high-quality few-shot prompts. A task is converted into a dense vector representation by a neural embedding model. This vector representation is then used to curate a pool of instructions previously utilized for similar tasks. Instructions are probabilistically sampled from this pool based on their instruction scores and incorporated into a few-shot prompt. The few-shot prompt is fed into the LLM. If the prompt is successful, the generated instruction is added to the population and the scores of the utilized instructions are updated.

contribute to prompt success will not yield successful prompts in the following turns. Because of this, instructions that do contribute to the the success will end up having higher scores in the long run.

When calculating sample weights, we take some steps to ensure desirable properties.

First, a valid probability distribution requires non-negative weights. However, cosine similarity is bounded in $[-1, 1]$. In order to ensure non-negativity, we add 1.0 to semantic similarity and move the bounds up to $[0, 2]$. While sampling based on past performance leads to a positive feedback loop where successful instructions are more likely to be sampled, our early experiments revealed it may also lead to undersampling instructions that are generated later in the training. To compensate, we apply the *log* function to the instruction scores and thereby soften the positive feedback loop. The default instruction score of 1 then becomes $log(1) = 0$. To prevent 0 sample weights by default, we add a $+1$ smoothing factor. Bringing all these steps together, our formula for calculating the sample weights is as follows:

$$(cosine\_similarity + 1.0) \cdot (log(instruction\_score) + 1.0) \tag{3.1}$$

Figure 3.2 demonstrates how the relevant tasks are selected for a few-shot prompt based on an arbitrary example task.

3.1.2. Testing

Before testing, we normalize the instruction scores based on the time they were added to the population. The intuition behind this is that the instructions that were present in the population for a longer time will have had more opportunities to be sampled, which, all other things being equal, will naturally lead to a higher instruction score. We normalize instruction scores against a fitted log curve:

$$instruction\_score = \beta_0 + \beta_1 \cdot log(instruction\_id)$$

Where instruction ids are sequential and increasing.

Apart from the normalization of the instruction scores, we follow the same pooling + sampling approach as we had during training. Additionally, we propose multiple sampling approaches that can be used during testing time:

- **Similarity adjusted probabilistic sampling:** Same as in training.
- **Similarity adjusted top-N sampling:** Calculate sample weights like training and select instructions with top weights.
- **Probabilistic sampling:** Only use instruction scores to derive sample weights.
- **Top-N sampling:** Select instructions with top scores.

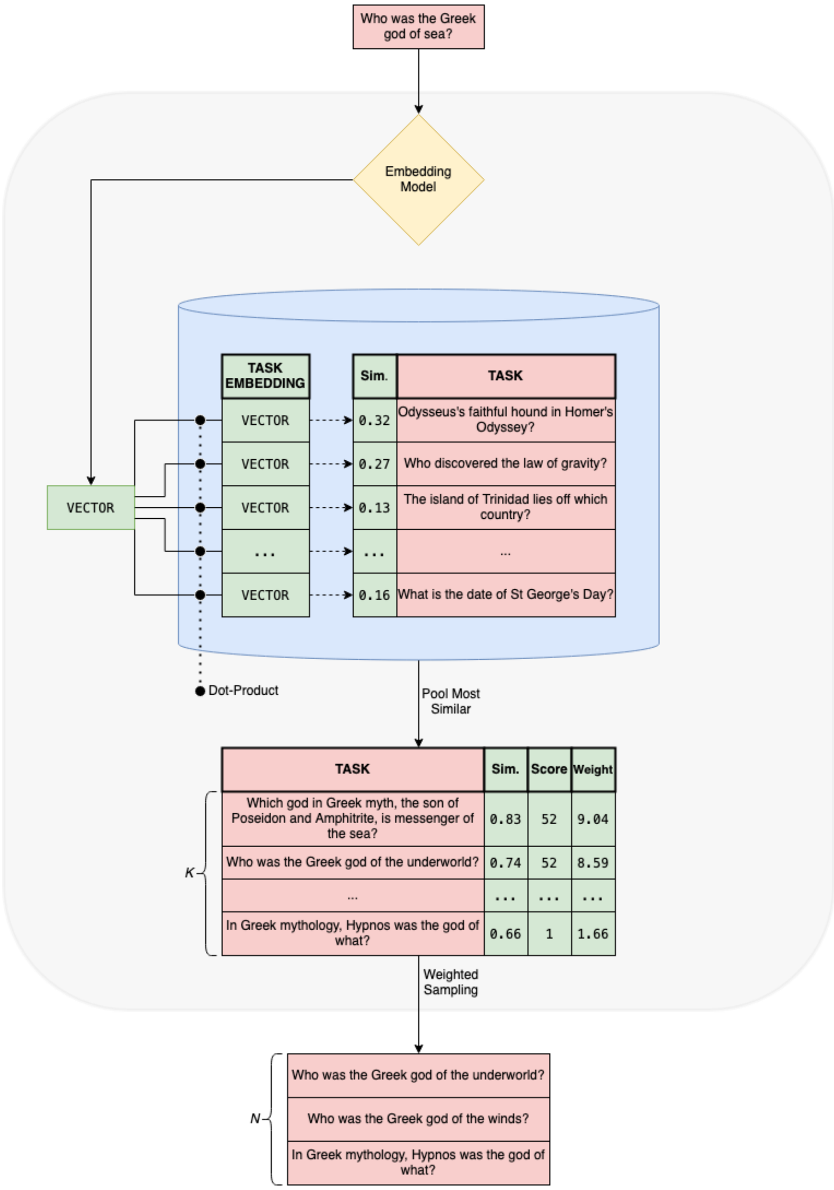

Figure 3.2.: A demonstration of our pipeline for selecting relevant tasks. First, an embedding model encodes the question *Who was the Greek god of sea?* into a vector representation. The dot-products between this vector and vectors of all known tasks are calculated as a measure of task similarity. We select K most similar tasks to form a pool, and then sample from this pool N tasks based on weights calculated via Equation 3.1.

4. Experiments

While our approach is generally adaptable; in our experiments, we focus on question answering with a RAG agent. We test our approach by generating few-shot prompts for a Planner agent from the PWS framework. We implement a version of this framework that is able to power multiples agent with a single local LLM.[1] Along with the usual Planner and Solver agents, we employ two types of worker agents: LLM-Worker and Wiki-Worker. We utilize the ~2000 instructions published by the original authors as the seed population. For all our experiments we set pool size $K = 50$ and number of instructions $N = 3$ unless stated otherwise. We leverage a LLM to extract minimum viable answer spans from the Solver response and we report mean exact-match accuracy between the extracted answers and the ground truth labels as our performance metric.

We utilize the StableBeluga (Mahan et al. n.d.) (Touvron et al. 2023) (Mukherjee et al. 2023) models as the LLM and the sentence transformer model `all-MiniLM-L6-v2` (Reimers and Gurevych 2019) as the embedding model. We select the LLM text generation hyper-parameters via grid search.[2] We use HuggingFace Accelerate (Gugger et al. 2022) for GPU acceleration during training and testing.

4.1. Training

We train our system on the TriviaQA (Joshi et al. 2017) training set using an 8-bit quantized 13B parameter LLM. We perform a single pass over the training set.

4.2. Testing

We test our system in multiple settings:

- **Sampling Tests:** We test the various sampling approaches mentioned in the previous chapter using a 13B model on the TriviaQA validation set. For ablation, we test zero-shot direct prompting without PWS, PWS with random instructions and PWS with instructions randomly sampled from the pool of similar tasks.

[1]For an in-detail explanation of our PWS implementation, please see Appendix A.
[2]The details of the hyper-parameter optimization process can be seen in Appendix B.

- **Prompt Size Tests:** We test the effects of prompt size by varying the number of instructions in the few-shot prompt. We test for $N \in \{3, 5, 8, 12\}$ on the TriviaQA validation set.

- **Cross-model & Cross-distributional Tests:** We test our system under three conditions; using a different model in the same distribution (70B model on TriviaQA validation set), using the same model in the cross-distribution setting (13B model on HotpotQA (Z. Yang et al. 2018) validation set), and using a different model in the cross-distribution setting (70B model on HotpotQA validation set). For ablation, we test alternative strategies as mentioned in the sampling tests.

5. Results & Discussion

5.1. Training

Our system takes ~111 hours to process ~138K question/answer pairs with 8-fold parallelization on a cluster of 8 NVIDIA RTX A6000 GPUs. This results in the generation of ~75K novel instructions that lead to correct answers, representing a ~56% accuracy rate.

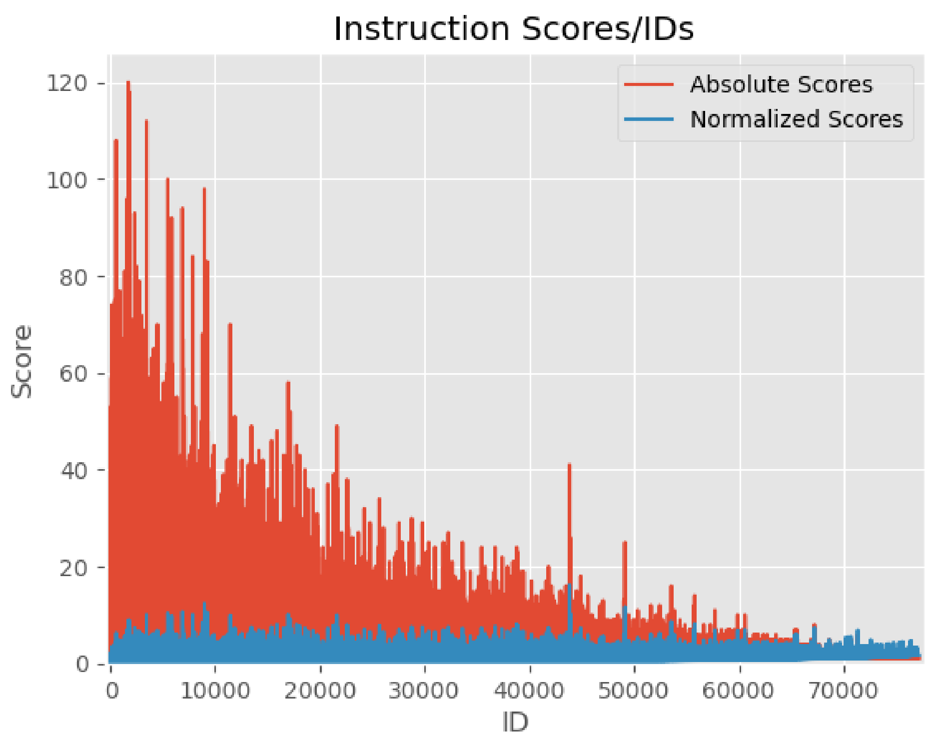

Figure 5.1.: Instruction scores and normalized instruction scores plotted against instruction ids. The outliers in the normalized instruction scores serve as evidence for the intuition that instruction quality has an impact on model performance.

Figure 5.1 shows a plot of instruction scores and normalized instruction scores plotted against instruction ids. The effects of early vs late inclusion is clearly visible in the plot where the absolute instruction scores follow a negative log curve, motivating the need for normalization Normalizing the scores flattens the curve, yet outliers remain present. This could be seen as a confirmation of our intuition that instruction quality has a measurable impact on model performance. In order to illustrate this, let's consider the following: We calculate the instruction scores as:

$$s_x = \sum_{Y_x} y$$

where Y_x denotes the set of binary exact match accuracy for all the model outcomes that include instruction x in the few-shot prompt. If instruction quality and the model performance are independent, then:

$$Pr(y = 1 | x \in prompt) \equiv Pr(y = 1)$$

Assuming that the model performance on the tasks from the same dataset is identically and independently distributed:

$$E[s_x] = E[\sum_{Y_x} y] \sim |Y_x|$$

Which means that the expected value of an instruction's score would be proportional to the number of times it was sampled. After normalizing for differences in sampling frequency that result from the time of introduction into the population, we would expect the scores to show a flat line. As such, outliers in the normalized instruction scores serve as counter evidence to the idea that instruction quality and model performance are independent.

We plot the rolling average of exact match accuracy in Figure 5.2 in order to demonstrate that our system is learning. Even though the rolling average seems unstable, fitting a linear model on question id as the predictive variable and the exact match score as the target variable shows that the improvement in model performance is statistically significant with a p-value of ~0.00137.

5.2. Testing

5.2.1. Sampling Tests

We test our sampling strategies on TriviaQA validation set using a 13B parameter model. (Table 5.1)

Adjusting for similarity seems to benefit when sampling probabilistically but not when selecting top values. Probabilistic sampling with similarity adjusted scores

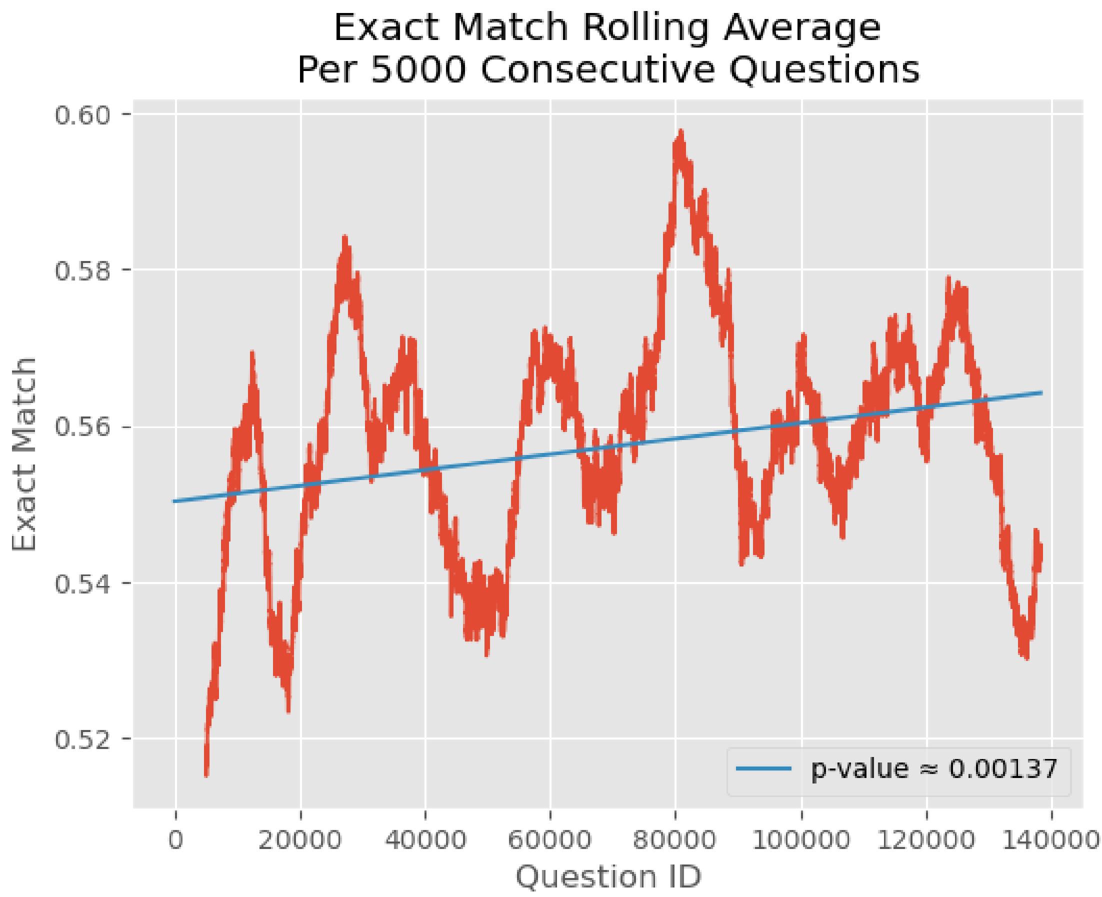

Figure 5.2.: Exact match accuracy rolling average per 5000 consecutive questions. The performance improvement is statistically significant, albeit minor.

achieve the best results. Based on this, we use the probabilistic sampling with similarity adjusted scores for the subsequent tests.

In order to illustrate a possible relation between the sample weights and the model performance, we plot in Figure 5.3 the sample weights of each instruction from each prompt curated through our best approach in the sampling tests. We additionally plot the sum of the sample weights for each prompt. The plots reveal no relation between the instruction scores and the model performance. We are as well not able to otherwise establish a statistically significant linear relation between the instruction scores and the model performance.

For ablation; we test zero-shot direct prompting without PWS, PWS with randomly

Mean Exact-Match	`Probabilistic`	`Top Selection`
`Similarity Adjusted`	**0.559**	0.528
`No Adjustment`	0.543	0.531

Table 5.1.: Sampling tests results. Values signify mean exact-match accuracy. *Similarity Adjusted Probabilistic Sampling* outperforms other sampling strategies.

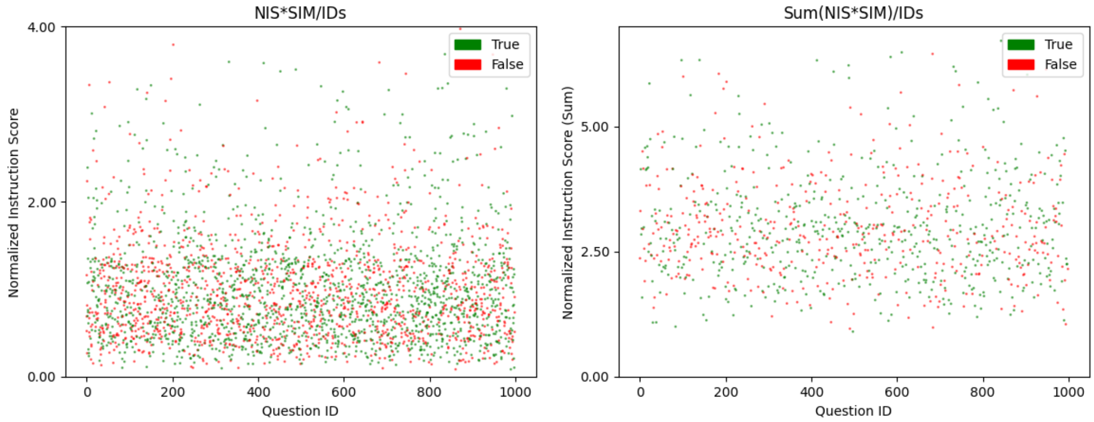

Figure 5.3.: On the left: the sample weights of each instruction from each prompt. On the right: sum of the sample weights for each prompt. The green dots represent the correct model outputs, whereas the red dots represent the incorrect outputs. The plots reveal no relation between the instruction scores and the model performance.

selected instructions and PWS with instructions randomly sampled from the similar pool of instructions. The results can be seen in Table 5.2. It is interesting to note that PWS under-performs direct prompting in all alternative sampling contexts. Our approach, on the other hand, outperforms all alternatives and improves ~1% over direct prompting. This could be seen as an empirical confirmation that our approach is able to discern high-quality instructions that lead to better few-shot performance.

5.2.2. Prompt Size Tests

We test the effects of prompt size as measured by the number of incorporated demonstrations into the few-shot prompt. (Table 5.3) We use a 13B parameter model for the prompt size tests.

Counter-intuitively, the shortest prompt achieves the best performance. This could be due to the length and complexity of the instructions making it hard for the model to

DP	*RS*	*SP+RS*	*Our Approach*
0.552	0.537	0.537	**0.559**

Table 5.2.: Ablation tests results. Values signify mean exact-match accuracy. *DP*, *RS* and *SP+RS* stand for *Direct Prompting*, *Random Sampling* and *Similarity Pooling + Random Sampling* ; respectively. Our approach refers to *Similarity Adjusted Probabilistic Sampling* that we established as the best sampling strategy in the earlier experiments. Our approach achieves the best performance.

	3	5	8	12
Exact Match	**0.559**	0.537	0.531	0.544

Table 5.3.: Results of the prompt size tests. Values signify mean exact-match accuracy. 3-Shot prompts achieve the best performance.

attend to a high number of demonstrative instructions. Another possibility is that the instruction scores are tied to the number of instructions present in the few-shot prompt during the training phase. In other words, our system might be selecting instructions that work the best in a 3-shot prompt since we train the system in the 3-shot setting.

5.2.3. Cross-Model & Cross-Distributional Tests

Next we observe the effects of switching models and distributional shifts. We perform cross-model tests using an 8-bit quantized 70B model. We perform cross-distributional tests on HotpotQA validation set. The results can be seen in Table 5.4.

Dataset	*Model Size*	*DP*	*RS*	*SP+RS*	*Our Approach*
`TriviaQA`	`70B`	0.616	**0.630**	0.625	0.629
`HotpotQA`	`13B`	0.212	0.244	**0.250**	0.245
`HotpotQA`	`70B`	0.261	0.272	**0.278**	0.264

Table 5.4.: Cross-Distributional & cross-model tests results. Values signify mean exact-match accuracy. *DP*, *RS* and *SP+RS* stand for *Direct Prompting*, *Random Sampling* and *Similarity Pooling + Random Sampling* ; respectively. Our approach refers to the best sampling approach established in the earlier experiments. (Similarity Adjusted Probabilistic Sampling)

Our approach outperforms direct prompting in all contexts but it does not lead to substantial improvements over random sampling in cross-model and cross-distributional settings. When the task-distribution is shifted, our approach remains on par with random sampling. When the model is switched, our approach is outperformed by random sampling. Performance degradation over random sampling is the greatest when the model is switched and the distribution is shifted at the same time. These observations indicate that the learned instruction scores are model and distribution specific.

6. Conclusion

6.1. Limitations

While our framework is generally applicable, there are some drawbacks that are worth mentioning. First, our approach depends on a model's in-context learning ability and is therefore inherently limited by it. It might not be feasible to learn all tasks in-context and some tasks may always require task specific fine-tuning. Another potential drawback is the fact that the few-shot setting is computationally more expensive compared to the zero-shot setting . For long instructions, incorporating multiple instructions in to the few-shot prompt may not be feasible, either due to a model's context length or due the increased computational costs. Furthermore, our experiments reveal that the learned instruction scores seem to be specific to the model and the task distribution. Finally, running LLMs is an inherently compute intensive process which prevents us from fully exploring the limits of our framework.

6.2. Future Work

Our approach could be extended in various research directions. One obvious follow-up study would be to fine-tune a model on the generated instructions. Fine-tuning on subsets of the generated instructions with differing average instruction scores could yield valuable information on the effects of instruction quality for instruction-tuning. Another potential extension could be to test our approach on cross-task adaptation to see if instructions for a certain type of task could be beneficial for another type of task. Applying our approach on code LLMs could also yield interesting results as they are by default trained on instruction sets. Performing multiple passes over data could stabilise the instruction scores and allow for better generalization. Our work could also be adapted as a continuous learning approach by introducing a periodical normalization of the instruction scores. Finally, our work could also be extended to the multi-modal context and be leveraged, for example, to instruct text-to-image models to produce higher quality images.

A. Planner-Worker-Solver

We implement an adapted version of the PWS framework that allows us to power all the agents with a single local LLM. Figure A.1 illustrates our implementation.

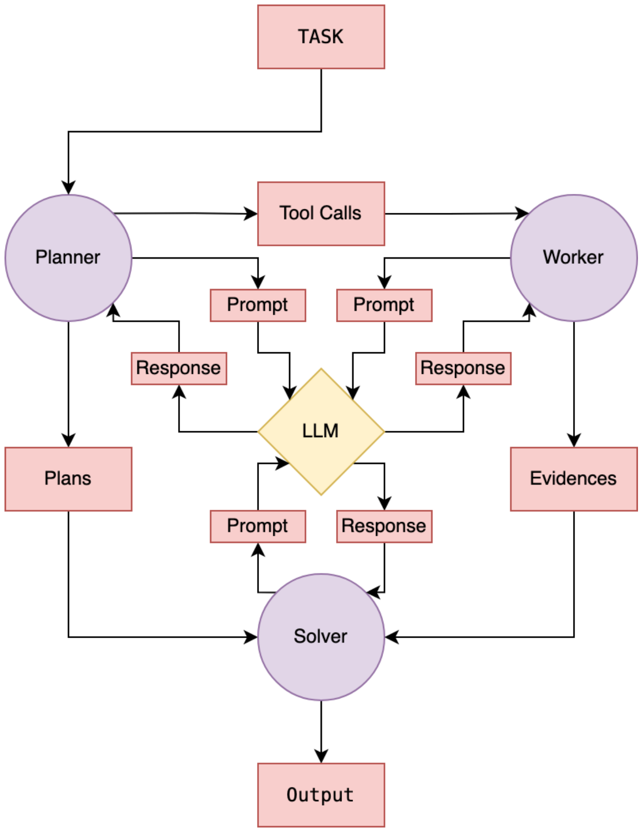

Figure A.1.: Our PWS implementation leverages a single local LLM to power all agents. Planner accepts a task, prompts the LLM to produce plans and then parses the LLM response to extract plans and tool calls. Tool calls are passed on to Worker, which executes them to produce evidences. Finally, plans and evidences are combined into a prompt by Solver, which leverages the LLM to integrate the available information and produce a final answer.

All agents leverage the same LLM via different prompts to fulfill their functionality. Planner utilizes the LLM to produce plans, Worker utilize tools (including the LLM) to collect evidences and Solver integrates plans and evidences using the LLMs. When decoupled from the LLM, each agent essentially becomes a prompt-builder and/or a response parser. To better illustrate how the agents interact, we list some example prompts and responses.

Listing A.1 shows an automatically generated Planner prompt. The prompt is constructed based on the given task (*Who invented the combustion engine?*) and the relevant task demonstrations obtained through our evolutionary framework. The Planner calls the LLM with this prompt, and receives in response Listing A.2.

Planner parses this response; passes the tool calls to Worker and the plans to Solver. Listing A.3 shows the Worker response based on the tool calls passed on by Planner.

Solver combines the plans from Planner and the evidences from the Worker response into the Solver prompt seen in Listing A.4. LLM response to the Solver prompt (Listing A.5) is the answer to our original question.

System:
You are an advanced AI capable of making plans to solve complex problems. For the following tasks, make plans that can solve the problem step−by−step. For each plan, indicate which external tool together with tool input to retrieve evidence. You can store the evidence into a variable #E that can be called by later tools. (Plan, #E1, Plan, #E2, Plan, ...)
Tools can be one of the following:
LLM[input]: A pretrained LLM like yourself. Useful when you need to act with general world knowledge and common sense. Prioritize it when you are confident in solving the problem yourself. Input can be any instruction.
Wikipedia[input]: Worker that search for similar page contents from Wikipedia. Useful when you need to get holistic knowledge about people, places, companies, historical events, or other subjects. The response are long and might contain some irrelevant information. Input should be a search query.
Describe your plans with rich details. Each Plan should be followed by only one #E. Answer each question directly with plans.

User:
Which engineer first used the term Horsepower?

Assistant:
Plan: Search for more information about the engineer who first used the term Horsepower.
#E1 = Wikipedia[Engineer who first used the term Horsepower]
Plan: Identify the engineer who first used the term Horsepower.
#E2 = LLM[Who first used the term Horsepower? Given context: #E1]

User:
Which English chemist and physicist invented the dynamo?

Assistant:
Plan: Search for more information about the English chemist and physicist who invented the dynamo.
#E1 = Wikipedia[English chemist and physicist who invented the dynamo]
Plan: Ask the LLM for the name of the English chemist and physicist who invented the dynamo.
#E2 = LLM[Who was the English chemist and physicist who invented the dynamo? Given context: #E1]

User:
What was the first name of Mr. Diesel, the inventor of the Diesel engine?

Assistant:
Plan: Search for information about the inventor of the Diesel engine.

#E1 = Wikipedia[Diesel engine]
Plan: Identify the first name of the inventor of the Diesel engine.
#E2 = LLM[What was the first name of Mr. Diesel, the inventor of the Diesel engine? Given context: #E1]

User:
Who invented the combustion engine?

Assistant:
———————————————————————————————————

Listing A.1: Few-Shot Planner Prompt

———————————————————————————————————
Plan: Search for information about the inventor of the combustion engine.
#E1 = Wikipedia[Combustion engine]
Plan: Identify the inventor of the combustion engine.
#E2 = LLM[Who invented the combustion engine? Given context: #E1]
———————————————————————————————————

Listing A.2: Planner Response

```
------------------------------------------------------------------
#E1 = Wikipedia[Combustion engine]
An internal combustion engine (ICE or IC engine) is a heat engine in which the combustion
    of a fuel occurs with an oxidizer (usually air) in a combustion chamber that is an integral
     part of the working fluid flow circuit. In an internal combustion engine, the expansion
    of the high-temperature and high-pressure gases produced by combustion applies
    direct force to some component of the engine. The force is typically applied to pistons (
    piston engine), turbine blades (gas turbine), a rotor (Wankel engine), or a nozzle (jet
    engine). This force moves the component over a distance, transforming chemical energy
    into kinetic energy which is used to propel, move or power whatever the engine is
    attached to.
The first commercially successful internal combustion engine was created by Etienne Lenoir
    around 1860, and the first modern internal combustion engine, known as the Otto engine
    , was created in 1876 by Nicolaus Otto. The term internal combustion engine usually
    refers to an engine in which combustion is intermittent, such as the more familiar two-
    stroke and four-stroke piston engines, along with variants, such as the six-stroke piston
    engine and the Wankel rotary engine. A second class of internal combustion engines use
    continuous combustion: gas turbines, jet engines and most rocket engines, each of which
     are internal combustion engines on the same principle as previously described. (
    Firearms are also a form of internal combustion engine, though of a type so specialized
    that they are commonly treated as a separate category, along with weaponry such as
    mortars and anti-aircraft cannons.) In contrast, in external combustion engines, such as
    steam or Stirling engines, energy is delivered to a working fluid not consisting of, mixed
    with, or contaminated by combustion products. Working fluids for external combustion
    engines include air, hot water, pressurized water or even boiler-heated liquid sodium.
While there are many stationary applications, most ICEs are used in mobile ap

#E2 = LLM[Who invented the combustion engine? Given context: #E1]
Etienne Lenoir invented the first commercially successful internal combustion engine around
    1860.
------------------------------------------------------------------
```

Listing A.3: Worker Response

```
-----------------------------------------------------------------------
### System:
You are an advanced AI capable of solving tasks based on evidence. Solve the following task
    or problem based on the provided plans and corresponding evidences. Keep your
    responses direct and concise.

### User:
Who invented the combustion engine?
Plan: Search for information about the inventor of the combustion engine.
Evidence: An internal combustion engine (ICE or IC engine) is a heat engine in which the
    combustion of a fuel occurs with an oxidizer (usually air) in a combustion chamber that
    is an integral part of the working fluid flow circuit. In an internal combustion engine, the
     expansion of the high-temperature and high-pressure gases produced by combustion
    applies direct force to some component of the engine. The force is typically applied to
    pistons (piston engine), turbine blades (gas turbine), a rotor (Wankel ...
Plan: Identify the inventor of the combustion engine.
Evidence: Etienne Lenoir invented the first commercially successful internal combustion
    engine around 1860....
Based on the provided evidence answer the following question directly and concisely: Who
    invented the combustion engine?

### Assistant:
-----------------------------------------------------------------------
```

Listing A.4: Solver Prompt

```
-----------------------------------------------------------------------
Etienne Lenoir invented the combustion engine.
-----------------------------------------------------------------------
```

Listing A.5: Solver response

B. Hyper-Parameter Optimization

We set the decoding hyper-parameters `Temperature` and `Repetition Penalty` via grid search over the parameter sets $\{0.01, 0.25, 0.50, 0.75, 1.0\}$ and $\{1.0, 1.1, 1.2, 1.3\}$, respectively. We evaluate each configuration on 30 samples and report exact match score in Table B.1.

	Repetition Penalty			
Temperature	1.0	1.1	1.2	1.3
0.01	**18**	**18**	12	9
0.25	17	17	12	6
0.50	15	16	12	10
0.75	16	15	12	4
1.00	**18**	14	11	6

Table B.1.: Hyper-Parameter optimization results. The table shows each configuration's exact match score on 30 samples.

As the top three configurations share the same score, we conduct another hyper-parameter optimization among these configurations over 100 samples. The results can be seen in Table B.2

Temperature/Repetition Penalty	Exact Match Score
0.01/1.0	**49**
0.01/1.1	**49**
1.0/1.0	47

Table B.2.: Second round hyper-parameter optimization results. The table shows each configuration's exact match score on 100 samples.

The second round of the hyper-parameter optimization shows that the top two configurations perform the same. While the configurations share the same `Temperature` value, the `Repetition Penalty` values are different. To break the tie, we set `Repetition Penalty` to `1.0` as it is the default value, and set `Temperature` to `0.01`.

Abbreviations

AI Artificial Intelligence

AGI Artificial General Intelligence

API Application Programming Interface

BERT Bidirectional Encoder Representations from Transformers

DL Deep Learning

EA Evolutionary Algorithm

GPT Generative Pre-Trained Transformer

LLM Large Language Model

LSTM Long Short-Term Memory

NLP Natural Language Processing

NN Neural Network

PWS Planner-Worker-Solver

RAG Retrieval Augmented Generation

RLHF Reinforcement Learning from Human Feedback

seq2seq Sequence to Sequence

SOTA State-of-the-Art

T5 Text-to-Text Transfer Transformer

List of Figures

List of Tables

Bibliography

Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). *Layer Normalization*. arXiv: 1607.06450 [stat.ML].

Banko, M. and E. Brill (July 2001). "Scaling to Very Very Large Corpora for Natural Language Disambiguation." In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*. Toulouse, France: Association for Computational Linguistics, pp. 26–33. DOI: 10.3115/1073012.1073017.

Bender, E. M., T. Gebru, A. McMillan-Major, and S. Shmitchell (2021). "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Virtual Event, Canada: Association for Computing Machinery, pp. 610–623. ISBN: 9781450383097. DOI: 10.1145/3442188.3445922.

Bommasani, R., D. A. Hudson, E. Adeli, R. B. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. S. Chatterji, A. S. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. D. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. S. Krass, R. Krishna, R. Kuditipudi, and et al. (2021). "On the Opportunities and Risks of Foundation Models." In: *CoRR* abs/2108.07258. arXiv: 2108.07258.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020). "Language Models Are Few-Shot Learners." In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc. ISBN: 9781713829546.

Bubeck, S., V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang (2023). *Sparks of Artificial General Intelligence: Early experiments with GPT-4*. arXiv: 2303.12712 [cs.CL].

Cai, T., X. Wang, T. Ma, X. Chen, and D. Zhou (2023). *Large Language Models as Tool Makers*. arXiv: 2305.17126 [cs.LG].

Chen, A., D. Dohan, and D. So (2023). "EvoPrompting: Language Models for Code-Level Neural Architecture Search." In: *Thirty-seventh Conference on Neural Information Processing Systems*.

Chen, W., X. Ma, X. Wang, and W. W. Cohen (2023). "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks." In: *Transactions on Machine Learning Research*. ISSN: 2835-8856.

Child, R., S. Gray, A. Radford, and I. Sutskever (2019). "Generating long sequences with sparse transformers." In: *arXiv preprint arXiv:1904.10509*.

Cho, K., B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (Oct. 2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

Christiano, P. F., J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei (2017). "Deep Reinforcement Learning from Human Preferences." In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc.

Dai, A. M. and Q. V. Le (2015). "Semi-Supervised Sequence Learning." In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'15. Montreal, Canada: MIT Press, pp. 3079–3087.

Davis, E. and S. Aaronson (2023). *Testing GPT-4 with Wolfram Alpha and Code Interpreter plug-ins on math and science problems*. arXiv: 2308.05713 [cs.AI].

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (June 2019). "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, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

Du, N., Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, B. Zoph, L. Fedus, M. P. Bosma, Z. Zhou, T. Wang, E. Wang, K. Webster, M. Pellat, K. Robinson, K. Meier-Hellstern, T. Duke, L. Dixon, K. Zhang, Q. Le, Y. Wu, Z. Chen, and C. Cui (July 2022). "GLaM: Efficient Scaling of Language Models with Mixture-of-Experts." In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, pp. 5547–5569.

Dziri, N., X. Lu, M. Sclar, X. L. Li, L. Jiang, B. Y. Lin, P. West, C. Bhagavatula, R. L. Bras, J. D. Hwang, S. Sanyal, S. Welleck, X. Ren, A. Ettinger, Z. Harchaoui, and Y. Choi

(2023). *Faith and Fate: Limits of Transformers on Compositionality*. arXiv: 2305.18654 [cs.CL].

Fu, Y., H. Peng, L. Ou, A. Sabharwal, and T. Khot (2023). "Specializing Smaller Language Models towards Multi-Step Reasoning." In: *Proceedings of the 40th International Conference on Machine Learning*. ICML'23. Honolulu, Hawaii, USA: JMLR.org.

Gao, L., A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig (2023). "PAL: Program-Aided Language Models." In: *Proceedings of the 40th International Conference on Machine Learning*. ICML'23. Honolulu, Hawaii, USA: JMLR.org.

Goldberg, Y. (Sept. 2016). "A Primer on Neural Network Models for Natural Language Processing." In: *J. Artif. Int. Res.* 57.1, pp. 345–420. ISSN: 1076-9757.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Goodman, J. T. (2001). "A bit of progress in language modeling." In: *Computer Speech & Language* 15.4, pp. 403–434. ISSN: 0885-2308. DOI: https://doi.org/10.1006/csla.2001.0174.

Gudibande, A., E. Wallace, C. Snell, X. Geng, H. Liu, P. Abbeel, S. Levine, and D. Song (2023). *The False Promise of Imitating Proprietary LLMs*. arXiv: 2305.15717 [cs.CL].

Gugger, S., L. Debut, T. Wolf, P. Schmid, Z. Mueller, S. Mangrulkar, M. Sun, and B. Bossan (2022). *Accelerate: Training and inference at scale made simple, efficient and adaptable.* https://github.com/huggingface/accelerate.

Guo, H., J. Yang, J. Liu, L. Yang, L. Chai, J. Bai, J. Peng, X. Hu, C. Chen, D. Zhang, X. Shi, T. Zheng, L. Zheng, B. Zhang, K. Xu, and Z. Li (2023). *OWL: A Large Language Model for IT Operations*. arXiv: 2309.09298 [cs.CL].

Gutiérrez, L. and B. Keith (2019). "A Systematic Literature Review on Word Embeddings." In: *Trends and Applications in Software Engineering*. Ed. by J. Mejia, M. Muñoz, Á. Rocha, A. Peña, and M. Pérez-Cisneros. Cham: Springer International Publishing, pp. 132–141. ISBN: 978-3-030-01171-0.

He, H., H. Zhang, and D. Roth (2022). *Rethinking with Retrieval: Faithful Large Language Model Inference*. arXiv: 2301.00303 [cs.CL].

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep Residual Learning for Image Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

Heaven, W. D. (Feb. 2021). "GPT-3 / 10 Breakthrough Technologies 2021." In: *MIT Technology Review* 124.2. Ed. by D. Rotman and A. Nordrum, pp. 34–35.

Hendrycks, D., C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt (2021). "Measuring Mathematical Problem Solving With the MATH Dataset." In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren and S. Yeung. Vol. 1. Curran.

Hestness, J., N. Ardalani, and G. Diamos (2019). "Beyond Human-Level Accuracy: Computational Challenges in Deep Learning." In: *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*. PPoPP '19. Washington, District of Columbia: Association for Computing Machinery, pp. 1–14. ISBN: 9781450362252. DOI: 10.1145/3293883.3295710.

Hestness, J., S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou (2017). "Deep learning scaling is predictable, empirically." In: *arXiv preprint arXiv:1712.00409*.

Hochreiter, S., Y. Bengio, P. Frasconi, and J. Schmidhuber (2001). "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies." In: *A Field Guide to Dynamical Recurrent Neural Networks*. Ed. by S. C. Kremer and J. F. Kolen. IEEE Press.

Hochreiter, S. and J. Schmidhuber (Nov. 1997). "Long Short-Term Memory." In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

Hoffmann, J., S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre (2022). *Training Compute-Optimal Large Language Models*. arXiv: 2203.15556 [cs.CL].

Hong, S., M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber (2023). *MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework*. arXiv: 2308.00352 [cs.AI].

Howard, J. and S. Ruder (July 2018). "Universal Language Model Fine-tuning for Text Classification." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 328–339. DOI: 10.18653/v1/P18-1031.

Huang, J. and K. C.-C. Chang (July 2023). "Towards Reasoning in Large Language Models: A Survey." In: *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by A. Rogers, J. Boyd-Graber, and N. Okazaki. Toronto, Canada: Association for Computational Linguistics, pp. 1049–1065. DOI: 10.18653/v1/2023.findings-acl.67.

Joshi, M., E. Choi, D. Weld, and L. Zettlemoyer (July 2017). "TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension." In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by R. Barzilay and M.-Y. Kan. Vancouver, Canada: Association for Computational Linguistics, pp. 1601–1611. DOI: 10.18653/v1/P17-1147.

Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing (2nd Edition)*. USA: Prentice-Hall, Inc. ISBN: 0131873210.

Kaplan, J., S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei (2020). "Scaling Laws for Neural Language Models." In: *CoRR* abs/2001.08361. arXiv: 2001.08361.

Khattab, O., K. Santhanam, X. L. Li, D. Hall, P. Liang, C. Potts, and M. Zaharia (2023). *Demonstrate-Search-Predict: Composing retrieval and language models for knowledge-intensive NLP*. arXiv: 2212.14024 [cs.CL].

Kim, G., P. Baldi, and S. M. McAleer (2023). "Language Models can Solve Computer Tasks." In: *Thirty-seventh Conference on Neural Information Processing Systems*.

Kojima, T., S. (Gu, M. Reid, Y. Matsuo, and Y. Iwasawa (2022). "Large Language Models are Zero-Shot Reasoners." In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., pp. 22199–22213.

Kung, P.-N. and N. Peng (2023). *Do Models Really Learn to Follow Instructions? An Empirical Study of Instruction Tuning*. arXiv: 2305.11383 [cs.AI].

Lazaridou, A., E. Gribovskaya, W. J. Stokowiec, and N. Grigorev (2023). *Internet-augmented language models through few-shot prompting for open-domain question answering*.

Lehman, J., J. Gordon, S. Jain, K. Ndousse, C. Yeh, and K. O. Stanley (2024). "Evolution Through Large Models." In: *Handbook of Evolutionary Machine Learning*. Ed. by W. Banzhaf, P. Machado, and M. Zhang. Singapore: Springer Nature Singapore, pp. 331–366. ISBN: 978-981-99-3814-8. DOI: 10.1007/978-981-99-3814-8_11.

Lewis, P., E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 9459–9474.

Li, S., J. Chen, Y. Shen, Z. Chen, X. Zhang, Z. Li, H. Wang, J. Qian, B. Peng, Y. Mao, W. Chen, and X. Yan (2022). *Explanations from Large Language Models Make Small Reasoners Better*. arXiv: 2210.06726 [cs.CL].

Li, X., P. Yu, C. Zhou, T. Schick, L. Zettlemoyer, O. Levy, J. Weston, and M. Lewis (2023). *Self-Alignment with Instruction Backtranslation*. arXiv: 2308.06259 [cs.CL].

Li, X., R. Zhao, Y. K. Chia, B. Ding, S. Joty, S. Poria, and L. Bing (2023). *Chain-of-Knowledge: Grounding Large Language Models via Dynamic Knowledge Adapting over Heterogeneous Sources*. arXiv: 2305.13269 [cs.CL].

Lipton, Z. C. (2015). "A Critical Review of Recurrent Neural Networks for Sequence Learning." In: *CoRR* abs/1506.00019. arXiv: 1506.00019.

Liu, T. and B. K. H. Low (2023). *Goat: Fine-tuned LLaMA Outperforms GPT-4 on Arithmetic Tasks*. arXiv: 2305.14201 [cs.LG].

Magister, L. C., J. Mallinson, J. Adamek, E. Malmi, and A. Severyn (July 2023). "Teaching Small Language Models to Reason." In: *Proceedings of the 61st Annual Meeting of the*

Association for Computational Linguistics (Volume 2: Short Papers). Ed. by A. Rogers, J. Boyd-Graber, and N. Okazaki. Toronto, Canada: Association for Computational Linguistics, pp. 1773–1781. DOI: 10.18653/v1/2023.acl-short.151.

Mahan, D., R. Carlow, L. Castricato, N. Cooper, and C. Laforte (n.d.). *Stable Beluga models*.

Menick, J., M. Trebacz, V. Mikulik, J. Aslanides, F. Song, M. Chadwick, M. Glaese, S. Young, L. Campbell-Gillingham, G. Irving, and N. McAleese (2022). *Teaching language models to support answers with verified quotes*. arXiv: 2203.11147 [cs.CL].

Meyerson, E., M. J. Nelson, H. Bradley, A. Gaier, A. Moradi, A. K. Hoover, and J. Lehman (2023). *Language Model Crossover: Variation through Few-Shot Prompting*. arXiv: 2302.12170 [cs.NE].

Mikolov, T., E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin (May 2018). "Advances in Pre-Training Distributed Word Representations." In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA).

Mikolov, T., I. Sutskever, K. Chen, G. Corrado, and J. Dean (2013). "Distributed Representations of Words and Phrases and Their Compositionality." In: NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., pp. 3111–3119.

Mikolov, T., W.-t. Yih, and G. Zweig (June 2013). "Linguistic Regularities in Continuous Space Word Representations." In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, pp. 746–751.

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). "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 Y. Bengio and Y. LeCun.

Mukherjee, S., A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and A. Awadallah (2023). *Orca: Progressive Learning from Complex Explanation Traces of GPT-4*. arXiv: 2306.02707 [cs.CL].

Nye, M., A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, C. Sutton, and A. Odena (2022). *Show Your Work: Scratchpads for Intermediate Computation with Language Models*.

Ouyang, L., J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe (2022). "Training language models to follow instructions with human feedback." In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., pp. 27730–27744.

Paranjape, B., S. Lundberg, S. Singh, H. Hajishirzi, L. Zettlemoyer, and M. T. Ribeiro (2023). *ART: Automatic multi-step reasoning and tool-use for large language models*. arXiv: 2303.09014 [cs.CL].

Patil, S. G., T. Zhang, X. Wang, and J. E. Gonzalez (2023). *Gorilla: Large Language Model Connected with Massive APIs*. arXiv: 2305.15334 [cs.CL].

Pennington, J., R. Socher, and C. Manning (Oct. 2014). "GloVe: Global Vectors for Word Representation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer (June 2018). "Deep Contextualized Word Representations." In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237. DOI: 10.18653/v1/N18-1202.

Press, O., M. Zhang, S. Min, L. Schmidt, N. A. Smith, and M. Lewis (2023). *Measuring and Narrowing the Compositionality Gap in Language Models*. arXiv: 2210.03350 [cs.CL].

Qin, Y., S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, R. Tian, R. Xie, J. Zhou, M. Gerstein, D. Li, Z. Liu, and M. Sun (2023). *ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs*. arXiv: 2307.16789 [cs.AI].

Radford, A., K. Narasimhan, T. Salimans, I. Sutskever, et al. (2018). "Improving language understanding by generative pre-training." In.

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. (2019). "Language models are unsupervised multitask learners." In: *OpenAI blog* 1.8, p. 9.

Raffel, C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu (Jan. 2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." In: *J. Mach. Learn. Res.* 21.1. ISSN: 1532-4435.

Rajani, N. F., B. McCann, C. Xiong, and R. Socher (July 2019). "Explain Yourself! Leveraging Language Models for Commonsense Reasoning." In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by A. Korhonen, D. Traum, and L. Màrquez. Florence, Italy: Association for Computational Linguistics, pp. 4932–4942. DOI: 10.18653/v1/P19-1487.

Reimers, N. and I. Gurevych (Nov. 2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Rogers, A., O. Kovaleva, and A. Rumshisky (2020). "A Primer in BERTology: What We Know About How BERT Works." In: *Transactions of the Association for Computational Linguistics* 8, pp. 842–866. DOI: 10.1162/tacl_a_00349.

Schick, T., J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom (2023). "Toolformer: Language Models Can Teach Themselves to Use Tools." In: *Thirty-seventh Conference on Neural Information Processing Systems*.

Shen, Y., K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang (2023). *HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face*. arXiv: 2303.17580 [cs.CL].

Shumaker, R., K. Walkup, B. Beck, and G. Burghardt (2011). *Animal Tool Behavior: The Use and Manufacture of Tools by Animals*. Animal Tool Behavior. Johns Hopkins University Press. ISBN: 9781421401287.

Sloss, A. N. and S. Gustafson (2020). "2019 Evolutionary Algorithms Review." In: *Genetic Programming Theory and Practice XVII*. Ed. by W. Banzhaf, E. Goodman, L. Sheneman, L. Trujillo, and B. Worzel. Cham: Springer International Publishing, pp. 307–344. ISBN: 978-3-030-39958-0. DOI: 10.1007/978-3-030-39958-0_16.

Sun, Z., Y. Shen, Q. Zhou, H. Zhang, Z. Chen, D. Cox, Y. Yang, and C. Gan (2023). *Principle-Driven Self-Alignment of Language Models from Scratch with Minimal Human Supervision*. arXiv: 2305.03047 [cs.LG].

Sutskever, I., O. Vinyals, and Q. V. Le (2014). "Sequence to Sequence Learning with Neural Networks." In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, pp. 3104–3112.

Talmor, A., J. Herzig, N. Lourie, and J. Berant (June 2019). "CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge." In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4149–4158. DOI: 10.18653/v1/N19-1421.

Talmor, A., O. Tafjord, P. Clark, Y. Goldberg, and J. Berant (2020). "Leap-of-Thought: Teaching Pre-Trained Models to Systematically Reason over Implicit Knowledge." In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc. ISBN: 9781713829546.

Taori, R., I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto (2023). *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca.

Touvron, H., L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian,

X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom (2023). *Llama 2: Open Foundation and Fine-Tuned Chat Models*. arXiv: 2307.09288 [cs.CL].

Turian, J., L.-A. Ratinov, and Y. Bengio (July 2010). "Word Representations: A Simple and General Method for Semi-Supervised Learning." In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, pp. 384–394.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). "Attention is All You Need." In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., pp. 6000–6010. ISBN: 9781510860964.

Wang, B., S. Min, X. Deng, J. Shen, Y. Wu, L. Zettlemoyer, and H. Sun (July 2023). "Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters." In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by A. Rogers, J. Boyd-Graber, and N. Okazaki. Toronto, Canada: Association for Computational Linguistics, pp. 2717–2739. DOI: 10.18653/v1/2023.acl-long.153.

Wang, Y., Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi (July 2023). "Self-Instruct: Aligning Language Models with Self-Generated Instructions." In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by A. Rogers, J. Boyd-Graber, and N. Okazaki. Toronto, Canada: Association for Computational Linguistics, pp. 13484–13508. DOI: 10.18653/v1/2023.acl-long.754.

Wei, J., M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le (2022). "Finetuned Language Models are Zero-Shot Learners." In: *International Conference on Learning Representations*.

Wei, J., Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H.-h. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus (2022). "Emergent Abilities of Large Language Models." In: *Trans. Mach. Learn. Res.* 2022.

Wei, J., X. Wang, D. Schuurmans, M. Bosma, brian ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou (2022). "Chain of Thought Prompting Elicits Reasoning in Large Language Models." In: *Advances in Neural Information Processing Systems*. Ed. by A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho.

Weiss, K., T. M. Khoshgoftaar, and D. Wang (2016). "A survey of transfer learning." In: *Journal of Big Data* 3.1, p. 9. DOI: 10.1186/s40537-016-0043-6.

Xu, B., Z. Peng, B. Lei, S. Mukherjee, Y. Liu, and D. Xu (2023). *ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models*. arXiv: `2305.18323 [cs.CL]`.

Xu, C., Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang (2023). *WizardLM: Empowering Large Language Models to Follow Complex Instructions*. arXiv: `2304.12244 [cs.CL]`.

Yang, C., X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen (2023). *Large Language Models as Optimizers*. arXiv: `2309.03409 [cs.LG]`.

Yang, Z., P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning (2018). "HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yao, S., D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan (2023). "Tree of thoughts: Deliberate problem solving with large language models." In: *arXiv preprint arXiv:2305.10601*.

Yao, S., J. Zhao, D. Yu, I. Shafran, K. R. Narasimhan, and Y. Cao (2022). "ReAct: Synergizing Reasoning and Acting in Language Models." In: *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Yue, X., X. Qu, G. Zhang, Y. Fu, W. Huang, H. Sun, Y. Su, and W. Chen (2023). *MAmmoTH: Building Math Generalist Models through Hybrid Instruction Tuning*. arXiv: `2309.05653 [cs.CL]`.

Zelikman, E., Y. Wu, J. Mu, and N. Goodman (2022). "STaR: Bootstrapping Reasoning With Reasoning." In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., pp. 15476–15488.

Zhang, S., L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu, et al. (2023). "Instruction Tuning for Large Language Models: A Survey." In: *arXiv preprint arXiv:2308.10792*.

Zhao, W. X., K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen (2023). *A Survey of Large Language Models*. arXiv: `2303.18223 [cs.CL]`.

Zhou, C., P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, S. Zhang, G. Ghosh, M. Lewis, L. Zettlemoyer, and O. Levy (2023). *LIMA: Less Is More for Alignment*. arXiv: `2305.11206 [cs.CL]`.

Zhou, D., N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. V. Le, and E. H. Chi (2023). "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models." In: *The Eleventh International Conference on Learning Representations*.

Zhou, X., G. Li, and Z. Liu (2023). *LLM As DBA*. arXiv: `2308.05481 [cs.DB]`.

Ziegler, D. M., N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving (2019). "Fine-tuning language models from human preferences." In: *arXiv preprint arXiv:1909.08593*.