LARGE LANGUAGE MODELS ARE HUMAN-LEVEL PROMPT ENGINEERS

Yongchao Zhou[1,2,*], **Andrei Ioan Muresanu**[2,3,*], **Ziwen Han**[1,2,*], **Keiran Paster**[1,2], **Silviu Pitis**[1,2], **Harris Chan**[1,2], **Jimmy Ba**[1,2]
[1]University of Toronto [2]Vector Institute [3]University of Waterloo [*]Equal contribution
{yczhou,hanziwen,keirp,spitis,hchan,jba}@cs.toronto.edu
{andrei.muresanu}@uwaterloo.ca

ABSTRACT

By conditioning on natural language instructions, large language models (LLMs) have displayed impressive capabilities as general-purpose computers. However, task performance depends significantly on the quality of the prompt used to steer the model, and most effective prompts have been handcrafted by humans. Inspired by classical program synthesis and the human approach to prompt engineering, we propose *Automatic Prompt Engineer*[1] (APE) for automatic instruction generation and selection. In our method, we treat the instruction as the "program," optimized by searching over a pool of instruction candidates proposed by an LLM in order to maximize a chosen score function. To evaluate the quality of the selected instruction, we evaluate the zero-shot performance of another LLM following the selected instruction. Extensive experiments show that our automatically generated instructions outperform the prior LLM baseline by a large margin and achieve better or comparable performance to the instructions generated by human annotators on 24/24 Instruction Induction tasks and 17/21 curated BIG-Bench tasks. We conduct extensive qualitative and quantitative analyses to explore the performance of APE. We show that APE-engineered prompts are able to improve few-shot learning performance (by simply prepending them to standard in-context learning prompts), find better zero-shot chain-of-thought prompts, as well as steer models toward truthfulness and/or informativeness. [2]

1 INTRODUCTION

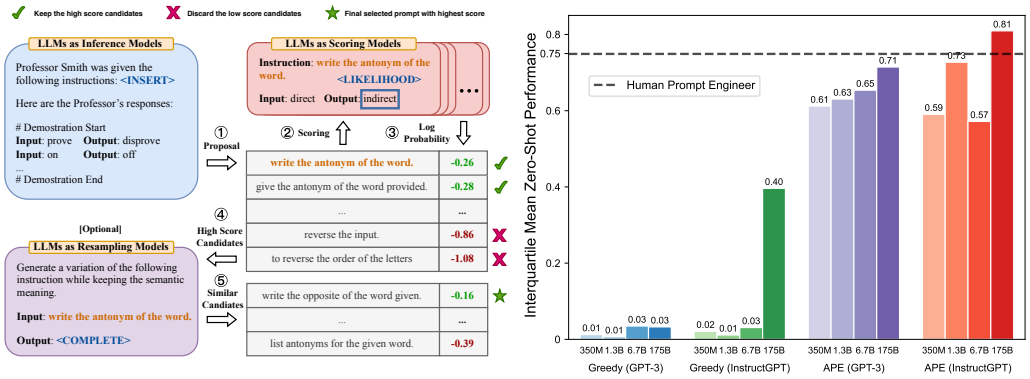
The combination of scale and attention-based architectures has resulted in language models possessing an unprecedented level of generality (Kaplan et al., 2020; Vaswani et al., 2017). These so-called "large language models" (LLMs) have shown remarkable, often superhuman, capabilities across a diverse range of tasks, including both zero-shot and few-shot setups (Brown et al., 2020; Srivastava et al., 2022). With generality, however, there comes a question of control: how can we make LLMs do what we want them to do?

To answer this question and steer LLMs toward desired behaviors, recent work has considered fine-tuning (Ouyang et al., 2022; Ziegler et al., 2019), in-context learning (Brown et al., 2020), and several forms of prompt generation (Gao, 2021), including both differentiable tuning of soft prompts (Qin & Eisner, 2021; Lester et al., 2021) and natural language prompt engineering (Reynolds & McDonell, 2021). The latter is of particular interest, as it provides a natural interface for humans to communicate with machines and may be of great relevance not only to LLMs but to other generalist models such as prompted image synthesizers (Rombach et al., 2022; Ramesh et al., 2022), for which public interest in prompt design and generation has also emerged (see Appendix A for examples).

Behind this interest is the fact that plain language prompts do not always produce the desired results, even when those results are possible to produce with alternative instructions. Thus, human users must experiment with a wide range of prompts to elicit desired behaviors, as they have little knowledge of how compatible instructions are with a particular model. We can understand this by viewing LLMs as black-box computers that execute programs specified by natural language instructions: while they

[1]We define "prompt engineering" as optimizing the language in a prompt in order to elicit the best possible performance. Notably, this does not include prompts that chain multiple LLM queries together or give the LLM access to external tools.

[2] Our code is available at https://github.com/keirp/automatic_prompt_engineer.



(a) Automatic Prompt Engineer (APE) workflow (b) Interquartile mean across 24 tasks

Figure 1: (a) Our method, **Automatic Prompt Engineer (APE)**, automatically generates instructions for a task that is specified via output demonstrations: it generates several instruction candidates, either via direct inference or a recursive process based on semantic similarity, executes them using the target model, and selects the most appropriate instruction based on computed evaluation scores. (b) As measured by the interquartile mean across the 24 NLP tasks introduced by Honovich et al. (2022), APE is able to surpass human performance when using the InstructGPT model (Ouyang et al., 2022).

can execute a broad range of natural language programs, the way these programs are processed may not be intuitive for humans, and the quality of instruction can only be measured when executing these instructions on a downstream task (Sanh et al., 2022; Wei et al., 2021).

To reduce the human effort involved in creating and validating effective instructions, we propose a novel algorithm using LLMs to generate and select instructions automatically. We call this problem *natural language program synthesis* and propose to address it as a black-box optimization problem using LLMs to generate and search over heuristically viable candidate solutions. In doing so, we leverage the generalist capabilities of LLMs in three ways. First, we use an LLM as an inference model (Ellis et al., 2021; Honovich et al., 2022) to generate instruction candidates based on a small set of demonstrations in the form of input-output pairs. Next, we guide the search process by computing a score for each instruction under the LLM we seek to control. Finally, we propose an iterative Monte Carlo search method where LLMs improve the best candidates by proposing semantically similar instruction variants. Intuitively, our algorithm asks LLMs to generate a set of instruction candidates based on demonstrations and then asks them to assess which instructions are more promising. We call our algorithm Automatic Prompt Engineer (APE). **Our main contributions are:**

- We frame instruction generation as natural language program synthesis, formulate it as a black-box optimization problem guided by LLMs, and propose both a naive and an iterative Monte Carlo search methods to approximate the solution.
- Our proposed method, APE, achieves human-level performance on zero-shot learning with model-generated instructions on 24/24 Instruction Induction and 17/21 Big-Bench tasks.
- We provide extensive qualitative and quantitative analyses exploring various facets of APE, and demonstrate applications of APE for improving few-shot learning, finding better zero-shot chain of thought prompts, and steering LLMs toward desired behaviors such as truthfulness and/or informativeness.

2 Related Work

Large Language Models Scaling up transformer-based language models in terms of model size, training data, and training compute has been shown to predictably improve performance on a wide range of downstream NLP tasks (Vaswani et al., 2017; Devlin et al., 2018; Brown et al., 2020). Many emergent abilities (Wei et al., 2022a) of LLMs have been discovered as a result of this scaling, including few-shot in-context learning, zero-shot problem solving, chain of thought reasoning, instruction following, and instruction induction (Cobbe et al., 2021; Wei et al., 2022b; Kojima et al.,

2022; Sanh et al., 2022; Wei et al., 2021; Ouyang et al., 2022; Honovich et al., 2022). In this paper, we view LLMs as black-box computers that execute programs specified by natural language instructions and investigate how to control an LLM's behavior using model-generated instructions.

Prompt Engineering Prompting offers a natural and intuitive interface for humans to interact with and use generalist models such as LLMs. Due to its flexibility, prompting has been widely used as a generic method for NLP tasks (Schick & Schütze, 2021; Brown et al., 2020; Sanh et al., 2022). However, LLMs require careful prompt engineering, either manually (Reynolds & McDonell, 2021) or automatically (Gao et al., 2021; Shin et al., 2020), as models do not seem to understand the prompts in the same way a human would (Webson & Pavlick, 2021; Lu et al., 2021). Though many successful prompt tuning methods perform optimization over a continuous space using gradient-based methods (Liu et al., 2021; Qin & Eisner, 2021; Lester et al., 2021), this becomes less practical with scale, as computing gradients becomes increasingly expensive and access to models shifts to APIs that may not provide gradient access. In our paper, we borrow components from discrete prompt search methods, such as prompt generation (Gao et al., 2021; Ben-David et al., 2021), prompt scoring (Davison et al., 2019) and prompt paraphrasing (Jiang et al., 2020; Yuan et al., 2021) to optimize instructions by searching directly in the natural language hypothesis space. As compared to this past work, which uses specialized models for each component and leans heavily on human templates, we show that the entire search can be conducted by a single LLM.

Program Synthesis Program synthesis involves the automatic search over a "program space" to find a program satisfying a particular specification (Gulwani et al., 2017). Modern program synthesis admits a wide variety of specifications, including input-output examples (Ellis et al., 2021; Wong et al., 2021) and natural language (Jain et al., 2022). The range of feasible program spaces to search over has also grown, from historically restrictive domain-specific languages to general-purpose programming languages (Austin et al., 2021). In contrast to prior approaches that require a suitable structured hypothesis space and library of components (Liang et al., 2010; Ellis et al., 2018), we leverage the structure provided by LLMs to search over the space of natural language programs. Using inference models is a standard practice to speed up the search by restricting the search space to a limited space of possible expressions (Menon et al., 2013; Lee et al., 2018; Devlin et al., 2017; Ellis et al., 2021). Inspired by this, we use LLMs as approximate inference models to generate program candidates based on a small set of demonstrations. Unlike classical program synthesis, our inference models do not require any training and generalize well to various tasks.

3 Natural Language Program Synthesis using LLMs

We consider a task specified by a dataset $\mathcal{D}_{\text{train}} = \{(Q, A)\}$ of input/output demonstrations sampled from population $\mathcal{X}$, and a prompted model $\mathcal{M}$. The goal of natural language program synthesis is to find a single instruction ρ such that, when $\mathcal{M}$ is prompted with the concatenation $[\rho; Q]$ of instruction and a given input, $\mathcal{M}$ produces the corresponding output A. More formally, we frame this as an optimization problem, where we seek instruction ρ that maximizes the expectation of some per-sample score $f(\rho, Q, A)$ over possible (Q, A):

$$\rho^\star = \arg\max_{\rho} f(\rho) = \arg\max_{\rho} \mathbb{E}_{(Q,A)} \left[f(\rho, Q, A) \right] \tag{1}$$

Note that in general, Q may be the empty string, such that we are optimizing ρ as a prompt that directly produces outputs $\{A\}$. While this task has been widely attempted by humans, we have little knowledge of how compatible any particular instruction is with model $\mathcal{M}$. Thus, we propose to treat this human-intractable question as a black-box optimization process guided by LLMs. Our algorithm, APE, uses LLMs in each of two key components, proposal and scoring. As shown in Figure 1 and summarized in Algorithm 1, APE first proposes a few candidate prompts, and then filters/refines the candidate set according to a chosen score function, ultimately choosing the instruction with the highest score. We discuss options for proposal and scoring next.

3.1 Initial Proposal Distributions

Due to the infinitely large search space, finding the right instruction can be extremely difficult, which has rendered natural language program synthesis historically intractable. Recent progress in NLP has shown language models are very good at generating diverse natural language text. Therefore, we

Algorithm 1 Automatic Prompt Engineer (APE)

Require: $\mathcal{D}_{\text{train}} \leftarrow \{(Q, A)\}_n$: training examples, $f : \rho \times \mathcal{D} \mapsto \mathbb{R}$: score function
1: Use LLM to sample instruction proposals $\mathcal{U} \leftarrow \{\rho_1, ..., \rho_m\}$. (See Section 3.1)
2: **while** not converged **do**
3: Choose a random training subset $\widetilde{\mathcal{D}}_{\text{train}} \subset \mathcal{D}_{\text{train}}$.
4: **for all** ρ in $\mathcal{U}$ **do**
5: Evaluate score on the subset $\widetilde{s} \leftarrow f(\rho, \widetilde{\mathcal{D}}_{\text{train}})$ (See Section 3.2)
6: **end for**
7: Filter the top k% of instructions with high scores $\mathcal{U}_k \subset \mathcal{U}$ using $\{\widetilde{s}_1, ..., \widetilde{s}_m\}$
8: Update instructions $\mathcal{U} \leftarrow \mathcal{U}_k$ or use LLM to resample $\mathcal{U} \leftarrow \text{resample}(\mathcal{U}_k)$ (See Section 3.3)
9: **end while**
Return instruction with the highest score $\rho^\star \leftarrow \arg\max_{\rho \in \mathcal{U}_k} f(\rho, \mathcal{D}_{\text{train}})$

consider leveraging a pretrained LLM to propose a good set $\mathcal{U}$ of candidate solutions that will guide our search procedure. While random samples from LLMs are unlikely to produce the desired (Q, A) pairs, we can instead ask the LLM to approximately infer the most likely instructions with a high score, given the input/output demonstrations; i.e., to approximately sample from $P(\rho \,|\, \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$.

Forward Mode Generation We consider two approaches to generate high-quality candidates from $P(\rho \,|\, \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$. First, we adopt an approach based on "forward" mode generation by translating this distribution $P(\rho \,|\, \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$ into words. For example, in our instruction induction experiments (Subsection 4.1), we follow Honovich et al. (2022) and prompt the LLM using Figure 2 (Top).

Reverse Mode Generation Although the "forward" model works out of the box for most of the pretrained LLMs, translating $P(\rho \,|\, \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$ into words requires custom engineering across different tasks. This is because while instructions are typically found in the beginning of passages, the "forward" model only generates text from left to right, which requires the instruction to be predicted at the end of the prompt. Therefore, we desire a more flexible approach such that the instruction can be anywhere in the text. To address this, we consider "reverse" mode generation, which uses an LLM with infilling capabilities—e.g., T5 (Raffel et al., 2020), GLM (Du et al., 2022), and InsertGPT (Bavarian et al., 2022)—to infer the missing instructions. Our "reverse" model directly samples from $P(\rho \,|\, \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$ by filling in the blank. We show an example of the such template in Figure 2 (Middle).

Customized Prompts Note that depending on the score function being used, there may exist more appropriate prompts than the samples above. For example, in our TruthfulQA experiments, we start with the human-designed instructions from the original dataset (Lin et al., 2022) and ask the the "reverse" model to propose initial instruction samples that fit the missing context (Figure 2 (Bottom)).

Forward Generation Template

I gave a friend an instruction and five inputs. The friend read the instruction and wrote an output for every one of the inputs. Here are the input-output pairs:

Input: $[Q_1]$ **Output**: $[A_1]$
Input: $[Q_2]$ **Output**: $[A_2]$
...

The instruction was **<COMPLETE>**

Reverse Generation Template

I instructed my friend to **<INSERT>**.

The friend read the instruction and wrote an output for every one of the inputs. Here are the input-output pairs:

Input: $[Q_1]$ **Output**: $[A_1]$
Input: $[Q_2]$ **Output**: $[A_2]$
...

Template for TruthfulQA

Professor Smith was given the following instructions: **<INSERT>**

Here are the Professor's responses:

Input: $[Q_1]$ **Output**: $[A_1]$
Input: $[Q_2]$ **Output**: $[A_2]$
...

Figure 2: Prompts for LLMs

3.2 Score Functions

To cast our problem as black-box optimization, we choose a score function that accurately measures the alignment between the dataset and the data the model generates. In our instruction induction experiments, we consider two potential score functions, described below. In the TruthfulQA experiments, we focused primarily on automated metrics proposed in Lin et al. (2022), similar to the execution accuracy. In each case, we evaluate the quality of a generated instruction using Equation (1), and take the expectation over a held-out test dataset $\mathcal{D}_{\text{test}}$.

Execution accuracy First, we consider evaluating the quality of an instruction ρ using the execution accuracy metric proposed by Honovich et al. (2022), which we denote as f_{exec}. In most cases,

execution accuracy is simply defined as the 0-1 loss, $f(\rho, Q, A) = \mathbb{1}\left[\mathcal{M}([\rho; Q]) = A\right]$. On some tasks, execution accuracy takes into account invariants; e.g., it may be an order invariant set matching loss, as described in Appendix A of Honovich et al. (2022).

Log probability We further consider a softer probabilistic score function, which we hypothesize might improve optimization by providing a more fine-grained signal when searching over low-quality instruction candidates. In particular, we consider the log probability of the desired answer given the instruction and question under the target model $\mathcal{M}$, which on a per sample basis, is $\log P(A \mid [\rho; Q])$.

Efficient score estimation Estimating the score by computing the score over the entire training dataset for all instruction candidates can be expensive. To reduce the computation cost, we adopt a filtering scheme where a promising candidate receives more computation resources while a low-quality candidate receives less computation. It can be achieved by using a multi-stage computation strategy on lines 2-9 Algorithm 1. We first evaluate all candidates with a small subset of the training dataset. For the candidates with a score greater than a certain threshold, we sample and evaluate a new non-overlapping subset from the training dataset to update the moving average of the score. Then, we repeat this process until a small set of candidates is left, which are evaluated on the entire training dataset. This adaptive filtering scheme significantly improves the computation efficiency by keeping the exact computation costs for the high-quality samples and drastically reducing the computation costs for low-quality candidates. We note that a similar score estimation scheme has been used in previous works (Li et al., 2022; Maclaurin & Adams, 2015).

3.3 Iterative Proposal Distributions

Despite our attempt to directly sample high-quality initial instruction candidates, it could be the case that the method described in Subsection 3.1 fails to produce a good proposal set $\mathcal{U}$, either because it lacks of diversity or does not contain any candidates with a suitably high score. In case of such challenges, we explore an iterative process for resampling $\mathcal{U}$.

Iterative Monte Carlo Search Instead of only sampling from the initial proposal, we consider exploring the search space locally around the current best candidates. This allows us to generate new instructions that are more likely to be successful. We call this variant *iterative APE*. At each stage, we evaluate a set of instructions and filter out candidates with low scores. Then, an LLM is asked to generate new instructions similar to those with high scores. We provide the prompt used for resampling in Figure 3. Figure 6 (Right) shows that although this approach improves the overall quality of the proposal set $\mathcal{U}$, the highest scoring instruction tends to remain the same with more stages. We conclude iterative generation provides marginal improvement over the relative simplicity and effectiveness of the generative process described in Subsection 3.1. Therefore, we use APE without iterative search as default unless otherwise stated.

Prompt for Resampling

Generate a variation of the following instruction while keeping the semantic meaning.

Input: [INSTRUCTION]

Output: **<COMPLETE>**

Figure 3: Resampling

4 Large Language Models are Human-Level Prompt Engineers

This section examines how APE can guide LLMs to desired behaviors. We investigate from four perspectives: zero-shot performance, few-shot in-context learning performance, zero-shot chain-of-thought reasoning, and truthfulness. Our experiments show that APE can find prompts that improve task performance, performing equal to or even better than those authored by humans. APE also often produces insightful tricks for how to best prompt language models that can be successfully transferred to new tasks (see Section 4.3).

4.1 Instruction Induction

We assess the effectiveness of zero-shot and few-shot in-context learning on 24 instruction induction tasks proposed in Honovich et al. (2022). The tasks span many facets of language understanding, from simple phrase structure to similarity and causality identification. We provide a detailed descriptions of each task in Appendix B. For each task, we sample five input-output pairs from the training data and select the best instruction using algorithm 1. Then, we evaluate the quality of the instruction

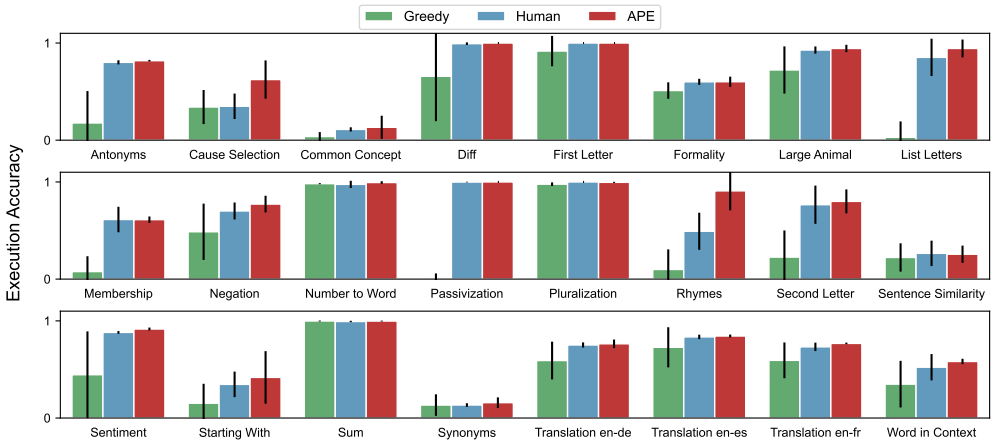


Figure 4: Zero-shot test accuracy on 24 Instruction Induction tasks. APE achieves human-level or better performance on all 24 out of 24 tasks.

by executing the instruction on InstructGPT [3]. We repeat our experiments five times with different random seeds to report the mean and standard deviation. The exact templates for our experiments can be found in Appendix (Table 5).

Zero-shot Learning We compare our method against two baselines: human prompt engineers (Human)[4] and the model-generated instruction algorithm proposed by Honovich et al. (2022). This algorithm can be thought of as a greedy version of APE, without a search and selection process; thus, we refer to it as "Greedy". Figure 4 shows the zero-shot performance of InstructGPT using human instructions and model generated instructions. Our algorithm outperforms "Greedy" on every task and achieves equal or better than human performance on 24 of 24 tasks. Moreover, the Interquartile Mean (IQM) (Agarwal et al., 2021) across all 24 tasks in Figure 1 suggests that APE with InstructGPT outperforms human-engineered prompts, obtaining an IQM of 0.810 vs humans' 0.749. We summarize the instruction selected by APE for each task in Appendix (Table 12).

Few-shot In-context Learning We evaluated APE-generated instructions in few-shot in-context learning, where we insert the instruction before the in-context demonstrations. Those instructions are selected based on zero-shot execution accuracy, and we denote this setting as "Instruction + In-context" in Figure 8. As shown in Figure 8, adding an instruction achieves a comparable or better test performance than the standard in-context learning performance on 21 of 24 tasks. Counter-intuitively, adding in-context examples for Rhymes, Large Animal, and Second Letters hurts model performance. We conjecture that it may be because the selected instructions overfit the zero-shot learning scenario and thus do not perform well on the few-shot case. Therefore, we experiment using few-shot execution accuracy as the selection metric. Figure 14 shows that the few-shot metric achieves comparable or slightly better than the zero-shot metric except for Rhymes. To have an intuitive understanding of what is happening, we provide a qualitative analysis in Appendix C.1.

4.2 BIGBENCH

To see whether APE can be applied to more challenging tasks, we propose and curate BIG-Bench Instruction Induction (BBII), a clean and tractable subset of 21 tasks that have a clear, human-written instruction that can be applied to all examples in the dataset. The selected tasks cover many facets of language understanding and includes all nine such problems from the BigBench-Hard Subset (Suzgun et al., 2022). In particular, it includes emotional understanding, context-free question answering, reading comprehension, summarization, algorithms, and various reasoning tasks (e.g., arithmetic, commonsense, symbolic, and other logical reasoning tasks). We provide a detailed description of the task and our selection criteria in Appendix B.

[3]We use the *text-davinci-002* via the OpenAI API (`https://beta.openai.com/`). Though not stated explicitly in the API, we assume the models are those reported by Ouyang et al. (2022).

[4]We use the gold annotations from Honovich et al. (2022), which were manually verified for correctness.

For each task, we used the reverse mode generation of InstructGPT to generate a set of instruction candidates and ranked the instructions based on their execution accuracy. Then, we executed the selected instruction on InstructGPT to compute the zero-shot performance on the test set and compared it with the default human prompt. As shown in Table 6, APE achieves comparable or better performance than the default human prompt on 17 out of 21 tasks.

4.3 Zero-shot Chain of Thought

Chain-of-thought reasoning has been shown to dramatically improve the ability of LLMs to complete complex reasoning tasks, such as solving math problems that require multiple steps. Early works (Nye et al., 2021; Betz et al., 2021; Wei et al., 2022b) on chain-of-thought used fine-tuning or in-context learning to get LLMs to show their work for such problems. One of the most influential recent works of prompt engineering was the discovery (Kojima et al., 2022) that LLMs could be made to give chain-of-thoughts simply by prepending "Let's think step by step." to the beginning of the LLM's response. Known as Zero-Shot-CoT, this prompting strategy improves the zero-shot performance of InstructGPT on MultiArith (Roy & Roth, 2016) from 17.7 to 78.7 and improves performance on GSM8K(Cobbe et al., 2021) from 10.4 to 40.7. As shown in Table 7, Kojima et al. (2022) found their prompt was the best performing out of at least nine human-designed prompts.

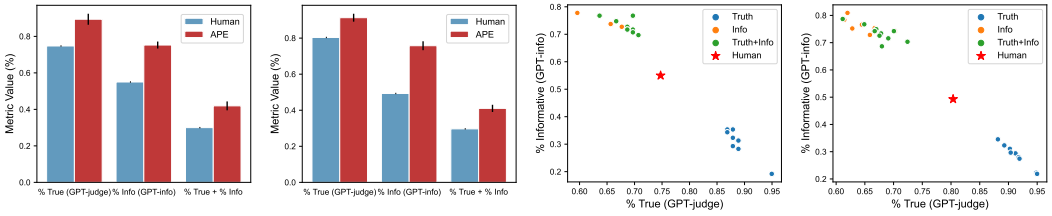
We used APE to automatically search for the best answer-prefix across the suite of tasks used in Kojima et al. (2022). Our approach to optimizing this prompt was inspired by Zelikman et al. (2022). First, we generate a dataset of questions and reasoning steps generated using InstructGPT with "Let's think step by step." Then, we remove any data points that had incorrect answers. Finally, we use APE to find a prompt starting with "Let's" that maximizes the likelihood of these correct reasoning steps. See Table 5 for the template used for prompt generation and evaluation. APE produces the prompt "Let's work this out in a step by step way to be sure we have the right answer." This generated prompt further improves performance from 78.7 to 82.0 on MultiArith and from 40.7 to 43.0 on GSM8K. We believe this general workflow represents a common use-case for APE where prompt engineers use APE to optimize parts of their exiting templates to improve performance. See Figure 10 for details on the performance of this prompt on other reasoning tasks.

4.4 TruthfulQA

We apply our method on TruthfulQA (Lin et al., 2022) to see how APE-generated instructions can steer an LLM to generate answers with different styles, and study the trade-off between truthfulness and informativeness. Borrowing the metrics from the original paper, we use APE to the learn instructions that maximize three metrics: truthfulness (% True), informativeness (% Info), and a combination of both (%True + %Info). Lin et al. (2022) used human evaluation to assess the model performance, but they found their automated metrics align with human prediction over 90% of the time. In our experiments, we rely on their fine-tuned GPT-judge and GPT-info to evaluate the scores.

Prompt Engineering in TruthfulQA We want to stress that the TruthfulQA dataset is intended to test pretrained models in zero-shot settings. Our results are not in any way compatible with the original benchmarks. Because we have optimized the instructions using a small portion of the question and answer pairs as training demonstrations, our results are not "true few-shot learning" (Perez et al., 2021). We randomly sampled 100 out of 817 questions for the actual experiments to form training demonstrations $\mathcal{D}_{\text{train}}$. To sample the proposal set $\mathcal{U}$, we ask a "reverse" model to generate instructions based on six randomly chosen demonstration pairs, similar to our previous experiments. Unlike in Instruction Induction, in TruthfulQA, we aim to find a single best instruction prompt that works well across all 38 categories of questions spanning health, law, politics, and fiction. It is worth noting all our generated instructions are very generic, e.g., "You will be asked a series of questions. For each question, you must either answer the question or decline to answer, in which case you must state that you have no comment", and do not contain any examples from the dataset.

Truthfulness vs Informativeness Trade-off We found that APE outperforms the human-engineered prompt with only 200 candidates proposed by InstructGPT (175B), as seen in Figure 5. We compared our generated prompt with the "help" prompt from Lin et al. (2022). The training and test performance are shown in Figure 5(a)-(b). We found that choosing the top 10 of 200 candidates on the training set generalizes well to the test set. We report the average performance across the top 10 instructions for the three metrics. This result by itself is not surprising as the human baseline is not carefully



(a) Average performance Train (b) Average performance Test (c) %True-%Info trade-off Training (d) %True-%Info trade-off Test

Figure 5: Comparison of APE and "help" (human) prompt on the TruthfulQA task. (a) Percentage of answers that were either true (% True), informative (% Info), or both (% True + % Info) on the 100 training examples. (b) Same data on the 717 test examples. (c) %True-%Info frontier computed on training data with top 10 instructions from each metric. (d) %True-%Info frontier on the test data.

chosen, as pointed out by Askell et al. (2021). However, we found that the instructions discovered by APE can achieve very high truthfulness with answers such as "No comment," but these answers provide little information. We used our top candidates to further investigate the trade-off between truthfulness and informativeness. We visualize the top 10 proposed samples across the three metrics on the truthfulness-informative plots shown in Figure 5(c) and Figure 5(d). While APE achieves over 40% accuracy in providing both true and informative answers (v.s. 30% by the "help" prompt from humans), the instructions discovered tend to target the two ends of this %true-%info Pareto frontier.

5 Quantitative Analysis

In this section, we conduct quantitative analyses to better understand the three main components of our method: proposal distribution, score functions, and iterative search. Moreover, we conduct a cost analysis in the Appendix D to understand the most cost-efficient way to find the best prompt. We observe the larger and more powerful language models are more cost-effective for generating the best prompt despite a higher per-token cost.

5.1 LLMs for Proposal Distribution

How does the proposal quality change as we increase the model size? To understand how the model size affects the quality of the initial proposal distribution, we examine eight different models[5] available via the OpenAI API. To assess the quality of the proposal distribution, we generate 250 instructions per model and compute the execution accuracy on 50 test data points. We visualize the survival function (percentage of instructions with test accuracy greater than a certain threshold) and the histogram of test accuracy for a simple task (i.e., Pluralization) in Figure 6 (a) and include a similar plot for a more challenging task (Start With) in the Appendix (Figure 28). As shown in both figures (and unsurprisingly), larger models tend to produce better proposal distributions than smaller ones, as do the models that were fine-tuned to follow human instructions. On the simple task, all instructions generated by the best model, InstructGPT (175B), have reasonable test accuracy. In contrast, half of the instructions are off-topic and perform poorly on the more challenging task.

5.2 LLMs for Selection

Does proposal quality matter under selection? If we sample more instructions from the LLMs, then it becomes more likely for us to find better instructions. To verify this hypothesis, we increase the sample size from 4 to 128 and evaluate the test accuracy change. Figure 7 (Left) shows a monotonically increasing trend with a diminishing return, as human-level performance is achieved with 64 instruction samples. Thus, we choose 50 as our default sample size. Under this configuration, we investigate how the proposal distribution affects the test accuracy of the best instruction selected by our algorithm. Figure 1(b) shows that though the small models may be less likely to generate good instructions, they nonetheless generate some good ones if we sample enough candidates. Therefore, we still find promising instructions with a small model by running our selection algorithm, explaining why our method outperforms the greedy approach Honovich et al. (2022) across all eight models.

[5]We use ada, babbage, curie, davinci, text-ada-001, text-babbage-001, text-curie-001, text-davanci-002

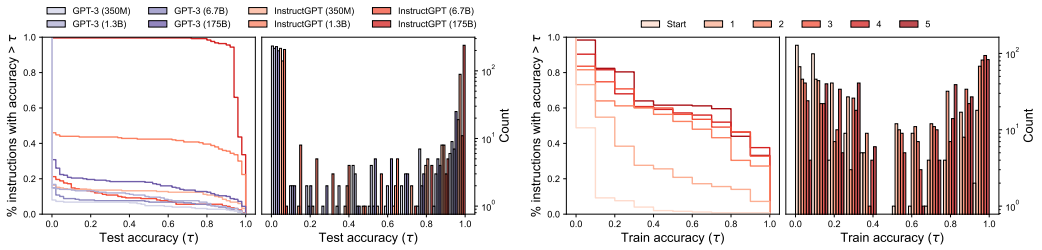


Figure 6: (Left) Quality of the proposal distribution of models with different size as assessed by test execution accuracy. (Right) Iterative Monte Carlo search improves the quality of the instruction candidates at each round.

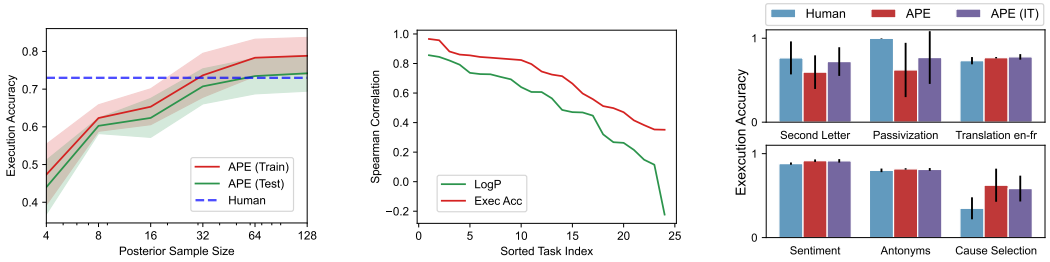


Figure 7: (Left) Test execution of the best instruction as we increase the number of instruction candidates. We report the mean and standard deviation across 6 different tasks. (Middle) Spearman Correlation between the test accuracy and two metrics on 24 tasks. (Right) Test execution accuracy of the best instruction selected using APE and iterative APE (APE (IT)).

Which scoring function is better? We compute the correlation between the test accuracy and two metrics on 24 instruction induction tasks to study how good our proposed metrics are. We generate 250 instructions per task using InstructGPT (175B) in "forward" mode and compute the metric score and test accuracy on 10 test data points. We visualize the Spearman correlation between the test accuracy and two metrics. Figure 7 (Middle) shows that the execution accuracy aligns better with the test performance across the tasks. Thus, we choose it as our default metric unless otherwise stated.

5.3 Iterative Monte Carlo Search

Does Iterative Search improve the instruction quality? We visualize the survival function and histogram of test accuracy on the "Passivization" task in Figure 6 (Right) and include five more tasks in the Appendix. The survival plot shows that the curves increase as the round goes up, which suggests that iterative search does result in a higher-quality proposal set. However, we observe diminishing returns to further selection rounds as the quality seems to stabilize after three rounds.

Do we need Iterative Search? We compare APE and iterative APE on six tasks[6]. As shown in Figure 7, the iterative search marginally improves performance on tasks where APE underperforms humans but achieves similar performance on the other tasks. This is consistent with our hypothesis that iterative search would be most useful on tasks where generating a good initial $\mathcal{U}$ is challenging.

6 Conclusion

Large language models can be seen as general-purpose computers that execute programs specified by natural language prompts. We automate the prompt engineering process by formulating it as a black-box optimization problem, which we propose to solve using efficient search algorithms guided by LLMs. Our method achieves human-level performance on various tasks with minimum human inputs. As recent LLMs demonstrate an impressive ability to follow human instruction, we expect many future models, including those for formal program synthesis, to have a natural language interface. This work builds the foundation to control and steer generative artificial intelligence.

ACKNOWLEDGMENTS

We would like to thank Or Honovich and Michael Zhang for their help and valuable feedback. JB was supported by NSERC Grant [2020-06904], CIFAR AI Chairs program, Google Research Scholar Program and Amazon Research Award. KP was supported by NSERC PGS-D. SP was supported by NSERC CGS-D. HC was supported by NSERC CGS-D and RBC Graduate Fellowship. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

Eyal Ben-David, Nadav Oved, and Roi Reichart. Pada: A prompt-based autoregressive approach for adaptation to unseen domains. *arXiv preprint arXiv:2102.12206*, 2021.

Gregor Betz, Kyle Richardson, and Christian Voigt. Thinking aloud: Dynamic context generation improves zero-shot reasoning performance of gpt-2. *arXiv preprint arXiv:2103.13033*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Joe Davison, Joshua Feldman, and Alexander M Rush. Commonsense knowledge mining from pretrained models. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 1173–1178, 2019.

Jacob Devlin, Rudy R Bunel, Rishabh Singh, Matthew Hausknecht, and Pushmeet Kohli. Neural program meta-induction. *Advances in Neural Information Processing Systems*, 30, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. GLM: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 320–335, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.26. URL https://aclanthology.org/2022.acl-long.26.

Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama, and Josh Tenenbaum. Learning libraries of subroutines for neurally–guided bayesian program induction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/7aa685b3b1dc1d6780bf36f7340078c9-Paper.pdf.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pp. 835–850, 2021.

Tianyu Gao. Prompting: Better ways of using language models for nlp tasks. *The Gradient*, 2021.

Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3816–3830, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.295. URL https://aclanthology.org/2021.acl-long.295.

Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, et al. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119, 2017.

Or Honovich, Uri Shaham, Samuel R Bowman, and Omer Levy. Instruction induction: From few examples to natural language task descriptions. *arXiv preprint arXiv:2205.10782*, 2022.

Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram Rajamani, and Rahul Sharma. Jigsaw: Large language models meet program synthesis. In *Proceedings of the 44th International Conference on Software Engineering*, pp. 1219–1231, 2022.

Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. Accelerating search-based program synthesis using learned probabilistic models. *ACM SIGPLAN Notices*, 53(4):436–449, 2018.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, 2021.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*, 2022.

Percy Liang, Michael I. Jordan, and Dan Klein. Learning programs: A hierarchical bayesian approach. In Johannes Fürnkranz and Thorsten Joachims (eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pp. 639–646. Omnipress, 2010. URL https://icml.cc/Conferences/2010/papers/568.pdf.

Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3214–3252, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.229. URL `https://aclanthology.org/2022.acl-long.229`.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *arXiv preprint arXiv:2103.10385*, 2021.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.

Dougal Maclaurin and Ryan Prescott Adams. Firefly monte carlo: Exact mcmc with subsets of data. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Aditya Menon, Omer Tamuz, Sumit Gulwani, Butler Lampson, and Adam Kalai. A machine learning framework for programming by example. In *International Conference on Machine Learning*, pp. 187–195. PMLR, 2013.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models. *Advances in Neural Information Processing Systems*, 34:11054–11070, 2021.

Guanghui Qin and Jason Eisner. Learning how to ask: Querying lms with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5203–5212, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–7, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*, 2016.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. In *The Tenth International Conference on Learning Representations*, 2022.

Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 255–269, 2021.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Albert Webson and Ellie Pavlick. Do prompt-based models really understand the meaning of their prompts? *arXiv preprint arXiv:2109.01247*, 2021.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2021.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022a.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022b.

Catherine Wong, Kevin M Ellis, Joshua Tenenbaum, and Jacob Andreas. Leveraging language to learn program abstractions and search heuristics. In *International Conference on Machine Learning*, pp. 11193–11204. PMLR, 2021.

Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore: Evaluating generated text as text generation. *Advances in Neural Information Processing Systems*, 34:27263–27277, 2021.

Eric Zelikman, Yuhuai Wu, and Noah D Goodman. Star: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.