LARGE LANGUAGE MODELS ARE HUMAN-LEVEL PROMPT ENGINEERS

Anonymous authors
Paper under double-blind review

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* (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. Experiments on 24 NLP tasks 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 19/24 tasks. We conduct extensive qualitative and quantitative analyses to explore the performance of APE. We show that APE-engineered prompts can be applied to steer models toward truthfulness and/or informativeness, as well as to improve few-shot learning performance by simply prepending them to standard in-context learning prompts.

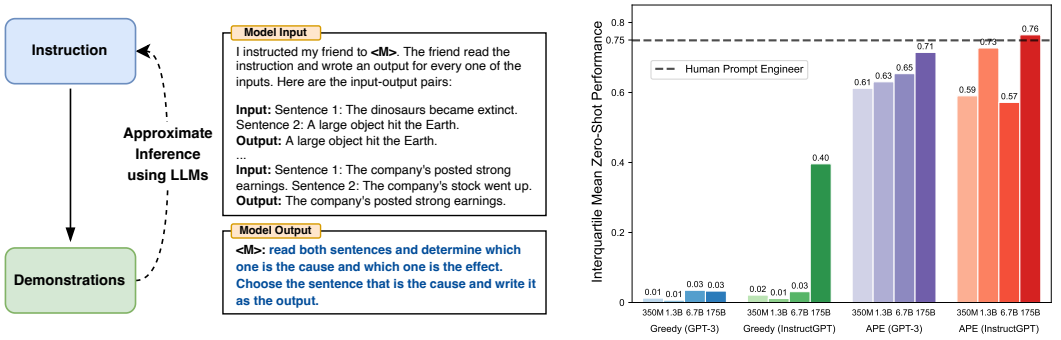
1 INTRODUCTION

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 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



(a) (b) LLMs as inference models (c) Interquartile mean across 24 tasks

Figure 1: (a) Natural language program synthesis finds an appropriate instruction (the program) that generates the observed demonstrations when executed by the model. We frame this as a black-box optimization problem guided by an inference procedure. (b) We use LLMs as inference models to fill in the blank; our algorithm involves a search over candidates proposed by the inference models. (c) 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).

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 19/24 NLP tasks.
- We provide extensive qualitative and quantitative analyses exploring various facets of APE, and demonstrate applications of APE for improving few-shot learning 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., 2021b; Qin & Eisner, 2021; Lester et al., 2021), this becomes less practical with scale, as computing computing gradients becomes increasingly expensive and access to models shifts to APIs

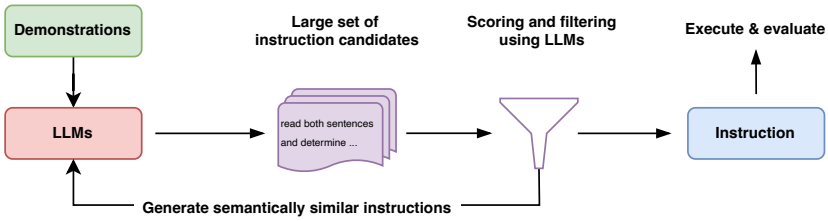


Figure 2: Our method, **Automatic Prompt Engineer (APE)**, automatically generates instructions for a task that is specified via output demonstrations: it generates several instruction candidates, either via direct inference or a recursive process based on semantic similarity, executes them using the target model, and selects the most appropriate instruction based on computed evaluation scores.

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 2 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

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

Use LLM to sample instruction proposals $\mathcal{U} \leftarrow \{\rho_1, ..., \rho_m\}$
while not converged **do**
  Choose a random training subset $\widetilde{\mathcal{D}}_{\text{train}} \subset \mathcal{D}_{\text{train}}$
  for all ρ in $\mathcal{U}$ **do**
    Evaluate score according to Sec 3.2 on the subset $\widetilde{s} \leftarrow f(\rho, \widetilde{\mathcal{D}}_{\text{train}})$
  end for
  Filter the top-k instructions $\mathcal{U}_k \subset \mathcal{U}$ using $\{\widetilde{s}_1, ..., \widetilde{s}_m\}$
  Update instructions with LLM $\mathcal{U} \leftarrow \text{resample}(\mathcal{U}_k)$
end while

Return instruction with the highest score $\rho^\star \leftarrow \arg\max_{\rho \in \mathcal{U}_k} f(\rho, \mathcal{D}_{\text{train}})$

has shown language models are very good at generating diverse natural language text. Therefore, we 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 $P(\rho \mid \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$.

Forward Mode Generation We consider two approaches to generate high-quality candidates from $P(\rho \mid \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$. First, we adopt an approach based on "forward" mode generation by translating this distribution $P(\rho \mid \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$ into words. For example, in our instruction induction experiments (Subsection 4.1), we follow Honovich et al. (2022) and prompt the LLM using Figure 3 (Top). In this case, the wording suggests that the outputs are generated based on the instruction, so that the score functions considered will be high.

Reverse Mode Generation Although the "forward" model works out of the box for most of the pretrained LLMs, translating $P(\rho \mid \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$ into words requires custom engineering across different tasks. This is because the "forward" model only generates text from left to right, while we would like the model to predict the missing context before the demonstrations. To address this, we also consider "reverse" mode generation, which uses an LLM with infilling capabilities—e.g., T5 (Raffel et al., 2020) and InsertGPT (Bavarian et al., 2022)—to infer the missing instructions. Our "reverse" model directly samples from $P(\rho \mid \mathcal{D}_{\text{train}},\ f(\rho) \text{ is high})$ by filling in the blank, making it a more versatile approach than the "forward" completion models. For example, in our instruction induction experiments we use the template in Figure 3 (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). As shown in Figure 3 (Bottom), the "reverse" model is asked to propose initial instruction samples that fit the missing context.

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 3: 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 4.

Prompt for Resampling

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

Input: [INSTRUCTION]

Output: **<COMPLETE>**

Figure 4: Resampling

Figure 8 (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 that 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 in our experiments unless otherwise stated.

4 Large Language Models are Human-Level Prompt Engineers

This section examines how APE can guide LLMs to desired behaviors. We investigate from three perspectives: zero-shot performance, few-shot in-context learning performance, and truthfulness.

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 refer the reader to Appendix A of Honovich et al. (2022) for detailed descriptions of each task. For each task, we sample five input-output pairs from the training data and select the best instruction using algorithm 1. Then, we

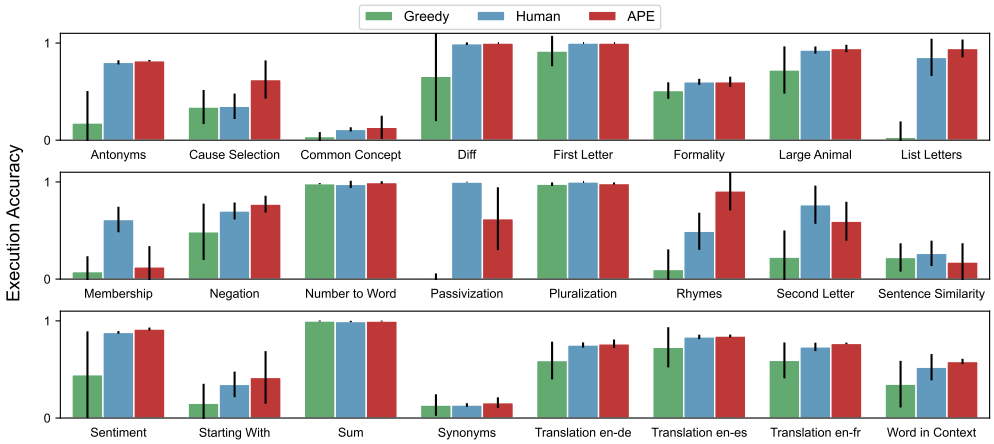


Figure 5: Zero-shot test accuracy on 24 Instruction Induction tasks. APE achieves human-level performance on 19 out of 24 tasks. See best performing instruction performance in Figure 10.

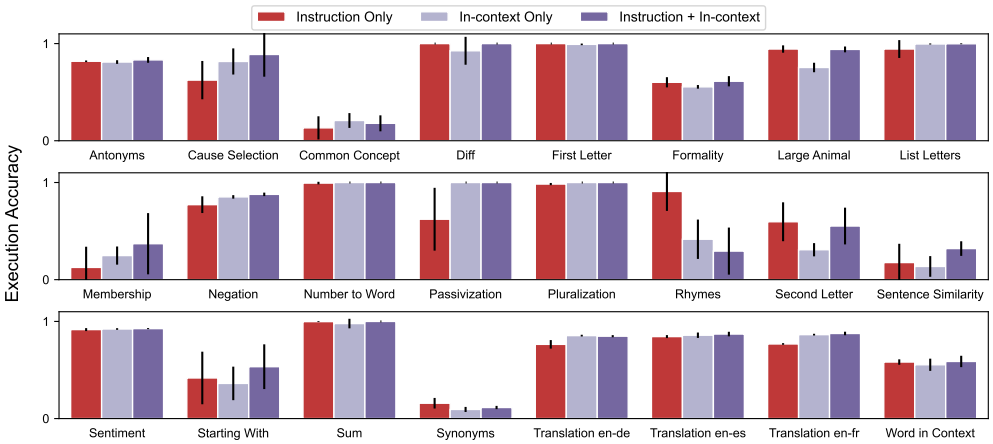


Figure 6: Few-shot in-context test accuracy on 24 Instruction Induction tasks. The APE improves the few-shot in-context learning performance on 21 out of 24 tasks. See best performing instruction performance in Figure 11.

evaluate the quality of the instruction by executing the instruction on InstructGPT (175B)[1]. We repeat our experiments five times with different random seeds to report the mean and standard deviation of the best performing result in each seed and report the best overall performance in Appendix (Figure 10). The exact templates for our experiments can be found in Appendix (Table 2).

Zero-shot Learning We compare our method against two baselines: human prompt engineers (Human)[2] 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 5 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 19 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 0.765 vs humans' 0.749. We summarize the instruction selected by APE for each task in Appendix (Table 8).

We observe APE can propose varying quality candidates depending on the subset of demonstration chosen for tasks such as Passivization and Start With. Similar to in-context learning findings (Liu et al., 2021a), a different combination of the in-context examples can yield significantly different results. As shown in Figure 5, our method achieves worse average performance than humans in Passivization and Sentence Similarity. However, selecting the best instruction still outperforms the

[1]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).

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

human in Figure 10. These results highlight the importance of the initial proposal stage. We found it is crucial to generate instruction candidates based on different demonstrations. Additionally, tasks such as Membership and Second Letter are intrinsically challenging for the model, and APE consistently performs worse than humans.

Zero-shot Qualitative Analysis To better understand the weaknesses of APE, we manually examined instructions in three tasks where APE underperforms humans in the zero-shot setting: Passivization, Membership, and Second Letter. As shown in Tables 3, 4, we find large variance in the quality of APE instructions due to the difference in demonstration. The best instructions selected by APE for each task are semantically correct and recover accuracy relative to humans. In contrast, the worst instructions are all semantically incorrect and perform poorly. Interestingly, we observe no semantic difference in the demonstrations used to generate the best and worst instruction under human inspection.

Few-shot In-context Learning We also evaluate APE-generated instructions in the few-shot in-context learning scenario, where we insert the instruction before the in-context demonstrations. We denote this as "Instruction + In-context" in Figure 6. As shown in Figure 6, 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, which we explore below.

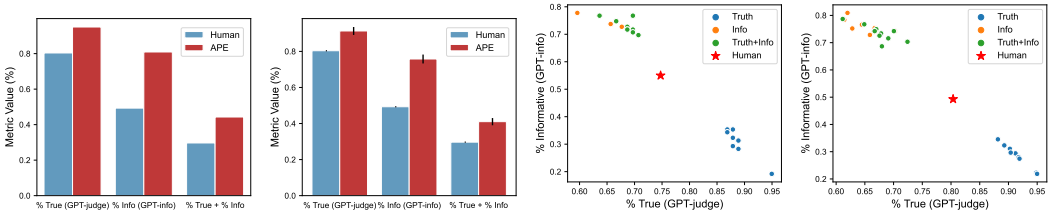
Few-shot Qualitative Analysis We found an adversarial case on Rhymes when combining the instruction and in-context prompts. Table 5 reveals that 4 of 5 filtered instructions ask to echo the input word, effectively hacking the evaluation with near-perfect test accuracy, as every word rhymes with itself trivially. However, adding in-context examples for these four instructions creates a misalignment between instruction (induces trivial rhymes) and context (induces non-trivial rhymes), resulting in a significant drop in performance. Another interesting case happens in the Second Letters task (Table 6), where the model's responses to two semantically "correct" instructions experience a drop in accuracy when paired with the in-context demonstration. In this case, though there is no semantic difference between the instruction and demonstration, adding in-context examples makes it difficult for the LLM to identify the correct program to execute. In contrast, these behaviors are not observed on the last task (i.e., Large Animal), where the in-context examples only have little impact on performance (Table 7)

4.2 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 7. We compared our generated prompt with the "help" prompt from Lin et al. (2022). The training and test performance are shown in Figure 7(a)-(b). We found that choosing the top 10 of 200 candidates on the training



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

Figure 7: 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.

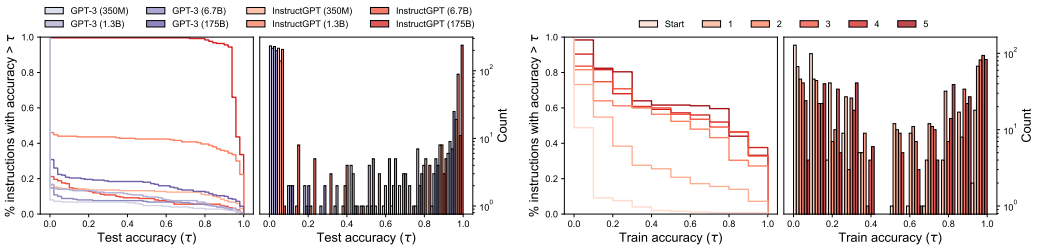


Figure 8: (Left) Quality of the proposal distribution of models with different size as assessed by test execution accuracy. (Right) Iterative Monte Carlo search improves the quality of the instruction candidates at each round.

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 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 7(c) and Figure 7(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 the three main components of our method, namely, the proposal distribution, score functions, and iterative search.

5.1 LLMs for Proposal Distribution and Scoring

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[3] 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 8 (a) and include a similar plot for a more challenging task (Start With) in the Appendix (Figure 13). 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 easy 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.

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

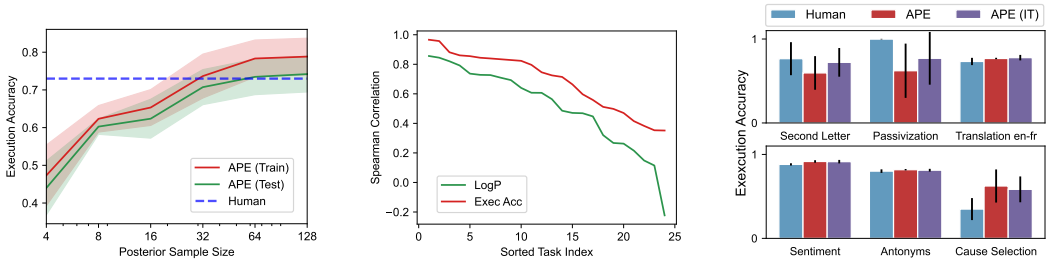


Figure 9: (Left) Test execution of the best instruction as we increase the number of instruction candidates. We report the mean and standard deviation across 6 different tasks. (Middle) Spearman Correlation between the test accuracy and two metrics on 24 tasks. (Right) Test execution accuracy of the best instruction selected using APE and iterative APE (APE (IT)).

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 test accuracy change. Figure 9 (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(c) shows that though the small models have a low chance of generating good instructions, they nonetheless generate some good ones if we sample enough candidates. Therefore, we can still find promising instructions with a small model by running our selection algorithm, explaining why our method performs significantly better than the greedy approach across eight models.

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 9 (Middle) shows execution accuracy aligns better to the test performance across the tasks. Thus, we choose it as our default metric unless otherwise stated.

5.2 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 8 (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[4]. As shown in Figure 9, 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

LLMs 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 AIs.

[4]The six datasets are chosen such that two of them are worse than humans, and the other four are human-level. They cover six categories (spelling, morphosyntax, lexical semantics, semantics, multi-lingual, and GLUE).

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.

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.

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.

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.

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`.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021a.

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

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.

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.

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.

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.

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.

A PROMPT ENGINEERING IN THE WILD

Large models with natural language interfaces, including models for text generation and image synthesis, have seen an increasing amount of public usage in recent years. As finding the right prompt can be difficult for humans, a number of guides on prompt engineering as well as tools to aid in prompt discovery have been developed. Among others, see, for example:

- https://blog.andrewcantino.com/blog/2021/04/21/prompt-engineering-tips-and-tricks/
- https://techcrunch.com/2022/07/29/a-startup-is-charging-1-99-for-strings-of-text-to-feed-to-dall-e-2/
- https://news.ycombinator.com/item?id=32943224
- https://promptomania.com/stable-diffusion-prompt-builder/
- https://huggingface.co/spaces/Gustavosta/MagicPrompt-Stable-Diffusion

In this paper we apply APE to generate effective instructions for steering LLMs, but the general framework Algorithm 1 could be applied to steer other models with natural language interfaces so long as an appropriate proposal method and scoring function can be designed.

B IMPLEMENTATION DETAILS

Table 1: For convenience, Table 1 from Honovich et al. (2022) is duplicated here. This describes the 24 NLP instruction induction tasks.

Category	Task	Instruction	Demonstration
Spelling	First Letter	Extract the first letter of the input word.	cat → c
	Second Letter	Extract the second letter of the input word.	cat → a
	List Letters	Break the input word into letters, separated by spaces.	cat → c a t
	Starting With	Extract the words starting with a given letter from the input sentence.	The man whose car I hit last week sued me. [m] → man, me
Morpho-syntax	Pluralization	Convert the input word to its plural form.	cat → cats
	Passivization	Write the input sentence in passive form.	The artist introduced the scientist. → The scientist was introduced by the artist.
Syntax	Negation	Negate the input sentence.	Time is finite → Time is not finite.
Lexical Semantics	Antonyms	Write a word that means the opposite of the input word.	won → lost
	Synonyms	Write a word with a similar meaning to the input word.	alleged → supposed
	Membership	Write all the animals that appear in the given list.	cat, helicopter, cook, whale, frog, lion → frog, cat, lion, whale
Phonetics	Rhymes	Write a word that rhymes with the input word.	sing → ring
Knowledge	Larger Animal	Write the larger of the two given animals.	koala, snail → koala
Semantics	Cause Selection	Find which of the two given cause and effect sentences is the cause.	Sentence 1: The soda went flat. Sentence 2: The bottle was left open. → The bottle was left open.
	Common Concept	Find a common characteristic for the given objects.	guitars, pendulums, neutrinos → involve oscillations.
Style	Formality	Rephrase the sentence in formal language.	Please call once you get there → Please call upon your arrival.
Numerical	Sum	Sum the two given numbers.	22 10 → 32
	Difference	Subtract the second number from the first.	32 22 → 10
	Number to Word	Write the number in English words.	26 → twenty-six
Multi-lingual	Translation	Translate the word into German / Spanish / French.	game → juego
GLUE	Sentiment Analysis	Determine whether a movie review is positive or negative.	The film is small in scope, yet perfectly formed. → positive
	Sentence Similarity	Rate the semantic similarity of two input sentences on a scale of 0 - definitely not to 5 - perfectly.	Sentence 1: A man is smoking. Sentence 2: A man is skating. → 0 - definitely not
	Word in Context	Determine whether an input word has the same meaning in the two input sentences.	Sentence 1: Approach a task. Sentence 2: To approach the city. Word: approach → not the same

Table 2: Raw templates used for model prompting in our experiments

Usage	Template
Zero-shot Evaluation	**Instruction**: [INSTRUCTION] **Input**: [Q_{test}]\n**Output**:**<COMPLETE>**
Few-shot Evaluation	**Instruction**: [INSTRUCTION] **Input**: [Q_1]\n**Output**: [A_1]\n\n**Input**: [Q_2]\n**Output**: [A_2] ... **Input**: [Q_{test}]\n**Output**:**<COMPLETE>**
Forward Generation	I gave a friend an instruction and five inputs. The friend read the instruction and wrote an output for every one of the inputs.\nHere are the input-output pairs: **Input**: [Q_1]\n**Output**: [A_1]\n\n**Input**: [Q_2]\n**Output**: [A_2] ... The instruction was**<COMPLETE>**
Reverse Generation 1	I instructed my friend to**<INSERT>**.The friend read the instruction and wrote an output for every one of the inputs.\nHere are the input-output pairs: **Input**: [Q_1]\n**Output**: [A_1]\n\n**Input**: [Q_2]\n**Output**: [A_2] ...
Reverse Generation 2	Professor Smith was given the following instructions:**<INSERT>**\nHere are the Professor's responses: **Q**: [Q_1]\n**A**: [A_1]\n\n**Q**: [Q_2]\n**A**: [A_2] ...
Resample Instruction	Generate a variation of the following instruction while keeping the semantic meaning. **Input**: [INSTRUCTION]\n**Output**:**<COMPLETE>**

C GENERATED INSTRUCTIONS

Table 3: Best APE selected instructions for underperforming tasks in zero-shot setting

Task	Best instruction	Zero-shot test accuracy
Passivization	to use the passive voice.	1
Membership	to choose the animals from the list	0.5
Second Letter	most likely "Find the second letter in each word."	0.84

Table 4: Worst APE selected instructions for underperforming tasks in zero-shot setting

Task	Worst instruction	Zero-shot test accuracy
Passivization	to reverse the order of the subject and object.	0.17
Membership	probably to sort the inputs alphabetically.	0
Second Letter	write the middle letter of the word.	0.32

Table 5: APE selected Rhyme instructions with zero and few-shot performance

Instruction	Zero-shot test accuracy	Few-shot test accuracy
probably “Write a word that rhymes with each of the following words.”	**0.55**	**0.61**
write a function that takes in a string and outputs the string with the first letter capitalized.	1	0.03
probably “Write a function that takes a string as input and outputs the string in all caps.”	0.99	0.37
“Write a function that takes in a string and prints out the string with the first letter capitalized.”	1	0.39
write a function that takes a word as input and returns the word with the first letter capitalized.	1	0.07

Table 6: APE selected Second Letters instructions with zero and few-shot performance

Instruction	Zero-shot test accuracy	Few-shot test accuracy
most likely "Find the second letter in each word."	**0.84**	**0.69**
to write the letter that appears second in the word.	**0.72**	**0.64**
to find the first vowel in each word.	0.60	0.62
to "write the vowel that comes before the first double letter in the word."	0.50	0.59
write the middle letter of the word.	0.32	0.22

Table 7: APE selected Large Animal instructions with zero and few-shot performance

Instruction	**Zero-shot test accuracy**	**Few-shot test accuracy**
"Identify which animal is larger."	0.97	0.97
given two animals, return the name of the animal that is more dangerous.	0.88	0.89
to choose the animal that is larger.	0.96	0.95
"write the name of the animal that is bigger."	0.97	0.94
"Choose the animal that is bigger."	0.94	0.95

Table 8: The best instruction under zero-shot test accuracy generated by APE for each of the 24 tasks in the Instruction-Induction benchmark

Category	Task	Best Instruction Generated by APE	Zero-Shot Test Accuracy
Spelling	First Letter	most likely "Write the first letter of the word."	1.00
	Second Letter	most likely "Find the second letter in each word."	0.84
	List Letters	to write the inputted word out letter by letter with a space in between each letter.	0.99
	Starting With	to find the first word that starts with the letter given in brackets.	0.68
Morpho-syntax	Pluralization	to pluralize the word.	1.00
	Passivization	to use the passive voice.	1.00
Syntax	Negation	" negate the statement" and the inputs were all factually correct statements.	0.83
Lexical Semantics	Antonyms	to write the opposite of the word given.	0.83
	Synonyms	to write a synonym for each input.	0.22
	Membership	to choose the animals from the list.	0.50
Phonetics	Rhymes	write a function that takes in a string and outputs the string with the first letter capitalized.	1.00
Knowledge	Larger Animal	"Identify which animal is larger."	0.97
Semantics	Cause Selection	"For each input, write the sentence that comes first chronologically."	0.84
	Common Concept	"List things that" and the inputs were " poker, displays of embarrassment, toilets" so the output should have been "involve flushes."	0.27
Style	Formality	"Translate the following phrases into more formal, polite language."	0.65
Numerical	Sum	"Add the two inputs together and output the result."	1.00
	Difference	"Subtract the second number from the first number."	1.00
	Number to Word	probably something like "Convert this number to words."	1.00
Multi-lingual	Translation English-German	to use the German cognate for each word.	0.82
	Translation English-Spanish	write a Spanish word for each English word.	0.86
	Translation English-Spanish	write the French word for each English word.	0.78
GLUE	Sentiment Analysis	write "positive" if the input is a positive review and "negative" if the input is a negative review.	0.94
	Sentence Similarity	"Determine whether two sentences are about the same thing" and the inputs were two sentences. The outputs were "0 - definitely not," "1 - probably not," "2 - possibly," "3 - probably," "4 - almost perfectly	0.43
	Word in Context	to compare the sentences and see if the word is used in the same context. "Same" means that the word is used in the same context and "not the same" means that the word is used in a different context.	0.62

D MORE VISUALIZATIONS

Table 9: Number of tasks that achieves human-level performance on zero-shot learning and few-shot learning.

Task	LogP		ExecACC	
	Forward	Insert	Forward	Insert
Beat Zero-shot human (Mean)	14	16	**19**	13
Beat Zero-shot human (Best)	19	18	**21**	19
Beat In-context w/o instr (Mean)	**21**	18	**21**	18
Beat In-context w/o instr (Best)	**23**	21	**23**	19
Beat In-context human (Mean)	**13**	11	**12**	11
Beat In-context human (Best)	**15**	12	**13**	12

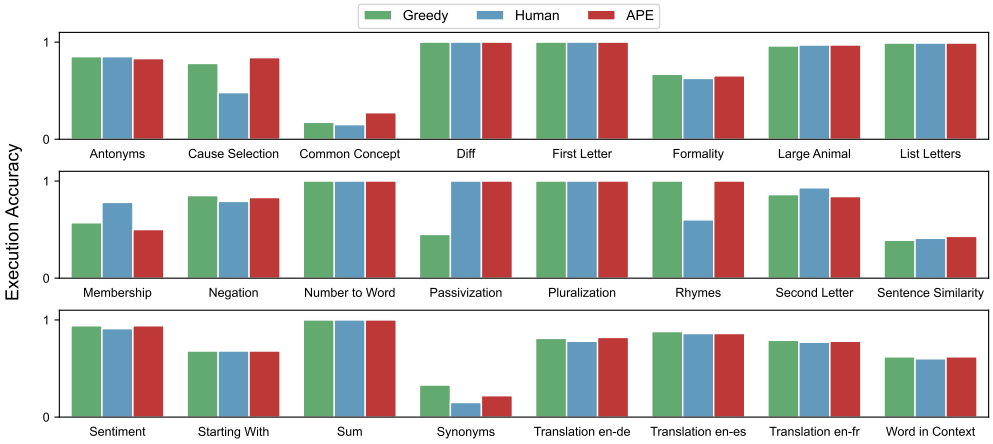


Figure 10: Zero-shot test accuracy of best performing instruction on 24 Instruction Induction tasks. APE achieves human-level performance on 21 out of 24 tasks.

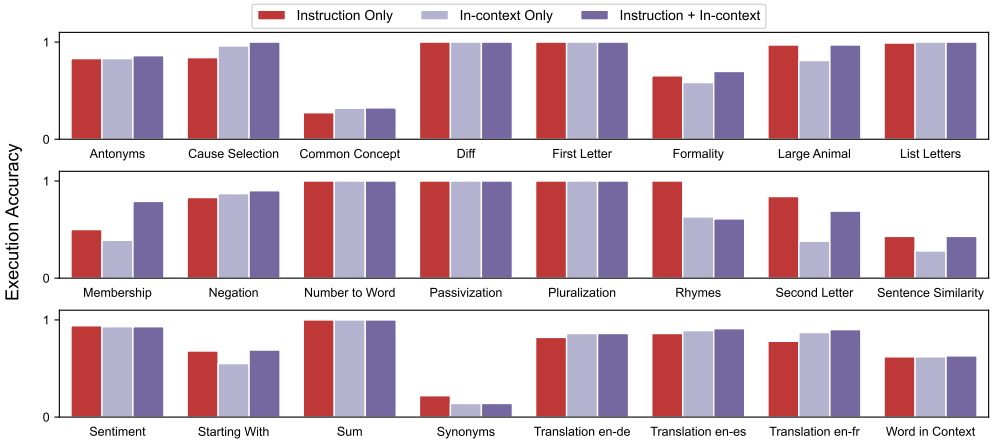


Figure 11: Few-shot in-context test accuracy of best performing instruction on 24 Instruction Induction tasks. The APE -generated instruction improves the few-shot in-context learning performance on 23 out of 24 tasks.

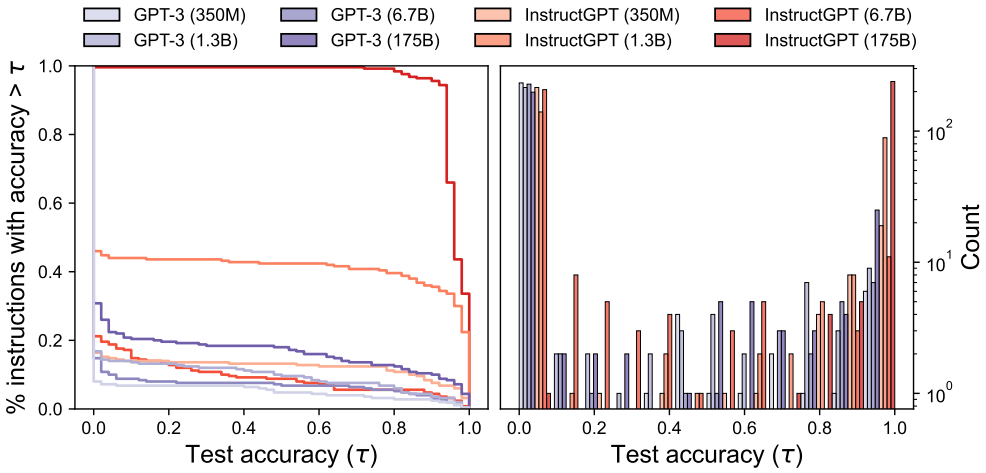


Figure 12: Survival function and the histogram of test accuracy on a simple task (i.e. Pluralization)

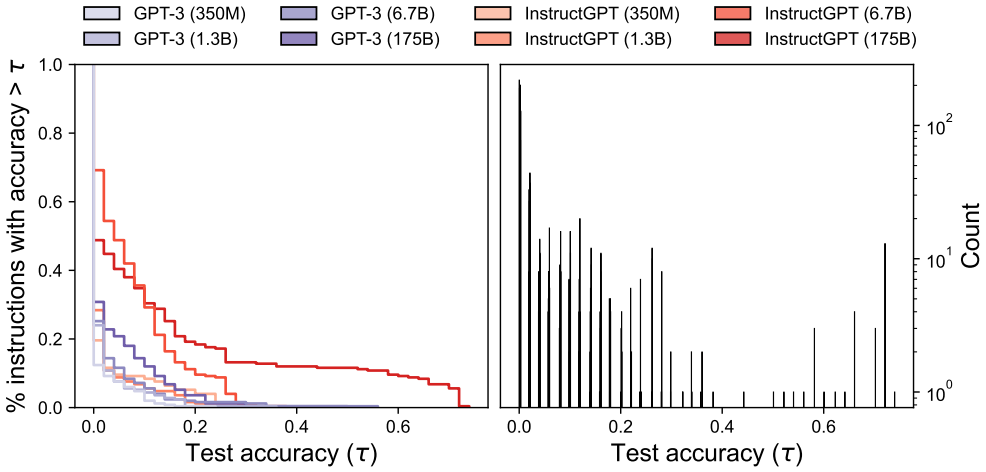
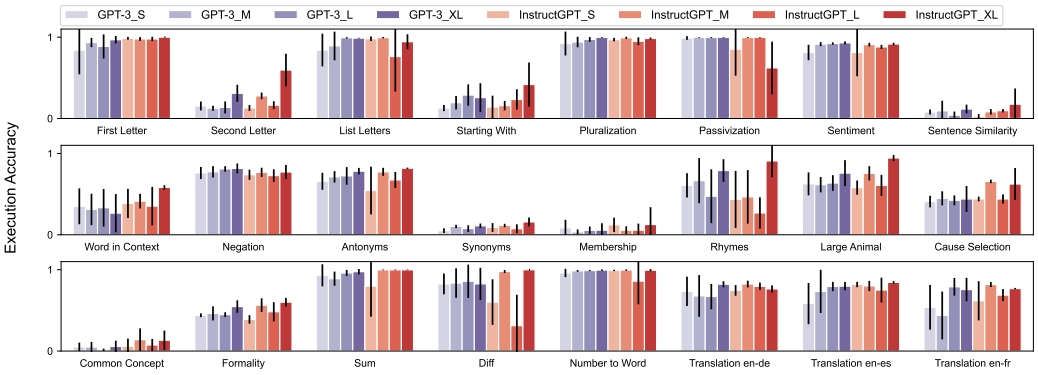
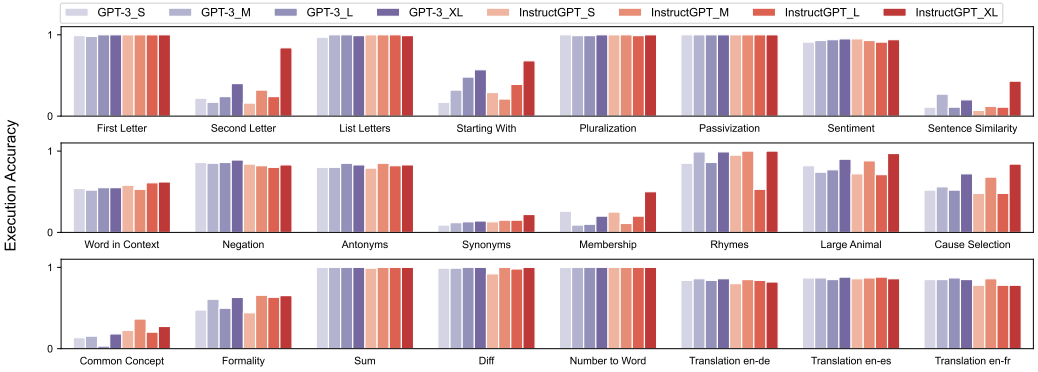


Figure 13: Survival function and the histogram of test accuracy on a challenging task (i.e. Start With)



(a)

Figure 14: Zero-shot test accuracy on 24 Instruction Induction tasks using eight different LLM models.



(a)

Figure 15: Zero-shot test accuracy of best performing instruction on 24 Instruction Induction tasks using eight different LLM models.

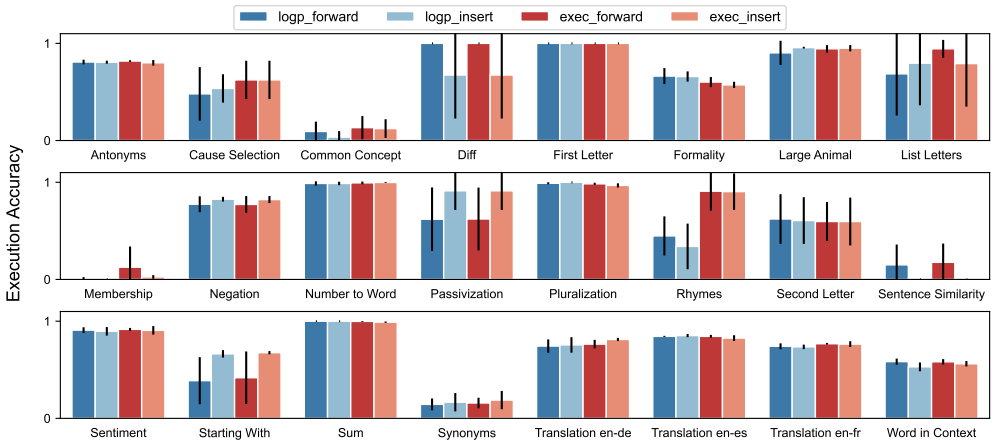


Figure 16: Zero-shot test accuracy on 24 Instruction Induction tasks using two different metrics and two different LLM models.

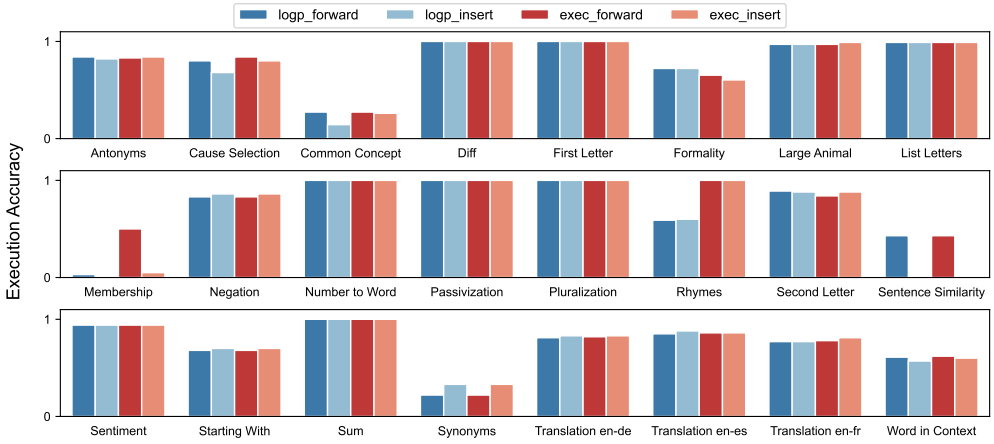


Figure 17: Zero-shot test accuracy of best performing instruction on 24 Instruction Induction tasks using two different metrics and two different LLM models.

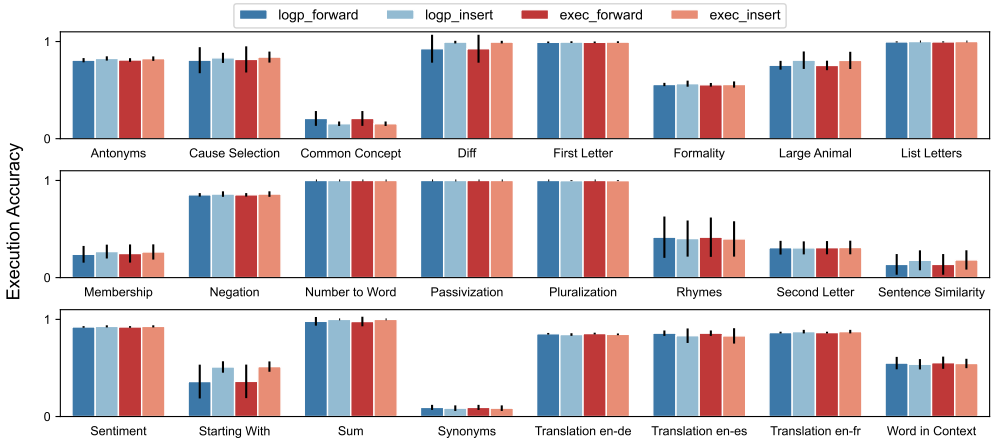


Figure 18: In-Context learning without instruction on 24 Instruction Induction tasks using two different metrics and two different LLM models.

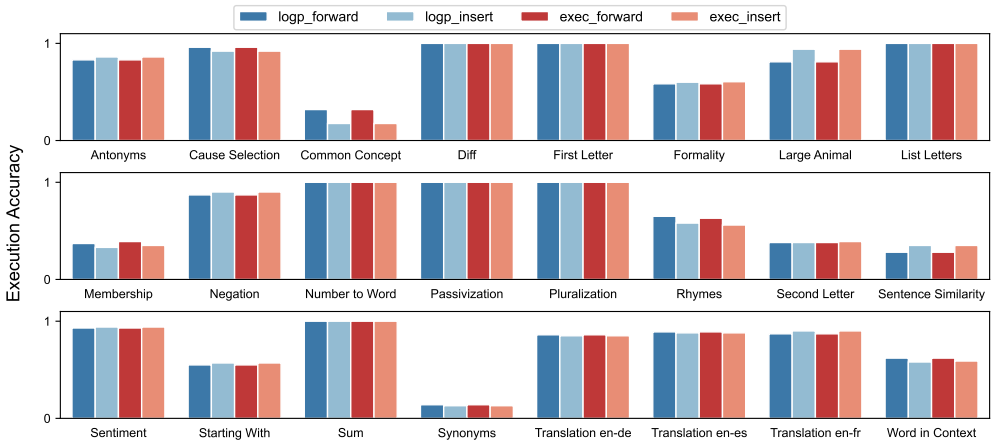


Figure 19: In-Context learning without instruction on 24 Instruction Induction tasks using two different metrics and two different LLM models.

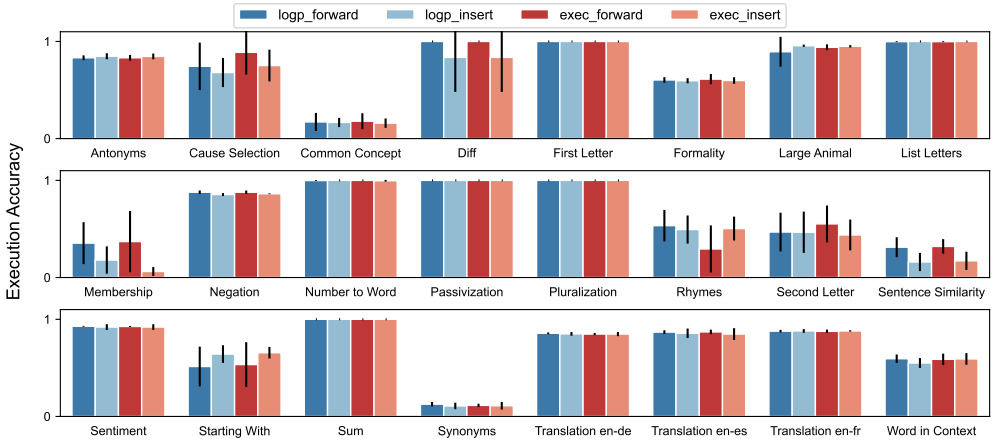


Figure 20: Test accuracy of in-Context learning with instruction on 24 Instruction Induction tasks using two different metrics and two different LLM models.

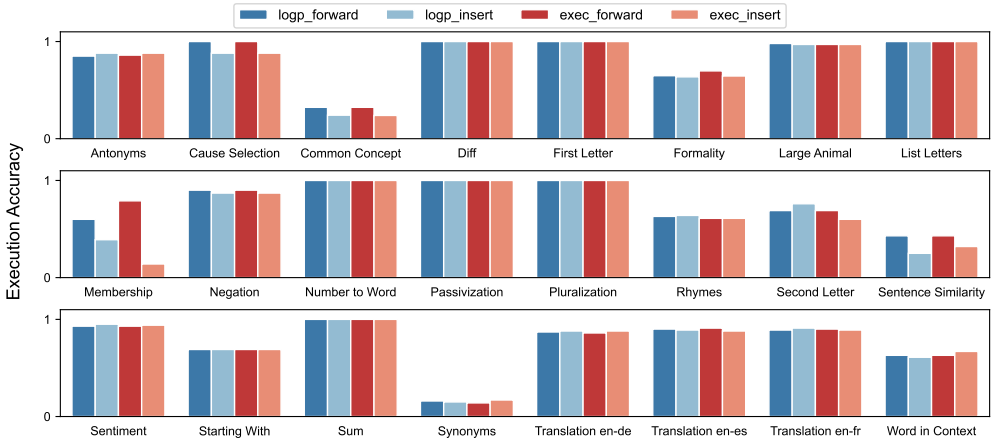


Figure 21: Test accuracy of in-Context learning with best performing instruction on 24 Instruction Induction tasks using two different metrics and two different LLM models.

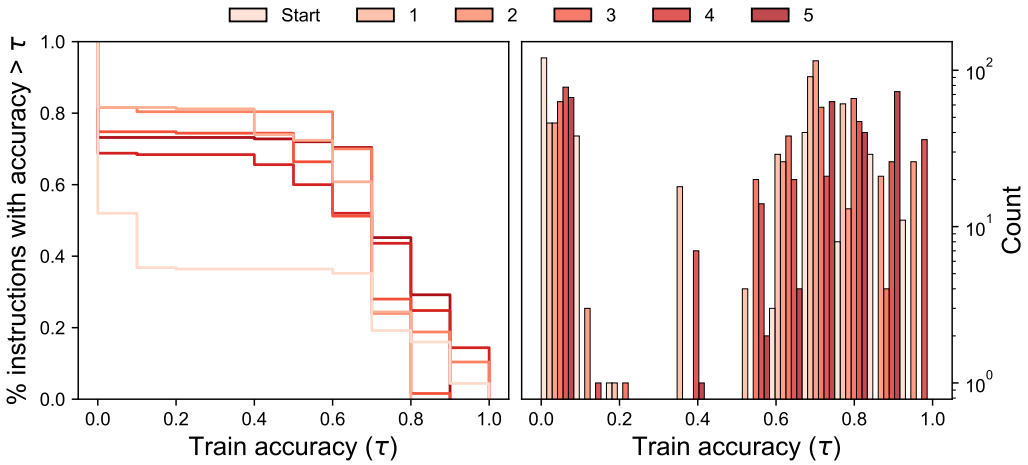


Figure 22: Iterative Monte Carlo search improves the quality of the instruction candidates at each round. Task: Antonyms.

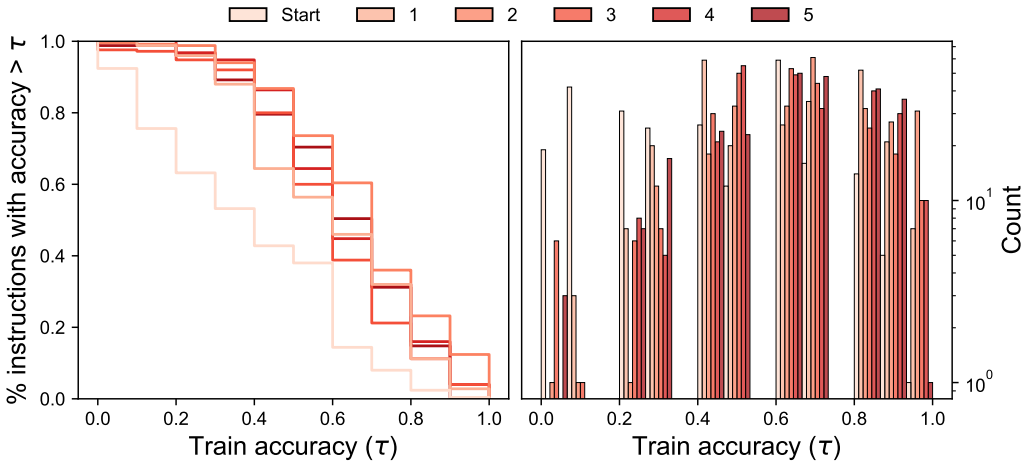


Figure 23: Iterative Monte Carlo search improves the quality of the instruction candidates at each round. Task: Cause Selection.

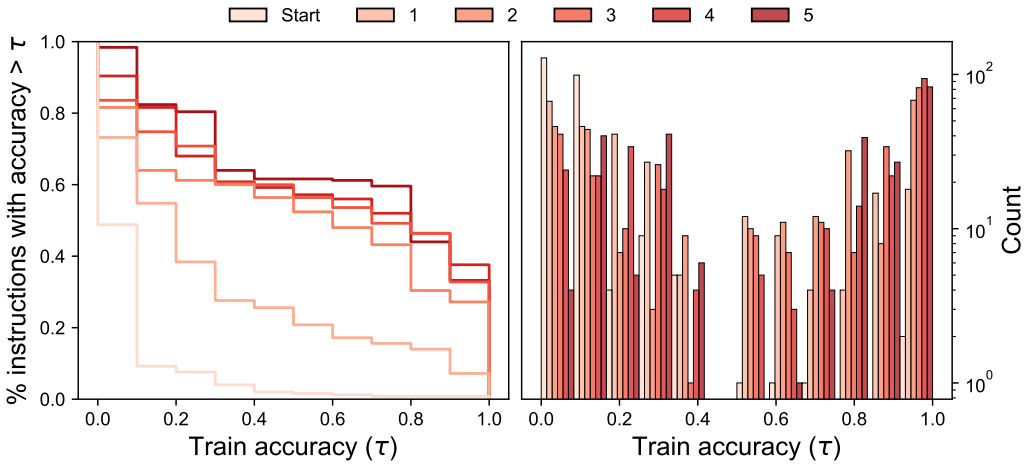


Figure 24: Iterative Monte Carlo search improves the quality of the instruction candidates at each round. Task: Passivization.

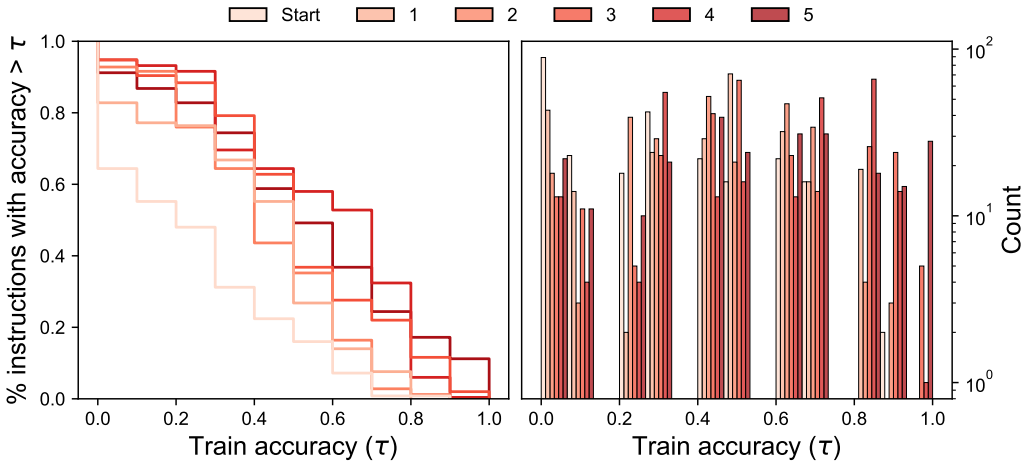


Figure 25: Iterative Monte Carlo search improves the quality of the instruction candidates at each round. Task: Second Letter.

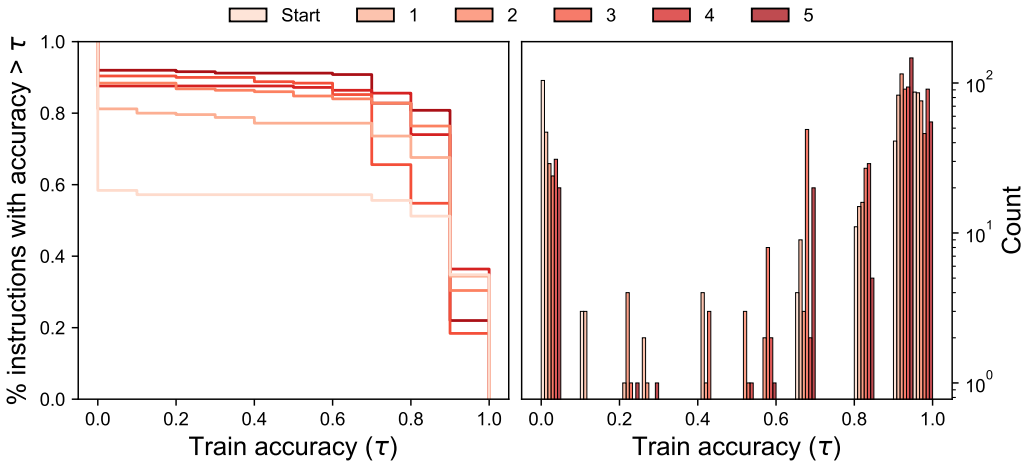


Figure 26: Iterative Monte Carlo search improves the quality of the instruction candidates at each round. Task: Sentiment.

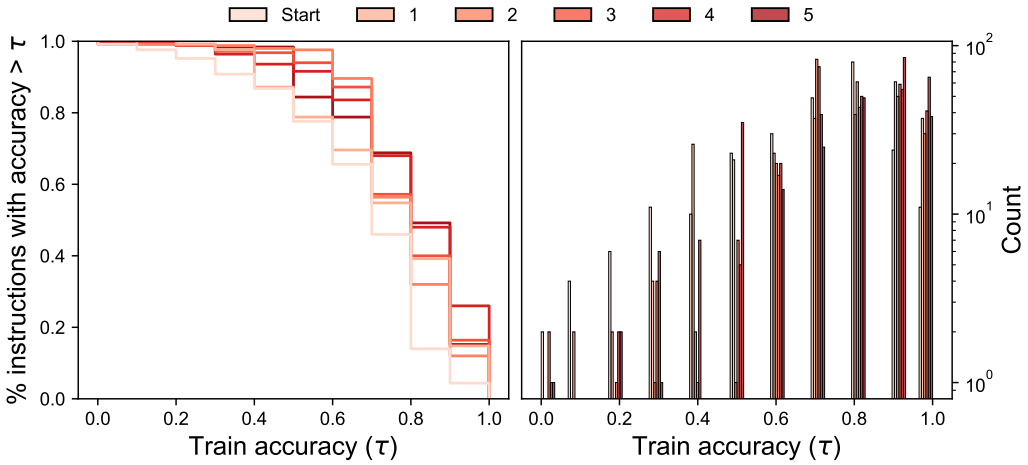


Figure 27: Iterative Monte Carlo search improves the quality of the instruction candidates at each round. Task: Translation en-fr.