Using an LLM to improve prompts to an LLM

References

- LLMs are Human-Level Prompt Engineers (https://arxiv.org/pdf/2211.01910.pdf)
    - APE summary (https://sites.google.com/view/automatic-prompt-engineer)

The Automatic Prompt Engineer (APE) is a system to *improve* upon prompts

- given a prompt
- APE will create a prompt that is *more effective*

It uses an LLM for multiple purposes

- to create variations of the given prompt
- to evaluate which variation is best

Using APE to improve upon Instruction Following

APE has been demonstrated to improve prompts **for a specific task** (not improvement of general prompts)

The task is: creating *instructions*

- to use in fine-tuning a raw LLM into an helpful Assistant
- we described Instruction Following in the module Synthetic data for Instruction Following (LLM_Instruction_Following_Synthetic_Data.ipynb)

In order to create an example of Instruction Following behavior we need a prompt with multiple parts

- a textual *task description* (*instruction*) that describes a task to be performed
- zero or more exemplars: demonstrating the input/output relationship described by the instruction

Given just the exemplars (the second part)

- we want APE to *create* the "best" instruction (first part)

The APE method for automatically generating "good" instructions is conceptually simple.

As a first step, we get an LLM to generate plausible instructions.

- Given the Input/Output exemplars
- Create a prompt
- Whose "response" is an *instruction* that is a plausible description of the input/output relation in the exemplars

We do this several times to generate multiple plausible instructions, conditional on the exemplars.

In a second step. we rank the multiple instruction candidates created in the first step.

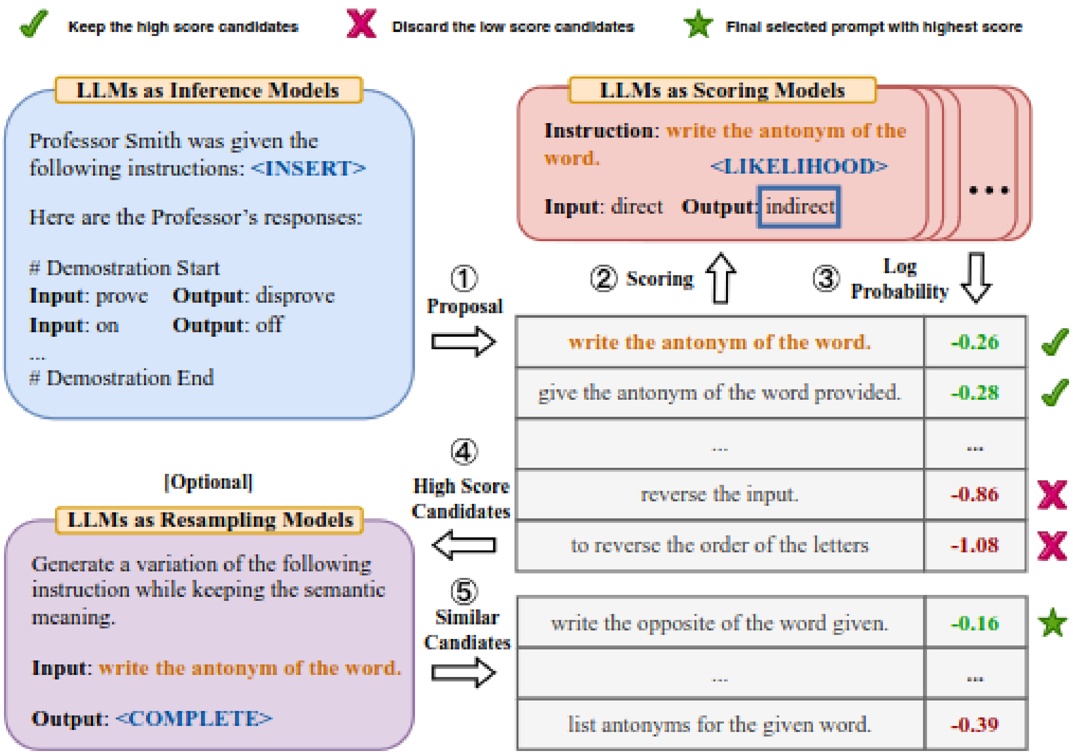
- we create a prompt requesting that the LLM *rank* the instructions created in the first step
- we filter the instructions to the most highly ranked results

As an optional third step, we can improve upon the *diversity* of the instructions selected in Step 2

- create a prompt requesting that the LLM generate a *variation* of a selected instruction
- Use an LLM to create a distribution of instruction, conditional on the exemplars

Here is a picture of the workflow.

APE Workflow



(a) Automatic Prompt Engineer (APE) workflow

Attribution: https://arxiv.org/pdf/2211.01910.pdf#page=2

Step 1: LLM as Inference Model

Goal: Get the LLM to create multiple *instruction* candidates, given the intended response

- intended *response* are exemplars demonstrating the input/output relation
  - `Input: prove, Output: disprove`

Given the context (response), the LLM is prompted to generate an *instruction* that could cause the given response

- prompt is a Masked Language Modeling task: fill in the mask (`<INSERT>`)

The LLM predicts the Instruction (`<INSERT>`) that best describes the task

- as illustrated by the exemplars

We obtain several candidates for the Instruction

- and rank them in a subsequent step
- that also uses an LLM for the ranking

This is in the same spirit as Backtranslation (LLM_Instruction_Following_Synthetic_Data.ipynb#Generating-Instructions-via--Backtranslation)

- generate an instruction from a response
- by learning a new model to map from response to instruction
- rather than adapting an LLM to do the same

Note

We will explore the context in which Backtranslation is used in more detail

- when we discuss the topic of Synthetic Data

For now: we only wish to illustrate the concept of using an LLM

- to map a response to an input
- rather than the usual mapping of input to a response

Steps 2 and 3: LLM as Scoring Model

Step 1 has created multiple possible instructions that are consistent with the exemplars.

We wish to rank them and choose the best.

The ranking is performed by prompting the LLM to compute the likelihood

- that a given response (an exemplar)

  ```
  Input: direct, Output: indirect
  ```

- is consistent with each candidate instruction

The candidate instructions are ranked from highest Likelihood to lowest.

Note

Likelihood is expressed as the log probability

- is a negative number since probabilities are fractions
    - less negative numbers are higher probabilities
- is the probability of the generated sequence
    - product of the individual probabilities of the tokens in the sequence

Steps 4 (Optional): LLM as Re-sampling model

Here, we create multiple variants of a highly ranked instruction.

Given the candidate Instruction selected by the previous step

- generate a variation of the instruction
- by asking the LLM to create it via text completion

```
Generate a variation of the following instruction ...

Input: write the antonym of the word; Output: <COMPLETE>
```

Each step is implemented as an instance of the pre-trained LLM's ability to complete text (or fill in a mask).

No fine-tuning or adaptation of weights is involed.

Forward/Reverse mode generation of candidates

This is a minor technical point.

The prompt in Step 1 of the workflow above is not in the format consistent with text-continuation

- format is called *forward generation*
- so must use an LLM that solves Masked Language task, rather than text-continuation

An alternate prompt can be used that is consistent with text-continuation

- format is called *reverse generation*

APE Forward/Reverse Generation templates

Attribution: https://arxiv.org/pdf/2211.01910.pdf#page=4

Forward Generation Template

I gave a friend an instruction and five

Reverse Generation Template

APE evaluation: super-human performance !

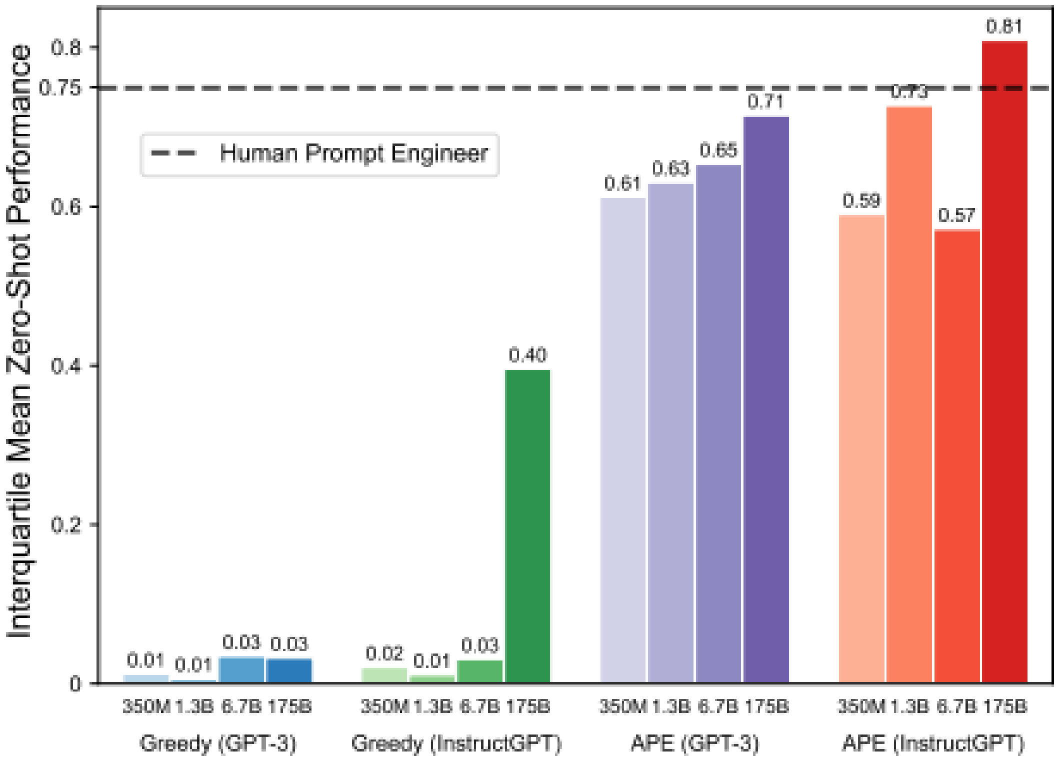
Here is a comparison of APE generated prompts

- versus
    - an alternate method (previously published), labeled "Greedy"
    - a human engineer (horizontal dotted line)
- evaluated on models of various sizes
    - GPT-3
    - Instruct GPT-3 (fine-tuned for instruction following)
- using 24 NLP tasks

Note

The reported statistic is *interquartile mean* (i.e., average after dropping the upper and lower 25% of results)

APE Workflow and Results



(b) Interquartile mean across 24 tasks

Attribution: https://arxiv.org/pdf/2211.01910.pdf#page=2

Zero-shot: Improving on "Let's think step by step"

[Chain of Thought (CoT) prompting (NLP_Beyond_LLM.ipynb#Chain-of-thought-prompting)](NLP_Beyond_LLM.ipynb#Chain-of-thought-prompting)

- is a simple technique
- for create prompts with better performance
- for multi-step reasoning problems

In the few-shot setting

- exemplars demonstrate step by step reasoning
- eliciting the LLM to produce text continuation that also exhibits step by step reasoning

In the zero-shot setting, it simply involves appending

Let's think step by step

to the prompt

Chain of Thought Prompting

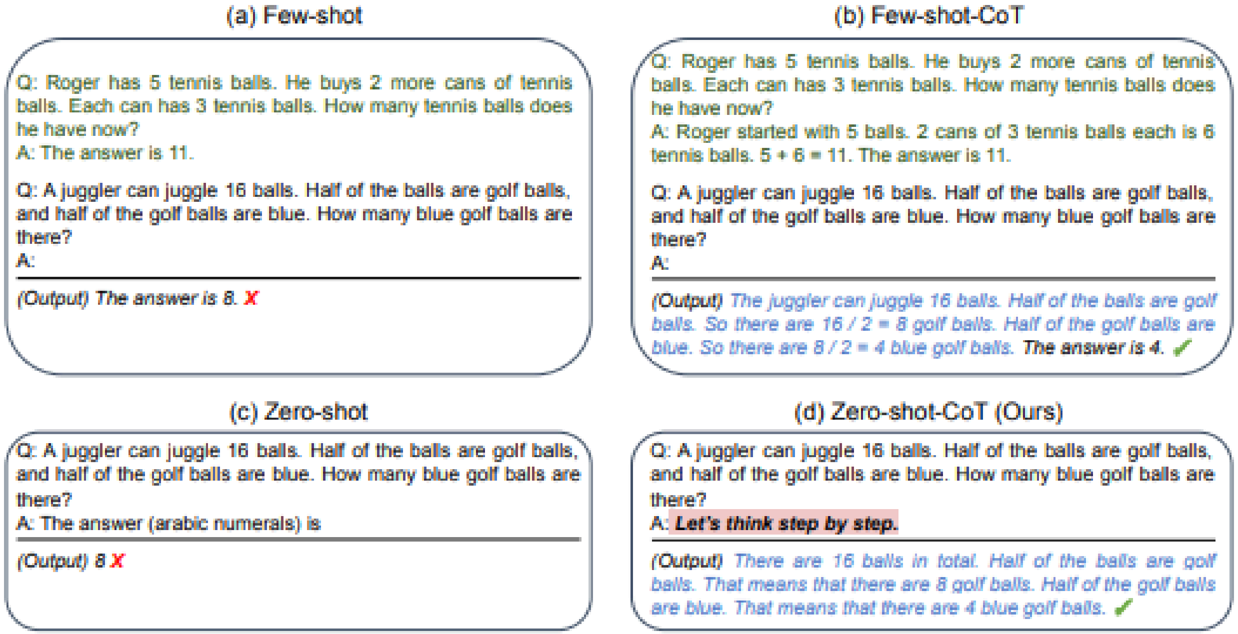


Figure 1: Example inputs and outputs of GPT-3 with (a) standard Few-shot ([Brown et al., 2020]), (b) Few-shot-CoT ([Wei et al., 2022]), (c) standard Zero-shot, and (d) ours (Zero-shot-CoT). Similar to Few-shot-CoT, Zero-shot-CoT facilitates multi-step reasoning (blue text) and reach correct answer where standard prompting fails. Unlike Few-shot-CoT using step-by-step reasoning examples **per task**, ours does not need any examples and just uses the same prompt "Let's think step by step" *across all tasks* (arithmetic, symbolic, commonsense, and other logical reasoning tasks).

Attribution: https://arxiv.org/pdf/2201.11903.pdf

Let's use APE (https://arxiv.org/pdf/2211.01910.pdf#page=19) to find a zero-shot prompt appendage that improves upon

> *Let's think step by step*

That is: the instruction consists of

- the task description
- followed by a "magic suffix" (e.g., "let's think step by step")

We want to find the best "magic suffix".

The authors use the following template (where `INPUT` and `OUTPUT` are place-holders for an actual question and answer pair).

```
Instruction: Answer the following question

Q: INPUT
A: Let's <INSERT> OUTPUT
```

We are using forward-mode generation to get APE

- to create a phrase that follows the `INPUT`
- that begins with the word "Let's"
- and is followed by

APE will solve for the best value of .

APE creates

> *Let’s work this out in a step by step way to be sure we have the right answer.*

and the author's demonstrate improved performance on several benchmarks.

This is a nice demonstration of using an LLM to help craft better prompts to LLM's.

In [2]:

```
print("Done")
```

```
Done
```