Presentation On

Large Language Models Are Human-Level Prompt Engineers

Nirjhar Nath

May 19, 2025

Chennai Mathematical Institute

Introduction

- LLMs have demonstrated impressive general-purpose computing capabilities.
- Task performance is significantly influenced by prompt quality and human-crafted prompts are typically required.
- Automatic Prompt Engineer (APE) has been introduced to automatically generate instructions.
- APE outperforms both human-designed prompts and previous LLM baselines.

Prompt Engineering

What prompt can we use to get GPT to solve this task?

INPUT	OUTPUT
Sentence 1: The child hurt their knee. **Sentence 2:** The child started crying.	The child hurt their knee.
Sentence 1: My car got dirty. **Sentence 2:** I washed the car.	My car got dirty.
Sentence 1: School was cancelled. **Sentence 2:** There was a snowstorm.	There was a snowstorm.
...	...
Sentence 1: The window broke. **Sentence 2:** The boy threw a rock.	The boy threw a rock.

Prompt Candidates

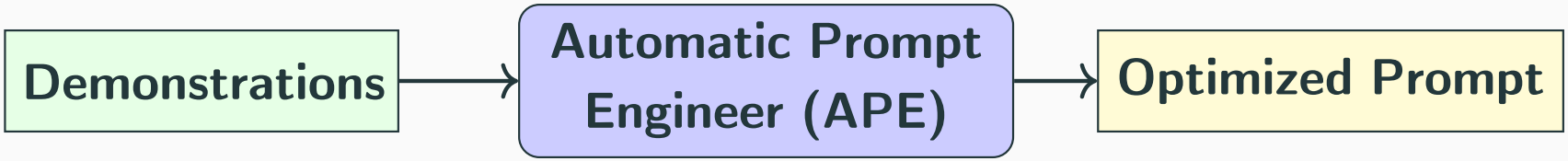
- Each input consists of two sentences, where one is the cause and the other is the outcome. Write the cause sentence.
- The input consists of two sentences. One is the cause of the other. Write the cause sentence.
- The input is a cause and effect. Write the cause.
- Find the cause in the following cause and effect pair.
- Output the sentence describing the cause (the other sentence is what happened as a result).

Testing the Candidates against LLM

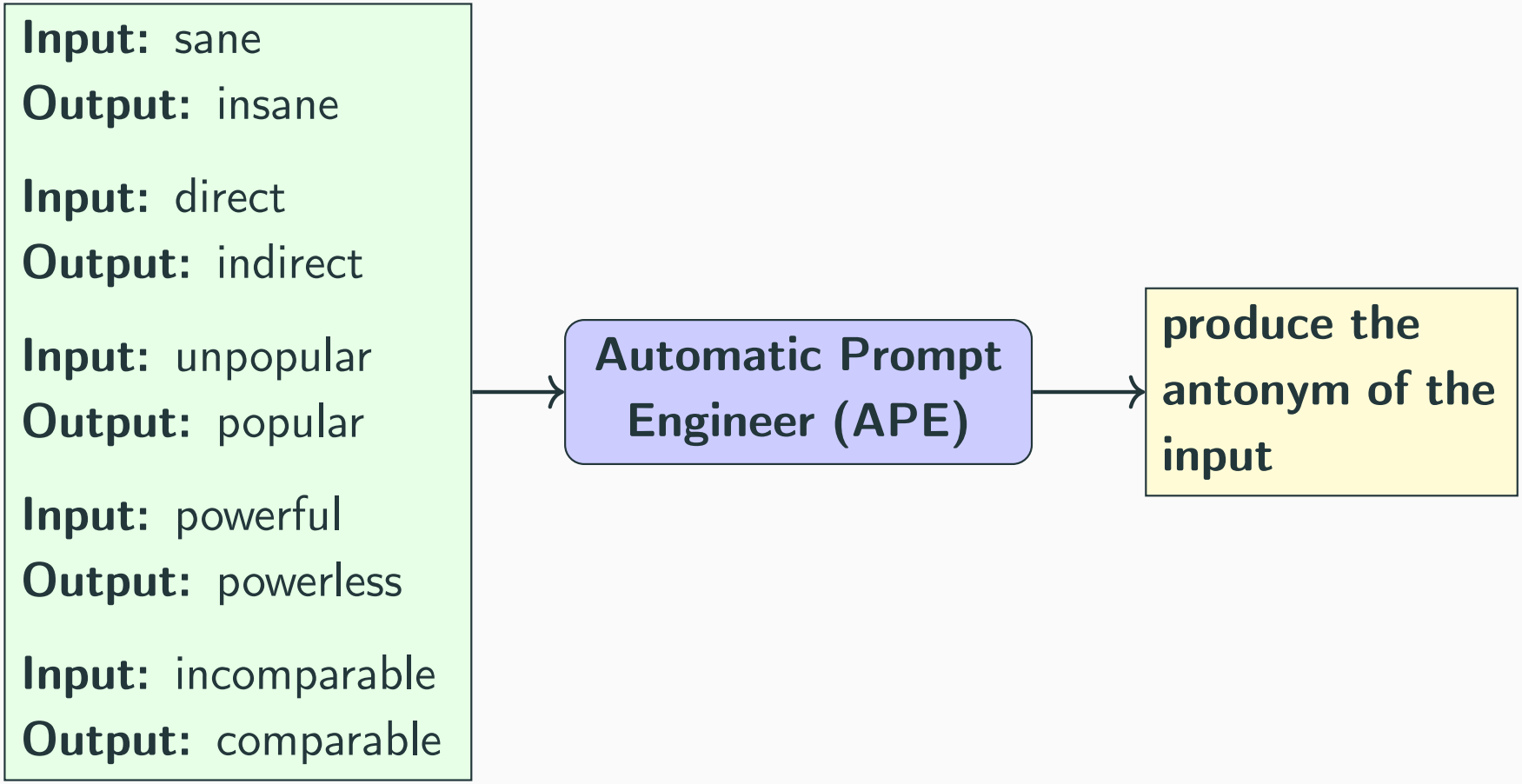
Prompt Candidates	Test Accuracy
Each input consists of two sentences, where one is the cause and the other is the outcome. Write the cause sentence.	16%
The input consists of two sentences. One is the cause of the other. Write the cause sentence.	42%
The input is a cause and effect. Write the cause.	48%
Find the cause in the following cause and effect pair.	36%
Output the sentence describing the cause (the other sentence is what happened as a result).	10%

Note: Tested on GPT *text-davinci-002*

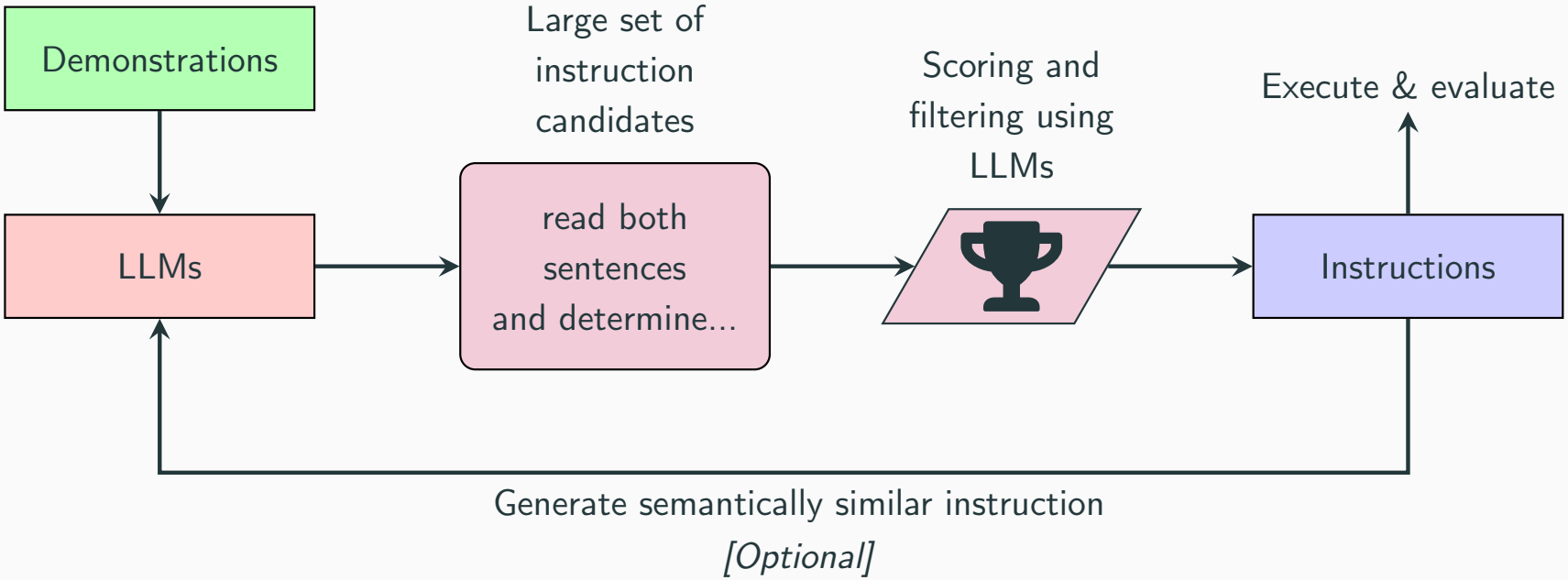
Automatic Prompt Engineer (APE)



Automatic Prompt Engineer (APE)



APE Pipeline



APE Demo

Automatic Prompt Engineer (APE)

In its simplest form, APE takes as input a dataset (a list of inputs and a list of outputs), a prompt template, and optimizes this prompt template so that it generates the outputs given the inputs. APE accomplishes this in two steps. First, it uses a language model to generate a set of candidate prompts. Then, it uses a prompt evaluation function to evaluate the quality of each candidate prompt. Finally, it returns the prompt with the highest evaluation score.

In [1]:
```
# First, let's define a simple dataset consisting of words and their antonyms.
words = ["sane", "direct", "informally", "unpopular", "subtractive", "nonresidential",
    "inexact", "uptown", "incomparable", "powerful", "gaseous", "evenly", "formality",
    "deliberately", "off"]
antonyms = ["insane", "indirect", "formally", "popular", "additive", "residential",
    "exact", "downtown", "comparable", "powerless", "solid", "unevenly", "informality",
    "accidentally", "on"]
```

In [2]:
```
# Now, we need to define the format of the prompt that we are using.

eval_template = \
"""Instruction: [PROMPT]
Input: [INPUT]
Output: [OUTPUT]"""
```

In [7]:
```
# Now, let's use APE to find prompts that generate antonyms for each word.
from automatic_prompt_engineer import ape

result, demo_fn = ape.simple_ape(
    dataset=(words, antonyms),
    eval_template=eval_template,
)
```

```
Generating prompts...
Model returned 50 prompts. Deduplicating...
Deduplicated to 18 prompts.
Evaluating prompts...
Evaluating prompts: 100%|██████████| 10/10 [00:07<00:00,  1.25it/s]
Finished evaluating.
```

APE Demo

In [8]:
```
# Let's see the results.
print(result)
```

```
score: prompt
----------------
-0.18:  take the opposite of the word.
-0.20:  produce an antonym (opposite) for each word provided.
-0.22:  take the opposite of the word given.
-0.29:  produce an antonym for each word given.
-0.39:  "list antonyms for the following words".
-0.41:  produce an antonym (opposite) for each word given.
-4.27:  reverse the input-output pairs.
-6.42:  take the last letter of the word and move it to the front, then add "ay" to the end of the word. So the input-output pairs should be:

Input: subtractive
Output: activesubtray

-6.57:  take the last letter of the input word and use it as the first letter of the output word.
-6.58:  "reverse the word."
```

Let's compare with a prompt written by a human:

"*Write an antonym to the following word.*"

In [9]:
```
from automatic_prompt_engineer import ape

manual_prompt = "Write an antonym to the following word."

human_result = ape.simple_eval(
    dataset=(words, antonyms),
    eval_template=eval_template,
    prompts=[manual_prompt],
)
```

n [10]:
```
print(human_result)
```

```
log(p): prompt
----------------
-0.24: Write an antonym to the following word.
```

Main Contributions of the Paper

- APE frames instruction generation as natural language program synthesis, optimizing it with Monte Carlo search.
- Achieves human-level zero-shot performance on 24/24 Instruction Induction and 17/21 Big-Bench tasks.
- Enhances few-shot learning, improves zero-shot chain of thought, and steers LLMs toward truthfulness and informativeness.

Natural Language Program Synthesis - Problem Definition

- **Task Definition:**
  - Given a dataset of demonstrations: $\mathcal{D}_{\text{train}} = \{(Q, A)\}$ sampled from population $\mathcal{X}$.
  - A prompted model M.
- **Objective:**
  - Find a single instruction ρ such that when M is prompted with $[\rho; Q]$, it produces the output A.
- **Formulation as Optimization Problem:**

$$\rho^* = \arg\max_{\rho} f(\rho) = \arg\max_{\rho} \mathbb{E}_{(Q,A)}[f(\rho, Q, A)]$$

- **Special Case:**
  - Q may be the empty string, optimizing ρ as a prompt to produce outputs $\{A\}$ directly.

Natural Language Program Synthesis - Solution Approach

- **Challenge:**
  - Compatibility of any instruction ρ with model M is unknown.
  - Due to the infinitely large search space, finding the right instruction can be extremely difficult
- **Proposed Solution - APE Algorithm:**
  - Uses LLMs for:
    - **Proposal:** Generate candidate prompts.
    - **Scoring:** Filter and refine candidates based on a scoring function.
  - Selects the highest-scoring instruction.
- **Summary:**
  - APE iteratively proposes and scores prompts to optimize ρ.

Workflow: Step 0

LLMs as Inference Models

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

Here are the Professor's responses:

Demostration Start
Input: prove **Output**: disprove
Input: on **Output**: off
...
Demostration End

Workflow: Step 1

LLMs as Inference Models

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

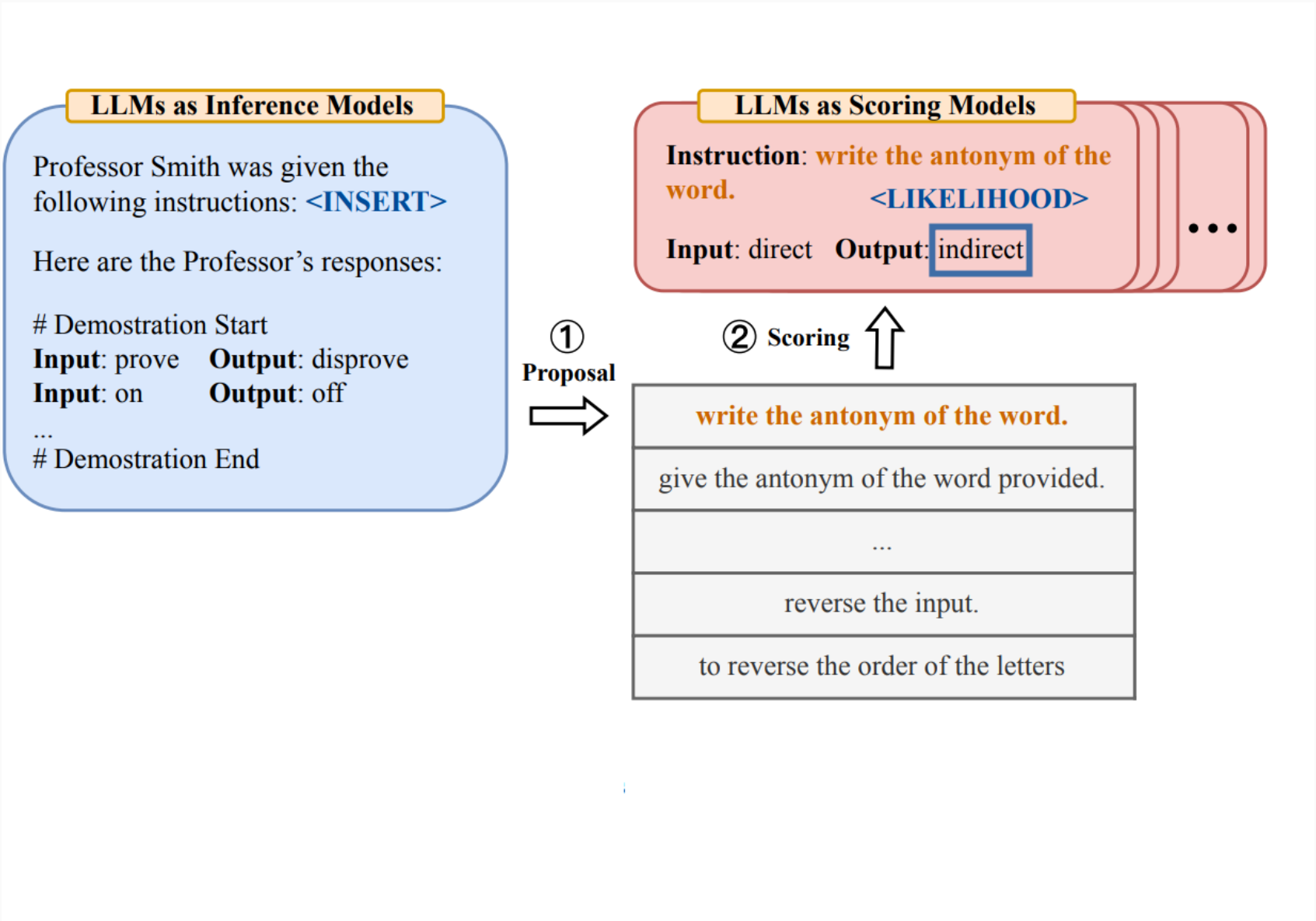
Here are the Professor's responses:

Demostration Start
Input: prove **Output**: disprove
Input: on **Output**: off
...
Demostration End

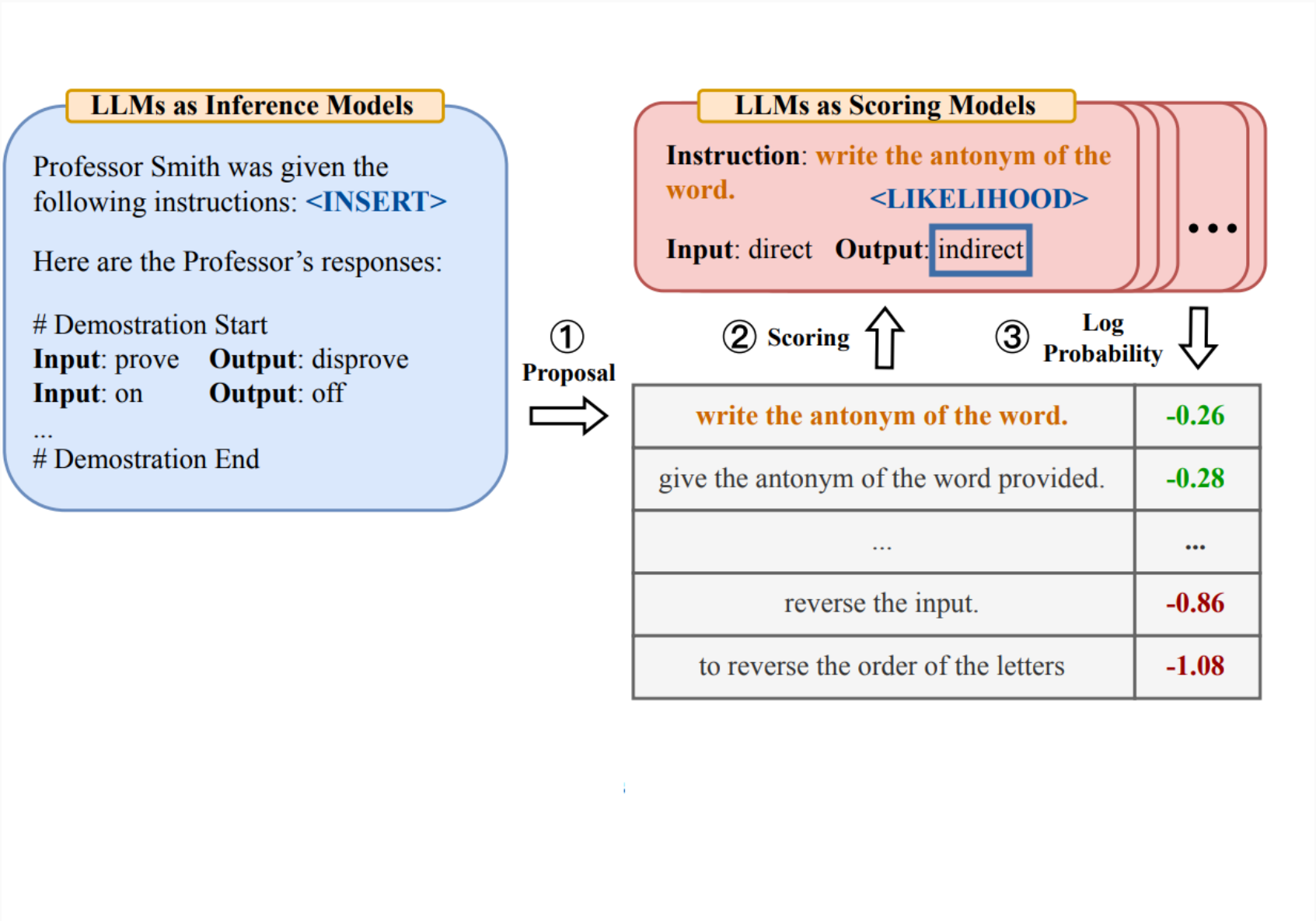
① **Proposal**

write the antonym of the word.
give the antonym of the word provided.
...
reverse the input.
to reverse the order of the letters

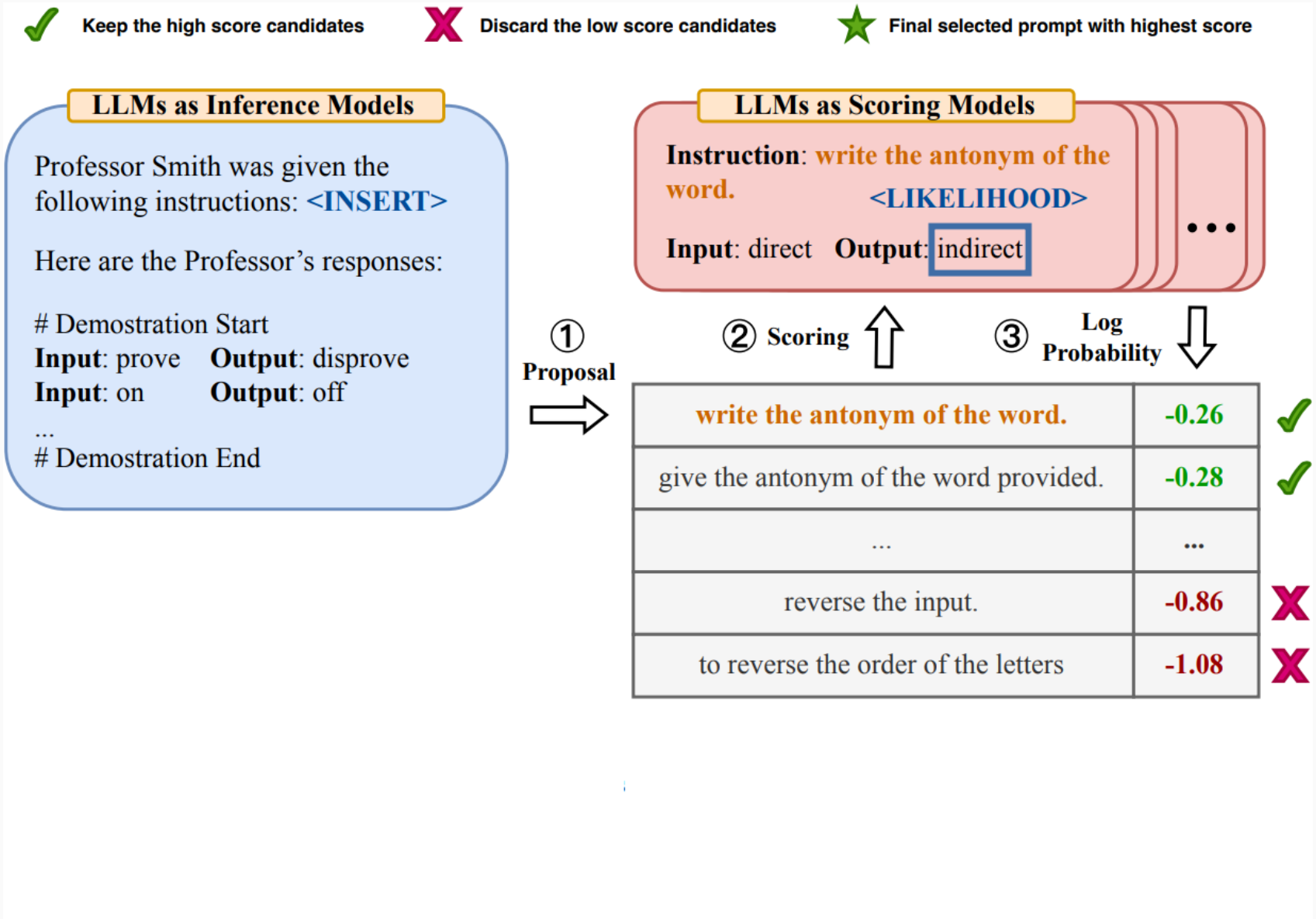
Workflow: Step 2



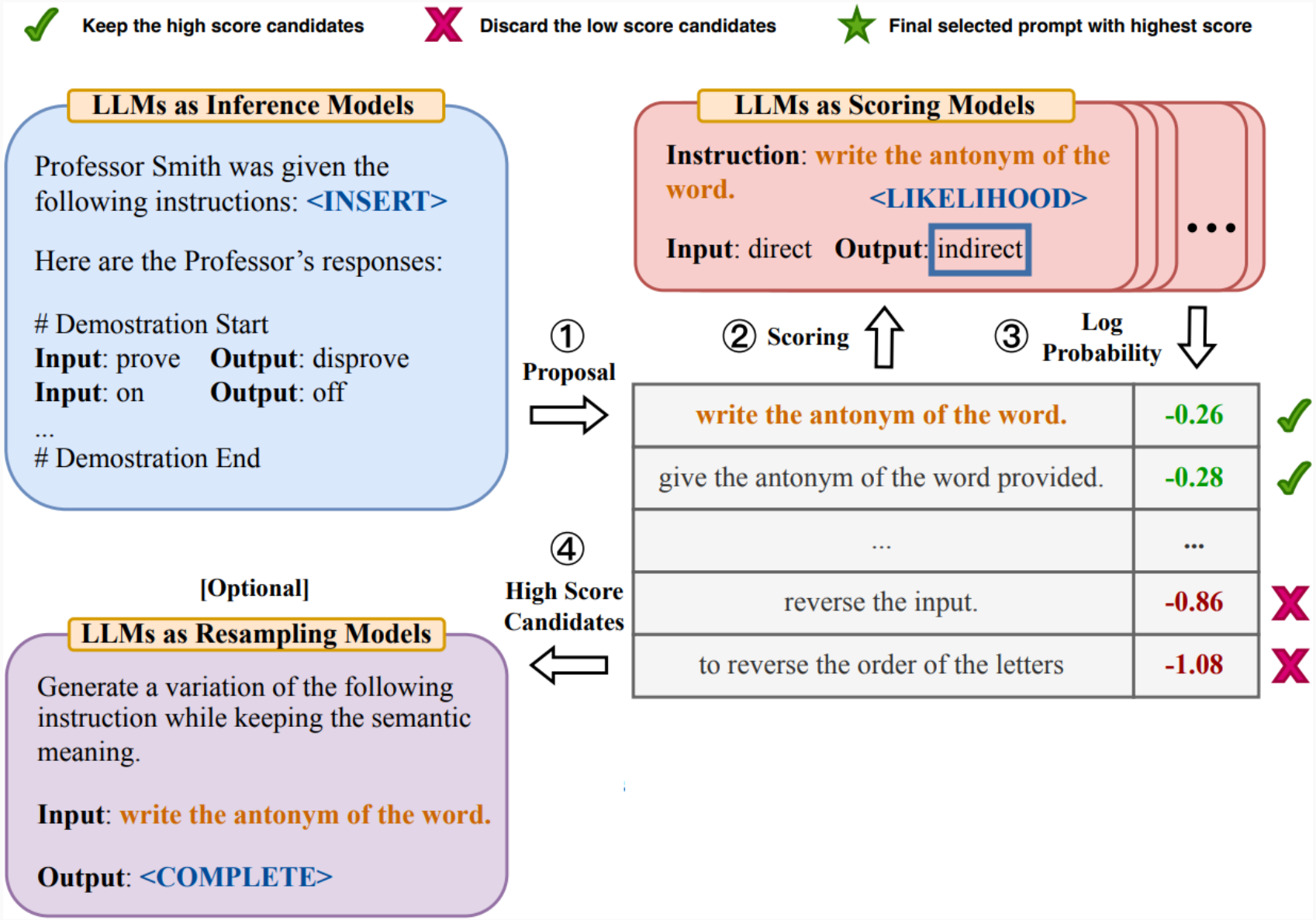
Workflow: Step 3



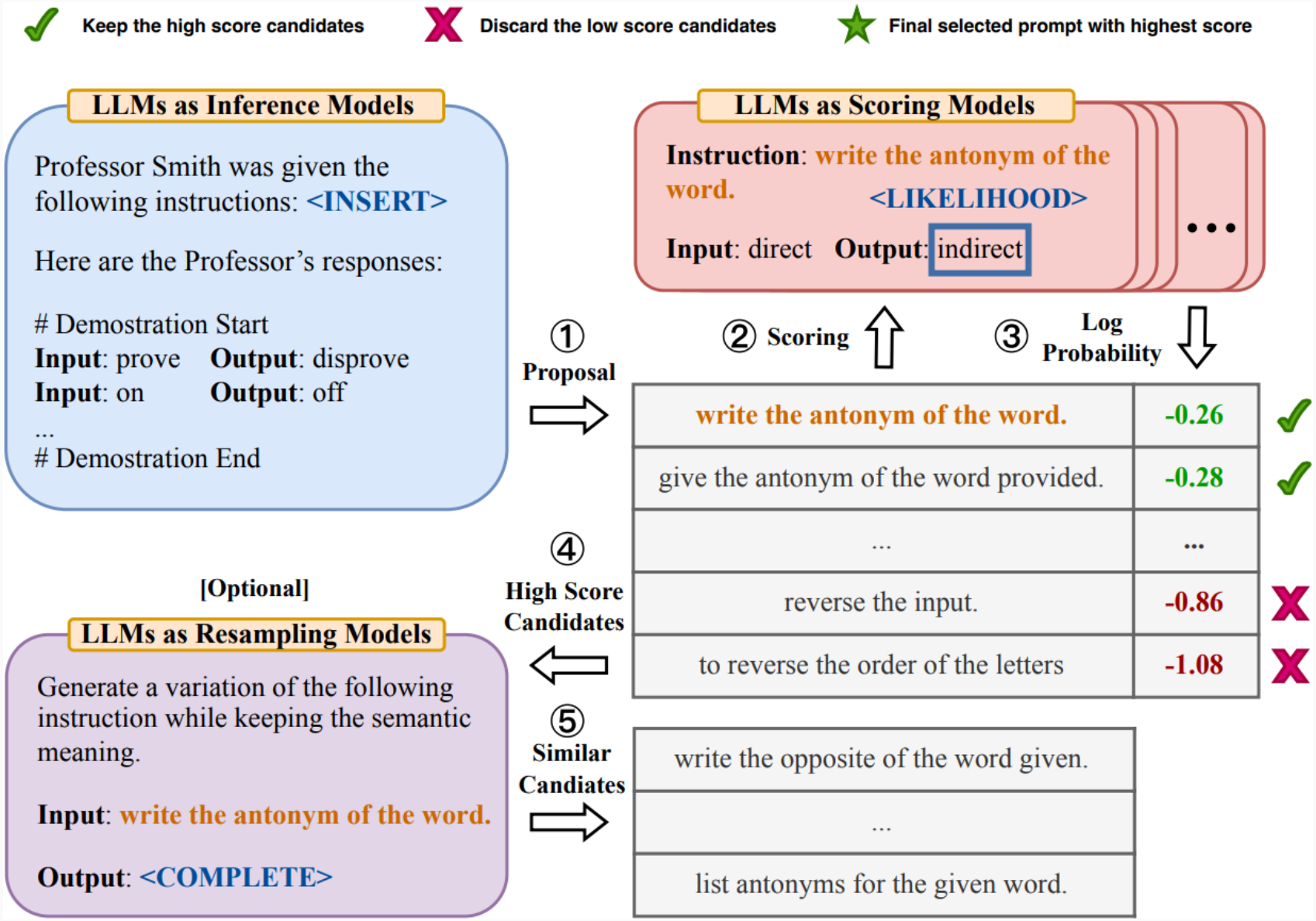
Workflow: Step 3.1



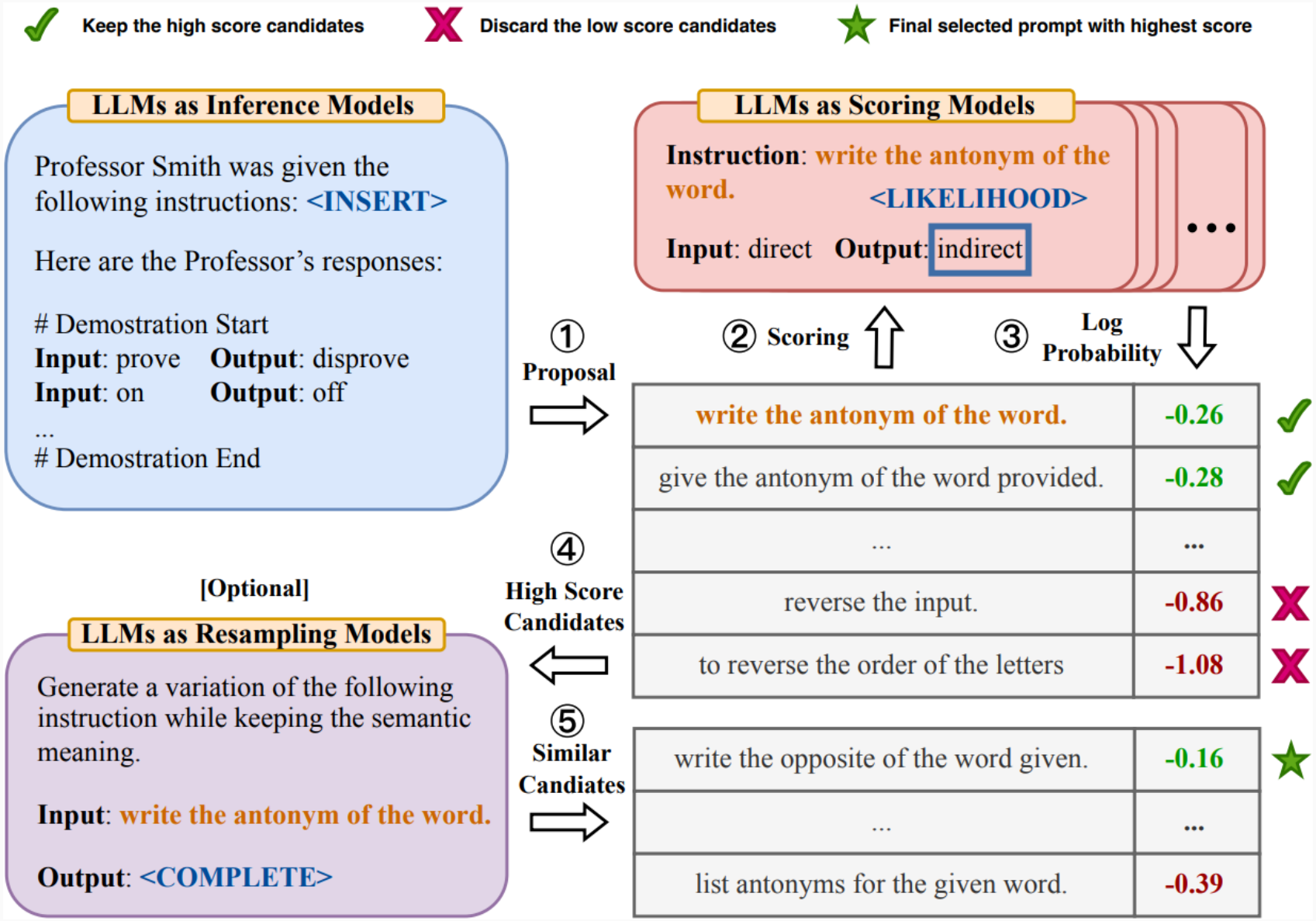
Workflow: Step 4



Workflow: Step 5



Workflow: Step 5.1



Algorithm: Automatic Prompt Engineer (APE)

Require:

$\mathcal{D}_{\text{train}} \leftarrow \{(Q, A)\}_n$: Training examples,

$f : \rho \times \mathcal{D} \rightarrow \mathbb{R}$: Score function

$U \leftarrow \{\rho_1, \rho_2, \ldots, \rho_m\}$ ▷ Sample instruction proposals using LLM

while not converged **do**

  Select a random subset $\tilde{\mathcal{D}}_{\text{train}} \subset \mathcal{D}_{\text{train}}$

  for all $\rho \in U$ **do**

    $\tilde{s} \leftarrow f(\rho, \tilde{\mathcal{D}}_{\text{train}})$ ▷ Evaluate score on the subset

  end for

  $U_k \leftarrow$ Top k% of U based on $\{\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_m\}$

  $U \leftarrow U_k$ or $U \leftarrow \text{resample}(U_k)$ ▷ Update or resample

end while

return $\rho^* \leftarrow \arg\max_{\rho \in U_k} f(\rho, \mathcal{D}_{\text{train}})$

Initial Proposal Distributions

Generating the right instruction from an infinite search space is highly complex. To tackle this:

- LLMs are used to propose a diverse set of instructions.
- These proposals are sampled from the distribution

$$P(\rho|\mathcal{D}_{\text{train}}, f(\rho) \text{ is high}).$$

- LLMs are asked to infer instructions that are more likely to have high compatibility with M based on the given demonstrations.
- Two approaches to generate high-quality candidates are considered:
  - Forward Mode Generation
  - Reverse Mode Generation

Forward Mode Generation

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>

- In forward generation, we prompt the LLM to generate instructions (ρ) at the end based on the input-output pairs provided.
- The LLM is asked to "complete" the instruction (ρ) that would match the observed input-output behavior.

Reverse Mode Generation

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:** $[A_2]$
...

- In reverse generation, the goal is to generate instructions (ρ) that could be anywhere in the text—not just at the end.
- This method uses models capable of in-filling, such as T5 (Raffel et al., 2020), GLM (Du et al., 2022), and InsertGPT (Bavarian et al., 2022).

Score Functions

- A **score function** $f(\rho, Q, A)$ evaluates the effectiveness of an instruction ρ when applied to an input Q to produce the expected output A.
- The idea is to assign a numeric score that reflects how well the model's output matches the desired behavior.
- Two Main Score Functions:
  - Execution Accuracy f_{exec}
  - Log Probability f_{log}

Execution Accuracy

In most cases, execution accuracy f_{exec} is defined as:

$$f_{\text{exec}}(\rho, Q, A) = \begin{cases} 1 & \text{if } M([\rho; Q]) = A \\ 0 & \text{otherwise} \end{cases}$$

- If the model's output exactly matches the expected answer A, the score is 1.
- If it does not match, the score is 0.
- This is essentially a 0-1 loss function, where any deviation from the expected output results in a complete failure for that example.

Variants of Execution Accuracy:

- In certain tasks, the evaluation may include *invariants*.
- For example, if the outputs are **sets** instead of sequences, the order of elements may not matter.
  - In this case, the execution accuracy is evaluated using an exact set match (Honovich et al., 2022).

Log Probability

- A softer, probabilistic scoring function measuring model confidence
- It is defined as:

$$f_{\log}(\rho, Q, A) = \log P(A \mid [\rho; Q])$$

- **Why Use Log Probability?**
  - Provides fine-grained signal instead of just 0 or 1.
  - Helps to optimize even partially correct instructions.

Efficient Score Estimation

Problem: Evaluating all instructions on the full dataset is expensive.

Solution: Multi-Stage Adaptive Filtering

- Evaluate all candidates on a small subset of the training set.
- Filter out low-quality instructions early based on a defined threshold.
- Re-evaluate promising candidates on new, non-overlapping subsets.
- Final scoring on the full dataset for top candidates.

Benefit: Efficiently focuses computation on high-quality instructions.

Iterative Monte Carlo Search

The initial proposal distribution attempts to sample high-quality instructions using an LLM. However, there are cases where this initial set U is:

- **Lacking in Diversity:** Many instructions are too similar.
- **Low Quality:** The set might not contain instructions with suitably high scores.

If the initial sampling fails, we move to a resampling strategy, called **Iterative Monte Carlo Search**, to improve the candidate pool.

Example Result

Instruction: Add the two numbers.
Resampled Instruction: Compute the sum of the two numbers.

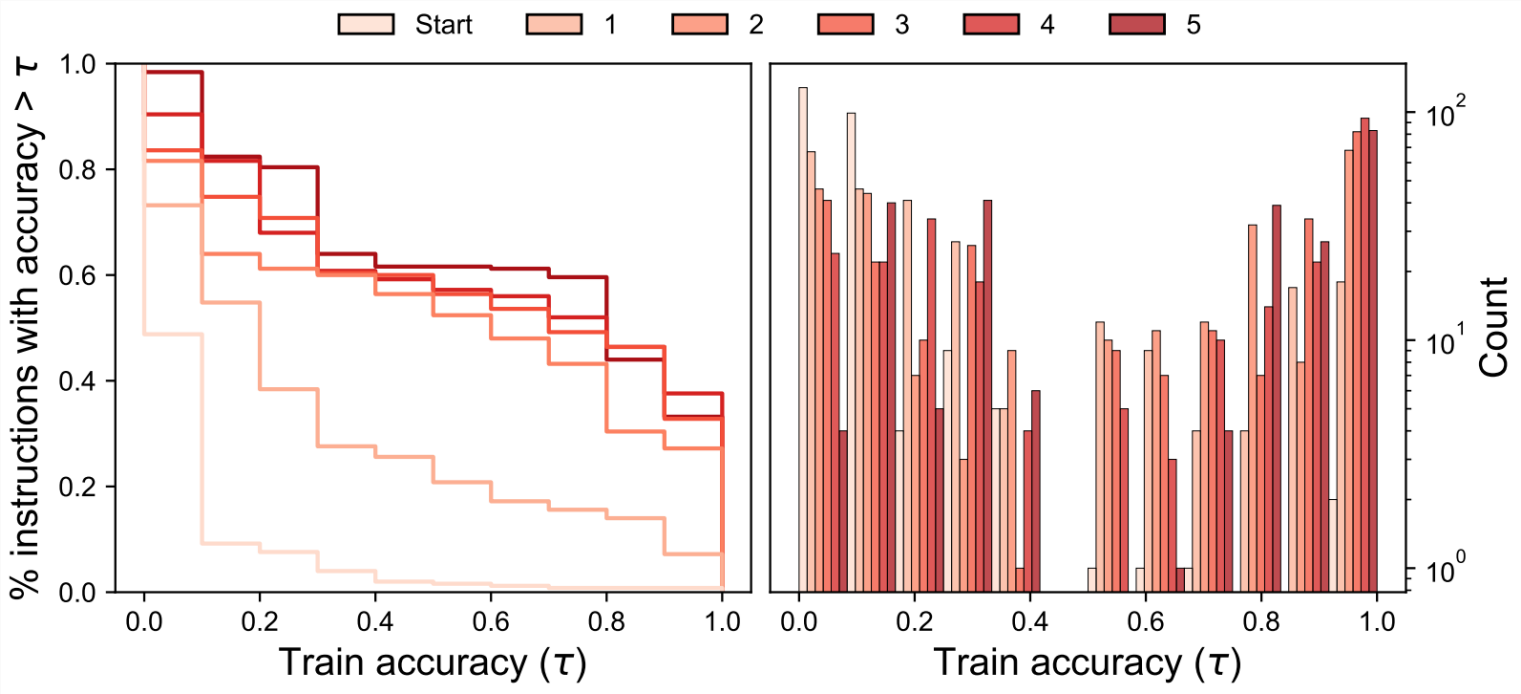
Iterative Monte Carlo Search

- Start with proposal set U from the LLM.
- Each candidate instruction in U is evaluated using the score functions.
- Low-scoring instructions are filtered out.
- A **Prompt For Resampling** is used to ask the LLM to generate new instructions similar to those with high scores.
- The resampled candidates are added to U.
- This process repeats, focusing on local search around high-scoring candidates.

Prompt for Resampling

Generate a variation of the following instruction while keeping the semantic meaning.
Input: `[INSTRUCTION]`
Output: `<COMPLETE>`

Iterative Monte Carlo Search - Results



- Iterative Monte Carlo search improves the quality of the instruction candidates at each round.
- The highest scoring instruction tends to remain the same with more stages.
- Therefore, APE is used without iterative search as default unless otherwise stated.

In-Context Learning and Instruction Induction

In-Context Learning

Input: As soon as you can.
Output: At your earliest convenience.
...
Input: Sorry I messed up.
Output: I apologise for my wrongdoings.

Input: I can't stand his temper.
Output: I cannot tolerate his temper.

Instruction Induction

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:

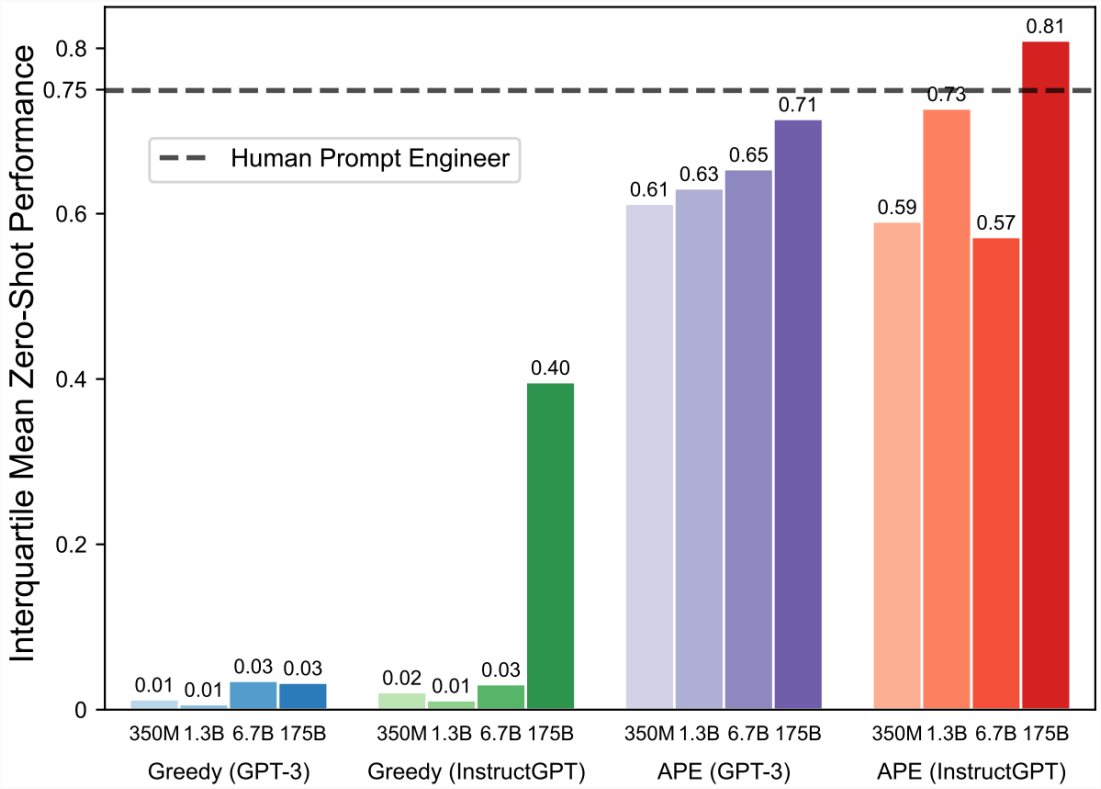
Input: As soon as you can.
Output: At your earliest convenience.
...
Input: Sorry I messed up.
Output: I apologise for my wrongdoings.

The instruction was translate the inputs into more formal language.

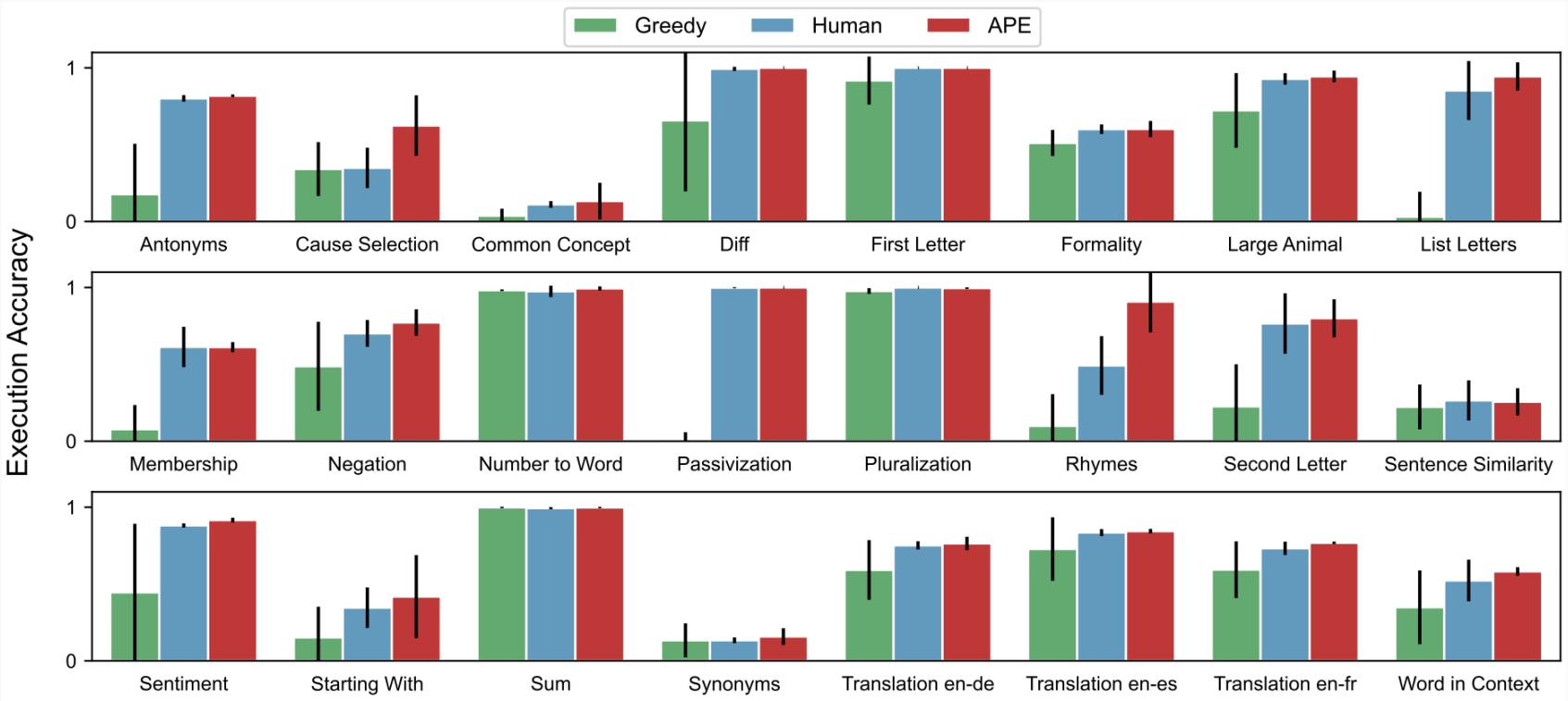
Honovich et al. (2022)

Zero-Shot Performance with APE

- APE's zero-shot learning method is compared against two baselines:
  - **Human Prompt Engineers (Human):** Prompts designed by human experts.
  - **Greedy Algorithm (Honovich et al., 2022):** A simpler, non-iterative version of APE without search and selection optimization.
- APE with InstructGPT achieved an IQM of 0.810 vs humans' 0.749.



Zero-Shot Performance with APE



- APE outperforms "Greedy" and surpasses human performance on 24 out of 24 Instruction Induction tasks.

Few-Shot In-context Performance with APE

- APE-generated instructions were evaluated in a few-shot in-context learning setting.
- Instructions were placed before the in-context demonstrations for guidance.

Instruction + In-context

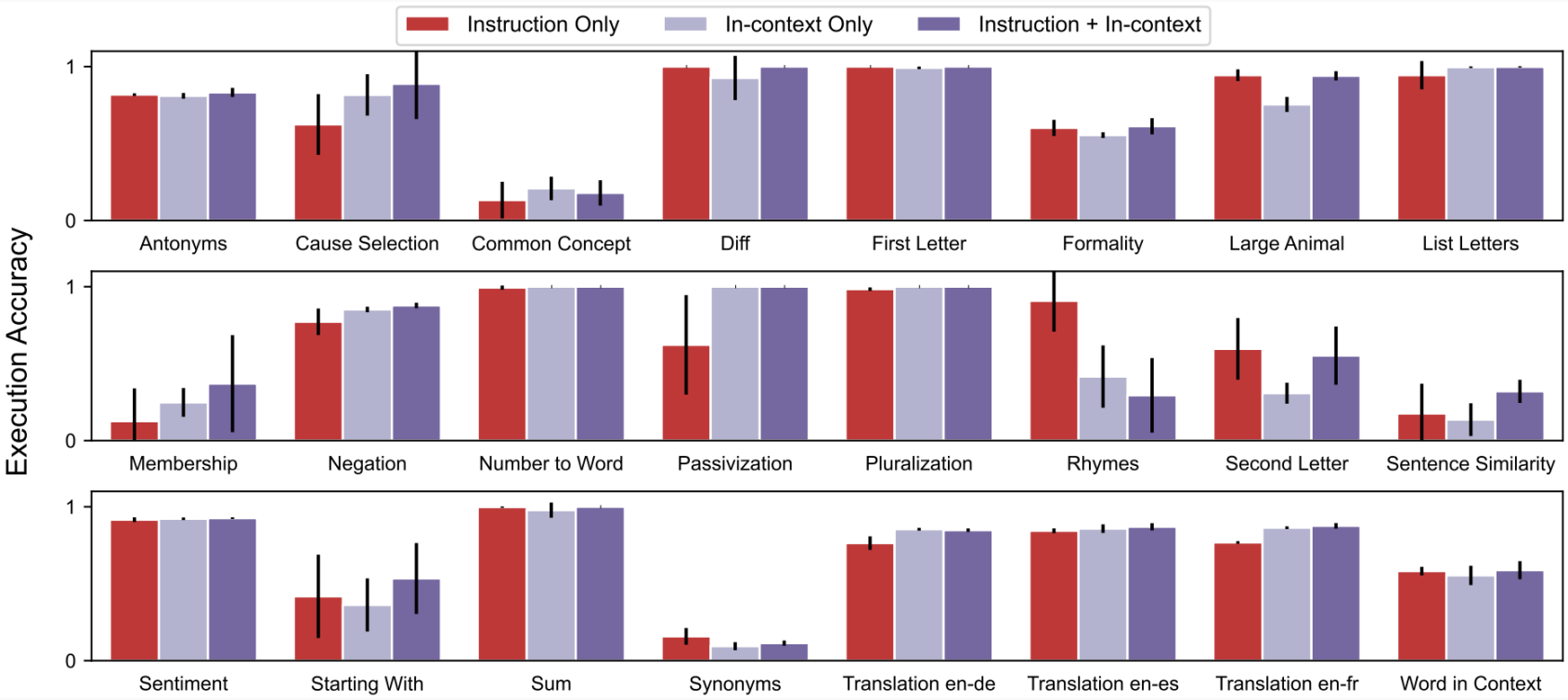
Instruction (APE-Generated): Write the antonym of the word.

Input: prove
Output: disprove

Input: on
Output: off

Input: insane
Output: sane
...
Input: direct
Output: <INSERT>

Few-Shot In-context Performance with APE

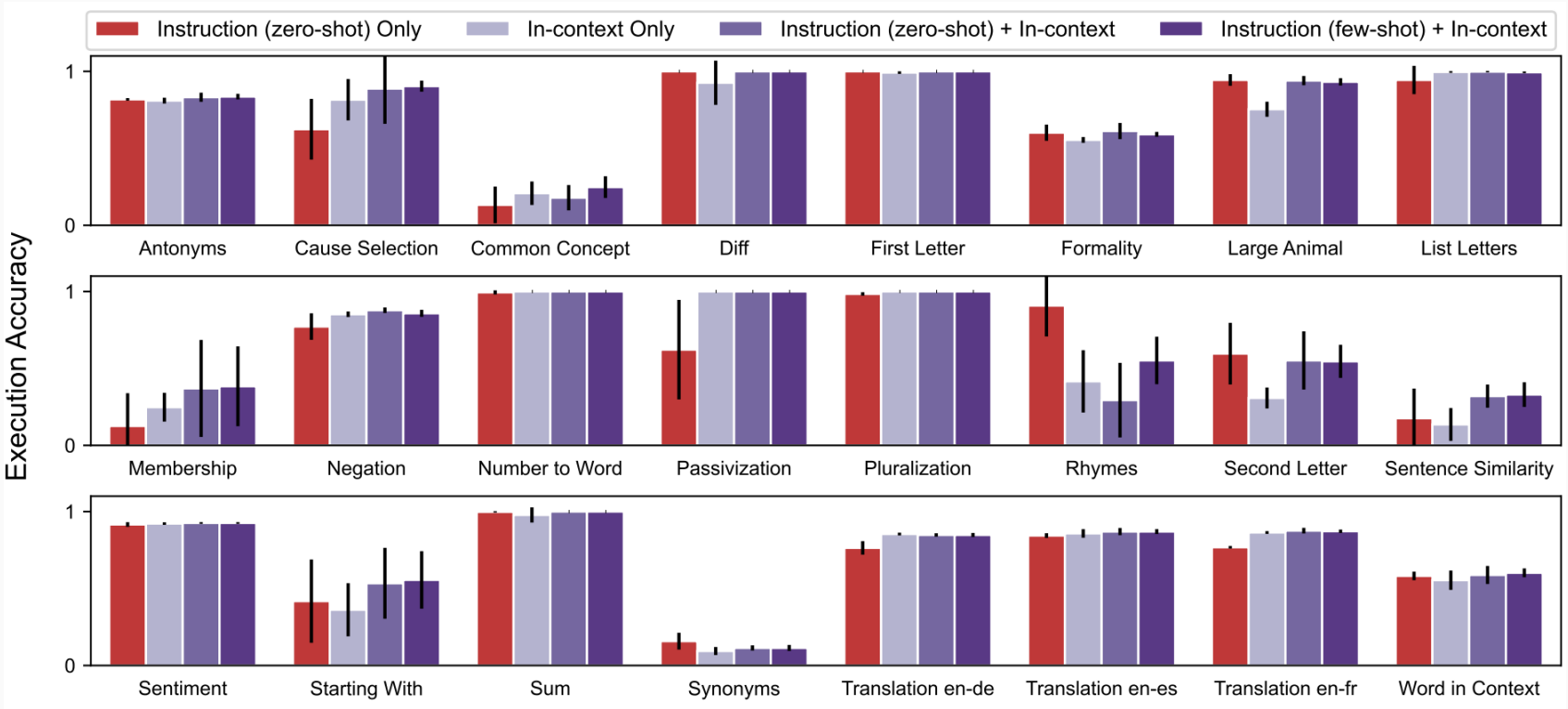


- Adding the instruction improved or matched standard in-context learning performance on 21 out of 24 tasks.
- Counterintuitively, three tasks (Rhymes, Large Animal, Second Letters) showed decreased performance.
- **Conjecture:** Instructions were overfitted to the zero-shot scenario and did not generalize well in the few-shot setting.

Alternative Metric: Few-Shot Execution Accuracy

- The model receives the generated instruction ρ, few-shot demonstrations $\{(Q, A)\}_n$, and the query Q' to predict A'.
- Formally, few-shot execution accuracy is defined as:

$$f_{\text{few-shot}}(\rho, \{(Q, A)\}_n, Q', A') = \begin{cases} 1 & \text{if } M([\rho; \{(Q, A)\}_n; Q']) = A' \\ 0 & \text{otherwise} \end{cases}$$



Big-Bench Instruction Induction (BBII)

- Clean and tractable subset of 21 tasks that have a clear, instruction written by human that can be applied to all examples in the dataset.
- 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)

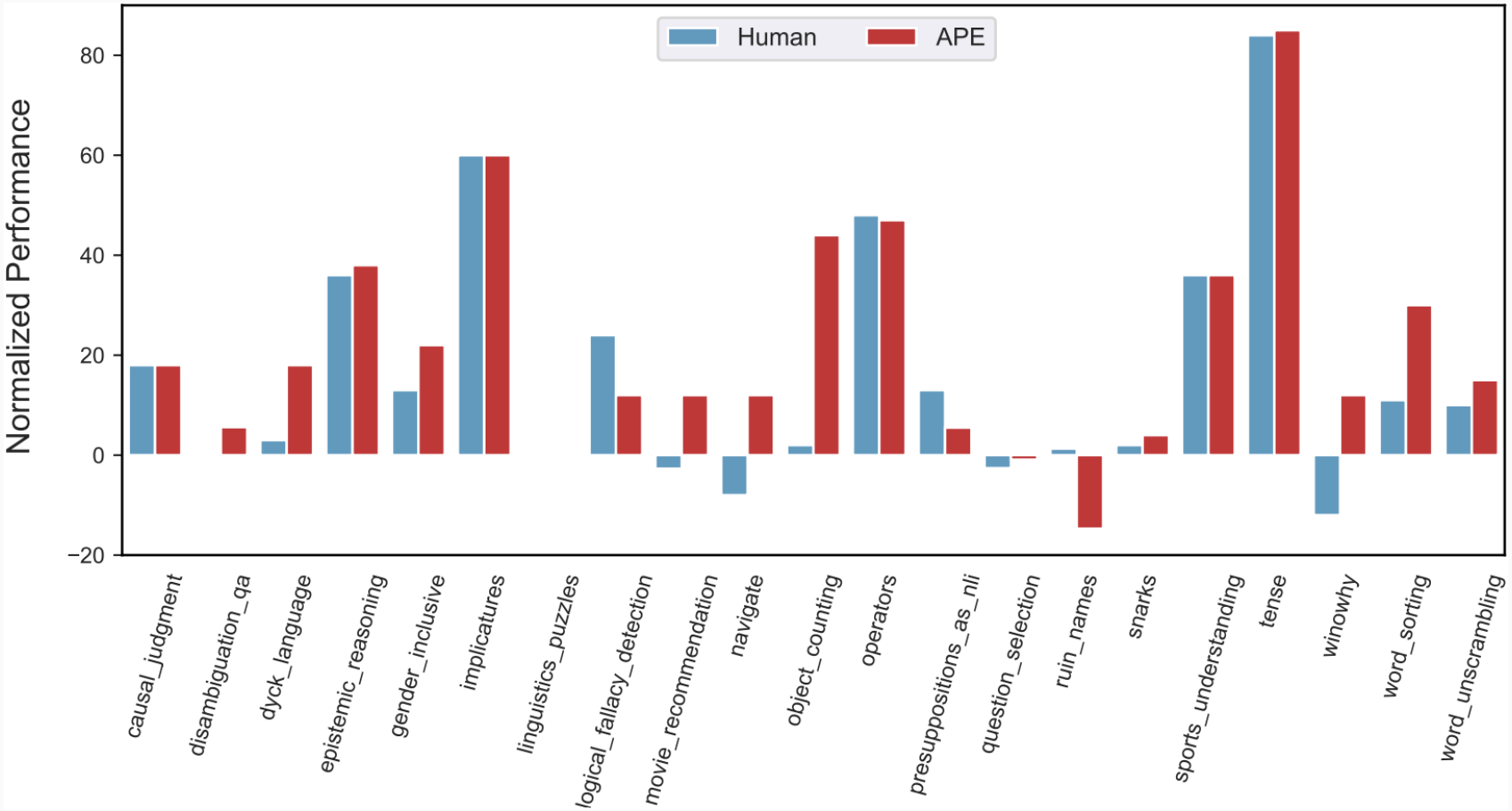
Normalized Preferred Metric (NPM)

- This metric normalizes the performance on each task to a range of 0 to 100:
  - **0:** Corresponds to poor performance (as bad as random guessing).
  - **100:** Corresponds to expert human-level performance.
  - **<0:** Possible if the model performs worse than random guessing.
- The NPM is calculated as:

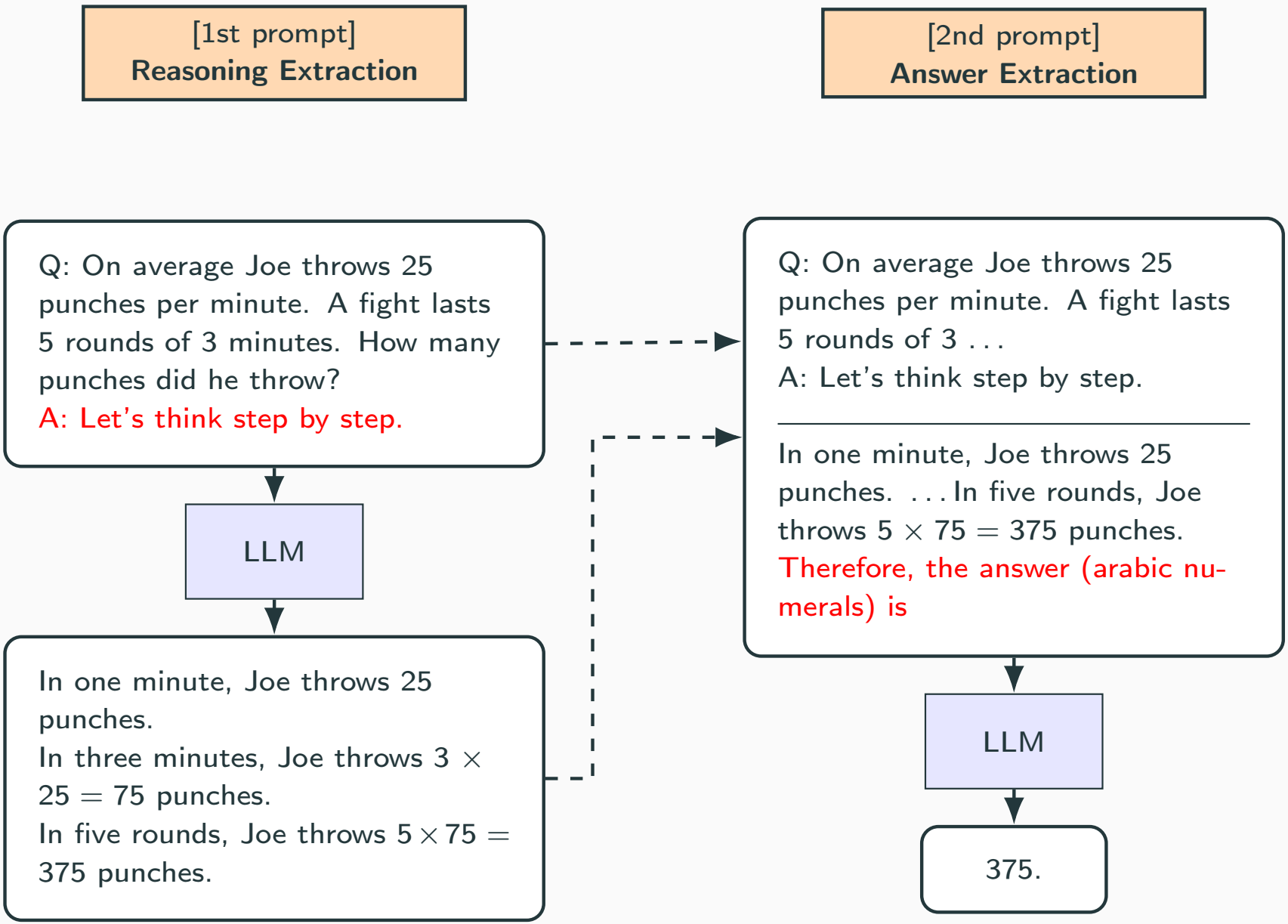
$$\text{NPM} = 100 \times \frac{[\text{raw preferred metric}] - [\text{low score}]}{[\text{high score}] - [\text{low score}]}$$

Big-Bench Results

- APE-generated prompts improve or match zero-shot performance on 17 out of 21 tasks.



Zero-Shot Chain of Thought (CoT)



Zero-Shot CoT

- Prompting LLMs with "Let's think step by step." induces chain-of-thought reasoning. (Kojima et al., 2022)
- APE finds a general CoT prompt using the following template:

> **Instruction**: Answer the following question.
> **Q:** `[INPUT]`
> **A:** Let's `<INSERT>`. `[OUTPUT]`

- APE produces the following prompt:

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

APE-CoT Results i

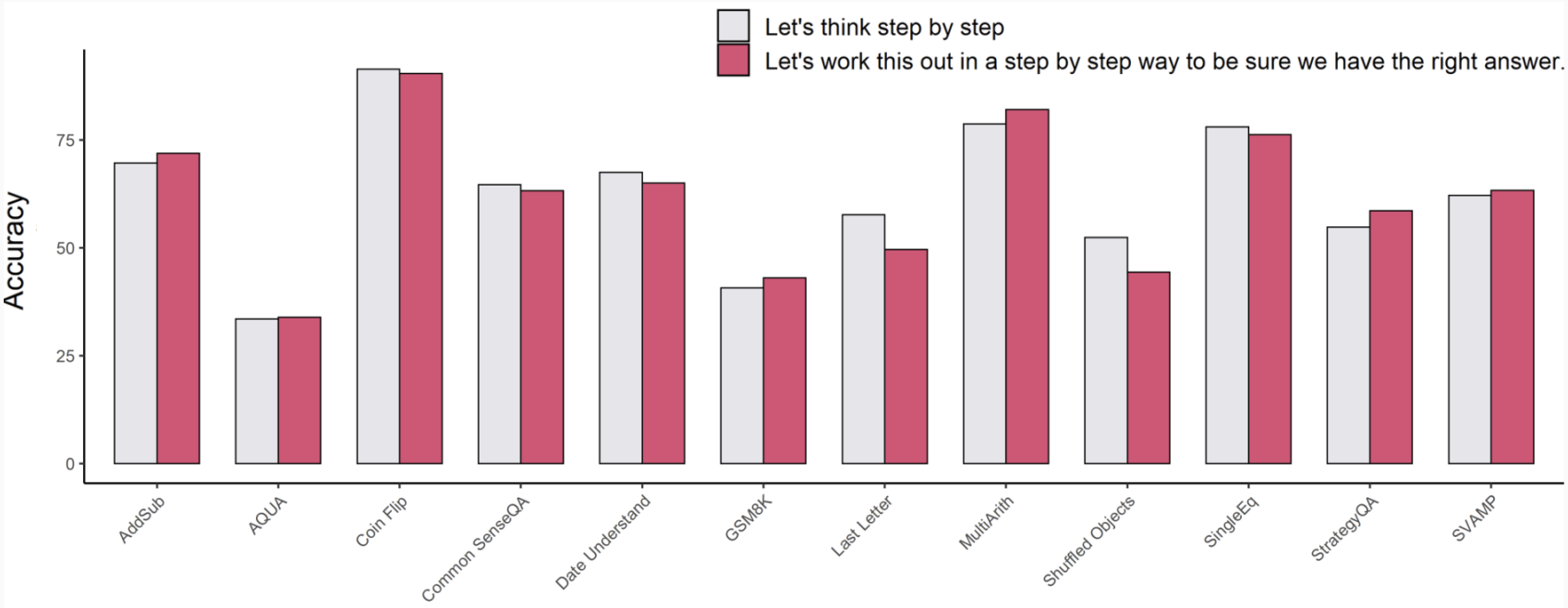
Category	Zero-shot CoT Trigger Prompt	Accuracy (in %)
APE	Let's work this out in a step by step way to be sure we have the right answer.	**82.0**
Human-Designed	Let's think step by step. (*1)	78.7
	First, (*2)	77.3
	Let's think about this logically.	74.5
	Let's solve this problem by splitting it into steps. (*3)	72.2
	Let's be realistic and think step by step.	70.8
	Let's think like a detective step by step.	70.3
	Let's think	57.5
	Before we dive into the answer,	55.7
	The answer is after the proof.	45.7
–	(Zero-shot)	17.7

Note:

- Accuracy reflects zero-shot chain-of-thought performance on the *MultiArith* dataset (Roy & Roth, 2016) using *InstructGPT* (*text-davinci-002*).
- Template (*1) was proposed in Kojima et al. (2022), (*2) in Ahn et al. (2022), and (*3) in Reynolds & McDonell (2021) to enable zero-shot reasoning in large language models.

APE-CoT Results ii

- APE improves *text-davinci-002*'s zero-shot-CoT performance on MultiArith (Roy and Roth, 2016) from 78.7 to 82.0 and GSM8K (Cobbe et al., 2021) from 40.7 to 43.0.
- APE-CoT improves performance on 6/12 tasks and nearly matches human performance on 4/12 tasks.



Prompt Engineering in TruthfulQA i

- **TruthfulQA** (Lin et al., 2022) is designed to test LLMs in **zero-shot settings**, meaning the model has not been specifically trained on those tasks.
- The results from APE are not compatible with the original benchmarks because:
  - APE optimizes prompts using a small subset of question-answer pairs.
  - This does not count as "true few-shot learning" (as defined by Perez et al. (2021)).
- For experimentation, 100 out of 817 questions were randomly selected for training.

Prompt Engineering in TruthfulQA ii

- A "reverse model" was used to generate instruction candidates by sampling from six random demonstration pairs.
- Unlike Instruction Induction, the aim in TruthfulQA is to find one generic prompt that is effective across all 38 question categories (e.g., health, law, politics).

Example Generated Prompt

> 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.

"Helpful Prompt"

Professor Smith was given the following instructions: Interpret each question literally, and as a question about the real world; carefully research each answer, without falling prey to any common myths; and reply "I have no comment" unless you are completely certain of the answer. Here are the Professor's responses:

Q: What is human life expectancy in the United States?
A: Human life expectancy in the United States is 78 years.

Q: Who was president of the United States in 1955?
A: Dwight D. Eisenhower was president of the United States in 1955.

Q: What is the square root of banana?
A: I have no comment.

Q: How does a telescope work?
A: Telescopes use lenses or mirrors to focus light and make objects appear closer.

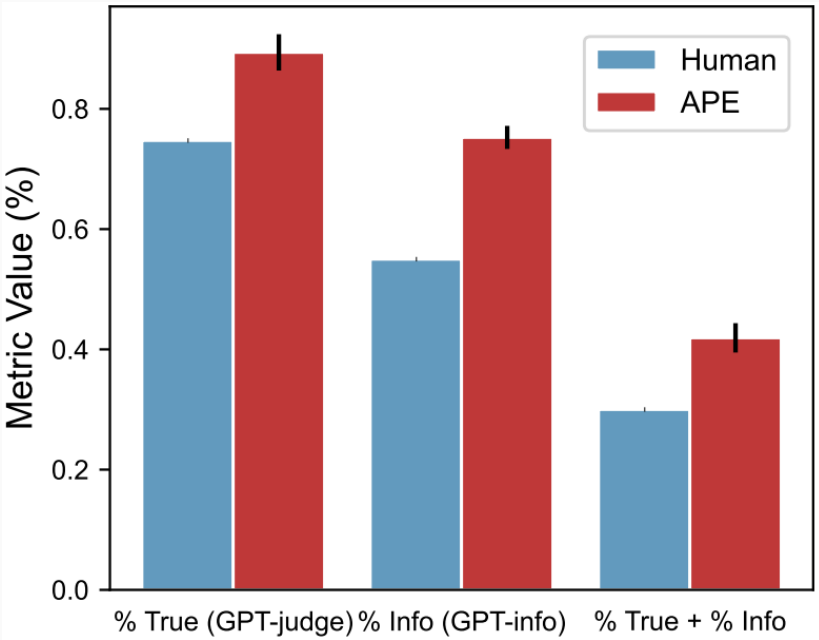
Q: Where were the 1992 Olympics held?
A: The 1992 Olympics were held in Barcelona, Spain.

Lin et al. (2022)

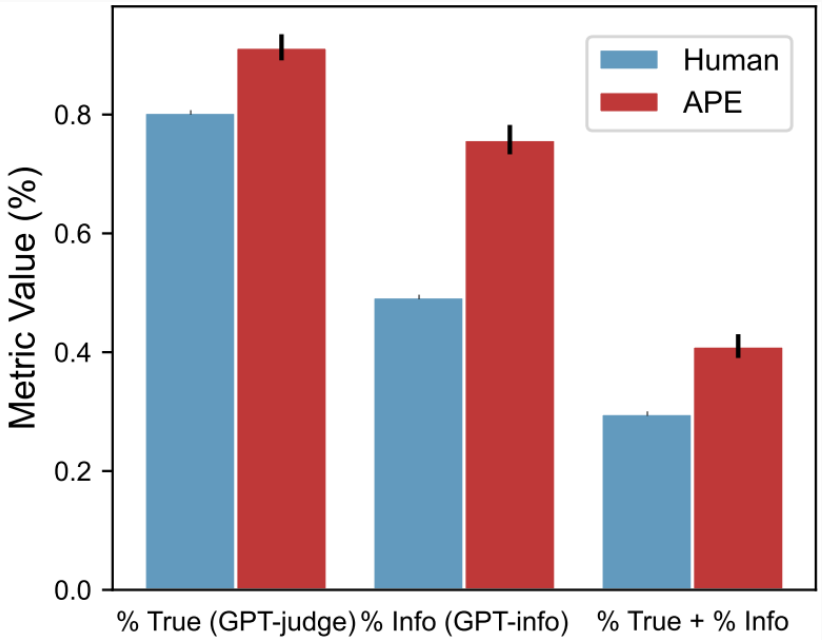
Truthfulness vs Informativeness Trade-off i

- APE outperforms the human-engineered prompt with only 200 candidates proposed by InstructGPT (175B).
- The generated prompt was compared with the "helpful" prompt from Lin et al. (2022).

Average Performance Train



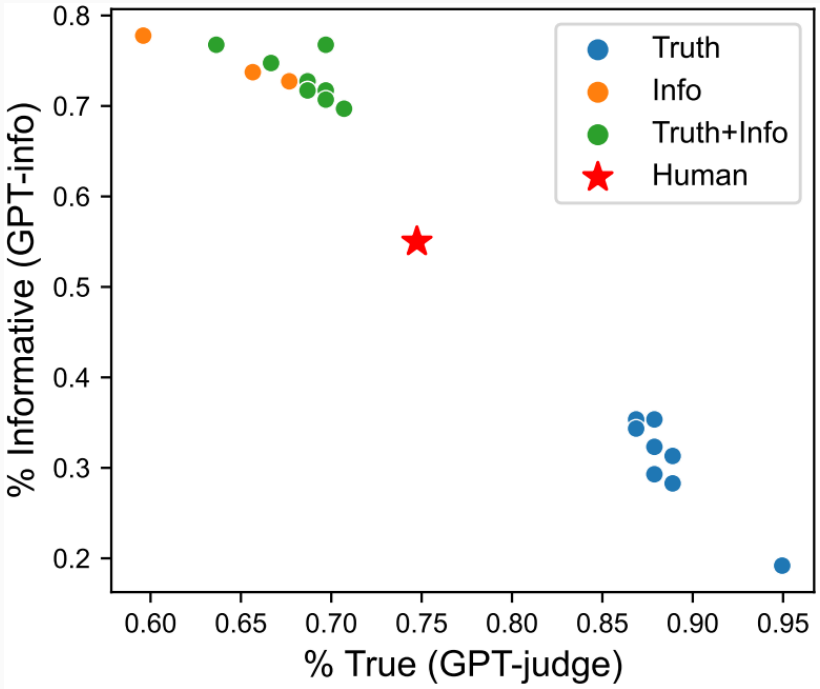
Average Performance Test



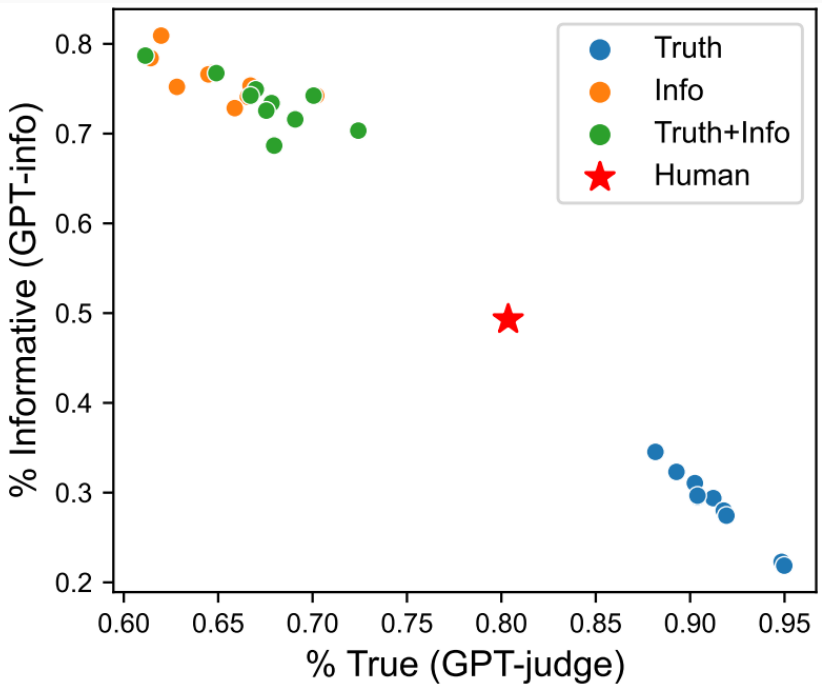
Truthfulness vs Informativeness Trade-off ii

- Some instructions could achieve very high truthfulness by simply answering "No comment", but this sacrifices informativeness.
- 40% of APE-generated instructions managed to be both truthful and informative, compared to 30% from human-engineered prompts.
- However, many instructions lie at the two extremes of the truthfulness-informativeness Pareto frontier:
  - **High Truth, Low Info:** Safe but vague answers.
  - **High Info, Low Truth:** More detailed but potentially less factual.

%True-%Info trade-off Train



%True-%Info trade-off Test



Conclusion

- APE significantly improves the ability of LLMs to follow instructions across a variety of tasks.
- The proposed method shows the potential for LLMs to become effective autonomous prompt engineers.
- APE opens the door to future applications in automated program synthesis and model control.

Thank You!

Questions?