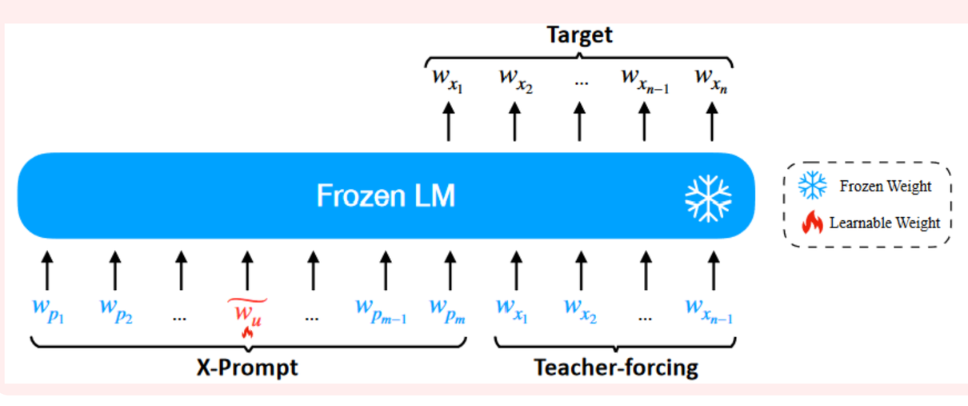
ICL+Prompt

Prompt Tuning/Prompt Engineering

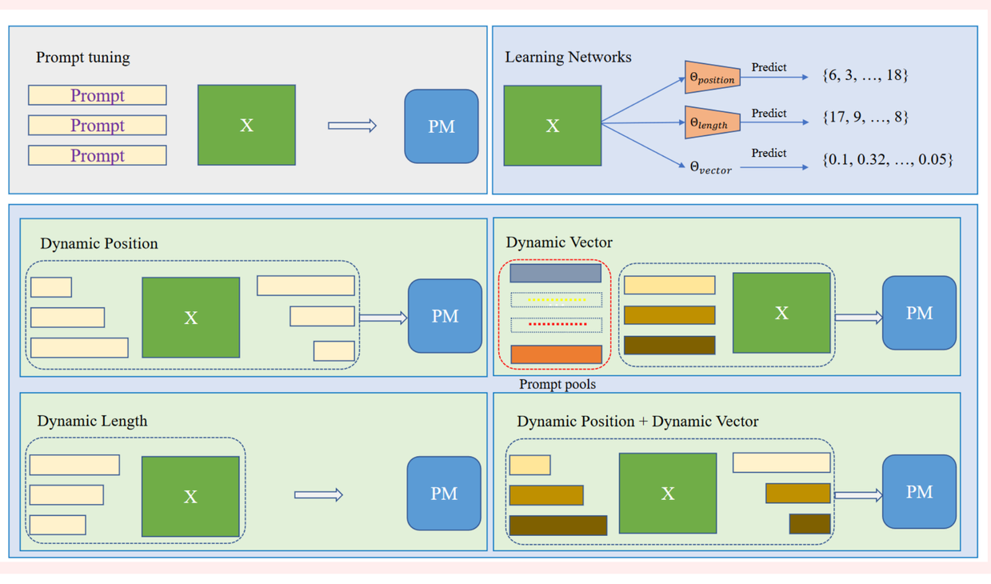
Paper: Extensible Prompts for Language Models on Zero-shot Language Style Customization

X-Prompt主要的思想是将**imaginary words**注入到natural language中构成prompt，其中只有**imaginary words**是learnable。文章还提出了Context-Augmented Learning(CAL)的概念帮助更好地学习imaginary words(增强在OOD上的表现)。采用的task类型为open-ended text generation和style transfer



Paper: Dynamic Prompting: A Unified Framework for Prompt Tuning

Dynamic Prompting主要思想是根据不同的task/instance的情况来调整相对应prompt的position、length、representation。



Paper: When Do Prompting and Prefix-Tuning Work? A Theory of Capabilities and Limitations(粗读)

这篇paper比较理论，公式也比较多，所以没怎么看，这里只记下结论。

1. content-based fine-tuning只能激发LLM原有的skill而不能学一个全新的task
2. a hierarchy: prompting<soft prompting<prefix-tuning, prefix-tuning的效果最好，prompting和in-context learning是prefix-tuning的特例
3. 对于Transformer而言，变化一个virtual token比变化一个hard token，会产生更多的completion
4. prefix-tuning only adds a bias to the attention block output，这个bias能激发出LLM原有的skill
5. 参数相同的情况下，LoRA可以学会全新的task，prefix-tuning不可以

利用ICL优化Prompt

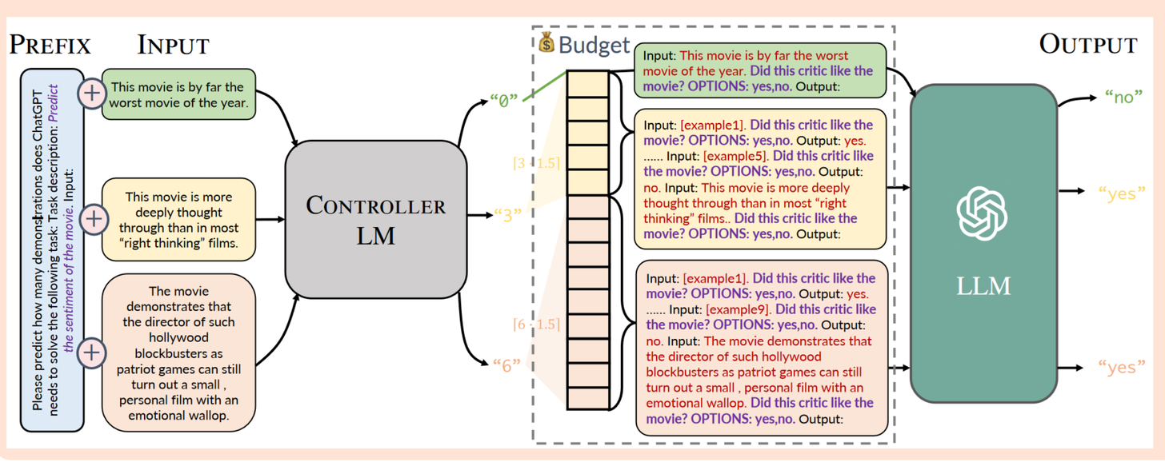
Paper: How Does In-Context Learning Help Prompt Tuning

这篇paper主要是对比了PT(Prompt Tuning)、ICL(In-Context Learning)、IPT(Instruction Prompt Tuning)三种方法的效果来探究ICL对prompt tuning的影响。task类型选用的是language generation task(data-to-text generation、logic-to-text generation、semantic parsing)

得到的一些结论：1.ICL表现得比PT差 2.PT与IPT表现难分伯仲(取决于task类型/tunable parameter数目等) 3.当demonstration跟test input类似时，IPT可以很好work(别的论文有结论：in-context examples主要是帮助model学习output label space和distribution of input text) 4.IPT在有更多soft prompt tokens时表现得比PT更稳定 5.在有in-context demonstration的情况下，prompt embeddings对于新的task是transferable

Paper: Efficient Prompting via Dynamic In-Context Learning

DYNAICL(Dynamic In-Context Learning)的提出是为了有效解决performance-efficiency trade-off问题，因为in-context learning使用的demonstration数目会影响prompt长度，而prompt长度过大会导致低效，因此动态分配demonstration的数目可以有效解决此问题。论文的核心是train一个meta controller可以动态分配in-context demonstration的数目，train分为两个阶段：1.使得最后用的generalist model生成"good output"的同时利用最少的demonstrations 2.利用RL来微调



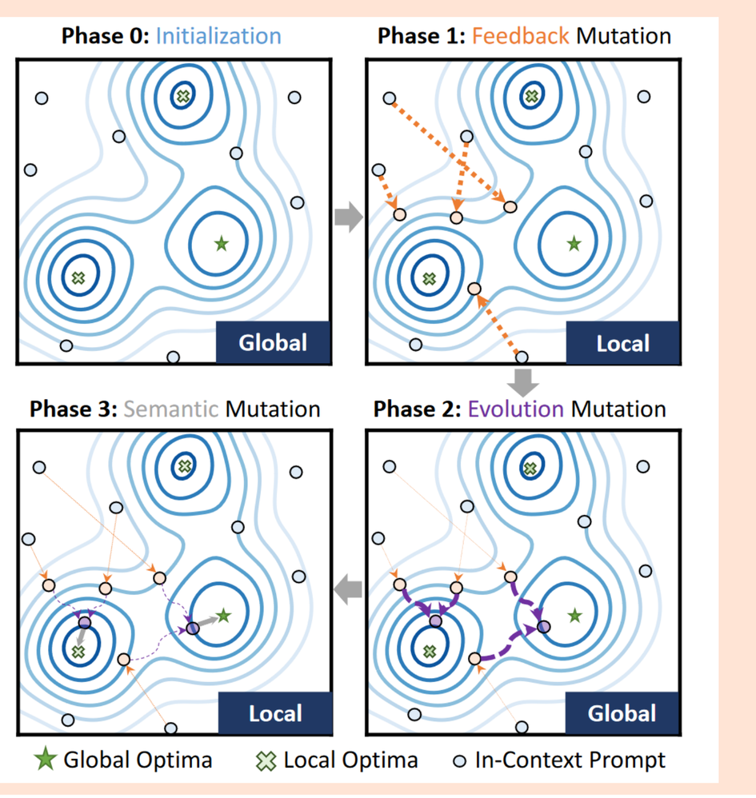
Paper: PhaseEvo: Towards Unified In-Context Prompt Optimization for Large Language Models(粗读)

Submitted on 17 Feb 2024

这篇paper主要提出了一个统一的in-context prompt优化的框架PhaseEvo，核心算法是进化算法(Evolutionary Algorithms)，采用了Exploration、Exploitation两种不同的优化策略。整个框架分为四个阶段：

1. global initialization
2. local feedback mutation
3. global evolution mutation
4. local semantic mutation

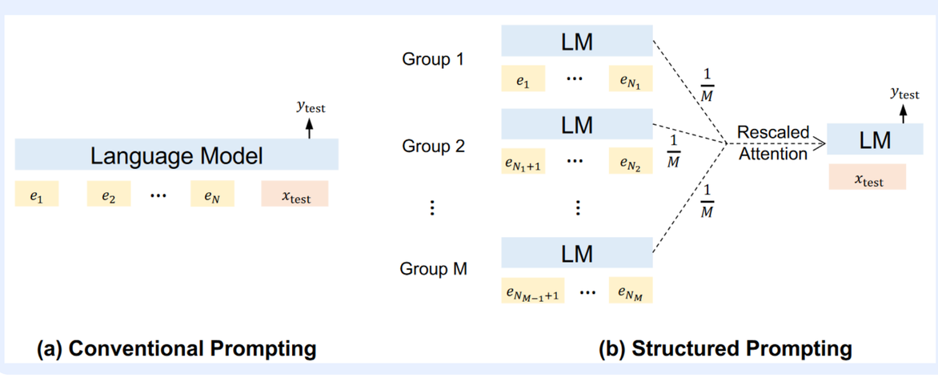
local通常是为了加速收敛/逼近局部最优，global是为了防止stuck in局部最优



In Context Learning

Paper: Structured Prompting: Scaling In-Context Learning to 1,000 Examples

该paper主要目的是解决传统Prompting增加demonstrations数目难度比较大的问题，实现了使用较多数目的demonstrations进行train。**Structured Prompting**主要思路是把所有demonstrations进行分组，然后不同group进行独立encode，之后喂进Rescaled Attention对attention score进行正则化，最终连同test input喂进LM中。



Paper: Pre-Training to Learn in Context

该paper提出了Pre-training for In-Context Learning(PICL)，基于Corpus中很多paragraphs都包含intrinsic tasks的假设，用Retriever将具有相同类型intrinsic task的paragraph检索出来，并连接在一起构建一个meta-training dataset，用contrastive learning的方法train一个Encoder使得具有相同类型intrinsic task的paragraph具有类似的Embedding

Paper: Active Example Selection for In-Context Learning(粗读)

这篇paper主要提出了为In-Context Learning挑选example的方法。方法提出的背景是Reorder/Calibration都不能很好解决instability/variance的问题，基于此，作者将example selection视为sequential decision问题。整个过程类似于active learning，作者将active example selection视作MDP，然后采用Q-learning算法解决

LLM as Prompt Optimizers

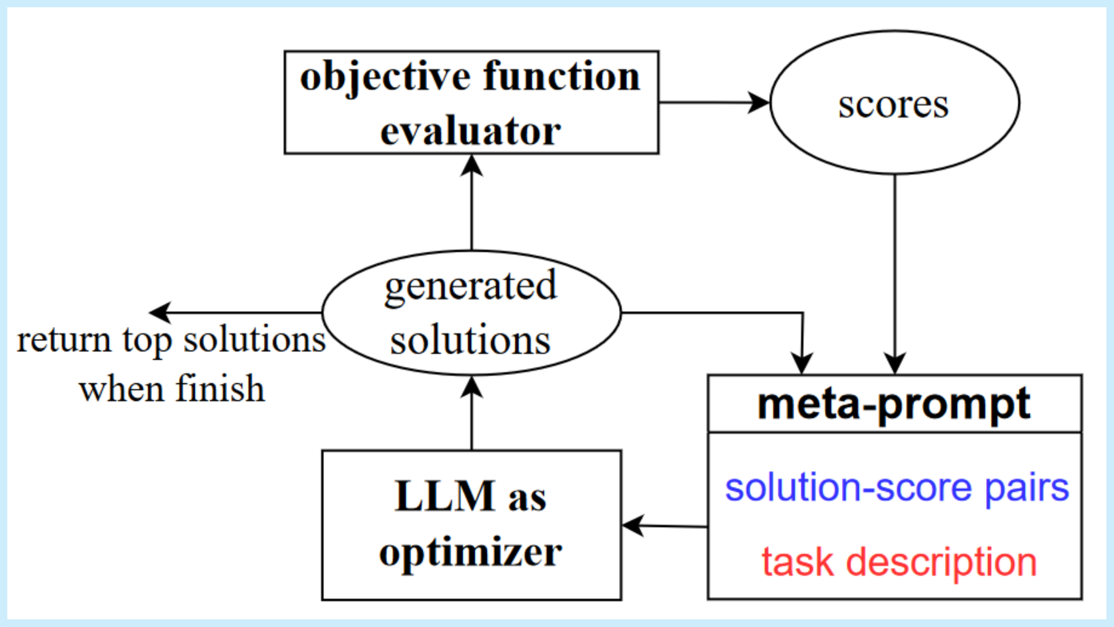
Paper: Large Language Models as Optimizers

这篇文章提出**OPRO**，也就是用LLM作为优化器来生成使得task accuracy最高的hard prompt(自然语言形式)。整个工作是通过调用model api实现的，适用于闭源大模型，流程图如下图所示：

一个LLM负责优化，称为optimizer LLM；另一个LLM负责评估，称为scorer LLM。optimizer LLM的输出称为an instruction，用于连接到每个exemplar的question part然后prompt scorer LLM，可以插入的位置是Q_begin、Q_end、A_begin。

最后文章探究了**Meta-prompt design**(previous instructions的顺序、instruction score的表示方式、exemplar的数量)、**generated instructions的数量**、**starting point**、**每步的diversity**(LLM temperature设置)等对实际效果的影响。

Future Work:
1. exemplar的选取要充分利用ICL
2. weaker starting point的收敛速度优化



Paper: Automatic Prompt Optimization with "Gradient Descent" and Beam Search

这篇paper提出了APO算法(Prompt Optimization with Textual Gradients)，主要思想就是用LLM feedback代替differentiation、用LLM editing代替backpropagation，以此来类比传统的gradient descent。整体过程如右图1所示。

算法过程如右图2所示，每次迭代都会产生新的candidate prompts(**expansion**)，然后进行**selection**。expansion算法过程如右图3所示，selection包括UCB Bandits、Successive Rejects两种方法，分别如右图4、5所示。

图1

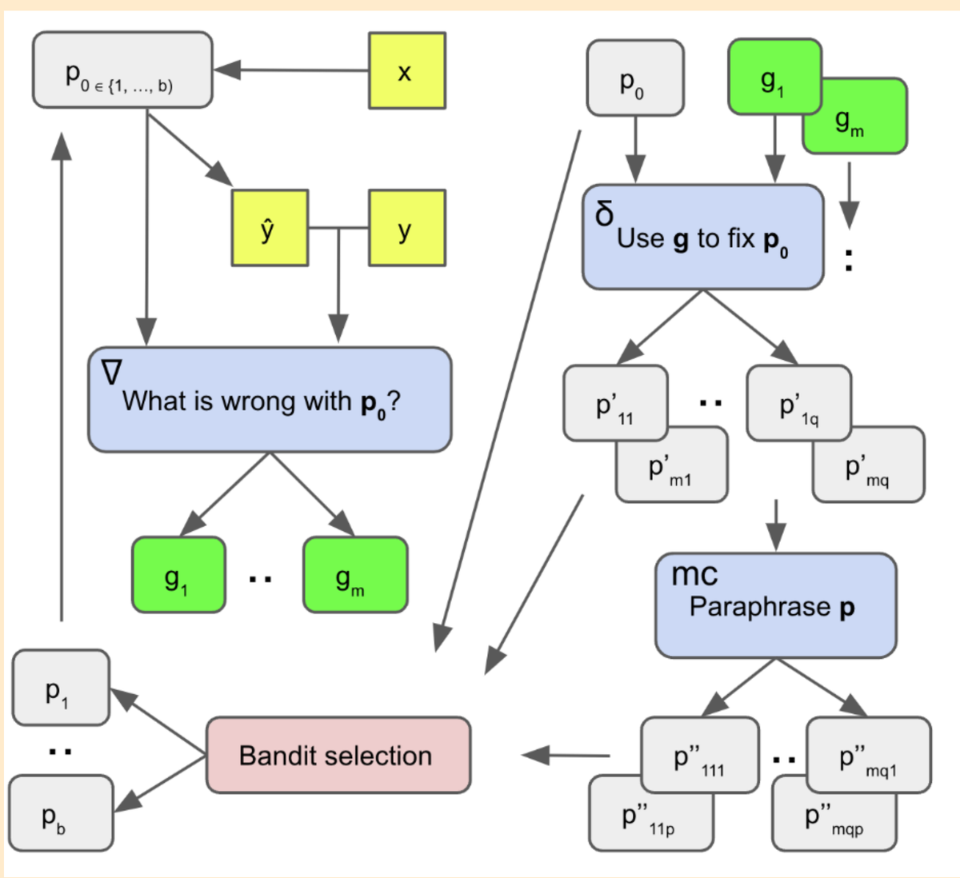


图2

Algorithm 1 Prompt Optimization with Textual Gradients (ProTeGi)

Require: p_0: initial prompt, zb: beam width, r: search depth, m: metric function
1: $B_0 \leftarrow \{p_0\}$
2: **for** $i \leftarrow 1$ to $r-1$ **do**
3: $C \leftarrow \emptyset$
4: **for all** $p \in B_i$ **do**
5: $C \leftarrow C \cup Expand(p)$
6: **end for**
7: $B_{i+1} \leftarrow Select_b(C, m)$
8: **end for**
9: $\hat{p} \leftarrow argmax_{p \in B_r} m(s)$
10: **return** $\hat{p}$

图3

Algorithm 2 $Expand(\cdot)$ - line 5 of Algorithm 1

Require: p: prompt candidate, $\mathcal{D}_{tr}$: train data
1: Sample minibatch $\mathcal{D}_{mini} \subset \mathcal{D}_{tr}$
2: Evaluate prompt p on minibatch $\mathcal{D}_{mini}$ and collect errors $e = \{(x_i, y_i) : (x_i, y_i) \in \mathcal{D}_{mini} \wedge LLM_p(x_i) \neq y_i\}$
3: Get gradients: $\{g_1, ..., g_m\} = LLM_\nabla(p, e)$
4: Use the gradients to edit the current prompt: $\{p'_{i1}, ..., p'_{iq}\} = LLM_\delta(p, g_i, e)$
5: Get more monte-carlo successors: $\{p''_{ij1}, ..., p''_{ijm}\} = LLM_{mc}(p'_{ij})$
6: **return** $\{p'_{11}, ..., p'_{mq}\} \cup \{p''_{111}, ..., p''_{mqp}\}$

图4

Algorithm 3 $Select(\cdot)$ with UCB Bandits - line 7 of Algorithm 1

Require: n prompts $p_1, ..., p_n$, dataset $\mathcal{D}_{tr}$, T time steps, metric function m
1: Initialize: $N_t(p_i) \leftarrow 0$ for all $i = 1, \ldots, n$
2: Initialize: $Q_t(p_i) \leftarrow 0$ for all $i = 1, \ldots, n$
3: **for** $t = 1, \ldots, T$ **do**
4: Sample uniformly $\mathcal{D}_{sample} \subset \mathcal{D}_{tr}$
5: $p_i \leftarrow \begin{cases} \arg\max_p \left\{Q_t(p) + c\sqrt{\frac{\log t}{N_t(p)}}\right\} & \text{(UCB)} \\ \arg\max_p \left\{Q_t(p) + c\sqrt{\frac{c}{N_t(p)}}\right\} & \text{(UCB E)} \end{cases}$
6: Observe reward $r_{i,t} = m(p_i, \mathcal{D}_{sample})$
7: $N_t(p_i) \leftarrow N_t(p_i) + |\mathcal{D}_{sample}|$
8: $Q_t(p_i) \leftarrow Q_t(p_i) + \frac{r_{i,t}}{N_t(p_i)}$
9: **end for**
10: **return** $SelectTop_b(Q_T)$

图5

Algorithm 4 $Select(\cdot)$ with Successive Rejects - line 7 of Algorithm 1

Require: n prompts $p_1, ..., p_n$, dataset $\mathcal{D}_{tr}$, metric function m
1: Initialize: $S_0 \leftarrow \{p_1, \ldots, p_n\}$
2: **for** $k = 1, \ldots, n-1$ **do**
3: Sample $\mathcal{D}_{sample} \subset \mathcal{D}_{tr}$, $|\mathcal{D}_{sample}| = n_k$
4: Evaluate $p_i \in S_{k-1}$ with $m(p_i, \mathcal{D}_{sample})$
5: $S_k \leftarrow S_{k-1}$, excluding the prompt with the lowest score from the previous step
6: **end for**
7: **return** Best prompt $p^* \in S_{n-1}$

Paper: Large Language Models Are Human-Level Prompt Engineers

这篇paper提出**APE(Automatic Prompt Engineering)和Iterative APE**，整个工作流程如右图1所示，算法过程如右图2所示。

LLM承担的角色是proposal(生成a set of instruction candidates)、scoring(为生成的instruction打分)，对于proposal文中提出了Forward Mode Generation和Reverse Mode Generation两种方式，对于scoring文中提出了execution accuracy、log probability两种形式。

最后这篇paper还提出了Iterative Monte Carlo Search(Iterative APE)，在当前最优的instruction candidate附近保持语义进行search(而不是在原有的proposal中search)，一个template如右图3。

实验部分主要是在不同的数据集上测试了**zero-shot learning、few-shot in-context learning、zero-shot chain-of-thought reasoning、truthfulness**。

图1

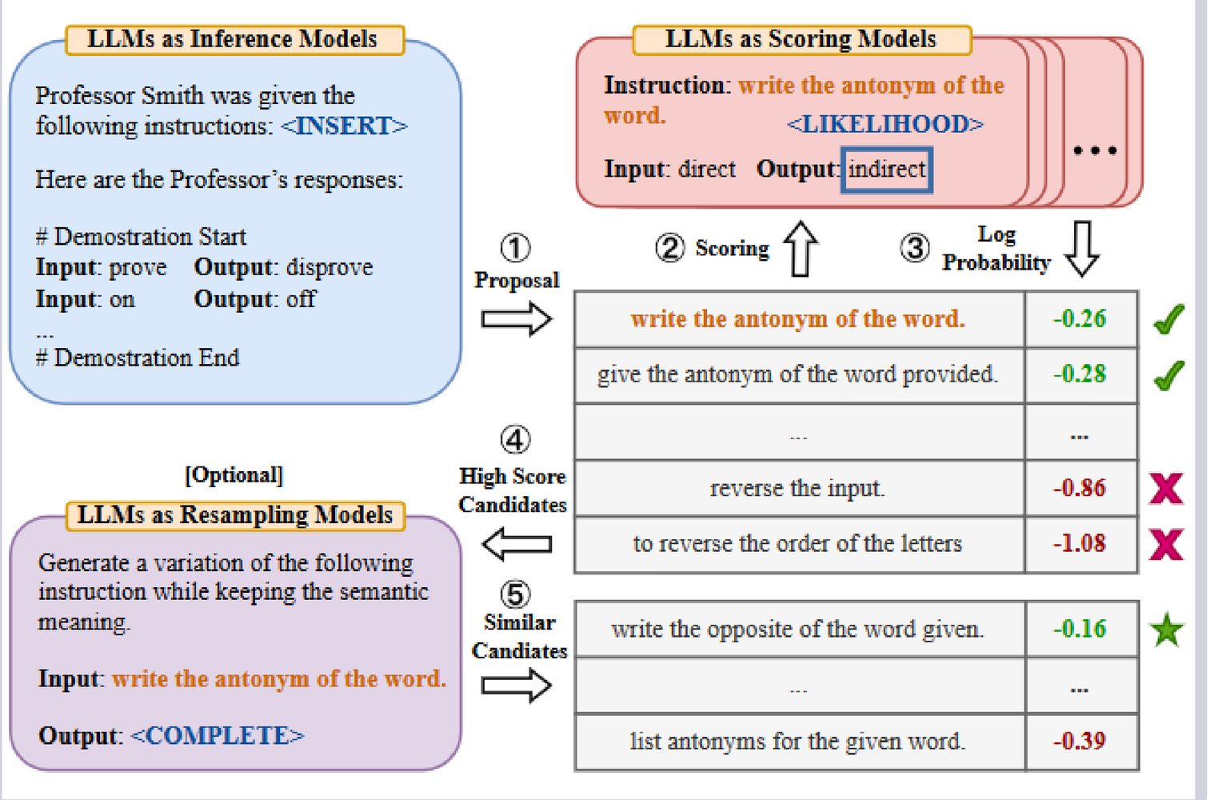


图2

Algorithm 1 Automatic Prompt Engineer (APE)

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

图3

Prompt for Resampling

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

Input: [INSTRUCTION]

Output: <COMPLETE>