Towards Automatic Prompt Optimization: Methods and Experimental Evaluation

ZhiCheng Wang wangzhicheng@mail.ustc.edu.cn

Abstract

大语言模型作为通用智能体，在执行自然语言指令完成特定任务上取得了令人印象深刻的表现。然而，模型在特定任务上的能力在很大程度上受到指导模型完成任务的提示词的影响。人工构造的提示词往往能够取得较为不错的表现，但构造过程需要对任务的深刻理解与大量的试错，带来时间与人力上的巨大开销。为了自动化这一过程，一些方法基于梯度下降等传统优化算法，在连续特征向量空间进行提示词优化，但存在可解释性差、依赖底层模型接口等问题。受到人类在构造提示词时“测试-反思-优化”的迭代过程的启发，我们改进了基于梯度的优化算法。我们的方法使用大语言模型作为优化器，以对错例的错因反思类比梯度计算过程，以基于错因的提示词转写类比反向传播过程，实现了对目标模型上的任务提示词的迭代优化。我们将这一方法以及其与搜索策略相结合的变体应用到数学推理任务，在 GSM8K 数据集上实现了显著的性能提升。此外，我们还验证了这一方法有助于为少样本的上下文学习带来持续的性能提升。实验代码仓库在 https://github.com/Starrybook/Automatic_Prompt_Optimization 。

Introduction

在大模型得广泛应用之后，为了激发模型在解决特定问题时的潜力，提示词学习（Prompt learning）越来越重要，适当的提示词能够显著提高模型在特定问题上的表现。在 CoT 开创性的研究[1]中展示了在零样本情境下短小的附加语句就可以激发大语言模型的数学推理能力。在提示词中加入少量样本输入输出样本也被观察到能够显著提高模型解决特定问题的能力。一些研究[2]发现在提示词中加入积极思考（Positive thinking）的语句也能够显著提高模型的能力，但其发挥作用的原因是难以直观解释的。这揭示了提示词学习问题的复杂性。

人工为特定任务构建合适的提示词需要对任务的深刻理解与大量的试错过程，这在现实中往往产生较大的时间与人力开销。另外，即使是人工完成的启发式提示词构造，也无法在庞大的语义空间中总是取得最优的结果，在研究[2]中观察到通过自动方式构建出的更优提示词可能包含人工构造过程中难以想到的内容。因此，为了降低提示词构建的开销，并提高提示词优化的效果，有必要通过自动化的方式来替代这一工作。

一些自动提示词优化的研究受到梯度下降方法的启发，在嵌入向量连续取值空间的层面对输入进行优化，利用梯度信息反向传播更新输入的各嵌入向量取值，但这一方法具有操作复杂、对编码器依赖大、可解释性差等缺点。为了解决如上问题，我们考虑重新定义梯度，并引入语义上的可解释性，从而得到了基于反思（Reflection-based）的提示词优化方法。

我们使用对随机采样的错例的原因反思来代替“梯度计算”的过程，利用反思原因对原提示词进行改写来代替“反向传播”过程。两个步骤均使用同一个作为优化器的大语言模型实现。通过这种方式，我们可以仅依靠目标模型的 API 接口实现针对目标模型的提示词优化，达到使用上的简便，并使得全过程具备更好的可解释性，这相对于依赖梯度计算的方法更有优势。在这一工作中，我们以数学推理任务为例，展现了基于反思的优化方法十分显著的优化效果，并通过与传统搜索算法的结合实现了更为高效的优化过程。

实验过程中，我们观察到，其它提示词学习方法比如 CoT 与 Positive thinking 的引入相对于初始的提示词能够产生较大的提升。然而，人工构造示例与提示词并进行不断试错的方式比较耗费人力与时间，在已经产生的优化效果的基础上很难进一步持续优化。我们以单样本的 CoT 为例，将包含一个有分步解释的示例的提示词作为初始提示指令，用基于反思的提示词优化算法对其进行持续优化，并证明可以达到优化效果。从而表明了基于反思的优化方法除了自身能够随迭代过程持续优化之外，也能够对其它提示词学习方法带来增益，从而解决了这些方法难以动态持续优化的问题。

Contribution

1. 梳理了自动提示词优化方法的动机、思路与不同的实现方式。
2. 验证了自动提示词优化的可行性，并讨论了其优势与劣势。
3. 以与单样本的 CoT 方法的对比为例，探讨了自动提示词优化的优势，以及与其它方法进行结合的可能性。

基于以上的经验分析与讨论，我们为提示词设计提供了一些建议：

1. 基于反思机制的提示词优化有助于寻找到更加适合任务的提示词，并能够起到持续优化的作用。
2. 提示词优化算法能够与其它利用上下文学习的方法相结合，在已有方法的基础上进一步优化，带来性能的持续增益。并且尤其适用于人工构造的任务初始提示词效果不佳的情形。

Methodology

Task abstraction

对一个由语言L中的文本输入输出对组成的数据集$D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$，将其划分为训练集$D_{train}$与测试集$D_{test}$，我们具有提示词$p$与一个目标黑盒模型$M$，将提示词与文本输入$x_i$连接后通过目标黑盒模型 API 得到文本回复$\hat{y}_i$，这里$\hat{y}_i = argmax_{y \in L} P_M(y|p, x_i)$。我们具有评测指标$E(y_i, \hat{y}_i)$对预测结果进行打分，从而确知目标黑盒模型的预测效果。我们的目标是寻找到最优的提示词$p^* \in L$，使得在测试集D_{test}上取得平均最佳的预测效果，即$p^* = argmax_{p \in L} \sum_{(x_i, y_i) \in D_{test}} E(y_i, \hat{y}_i)$。

由于语言空间L太过庞大，求出严格最优的p^*及其困难，因而需要采用一些近似的方式尝试寻找尽量更优的提示词。受到基于梯度的局部搜索方法的启发，我们可以将p^*的寻找视为一个在连续语言空间内的最优值搜索问题。这样建模带来一些问题：

1. 提示词点的梯度缺乏明确的定义。
2. 反向传播的实现缺乏明确的定义。

如果采用提示词在连续语言空间中的微分梯度，则需要训练辅助模型或者深入到较低的层次访问模型，从而获得提示词的微分表示，这样带来一些问题：

1. 操作复杂，除了模型 API 接口外可能需要额外的底层信息支持。
2. 由于不基于语义信息，而是需要利用到与模型相关的底层表示，因此对其他的模型或者不同的编码器难以适用。
3. 优化结果缺乏可解释性，无法以流畅自然语言的形式体现。

在研究[3]中，固定 encoder 的参数，利用投影的方式将每个嵌入向量投影到离散词汇表向量组成的嵌入矩阵中的最近邻，在投影后的点求梯度，并反向传播修改原提示词。这一做法与直接在当前提示词点直接求梯度稍有区别，不过仍然体现了上述三个缺点：需要额外的 encoder 协助优化；优化后的提示词对编码器具有强依赖性；优化结果非流畅自然语言，难以理解。

为了解决上述问题，我们要重新考虑对梯度的定义。上述研究将提示词点的梯度改为了在嵌入矩阵中的最近邻点的梯度，达到了更好的效果，因此我们考虑作出更加大的改动，让梯度的定义具有语义上的可解释性。

为了做到这一点，在研究[3]与研究[4]中引入了随机采样与基于反思的方法，我们将在下面详细讨论。

Random sampling method

研究[3]中使用在离散嵌入向量空间中寻找最近邻点计算梯度的方法得到梯度值，可以看作是将输入的每一个嵌入向量从连续的语言空间降维映射到离散的嵌入向量空间，然而由多个 token 拼接而成的结果并非流畅的自然语言，因而造成了可解释性的缺乏。为了增加梯度表示的可解释性，就需要将输入从自然语言映射到自然语言，这一任务是转写（rewrite）。由于语义表示的复杂性，这一步骤难以通过机械的算法实现，最适合完成这一任务的就是大语言模型本身。

因此，我们定义一个转写模型R，利用常量提示词指导其完成一条语句在保持原语义下的转写。研究[4]利用基于转写的随机采样方法实现了对提示词的优化，并且将蒙特卡洛模拟算法应用到这一方法中，在大量的随机模拟后能够得到效果较优的提示词。

然而，如果从梯度下降算法的视角来看，我们既没有实现梯度的计算，也没有实现基于反向传播的调整，而是通过无方向的采样算法从中选取到较优的结果。因而这一方法具有如下缺点：

1. 提示词的调整是无方向的，这意味着需要大量的试错才能够得到较优的提示词，而每一次评估的代价是高昂的，带来较大的 API 访问开销。
2. 提示词的语义被限制在原始语义的范围之内，由于每一次只进行轻微改动的转写，所以历经多次迭代之后也很难有较大的语义改动，这意味着算法只是在一个较小的语义空间内寻求最优结果，而忽视了语义差别更大的广阔空间内的可能性。

对于上述缺点，可以引入基于反思的机制加以解决。

Reflection-based method

如前所述，受到基于梯度的局部搜索方法的启发，我们希望能够将梯度和反向传播机制引入最优提示词的搜索算法，基于转写的采样方法难以实现这一点，但给了我们一些启示，也就是在保留可解释的语义信息的情况下定义梯度与反向传播。类比于人类在解决连续同类问题时的反应，很自然地可以发现如下类比：将梯度类比为对错误或者偏差的原因解释，将反向传播类比为基于解释而对原策略作出的主动修改。研究[5]中将这一过程称为反思（reflection）。

因此，我们定义一个优化器模型O。首先从原提示词P下预测错误或者偏差较大的结果中随机采样错例，将原提示词与每个错例对应的输入x_e、正确输出y_e，目标模型预测结果$\hat{y_e}$用常量提示词模板封装后作为输出传入优化器模型，得到反思的错因R_O，完成“梯度”的计算（reflection）。下一步将原提示词、错例与错因R_O用提示词常量封装之后的结果作为输入传入优化器模型O，得到改进后的提示词P'，完成“反向传播”（refinement）。注意这里将错例与错因封装在一起是由于后者对前者可能存在语义上的依赖关系。

按照这种方式，我们实现了对梯度与反向传播的模拟。基于这一模拟，我们可以实现基于贪心策略的搜索算法：

Algorithm 1: Reflection-based greedy search method

Require : P_{init} : prompt to get initial prompt, l : iteration limit,
O : Optimizer Model, T : Target Model, $D = \{(x_1, y_1), \dots (x_n, y_n)\}$

$p_0 \leftarrow GetInitPrompt(P_{init})$
for $i \leftarrow 1$ *to* l **do**
  $responses \leftarrow GetResponsesFromTarget(T, D)$
  $accuracy \leftarrow Evaluate(D, responses)$
  if $accuracy = 1.0$ **do**
    break
  $errorExamples \leftarrow GetErrorExamples(D, responses)$
  $reasons \leftarrow Reflection(O, p_0, errorExamples)$
  $p_0 \leftarrow Refinement(O, p_0, errroExamples, reasons)$
return p_0

需要注意的是，这种方法与对已知函数进行梯度下降算法不同。在这一方法中，由于固定传入优化器模型的提示词模板为常量，所以和优化器模型的整个交互过程是非参数化的，因而无法控制一个固定的步长。事实上，步长本身也很难定义。

由于缺乏对步长控制，我们难以控制迭代过程中会持续逼近局部的最优值。在实际的实验中，我们也观察到累次迭代下预测的评估值波动较大。因此，一个自然的动机是将传统的搜索算法应用到最优提示词的搜索上，结合传统搜索算法的高效性来提高搜索效率。

Apply search strategy to burst the performance

由于评估值随提示词变化的函数过于复杂，难以确知其性质，故难以利用基于梯度下降的方法寻找最优提示词，因此，为了达到较优的优化效果，需要结合其它的搜索算法。在研究[4]与研究[6]中使用蒙特卡洛搜索的方法，在研究[5]中结合了束搜索的方法，在研究[7]中引入了进化算法，均能够对原提示词的效果产生较大的增益。

在这一工作中，我们基于更优的提示词更有可能通过反思过程产生较优的提示词这一假设，认为束搜索的效率会更高。在后续的实验中，通过比较每次迭代产生的后继节点的均值证明了这一合理性，并证明了经过束搜索改进的提示词优化算法能够带来相当大的优化提升。

实验中，我们尝试了基于贪心策略的搜索算法和基于束搜索的算法。

Algorithm 2: Reflection-based beam search method

Require : P_{init} : prompt to get initial prompts, l : iteration limit,
b : beam width, s : expansion width
O : Optimizer Model, T : Target Model, $D = \{(x_1, y_1), \ldots (x_n, y_n)\}$

$B_0 \leftarrow \{GetInitPrompts(P_{init})\}$
$BestPrompt.inst \leftarrow B_0[0]$
$BestPrompt.accuracy \leftarrow 0.0$
for $i \leftarrow 1$ *to* l **do**
  for p **in** B_0 **do**
    $responses \leftarrow GetResponsesFromTarget(T, D)$
    $accuracy \leftarrow Evaluate(D, responses)$
    if $accuracy > BestPrompt.accuracy$ **do**
      $BestPrompt.inst \leftarrow p.inst$
      $BestPrompt.accuracy \leftarrow accuracy$
    if $BestPrompt.accuracy = 1.0$ **do**
      break
    $p.errorExamples \leftarrow GetErrorExamples(D, responses)$
  $SortByAccuracy(B_0)$
  $B \leftarrow \emptyset$
  for p **in** $B_0[0..b]$ **do**
    $B.push(p)$
    for $j \leftarrow 1$ *to* s **do**
      $samples \leftarrow SampleFrom(p.errorExamples)$
      $reasons \leftarrow Reflection(O, p.inst, samples)$
      $p' \leftarrow Refinement(O, p.inst, samples, reasons)$
      $B.push(p')$
  $B_0 \leftarrow B$
return p_0

由于在过程中观察到包含正例解答的 in-context learning 能够对评测结果产生显著的增益，我们想要探究我们的方法能否在此基础上带来进一步的提升，因此还完成了以 one-shot prompt 作为初始提示词的束搜索算法，并对此进行了进一步的讨论。

Experiments

Experimental setting

Dataset

数据集的选取对于结果的评判至关重要。我们的目标是寻找到一个具有挑战性的任务，从而说明提示词的优化能够达到其效果。介于目前大语言模型在数学推理，尤其是解决基本的、需要多步推理的数学任务上的不足，我们认为数学推理数据集 GSM8K 是验证提示词对大语言模型解决任务能力的影响的最优选择。

在 GSM8K 数据集中的问题数目相当多，仅测试集就有超过 1300 个问题。而在现实任务中，往往并不会具有如此大量的标准问答示例，因为其准备是十分消耗人力的[2]。因此，为了模拟这种标准数据的稀缺性，我们从全数据集中采样了 100 个问答作为训练集，100 个问答作为测试集，并不设置验证集，通过算法执行前后对测试集性能的增益来评价算法效果。为了弥补这种因样本缺乏而可能造成的不稳定性，我们在实验中选取了多个随机种子并进行重复实验。

Implementation

本节我们将介绍评估标准，以及算法实现时的一些细节。

Evaluation

在数据集 GSM8K 中，对每一个输入问题，输出的回答中包含一个唯一的数值答案，这意味着可以通过严格比对的方式来确定回答的正确性。因此，我们采用精确匹配（Exact match）作为评判指标。从目标模型的回复中解析出数值答案之后与标准值严格比对，从而确定回复是正例还是错例，所以对每一个输入x，对$\hat{y}$评估的结果是二值化的。

Format specification and output parsing

由于模型的输出为文本，而我们需要从中提取到部分的信息比如原因解释、优化后的提示词、问题的答案等，因而需要一种机制来完成这种提取。由于模型的输出词措较为灵活，并且对不同的输入回复的格式不具有一致性，因而难以通过流程化的算法提取结果。对此最佳的解决方法是在提示词中加入额外的格式说明常量，具体有以下三个主要的提示词格式说明常量：

1. Query：向目标模型传入输入x时，提示词中加入问题求解答案与解释格式的说明。
2. Reflection：向优化器模型传入原提示词与错例时，提示词中加入反思原因格式的说明。
3. Refinement：向优化器模型传入原提示词、错例与原因时，提示词中加入优化后的提示词格式的说明。

在研究[2]中采用示例的方式指导模型输出给定格式的结果。然而，我们认为这种方式容易与 few-shot prompt 相混淆，因为我们在后续的实验中发现少数示例会对模型的表现产生很大的影响。所以我们采用语句指令的方式实现对格式的规约，具体方法是指导模型在目标信息前后加入标记开始与结束的标记符号。

除了必要的格式说明常量，连接不同的输入部分以及指导模型给出结果也需要一些文本形式的常量。在实验中验证、溯因、优化三个部分包含这些常量的提示词模板如下：

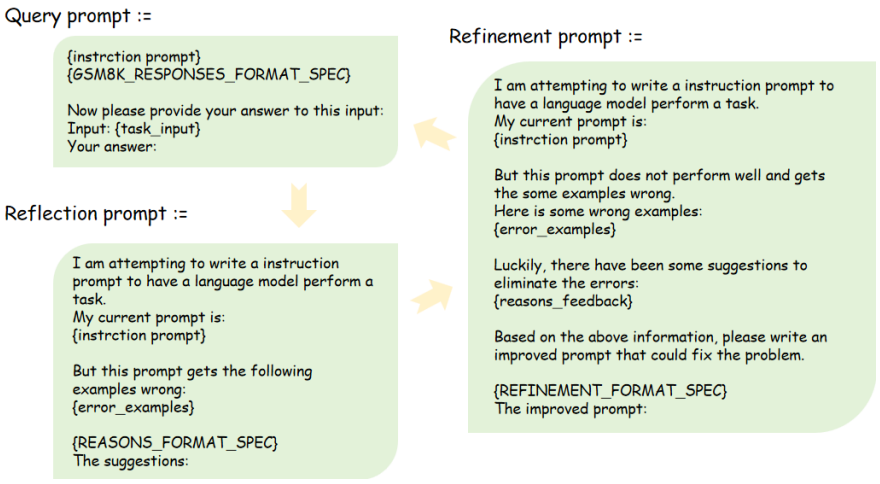

Figure 1: Constant prompt templates for three main steps

对目标模型反馈的问题解答，通过格式说明，我们可以在标记符号之间提取出所需的部分信息，完成第一步解析。此时的数值答案是以字符串的形式出现的，由于数据集中问题均为应用问题，因此模型可能根据上下文给出带有单位或者分隔符的文本结果，因而在与标准结果进行字符串比较时很容易造成假阴性的判断。因此，我们在得到文本形式的数值答案后会去除数字、符号、小数点等可能出现的符号以外的特殊符号，再将结果转换为数值类型，对标准答案也进行相同操作，对两个数值分别按照整数和浮点数的方式进行比对，如果有一种数值格式下的结果一致，则认为二者答案相同，精确匹配结果为真。

Settings

实验中的参数设置如下：

- 目标模型选用 GPT-3.5-turbo 。
- 优化器模型选用 GPT-3.5-turbo 。
- 训练集从 GSM8K 数据集中采样，大小为 50 。
- 测试集从 GSM8K 数据集中采样，与训练集无重复元素，大小为 100 。
- 迭代次数为 10 。
- 反思步骤采样的错例数为 3 ，实验中不存在错例少于这一数值的情况。
- 反思步骤反馈的错因条数为 2 。
- 所有实验为保证结论相对的普遍性，均采用三个不同的随机数种子进行三次重复实验，随机种子的设置将影响训练集的采样、提示词初始化、错例采样等步骤。
- 束搜索过程中，束大小为 6 ，每次选取前 $b = 2$ 个最优提示词进行扩展，各生成 $s = 2$ 个后继提示词，组成新束。
- Reflection 与 Refinement 步骤的常量提示词模板如前所述，实验过程中保持一致。

Greedy reflection-based method

Table 1: Procedure of reflection-based greedy algorithm

Iteration	1	2	3	4	5	6	7	8	9	10	Avg	Max	Init-test	Test
seed(0)	0.2	0.28	0.24	0.26	0.28	0.3	0.32	0.76	0.36	0.4	0.309	0.76	0.32	0.78
seed(1)	0.2	0.32	0.22	0.4	0.28	0.26	0.32	022	0.28	0.4	0.264	0.4	0.29	0.47

Iteration	1	2	3	4	5	6	7	8	9	10	Avg	Max	Init-test	Test
seed(2)	0.32	0.3	0.34	0.28	0.3	0.28	0.26	0.3	0.3	0.58	0.296	0.58	0.3	0.57

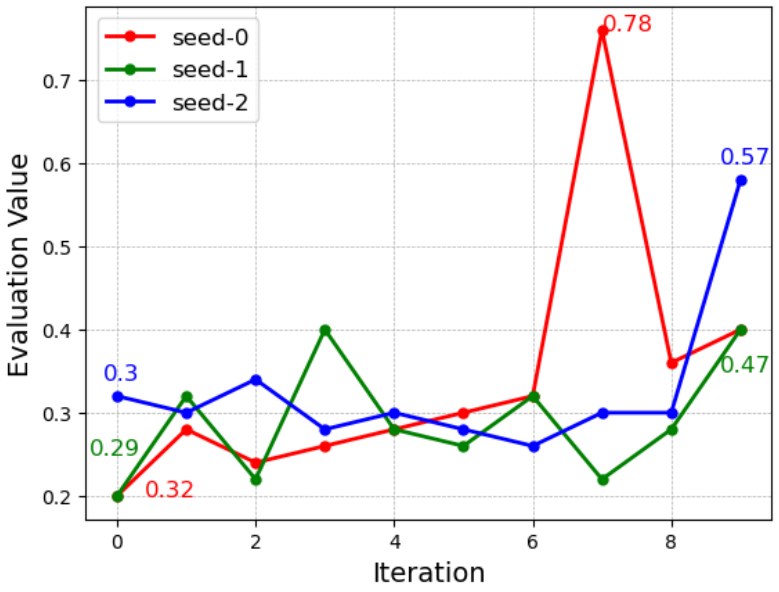

Figure 2: Fluctuations in evaluation values during the procedure of greedy strategy

基于贪心策略得到的结果如表1所示。

我们记录所有迭代中出现的评估值最优的指令，将其作为算法的输出。对于三个不同的随机种子，在仅 10 次迭代之后，算法输出的最优指令分别带来评估值 $144\%, 62\%, 90\%$ 的提升，并在测试集上达到了相近的评估值。这说明基于反思的方法具有寻找到更优后继的能力。

但同时可以看到，贪心策略随迭代次数增加，波动较为剧烈，对$seed(0)$的情况，在训练集上达到 0.76 的准确率之后下一次迭代便降至 0.36 ，说明反思优化步骤的不确定性较大。同时注意到训练集上平均的评估值与初始评估值相近，与每组的最优评估值具有较大差距，说明迭代过程中产生的提示词的平均效果不佳。

Reflection-based method with beam search

Table 2: Procedure of reflection-based beam search algorithm

Iteration	1	2	3	4	5	6	7	8	9	10	Init-test	Test
seed(0), avg.	0.19	0.243	0.28	0.353	0.417	0.4	0.51	0.49	0.527	0.56		
seed(0), max.	0.2	0.32	0.32	0.58	0.58	0.58	0.64	0.64	0.76	0.76	0.28	0.72
seed(1), avg.	0.18	0.237	0.3	0.413	0.47	0.393	0.453	0.51	0.487	0.483		
seed(1), max.	0.24	0.34	0.52	0.52	0.52	0.56	0.56	0.78	0.78	0.78	0.3	0.8
seed(2), avg.	0.31	0.34	0.343	0.46	0.577	0.62	0.627	0.633	0.623	0.617		
seed(2), max.	0.34	0.46	0.46	0.64	0.7	0.7	0.7	0.7	0.7	0.7	0.31	0.64

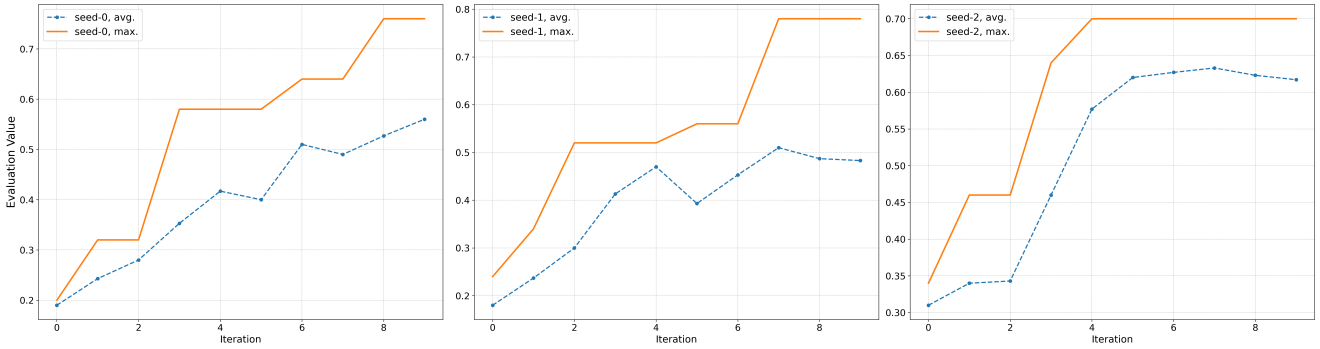

Figure 3: Maximum&Average evaluation values during beam search procedure

基于束搜索的算法得到的结果充分显示了基于反思的方法的优越性。如表2所示，在 10 次迭代之后，对于三个随机种子，算法输出的最优指令分别带来评估值 157%, 167%, 106% 的提升，并在测试集上得到了相近的评估值，最优指令达到了 0.8 的准确率，说明并没有产生严重的过拟合。

使用基于束搜索的算法得到的三组最佳提示词如下：

0. Break down the calculations for each individual item separately before combining them to ensure accuracy. Clearly define and explain the variables used in the equation to avoid confusion in the calculations. Provide a step-by-step breakdown of the total calculation process, including how to apply any discounts or conversions, to guarantee each part is accurately accounted for before arriving at the final result.

1. Break down the calculations step by step, clearly indicating how each part of the problem is being addressed. Provide detailed explanations for key concepts and operations involved in the problem to ensure a comprehensive understanding. Use different scenarios and values to demonstrate the concept of depreciation and its impact on the total price calculation over time. Ensure the logic and mathematical operations are outlined accurately in each calculation for each year of depreciation.

2. Break down the calculation of individual item costs into separate steps, clearly showing the multiplication for each item separately before combining them. Provide a step-by-step explanation of how to calculate the total cost before applying any discount, delineating the individual costs of each item and the final sum before moving on to the discount calculation. Ensure to identify key variables and information to accurately perform the calculations and equations. Make sure to break down complex tasks into smaller components and provide clear intermediate steps for each calculation.

Figure 4: Optimal prompts found for each random seed

从人类的角度能够理解其起作用的原因，因为它们包含问题拆分、分步解决、仔细检查等人类认为能够利于智能体求解问题的指令要求，而这些经检验并没有在初始获得的提示词中显式地出现。这充分显示了用语言模型的文本输出结果代表梯度的方法相对于显式地计算梯度并在嵌入向量层面更新提示词的方法在可解释性方面的优势。

Table 3: Investigation on the quality of generated prompts during training procedure

Iteration	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**r**
seed(0)-inherit	0.19	0.3	0.32	0.5	0.52	0.52	0.61	0.61	0.7	0.831
seed(0)-generated	0.27	0.27	0.37	0.375	0.34	0.505	0.43	0.485	0.49	
seed(1)-inherit	0.18	0.31	0.43	0.51	0.51	0.54	0.54	0.67	0.67	0.667
seed(1)-generated	0.265	0.295	0.405	0.45	0.335	0.41	0.495	0.395	0.39	
seed(2)-inherit	0.31	0.4	0.4	0.59	0.68	0.7	0.7	0.7	0.7	0.902
seed(2)-generated	0.355	0.315	0.49	0.57	0.59	0.59	0.6	0.585	0.575	

为了验证我们此前提到的假设：更优的提示词更有可能通过反思过程产生较优的提示词，我们对每一迭代步骤的数据进行了更细致的分析。注意到每一束评估值可以分为从上一轮继承来的 b 个提示词与对这 b 个提示词各进行 s 次错例采样生成的 $b \times s$ 个新的后继提示词，我们分别计算在每一轮迭代中二者的均值，得到表 3 。三组数据的相关系数表明采样生成的提示词的评估值与原提示词的

评估值具有强相关性，这验证了我们的假设，也表明了束搜索方法相对于蒙特卡洛模拟方法的高效性。

Enhance one-shot prompt

Table 4: The accuracy rate of several randomly generated one-shot prompt

Seed	Baseline accuracy	One-shot prompt accuracy
0	0.27	0.87
1	0.27	0.85
2	0.27	0.76

如表4所示，在实验中，我们观察到如果传入目标模型的提示词常量中包含完整的问题解答示例，则会对答复结果的准确性造成明显的增益。这是由于大语言模型卓越的少样本学习能力。我们并不希望引入这一因素干扰我们对提示词优化任务的研究，因而最终使用了格式说明文本常量来解决输出格式对齐的问题，而非给出示例解答，这与研究[2]中的做法不同。我们认为这一方式更为合理。

介于少量的完整解答示例就可以对提示词的性能造成极大的增益，并且据观察示例占用的 token 数量增长与优化提示词带来的 token 数量增长在相近的量级，并不会因为增加解答示例带来较大的访问开销，那么利用基于反思的方式进行提示词优化是否就没有意义？

在实验中，我们进一步讨论了这个问题。我们发现，即使对于效果较优的单样本提示词，我们的基于反思的提示词优化方法仍然能够在此基础上带来一定的性能提升。

Table 5: Procedure of reflection-based algorithm initialized with one-shot prompt

Iteration	1	2	3	4	5	6	7	8	9	10	Init-test	Test
seed(0), avg.	0.67	0.683	0.703	0.717	0.707	0.727	0.707	0.72	0.707	0.737		
seed(0), max.	0.72	0.72	0.74	0.76	0.76	0.76	0.76	0.76	0.76	0.76	0.78	0.8
seed(1), avg.	0.66	0.66	0.697	0.683	0.73	0.72	0.733	0.697	0.71	0.71		
seed(1), max.	0.68	0.7	0.74	0.74	0.78	0.78	0.78	0.78	0.78	0.78	0.75	0.79
seed(2), avg.	0.8	0.79	0.78	0.77	0.783	0.8	0.807	0.8	0.8	0.84		
seed(2), max.	0.82	0.82	0.82	0.82	0.9	0.9	0.9	0.9	0.9	0.9	0.76	0.83

如表 5 ，三组实验的初始提示词均包含一个正例的解答，评估值最值在 $0.78, 0.75, 0.76$，显著高于上一节实验中的 $0.28, 0.3, 0.31$。但三组实验均在前五次迭代内就带来了性能上的提升，在训练集上十次迭代后，最优提示词在测试集上的评估值增益分别为 $2.56\%, 5.33\%, 9.21\%$。

通过上述的实验，我们证明了基于反思的方法有助于提升 one-shot prompt 的效果。在现实应用中，one-shot prompt 的效果很大程度上与标准示例的选取有关，而由于大多数任务中初始正例的稀缺性，通过 one-shot prompt 所能够达到的最优效果是有限的，而利用基于反思迭代优化的方式可以达到性能上的持续提升，从而得到更优的结果。

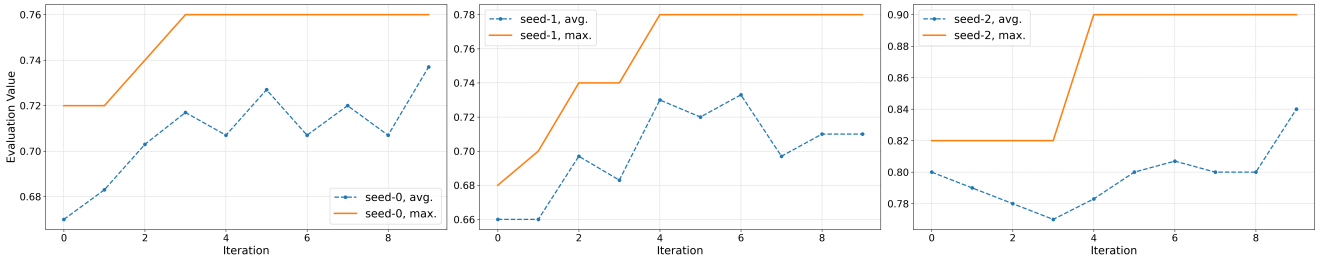

Figure 5: Beam search procedure initialized with one-shot prompt

Discussion

在这一节，我们主要来总结前述的一系列实验的结论，并讨论一些相关的问题。

实验中，我们首先对基于贪心策略与反思机制的提示词优化算法进行了测试，验证了反思机制有助于产生效果较优的后继。此后，为了改善贪心策略带来的评估值波动剧烈、平均效果不佳的问题，采用束搜索改进算法，在三组实验中均达到了相对较大的性能增益，这充分说明了基于反思机制的提示词优化方法的可行性。之后我们注意到包含示例解答的 in-context learning 能够给任务带来较大的性能增益，产生了将 one-shot prompt 作为初始提示词，使用基于反思的自动提示词优化方法对其进行持续的性能优化的想法，并实验验证能够带来一定的效果提升。

下面我们将讨论一些实验中观察到的现象与补充的实验细节。

1. 在进行基于反思机制的迭代提示词优化时，很可能出现反思过分偏重某一具体实例的情况，如下图所示，是在贪心策略的第一组实验中出现的一条优化后的提示词，其中反思步骤只有错例 2 反复提及银行一词，而经过 refinement 环节后的提示词特地强调了银行的例子，这对于其它的问题可能会对模型造成困扰，继而造成评估值指标的下降。

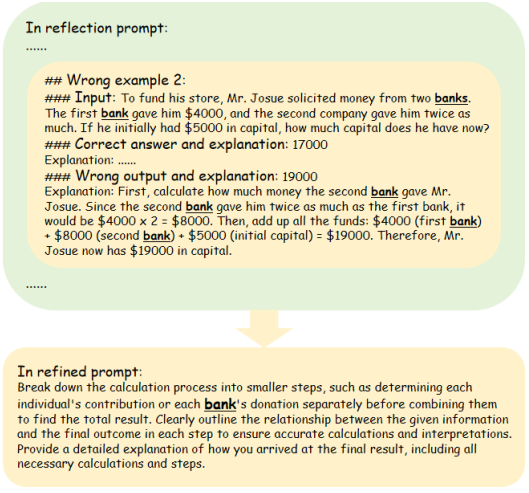

Figure 6: Example where the optimizer model focus on a specific situation

   在实验中，我们通过：

   1. 更改提示词常量：在 Reflection 与 Refinement 步骤的提示词常量模板中加入要求生成的提示词更具普遍性的语句。
   2. 增加错例数，并降低错例与反思原因个数的比值，以引导模型用更少的原因概括更多的问题。

   这两种方式尽量避免这一问题大量出现。

2. 由于资源开销较大，实验中难以在保证原迭代次数的情况下将束搜索的束宽度设为较大的值，因而可能会造成连续多次迭代选出的最优提示词为同一束，即连续多次对同一束进行随机采样，从而造成训练时最优评估值的停滞不变与平均评估值的上下波动。
3. 如第一点中所述，Reflection 与 Refinement 步骤进行优化时，常量提示词模板的内容可能会对结果造成较大的影响。比如研究[8]中通过改进 Reflection 步骤，基于 Chain-of-thought 的想法，让模型先进行案例分析，再提供建议，从而带来了预测性能的提升。当然，如果对将优化常量提示词模板视为一个提示词优化问题，或许能够寻找到更加高效的提示词模板，但其开销也将大量增加，这需要操作者根据资源与需求进行权衡。

Conclusion

Summary

在这一工作中，我们实现了基于反思机制的提示词优化算法，并与束搜索策略相结合，在数学推理任务上取得了显著的优化效果。这一方法只需利用到目标语言模型 API 接口，克服了基于梯度计算更新输入向量方法操作复杂、依赖模型底层接口的问题，更加便于实际应用。相对于基于少量样本或者 CoT 技术的上下文学习方法，我们的自动化方法能够带来性能的持续提升，克服了需要人工构建并试错来筛选最优提示词的缺点，并且能够在应用这些上下文学习方法的基础上进一步带来性能的增益。

Limitations and future work

这一工作存在的一些限制：

- 由于实验条件所限，没能够对更大的束宽度与迭代深度运行算法，因而目前的算法结果可能蕴含更多的不确定性。我们在实验中使用设置多个随机种子并观察到其对应的实验结果具有相同的分布趋势或者性质的方式作为一个弥补的方案，在一定程度上保证了结果的合理性。
- 算法过程中，每一次为了判断提示词的优劣，都需要在训练集上评测，这会带来大量的 API 调用开销。
- 在将自动优化算法与 in-context learning 相结合进行效果比较时，只考虑了 one-shot 的情况，并且没有对足够的数据进行验证，可能会造成结果中的偶然性因素。
- 由于时间所限，仅考虑了数学推理这一种任务情境与 GSM8K 这一个数据集，因此无法判断实验结论是否在其它任务情境中也适用。

可能的改进方式与其它的探索方向：

- 对更多数据集、更多的随机组数、更大的束宽度与迭代深度运行优化算法，从而提高数据的说服力，并验证这一方法在其他任务情境下的是否适用。
- 可以尝试使用错例对反思优化步骤生成的提示词进行预先验证，去除掉在错例集合上表现仍然很差的提示词，从而减少为了验证提示词效果er API 调用开销。
- 考虑对更多的 in-context learning 方法进行自动提示词优化，并验证随着迭代次数的加深，算法能否给原方法带来持续的效果提升。

---

1. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models ↩
2. The Unreasonable Effectiveness of Eccentric Automatic Prompts ↩ ↩ ↩ ↩ ↩
3. Hard Prompts Made Easy: Gradient-Based Discrete Optimization for Prompt Tuning and Discovery ↩ ↩ ↩
4. Large Language Models are Human-Level Prompt Engineers ↩ ↩ ↩
5. Automatic Prompt Optimization with "Gradient Descent" and Beam Search ↩ ↩
6. PROMPTAGENT: Strategic Planning with Language Models Enables Expert-level Prompt Optimization ↩
7. Connectiing Large Language Model with Evolutionary Algorithms Yields Powerful Prompt Optimizers ↩
8. Are Large Language Models Good Prompt Optimizers? ↩