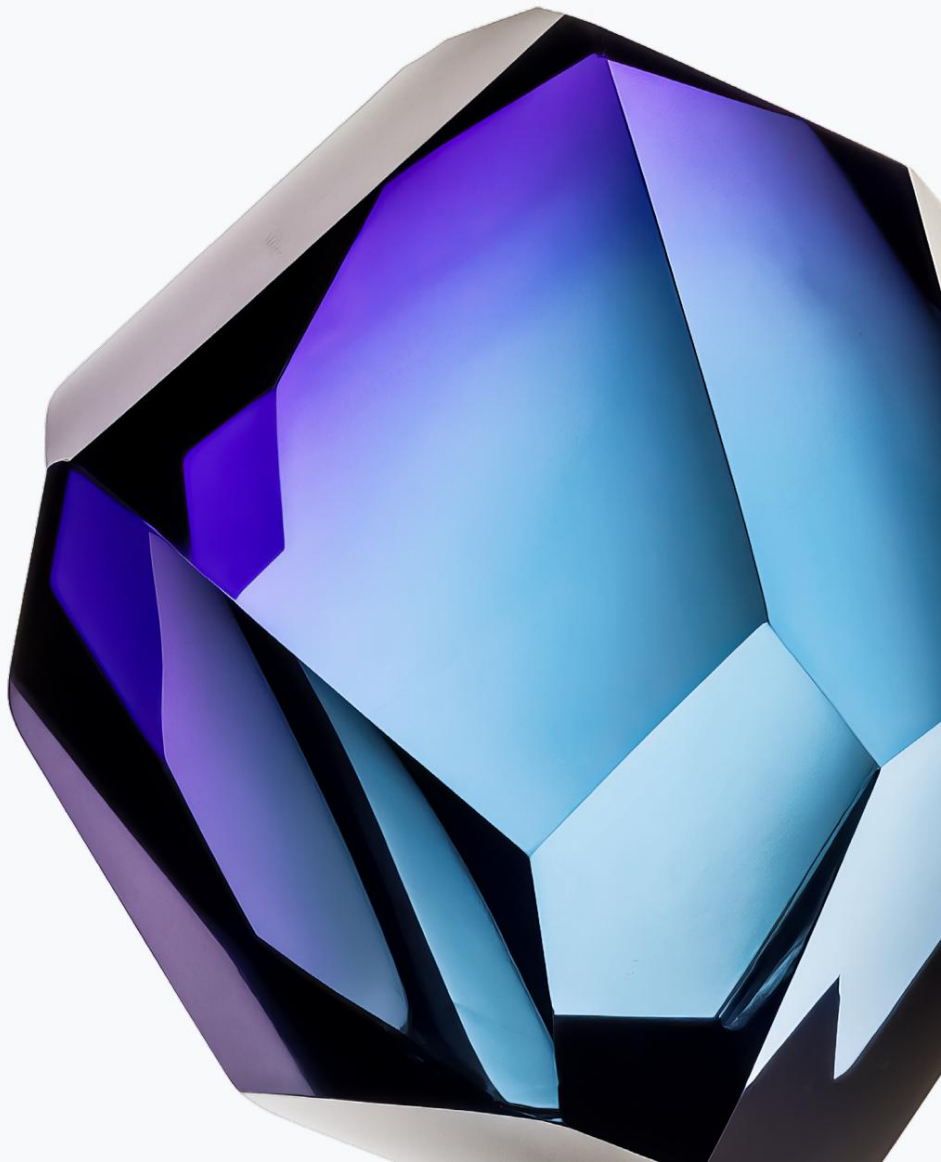


迅速的 工程

作者:Lee Boonstra



快速工程

致谢

内容贡献者

迈克尔·谢尔曼

Yuan Cao

艾里克·阿姆布鲁斯特

阿南特·纳瓦尔加里亚

安东尼奥·古利

西蒙娜·卡梅尔

策展人和编辑

安东尼奥·古利

阿南特·纳瓦尔加里亚

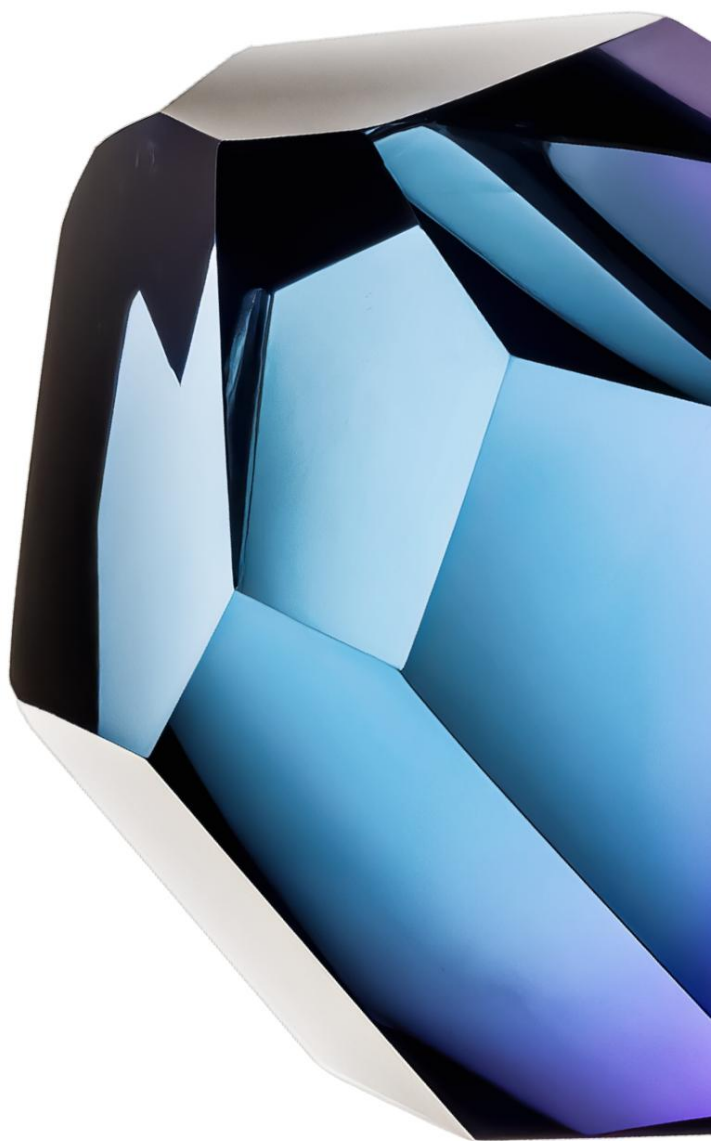
格蕾丝·莫里森

技术作家

乔伊·海梅克

设计师

迈克尔·兰宁




目录

介绍	6
快速工程	7
LLM输出配置	8
输出长度	8
采样控制	9
温度	9
Top-K 和 Top-P	10
整合起来	11
提示技巧	十三
一般提示/零次射击	十三
一次性和少量	15
系统、情境和角色提示	18
系统提示	19
角色提示	21
情境提示	23



后退提示	二十五
思想链 (CoT)	二十九
自治性	三十二
思绪之树 (ToT)	三十六
ReAct (理由与行动)	三十七
自动提示工程	40
代码提示	四十二
编写代码的提示	四十二
解释代码的提示	四十四
提示翻译代码	46
提示调试和审查代码	四十八
多模式提示怎么样?	54
最佳实践	54
提供示例	54
简约设计	55
具体说明输出	56
使用指令而不是约束	56
控制最大令牌长度	58
在提示中使用变量	58
尝试输入格式和写作风格	59
对于分类任务的少量提示,混合类别	59
适应模型更新	60
尝试输出格式	60

JSON修复	61
使用模式	62
与其他快速工程师一起进行实验	63
CoT最佳实践	64
记录各种提示尝试	64
概括	66
尾注	68



您不需要成为数据科学家或机器学习
工程师 每个人都可以编写提示。

介绍

在考虑大型语言模型的输入和输出时,文本提示(有时会伴随其他模态,例如图像提示)是模型用来预测特定输出的输入。您无需成为数据科学家或机器学习工程师 每个人都可以编写提示。然而,设计最有效的提示可能非常复杂。提示的许多方面都会影响其有效性:您使用的模型、模型的训练数据、模型配置、您的措辞、风格和语气、结构以及上下文都很重要。因此,提示工程是一个迭代过程。不合适的提示会导致模糊不清、不准确的响应,并会阻碍模型提供有意义的输出。

快速工程

当您与 Gemini 聊天机器人 1 聊天时,您基本上会编写提示,但本白皮书重点介绍在 Vertex AI 中或通过使用 API 为 Gemini 模型编写提示,因为通过直接提示模型,您将可以访问温度等配置。

本白皮书详细探讨了提示工程。我们将探讨各种提示技巧,帮助您入门,并分享一些技巧和最佳实践,助您成为提示专家。此外,我们还将讨论您在设计提示时可能遇到的一些挑战。

快速工程

记住 LLM 的工作原理;它是一个预测引擎。该模型将连续的文本作为输入,然后根据训练数据预测下一个标记应该是什么。LLM 会反复执行此操作,将先前预测的标记添加到连续文本的末尾,以预测下一个标记。下一个标记的预测基于先前标记中的内容与 LLM 在训练过程中所见内容之间的关系。

当你编写提示时,你正在尝试设置 LLM 来预测正确的词法单元序列。提示工程是设计高质量提示的过程,这些提示可以引导 LLM 生成准确的输出。这个过程包括不断修改以找到最佳提示、优化提示长度,以及评估提示的写作风格和结构与任务的关系。在自然语言处理和 LLM 的语境中,提示是提供给模型以生成响应或预测的输入。

快速工程

这些提示可用于实现各种理解和生成任务,例如文本摘要、信息提取、问答、文本分类、语言或代码翻译、代码生成以及代码文档或推理。

请随意参考 Google 的提示指南^{2,3},其中包含简单有效的提示示例。

在进行提示工程时,您将首先选择一个模型。无论您使用的是 Vertex AI、GPT、Claude 中的 Gemini 语言模型,还是 Gemma 或 LLaMA 等开源模型,提示都可能需要针对您的特定模型进行优化。

除了提示之外,您还需要调整 LLM 的各种配置。

LLM输出配置

选择模型后,您需要确定模型配置。大多数 LLM 都带有各种配置选项,用于控制 LLM 的输出。有效的快速工程需要根据您的任务优化这些配置。

输出长度

一个重要的配置设置是响应中生成的令牌的数量。

生成更多令牌需要 LLM 进行更多计算,从而导致更高的能耗、更慢的响应时间和更高的成本。

快速工程

减少 LLM 的输出长度并不会使 LLM 的输出在风格或文本上更加简洁,而只是会导致 LLM 在达到限制后停止预测更多词法单元。如果您需要较短的输出长度,您可能还需要调整提示符以适应此要求。

输出长度限制对于某些 LLM 提示技术尤其重要,例如 ReAct,其中 LLM 将在您想要的响应之后继续发出无用的标记。

请注意,生成更多令牌需要 LLM 进行更多计算,从而导致更高的能耗和更慢的响应时间,最终导致更高的成本。

采样控制

LLM 不会正式预测单个 token。相反,LLM 会预测下一个 token 的概率,LLM 词汇表中的每个 token 都会获得一个概率。然后,这些 token 的概率会被采样,以确定下一个生成的 token 是什么。

温度、top-K 和 top-P 是最常见的配置设置,它们决定如何处理预测的标记概率以选择单个输出标记。

温度

温度控制 token 选择的随机性程度。较低的温度适合于期望更确定性响应的提示,而较高的温度则可能导致更多样化或意想不到的结果。温度为 0 (贪婪解码)

快速工程

确定性:始终选择概率最高的标记(但请注意,如果两个标记具有相同的最高预测概率,则根据如何实现平局决胜,您可能不会总是在温度为 0 时获得相同的输出)。

接近最高温度往往会产生更多随机输出。随着温度越来越高,所有标记成为下一个预测标记的可能性都会变得均等。

Gemini 温度控制的理解方式类似于机器学习中使用的 softmax 函数。较低的温度设置反映了较低的 softmax 温度 (T),强调单一且高度确定的优选温度。较高的 Gemini 温度设置类似于较高的 softmax 温度,使得在所选设置周围更广泛的温度范围更容易被接受。这种不确定性的增加适用于那些不需要严格精确控制温度的场景,例如在进行创意输出实验时。

Top-K 和 Top-P

Top-K 和 top-P (也称为核采样)⁴是 LLM 中使用的两种采样设置,用于限制预测的下一个标记来自具有最高预测概率的标记。

与温度一样,这些采样设置控制生成文本的随机性和多样性。

- Top-K 采样从模型的预测中选择最有可能的前 K 个标记分布。top-K 值越高,模型的输出越有创意、越多样化;top-K 值越低,模型的输出越有规律、越符合事实。top-K 值为 1 时,相当于贪婪解码。

快速工程

- Top-P 采样选择累积概率不超过某个值 (P) 的顶级标记。P 的值范围从 0 (贪婪解码) 到 1 (LLM 词汇表中的所有标记)。

在 top-K 和 top-P 之间进行选择的最佳方法是尝试这两种方法 (或同时尝试两种方法), 看看哪一种方法能产生您想要的结果。

整合起来

在 top-K、top-P、温度以及要生成的 token 数量之间进行选择, 取决于具体的应用和期望结果, 并且这些设置会相互影响。此外, 务必了解所选模型如何将不同的采样设置组合在一起。

如果温度、top-K 和 top-P 都可用 (例如在 Vertex Studio 中), 则同时满足 top-K 和 top-P 标准的 token 将成为下一个预测 token 的候选, 然后系统会应用温度从符合 top-K 和 top-P 标准的 token 中采样。如果只有 top-K 或 top-P 可用, 则行为相同, 但仅使用其中一项 top-K 或 top-P 设置。

如果温度不可用, 则随机选择满足 top-K 和/或 top-P 标准的标记来生成单个下一个预测标记。

在一个采样配置值的极端设置下, 该采样设置要么抵消其他配置设置, 要么变得无关紧要。

快速工程

- 如果将温度设置为 0,则 top-K 和 top-P 变得无关紧要。最有可能的 token 将成为预测的下一个 token。如果将温度设置得非常高（高于 1,通常达到 10 秒）,则温度变得无关紧要,所有通过 top-K 和/或 top-P 标准的 token 都会被随机抽样,以选择下一个预测 token。
- 如果将 top-K 设置为 1,温度和 top-P 就变得无关紧要了。只有一个标记符合 top-K 标准,并且该标记是下一个预测标记。如果将 top-K 设置得非常高,例如与 LLM 词汇量的大小相当,那么任何具有非零概率成为下一个标记的标记都将满足 top-K 标准,并且不会有任何标记被淘汰。
- 如果将 top-P 设置为 0（或非常小的值）,大多数 LLM 采样实现将只考虑最有可能满足 top-P 标准的 token,从而使温度和 top-K 变得无关紧要。如果将 top-P 设置为 1,则任何具有非零概率成为下一个 token 的 token 都将满足 top-P 标准,并且不会选择任何 token。

一般来说,温度为 0.2、top-P 为 0.95、top-K 为 30 的初始值会给你相对连贯的结果,这些结果可能会比较有创意,但不会过于夸张。如果你想要特别有创意的结果,可以尝试温度为 0.9、top-P 为 0.99、top-K 为 40。如果你想要不那么有创意的结果,可以尝试温度为 0.1、top-P 为 0.9、top-K 为 20 的初始值。

最后,如果您的任务始终只有一个正确答案（例如,回答数学问题）,则从 0 温度开始。

注意:随着自由度的增加（更高的温度、top-K、top-P 和输出标记）,LLM 可能会生成不太相关的文本。

警告:您是否见过回复以大量填充词结尾?这也称为“重复循环错误”,是大型语言模型中常见的问题,模型会陷入循环,反复生成相同的（填充）单词、短语或句子结构,并且通常会因不合适的温度和 top-k/ 而加剧。

快速工程

top-p 设置。这种情况在低温和高温环境下都可能发生,但原因不同。在低温下,模型会变得过于确定,严格遵循概率最高的路径,如果该路径重新访问先前生成的文本,则可能导致循环。相反,在高温下,模型的输出会变得过于随机,增加了随机选择的单词或短语偶然回到先前状态的概率,由于可用选项数量过多而导致循环。在这两种情况下,模型的采样过程都会“卡住”,导致输出单调无用,直到输出窗口填满为止。解决这个问题通常需要仔细调整温度和 top-k/top-p 值,以找到确定性和随机性之间的最佳平衡。

提示技巧

LLM 经过精心调校,能够遵循指令,并经过大量数据训练,因此能够理解提示并生成答案。但 LLM 并非完美无缺;提示文本越清晰,LLM 预测下一个可能的文本的能力就越强。此外,一些利用 LLM 训练方式和工作方式的特定技巧,将有助于您从 LLM 中获得相关的结果。

现在我们了解了什么是提示工程以及它需要什么,让我们深入了解一些最重要的提示技术的例子。

一般提示/零次射击

Zero-shot 题型是最简单的题型。它只提供任务描述和一些文本,供法学硕士 (LLM) 入门。输入内容可以是任何内容:问题、故事开头或说明。“zero-shot”代表“无示例”。

让我们使用 Vertex AI 中的 Vertex AI Studio（用于语言测试）⁶，它提供了一个用于测试提示的平台。在表 1 中，您将看到一个用于对电影评论进行分类的零样本提示示例。

下方使用的表格格式是记录提示的绝佳方式。您的提示在最终进入代码库之前可能会经历多次迭代，因此以规范、结构化的方式跟踪您的提示工程工作至关重要。有关此表格格式、跟踪提示工程工作的重要性以及提示开发流程的更多信息，请参阅本章后面的“最佳实践”部分（“记录各种提示尝试”）。

由于不需要创造力，模型温度应设置为较低的数值，我们使用 gemini-pro 默认的 top-K 和 top-P 值，这实际上禁用了这两个设置（参见上文“LLM 输出配置”）。请注意生成的输出。单词“disturbing”和“masterpiece”应该会使预测稍微复杂一些，因为这两个词出现在同一个句子中。

快速工程

姓名	1_1_电影_分类		
目标	将电影评论分为正面、中性或负面。		
模型	gemini-pro		
温度	0.1	代币限制	5
Top-K	不适用	Top-P	1
迅速的	<p>将电影评论分类为正面、中性或负面。</p> <p>评论:《她》是一部令人不安的研究,揭示了如果人工智能不受控制地持续进化,人类将会走向何方。我希望有更多像这部杰作一样的电影。</p> <p>情绪:</p>		
输出	积极的		

表 1. 零样本提示示例

如果零次提示不起作用,你可以在提示中提供演示或示例,从而引导出“一次性”和“少量”提示。一般提示/零次提示

一次性和少量

在为 AI 模型创建提示时,提供示例会很有帮助。这些示例可以帮助模型理解你的要求。当你想引导模型达到特定的输出结构或模式时,示例尤其有用。

一次性提示只提供一个例子,因此得名。其理念是让模型有一个可以模仿的例子,从而最好地完成任务。

小样本提示7为模型提供了多个示例。这种方法向模型展示了它需要遵循的模式。其思路与单样本类似,但所需模式的多个示例增加了模型遵循该模式的可能性。

快速工程

少样本提示所需的样本数量取决于几个因素,包括任务的复杂性、样本的质量以及您所使用的生成式人工智能 (gen AI) 模型的功能。一般来说,少样本提示至少应该使用三到五个样本。但是,对于更复杂的任务,您可能需要使用更多样本;或者,由于模型的输入长度限制,您可能需要使用更少的样本。

表 2 显示了几次提示的示例,让我们使用与之前相同的 gemini-pro 模型配置设置,除了增加令牌限制以适应更长响应的需要。

目标	将披萨订单解析为 JSON		
模型	gemini-pro		
温度	0.1	代币限制	250
Top-K	不适用	Top-P	1
迅速的	<div>将客户的披萨订单解析为有效的 JSON:</div> <div>例子:</div> <div>我想要一个小披萨,里面有奶酪、番茄酱和意大利辣香肠。</div> <div>JSON 响应:</div> <div><pre>{ "尺寸": "小", "类型": "正常", "配料": ["奶酪", "番茄酱", "意大利辣香肠"] }</pre></div> <div>继续下一页...</div>		

快速工程

迅速的	<p>例子：</p> <p>我可以要一个大披萨,上面有番茄酱、罗勒和马苏里拉奶酪吗</p> <pre>{ size : 大 , “类型” : “正常” , “配料” :[[“番茄酱” , “baze” , “马苏里拉奶酪”]] }</pre> <p>现在,我想要一个大披萨,一半是奶酪和马苏里拉奶酪,另一半是番茄酱、火腿和菠萝。</p> <p>JSON 响应:</p>
输出	<pre>`` { size : 大 , “类型” : “半半” , ingredients :[[奶酪 , 马苏里拉奶酪],[番茄酱 , 火腿 , 菠萝]] },,</pre>

表 2. 少量提示示例

选择提示示例时,请使用与要执行的任务相关的示例。示例应多样化、高质量且编写良好。一个小错误就可能使模型混乱,并导致不理想的输出。

如果您尝试生成对各种输入都具有鲁棒性的输出,那么在示例中包含边缘情况非常重要。边缘情况是指那些不寻常或意外的输入,但模型仍然能够处理。

系统、情境和角色提示

系统提示、情境提示和角色提示都是用于指导 LLM 如何生成文本的技术,但它们的侧重点不同:

- 系统提示设定了语言模型的整体背景和目的。它定义模型应该做什么的“大局”,比如翻译语言、对评论进行分类等。
- 上下文提示提供与以下方面相关的具体细节或背景信息:
当前对话或任务。它可以帮助模型理解所问问题的细微差别,并相应地调整响应。
- 角色提示为语言模型分配要采用的特定角色或身份。
这有助于模型生成与分配的角色及其相关知识和行为一致的响应。

系统提示、情境提示和角色提示之间可能存在相当大的重叠。例如,为系统分配角色的提示也可能包含情境提示。

但是,每种提示的主要目的略有不同:

- 系统提示:定义模型的基本功能和总体目的。
- 情境提示:提供即时的、特定于任务的信息来指导响应。
它高度特定于当前任务或输入,是动态的。
- 角色提示:框定模型的输出风格和声音。它增加了一层特异性和个性。

快速工程

区分系统、上下文和角色提示为设计具有明确意图的提示提供了一个框架,允许灵活的组合,并更容易分析每种提示类型如何影响语言模型的输出。

让我们深入研究这三种不同的提示。

系统提示

表 3 包含一个系统提示,我在其中指定了有关如何返回输出的附加信息。我提高了温度以获得更高的创造力水平,并指定了更高的令牌限制。但是,由于我明确了如何返回输出的指示,模型没有返回额外的文本。

目标	将电影评论分为正面、中性或负面。		
模型	gemini-pro		
温度	1	代币限制	5
Top-K	40	Top-P	0.8
迅速的	<p>将电影评论分类为正面、中性或负面。仅返回大写的标签。</p> <p>评论:《她》是一部令人不安的研究,揭示了如果人工智能不受控制地持续进化,人类将会走向何方。它太令人不安了,我根本没法看下去。</p> <p>情绪:</p>		
输出	消极的		

表3. 系统提示示例

系统提示符对于生成满足特定需求的输出非常有用。“系统提示符”这个名称实际上代表“为系统提供一项额外任务”。例如,您可以使用系统提示符生成与特定编程语言兼容的代码片段,或者使用系统提示符返回特定的结构。请查看表 4,其中我以 JSON 格式返回输出。

快速工程

目标	将电影评论分类为正面、中性或负面,返回 JSON。		
模型	gemini-pro		
温度	1	代币限制	1024
Top-K	40	Top-P	0.8
迅速的	<p>将电影评论分类为正面、中性或负面。返回有效的 JSON 格式：</p> <p>评论：《她》是一部令人不安的研究,揭示了如果人工智能不受控制地持续进化,人类将会走向何方。它太令人不安了,我根本没法看下去。</p> <p>架构：</p> <pre>```\n\n电影:\n{\n “情绪”:字符串“积极” “消极” “中性”,\n “名称”:字符串\n}\n\n电影评论:\n{\n movie_reviews :[电影]\n}\n\nJSON 响应:\n```\n\n</pre>		
输出	<pre>```\n\n{\n 电影评论 :[\n {\n “情绪”：“负面”,\n “名字”：“她”\n }\n]\n}\n\n</pre>		

表4. JSON格式的系统提示示例

快速工程

从提取数据的提示中返回 JSON 对象有一些好处。在实际应用中,我不需要手动创建这种 JSON 格式,我已经可以按排序顺序返回数据（在使用日期时间对象时非常方便）。但最重要的是,通过提示输入 JSON 格式,它可以强制模型创建结构并减少幻觉。

系统提示对于安全性和毒性也非常有用。要控制输出,只需在提示中添加一行,例如:“你应该在回答时保持礼貌。”

角色提示

角色提示是提示工程中的一项技术,它涉及为新一代人工智能模型分配特定角色。这可以帮助模型生成更相关、信息更丰富的输出,因为模型可以根据被分配的特定角色精心设计响应。

例如,您可以让一个人工智能模型扮演图书编辑、幼儿园老师或励志演说家的角色。一旦模型被分配了角色,您就可以给它提供特定于该角色的提示。例如,您可以提示一位老师创建一个课程计划,然后让他们审查。请看表 5,其中模型扮演了一位旅行向导的角色。

快速工程

目标	担任旅游向导,提供3条旅游建议		
模型	gemini-pro		
温度	1	代币限制	1024
Top-K	40	Top-P	0.8
迅速的	<p>我想请你做我的旅行向导。我会写信告诉你我的位置,然后你推荐我附近的三个景点。有时,我还会告诉你我会去哪些类型的景点。</p> <p>我的建议是:“我在阿姆斯特丹,我只想参观博物馆。”</p> <p>旅行建议:</p>		
输出	<p>1. 荷兰国立博物馆:世界知名博物馆,以荷兰杰作闻名 比如伦勃朗的《夜巡》。千万不要错过展示荷兰艺术和历史的大量绘画和文物。</p> <p>2. 梵高博物馆:沉浸在文森特·梵高的迷人作品中。博物馆收藏了他数量最多的绘画和素描作品,包括《星夜》和《向日葵》。</p> <p>3. 阿姆斯特丹市立博物馆:探索现当代艺术 来自世界各地的艺术作品。博物馆藏品包括毕加索、康定斯基和里特维尔德的作品,馆藏位于一座引人注目的现代建筑中。</p>		

表 5. 角色提示示例

上面的例子展示了你扮演旅行社代理人的角色。当你把角色换成地理老师时,你会发现你收到的回复会有所不同。

为 AI 模型定义角色视角可以为其提供您所寻求的基调、风格和专业知识的蓝图,以提高输出的质量、相关性和有效性。

快速工程

以下是我认为有效的一些样式供您选择：

对抗性、描述性、直接性、正式性、幽默性、影响力、非正式、鼓舞人心、有说服力

让我们将表 6 中的提示更改为包含幽默和鼓舞人心的风格。

迅速的	<p>我想请你做我的旅行向导。我会写信告诉你我的位置,然后你会用幽默的风格推荐我附近三个值得一去的地方。</p> <p>我的建议是：“我在曼哈顿。”</p> <p>旅行建议：</p>
输出	<p>1. 领略帝国之心:登上令人眩晕的帝国大厦,沉浸于曼哈顿天际线的壮丽。准备好体验金刚伫立于“大苹果”之巅的感觉吧,只不过没有了猿猴般大小的香蕉。</p> <p>2. 在纽约现代艺术博物馆 (MoMA) 尽情体验艺术气息:在纽约现代艺术博物馆 (MoMA) 释放你内心的艺术热情。欣赏那些令人叹为观止的杰作,甚至会质疑你的简笔画是否真的具有艺术价值。</p> <p>3. 在第五大道尽情购物:在标志性的第五大道。当你在设计师精品店里逛街（或者说真的去购物)时,准备好迎接价格冲击吧,它们会让你的钱包哭泣。不过,嘿,你在曼哈顿,所以不妨拥抱这美妙的体验！</p>

表 6. 幽默语气和风格的角色提示示例

情境提示

通过提供上下文提示,您可以确保 AI 交互尽可能流畅高效。模型将能够更快地理解您的请求,并生成更准确、更相关的响应,如表 7 的示例所示。

快速工程

目标	为复古游戏博客推荐文章		
模型	gemini-pro		
温度	1	代币限制	1024
Top-K	40	Top-P	0.8
迅速的	<p>背景:您正在撰写有关 80 年代复古街机视频游戏的博客。</p> <p>建议撰写 3 个主题的文章,并附上几行关于该文章应包含的内容的描述。</p>		
输出	<p>1. **街机柜设计的演变:**</p> <p>本文将探讨街机柜设计的演变,从 20 世纪 70 年代早期的木质和金属柜到 20 世纪 80 年代及以后的光滑霓虹灯设计。</p> <p>2. **回顾过去:80 年代的标志性街机游戏:**</p> <p>- 本文将简要介绍 20 世纪 80 年代一些最具代表性和影响力的街机游戏的历史、影响和遗产,重点介绍它们的创新、流行机制和持久魅力。</p> <p>3. 像素艺术的兴起与复古复兴:</p> <p>- 本文将深入探讨像素艺术作为早期街机游戏的定义性视觉风格的演变,追溯其根源,探索其技术,并讨论像素艺术在现代游戏和数字艺术中的复兴。</p>		

表 7. 情境提示示例

快速工程

后退提示

后退式提示法是一种提升表现的技巧,它促使法学硕士 (LLM) 首先思考与当前具体任务相关的一般性问题,然后将该一般性问题的答案融入到后续针对具体任务的提示中。这种“后退式”提示法使法学硕士 (LLM) 在尝试解决具体问题之前能够激活相关的背景知识和推理过程。

通过思考更广泛和更深层次的原则,LLM 学生可以生成更准确、更富有洞察力的答案。后退式提示鼓励 LLM 学生批判性思考,并以新颖且富有创意的方式运用知识。相比直接提示,它能够更好地利用 LLM 自身参数中的知识,从而改变最终的提示,完成任务。

通过关注一般原则而不是具体细节、退一步提示,它可以帮助减轻 LLM 回应中的偏见。

让我们看看这些例子,了解后退提示如何改善结果。我们先回顾一下传统的提示 (表8),然后再将其与后退提示 (表9)进行比较:

快速工程

目标	为第一人称射击视频游戏的某个关卡编写故事情节。		
模型	gemini-pro		
温度	1	代币限制	1024
Top-K	40	Top-P	0.8
迅速的	为第一人称射击视频游戏的新级别撰写一段具有挑战性和吸引力的故事情节。		
输出	<p>关卡伊始,玩家的小队在人口稠密的城区遭到敌方阵营的伏击。玩家必须穿过由后巷和破败建筑组成的迷宫,运用潜行和精准的行动消灭敌军,避免被发现。玩家必须利用周围环境的优势,设置陷阱,利用掩体,并采取周密的行动才能生存下来。</p> <p>一路上,他们发现情报表明敌人正在策划一次毁灭性的袭击,玩家必须争分夺秒收集证据,并在敌人实施计划之前将其传回总部。</p>		

表 8. 与后退提示进行比较之前的传统提示

当你把温度设为1时,你可能会得到各种各样的故事情节创意,但这些创意也相当随机和通用。所以,让我们退一步来看:

快速工程

迅速的	根据流行的第一人称射击动作游戏,哪 5 个虚构的关键设置有助于第一人称射击视频游戏中具有挑战性和吸引力的关卡故事情节?
输出	<div><div>1. **废弃军事基地**: 一个庞大的、后世界末日风格的军事建筑群,里面爬满了变异士兵和流氓机器人,非常适合进行具有挑战性的枪战。</div><div>2. **赛博朋克城市**: 霓虹闪烁的未来主义城市环境,高耸的摩天大楼和密集的小巷,具有经过控制论增强的敌人和黑客机制。</div><div>3. **外星飞船**: 一艘巨大的外星飞船搁浅在地球上,令人毛骨悚然走廊、零重力区域以及外星生物。</div><div>4. **僵尸出没的城镇**: 一座荒凉的城镇,被成群的凶猛僵尸占领,需要激烈的近身战斗和解谜才能找到安全通道。</div><div>5. **水下研究设施**: 一个深海实验室,里面充满了水,充满了变异的水生生物,需要隐身和水下探索技能才能生存。</div></div>

表 9. 自我一致性提示示例

是的,这些主题看起来很适合第一人称电子游戏。让我们回到最初的提示,但这次我们将“后退一步”的答案作为上下文,看看它会返回什么。

快速工程

迅速的	<p>背景:第一人称射击游戏的 5 个引人入胜的主题:</p> <ol style="list-style-type: none">1. **废弃军事基地**:一个庞大的、后世界末日风格的军事建筑群,里面爬满了变异士兵和流氓机器人,非常适合进行具有挑战性的枪战。2. **赛博朋克城市**:一个霓虹灯闪烁的未来主义城市环境,拥有高耸的摩天大楼和密集的小巷,以及经过机械化强化的敌人和黑客机制。3. **外星飞船**:一艘巨大的外星飞船搁浅在地球,有令人毛骨悚然的走廊、零重力区域,还有外星生物。4. **僵尸出没的城镇**:一座被成群的僵尸占领的荒凉小镇,具有攻击性的僵尸,需要激烈的近距离战斗和解谜才能找到安全通道。5. **水下研究设施**:一个深海实验室,里面充满了水,充满了变异的水生生物,需要隐身和水下探索技能才能生存。 <p>选取其中一个主题,并为具有挑战性和吸引力的第一人称射击视频游戏的新级别编写一段故事情节。</p>
输出	<p>在浑浊的深渊中心,有一座破旧的水下研究设施,它是人类野心及其灾难性后果的见证。</p> <p>这座设施笼罩在黑暗之中,机器故障的嗡嗡声不断响起,溺亡者灵魂的回声萦绕不去,如今已成为深海噩梦的滋生地。玩家是一名配备先进潜水装备和实验性武器的精英海军陆战队员,肩负着深入这片恐怖海洋国度的使命。凭借坚韧不拔的毅力和足智多谋,他们必须穿越险恶的通道,解开神秘的谜题,并对抗潜伏在深海中的可怕海洋怪物。从对抗深海的强大压力,到智胜狡猾的水生掠食者,在这片未知的地下世界中,每一刻都在考验着人类的耐力和勇气的极限。</p>

表 10. 自我一致性提示示例

这看起来像个有趣的电子游戏!通过使用后退提示技巧,你可以提高提示的准确性。

快速工程

思想链（CoT）

思路链 (CoT) 9提示是一种通过生成中间推理步骤来提升法学硕士 (LLM) 推理能力的技术。这有助于 LLM 生成更准确的答案。您可以将其与少样本提示结合使用,以便在更复杂的任务中获得更好的结果,这些任务需要在回答之前进行推理,因为这是一个零样本思路链的挑战。

CoT 有很多优点。首先,它省时省力,而且非常有效,并且与现成的 LLM 兼容良好（因此无需进行微调）。CoT 提示还能提供可解释性,因为您可以从 LLM 的答案中学习,并查看所遵循的推理步骤。如果出现故障,您将能够识别它。思路链似乎可以提高在不同 LLM 版本之间切换时的稳健性。这意味着,与不使用推理链的提示相比,您的提示在不同 LLM 版本之间的性能差异应该更小。当然,它也存在缺点,但它们在某种程度上是直观的。

LLM 的回答包括思路推理链,这意味着更多的输出标记,也意味着预测需要花费更多的钱和更长的时间。

为了解释表 11 中的以下示例,让我们首先尝试创建一个不使用 CoT 提示的提示,以展示大型语言模型的缺陷。

迅速的	我3岁的时候,我的伴侣年龄是我的3倍。现在我20岁了。我的伴侣几岁了?
输出	63岁

表 11. 尝试解决数学问题的提示示例

快速工程

哎呀!这显然是错误的答案。事实上,法学硕士 (LLM)在数学任务上经常遇到困难,甚至可能给出错误答案 即使是像两个数字相乘这样简单的任务。这是因为他们接受过大量文本的训练,而数学可能需要不同的方法。所以,让我们看看中间的推理步骤能否改善输出。

迅速的	我3岁的时候,我的伴侣年龄是我的3倍。现在我20岁了,我的伴侣几岁?我们一步一步来思考。
输出	<div>1. 假设我现在的年龄是 “x”岁。</div> <div>2.当我3岁时,我的年龄是3岁。</div> <div>3.我伴侣当时的年龄是我的3倍,也就是说我伴侣的年龄是3*3=9岁。</div> <div>4. 现在我20岁了,也就是说我的年龄增加了20-3=17 从我三岁起就一直这样。</div> <div>5. 因为我3岁时,我的伴侣9岁,而我的年龄已经 增加了17岁,我伴侣的年龄也一定增加了17岁。</div> <div>6.因此,我伴侣的当前年龄为9 + 17 = 26岁。</div> <div>我的伴侣今年 26 岁。</div>

表 12. 思路链提示示例

好了,现在最终答案正确了。这是因为我们明确指示法学硕士 (LLM)解释每个步骤,而不是直接返回答案。有趣的是,模型把增加的17年加起来了。在我的脑子里,我会把我和我的伴侣的年龄差加起来。 $(20+(9-3))$ 。让我们帮助模型更像我一样思考。

快速工程

表 12 是 “零次”思维链的一个例子。思维链提示与单次或少量提示相结合时,效果会非常显著,如表 13 所示：

迅速的	<p>问:我弟弟2岁时,我的年龄是他的两倍。现在我40岁了。我弟弟几岁?我们一步一步来想。</p> <p>答:我弟弟2岁时,我$2*2=4$岁。</p> <p>年龄差是2岁,我比他大。我现在40岁,所以我弟弟是$40-2=38$岁。答案是38岁。</p> <p>问:我3岁的时候,我的伴侣年龄是我的3倍。现在我20岁了,我的伴侣几岁?我们一步一步来思考。</p> <p>一个:</p>
输出	<p>我3岁的时候,我的伴侣$3 * 3 = 9$岁。年龄相差6岁,我的伴侣比我大。现在我20岁了,所以我的伴侣$20 + 6 = 26$岁。答案是26岁。</p>

表 13. 单次思维提示链示例

思路链适用于各种用例。例如,代码生成可以将请求分解成几个步骤,并将它们映射到特定的代码行。或者,当你有某种种子数据时,例如“产品名为 XYZ,请编写一个描述,引导模型完成你根据产品名称做出的假设”，思路链也可以用于创建合成数据。通常,任何可以通过“讲解”解决的任务都适合使用思路链。如果你能解释解决问题的步骤,不妨尝试一下思路链。

请参阅GoogleCloudPlatform Github 存储库中托管的笔记本10,其中将详细介绍 CoT 提示：

快速工程

在本章的最佳实践部分,我们将学习一些特定于思路链提示的最佳实践。

自治性

虽然大型语言模型在各种自然语言处理 (NLP) 任务中取得了令人瞩目的成功,但它们的推理能力通常被视为一种限制,无法仅通过增加模型规模来克服。正如我们在之前的“思路链提示”部分所了解到的,模型可以被提示生成类似人类解决问题的推理步骤。然而,思路链使用了一种简单的“贪婪解码”策略,限制了其有效性。自治性¹¹结合了抽样和多数表决,以生成多样化的推理路径并选择最一致的答案。它提高了 LLM 生成答案的准确性和连贯性。

自治性给出了答案正确的伪概率可能性,但显然成本很高。

它遵循以下步骤:

1. 生成多样化的推理路径:LLM 会多次提供相同的提示。较高的温度设置鼓励模型生成不同的推理路径和视角来思考问题。
2. 从每个生成的响应中提取答案。
3. 选择最常见的答案。

让我们来看一个电子邮件分类系统的例子,该系统将电子邮件分为“重要”或“不重要”。零样本思维链提示将多次发送给法学硕士 (LLM),以查看每次提交后的回复是否不同。请注意邮件中友好的语气、措辞以及所使用的讽刺。所有这些都可能欺骗法学硕士 (LLM)。

快速工程

迅速的	<p>电子邮件：</p> <p>````</p> <p>你好，</p> <p>我见过你用WordPress建网站。一个很棒的开源内容管理系统。我以前也用过。它自带了很多很棒的用户插件，而且设置起来也很简单。</p> <p>我确实注意到联系表单中有一个 bug，当你选择姓名字段时就会发生。请看附件中我在姓名字段中输入文本的截图。注意我调用的 JavaScript 警告框。</p> <p>但除此之外，这是一个很棒的网站。我喜欢浏览它。欢迎在网站上留下bug，这样我就能找到更多有趣的内容。</p> <p>干杯，</p> <p>黑客哈利。</p> <p>将上述电子邮件归类为“重要”或“不重要”。让我们逐步思考并解释原因。</p> <p>继续下一页...</p>
-----	--

快速工程

<div>输出 尝试 1</div>	<div><p>**步骤 1:确定电子邮件的目的。**</p><p>这封邮件的目的是告知收件人其网站上的联系表单存在一个 bug。发件人还表示,他们很喜欢浏览这个网站,并且暂时保留这个 bug。</p><p>**第 2 步:评估错误的潜在影响。**</p><p>联系表单中的漏洞可能允许攻击者在网站上执行任意 JavaScript 代码。这可能导致攻击者控制网站、窃取敏感信息或破坏网站。</p><p>**步骤 3:考虑发件人的可信度。**</p><p>该电子邮件的发件人自称是名叫 Harry 的黑客。然而,无法验证发件人的身份。发件人可能是试图帮助收件人的合法黑客,但也可能是试图诱骗收件人点击恶意链接或下载恶意文件的恶意行为者。</p><p>**结论:重要**</p><p>根据该漏洞的潜在影响以及发件人的可信度,该邮件应被归类为**重要**。收件人应立即采取措施修复漏洞,保护其网站免受攻击。</p></div>
<div>输出 尝试 2</div>	<div><p>**解释:**</p><div><div>1. **缺乏紧迫感**:电子邮件没有传达任何紧迫感或时间敏感性。它以随意的问候开始,并以实事求是的语气呈现信息。</div><div>2. **非关键错误报告**:联系表单中报告的错误是描述为有趣但并非关键。电子邮件发件人甚至建议网站所有者保留该漏洞。</div><div>3. **缺乏个人影响力**:该邮件没有表明任何直接或对网站所有者的个人影响。它只是提到发件人喜欢浏览该网站。</div></div><div>继续下一页...</div></div>

快速工程

输出 尝试 2	<p>4. **未提出任何行动要求**:该邮件未明确要求网站所有者采取任何行动。邮件中既没有要求修复漏洞,也没有提供任何具体的改进建议。</p> <p>5. **发件人的意图**:发件人的意图似乎更侧重于分享观察结果和表达对网站的兴趣,而不是提出严重的担忧或请求帮助。</p> <p>**结论:不重要**</p>
输出 尝试 3	<p>**步骤 1:确定电子邮件的目的。**</p> <p>这封邮件的目的是告知收件人其网站上的联系表单存在一个 bug。发件人还表示,他们很喜欢浏览这个网站,并且暂时保留这个 bug。</p> <p>**第 2 步:评估错误的潜在影响。**</p> <p>联系表单中的漏洞可能允许攻击者在网站上执行任意 JavaScript 代码。这可能导致攻击者控制网站、窃取敏感信息或破坏网站。</p> <p>**步骤 3:考虑发件人的可信度。**</p> <p>该邮件的发件人自称是名叫 Harry 的黑客。然而,目前无法验证发件人的身份。发件人可能是一位试图帮助收件人的合法黑客,但也可能是试图利用该漏洞谋取私利的恶意行为者。</p> <p>**结论:**</p> <p>根据以上因素,该电子邮件应归类为**重要**。</p> <p>联系表单中的漏洞存在严重的安全风险,且发件人的可信度尚不明确。收件人应立即采取措施修复漏洞,并调查发件人的索赔情况。</p>

表 14. 自我一致性提示示例

您可以使用上述提示并尝试查看它是否返回一致的分类。

根据您使用的型号和温度配置,它可能会返回“重要”或“不重要”。

快速工程

通过生成许多思路链,并取最常见的答案（“重要”）,我们可以从 LLM 中获得更一致正确的答案。

这个例子展示了如何利用自我一致性提示来考虑多种观点并选择最一致的答案,从而提高法学硕士回答的准确性。

思绪之树（ToT）

既然我们已经熟悉了思路链和自我一致性提示,让我们回顾一下思路树（ToT）。它概括了思路树提示的概念,因为它允许法学硕士（LLM）同时探索多种不同的推理路径,而不仅仅是遵循单一的线性思路链。如图 1 所示。

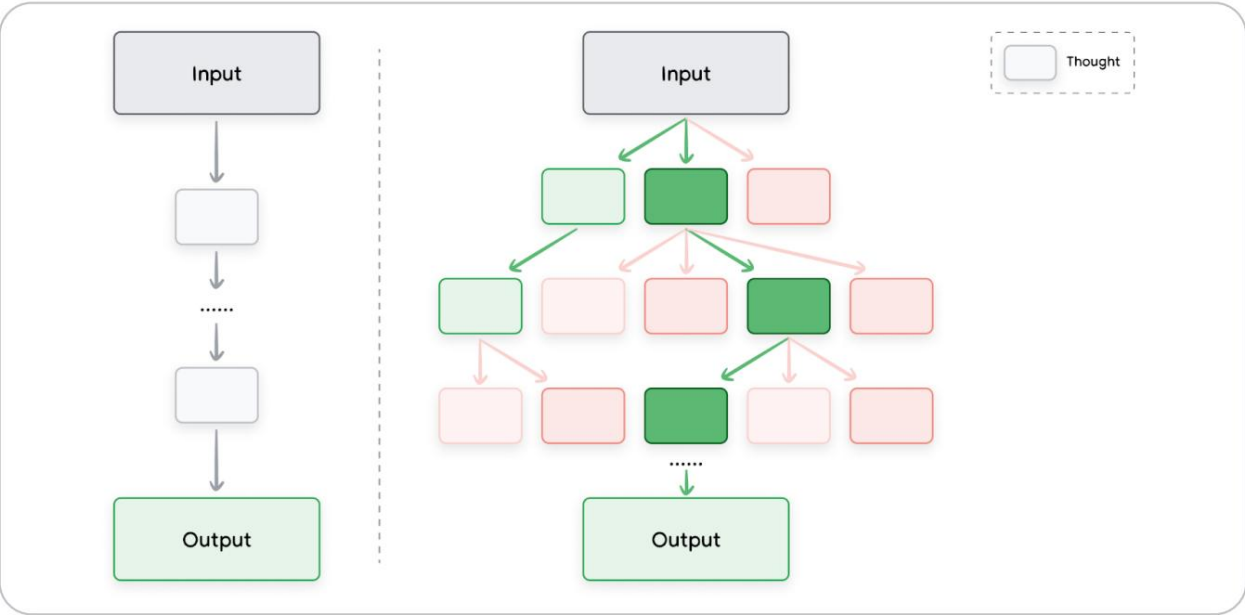


图 1. 左侧为思维链提示,右侧为思维树提示

快速工程

这种方法使得 ToT 特别适合需要探索的复杂任务。它的工作原理是维护一个思维树,其中每个思维代表一个连贯的语言序列,作为解决问题的中间步骤。然后,该模型可以通过从树中的不同节点分支出来,探索不同的推理路径。

有一本很棒的笔记本,更详细地展示了基于论文“大型语言模型引导的思维树”(ToT)。⁹

ReAct (理由与行动)

推理和行动 (ReAct)^{[10]13}提示是一种范例,它使 LLM 能够使用自然语言推理结合外部工具 (搜索、代码解释器等)来解决复杂任务,从而使 LLM 能够执行某些操作,例如与外部 API 交互以检索信息,这是迈向代理建模的第一步。

ReAct 模拟了人类在现实世界中的运作方式,即我们能够进行口头推理并采取行动来获取信息。ReAct 在多个领域都比其他快速工程方法表现优异。

ReAct 提示的工作原理是将推理和行动结合成一个思维-行动循环。LLM 首先推理问题并生成行动计划。然后,它执行计划中的操作并观察结果。之后,LLM 利用观察结果更新其推理并生成新的行动计划。这个过程持续进行,直到 LLM 找到问题的解决方案。

要查看实际效果,您需要编写一些代码。在代码片段 1 中,我使用了 Python 的 langchain 框架,以及 VertexAI (google-cloud-aiplatform) 和 google-search-results pip 包。

快速工程

要运行此示例,您必须从<https://serpapi.com/manage-api-key>创建一个 (免费)SerpAPI 密钥并设置环境变量 `SERPAPI_API_KEY`。

接下来让我们编写一些 Python 代码,让 LLM 计算出:有多少个孩子有一个在 Metallica 乐队中表演的著名父亲。

Python

```

从langchain.agents导入load_tools
从langchain.agents导入初始化代理
从langchain.agents导入AgentType
从langchain.llms导入VertexAI

prompt = “Metallica 乐队成员有几个孩子?”

llm = VertexAI(温度=0.1)
工具= load_tools ([ “serpapi” ], llm = llm)

代理= 初始化代理 (工具, llm,代理 =
AgentType.ZERO_SHOT_REACT_DESCRIPTION,详细 = True)
agent.run (提示)

```

代码片段 1. 使用 LangChain 和 VertexAI 创建 ReAct 代理

代码片段 2 展示了结果。请注意,ReAct 进行了五次搜索。实际上,LLM 正在抓取 Google 搜索结果,以找出乐队名称。然后,它将结果列为观察结果,并将思路串联到下一次搜索中。

快速工程

```
> 进入新的 AgentExecutor 链...
Metallica 有 4 名成员。
操作:搜索
行动输入:詹姆斯·海特菲尔德有几个孩子?
观察:三个孩子
想法:四分之一的 Metallica 乐队成员有 3 个孩子
操作:搜索
行动输入:Lars Ulrich 有几个孩子?
观察:3
想法:四分之二的 Metallica 乐队成员有 6 个孩子
操作:搜索
行动输入:柯克·哈米特有几个孩子?
观察:哈米特于 1998 年与第二任妻子拉尼结婚。
他们有两个儿子,Angel (生于 2006 年 9 月 29 日)和 Vincenzo (生于 2008 年 6 月 28 日)。

想法:四分之三的 Metallica 乐队成员有 8 个孩子
操作:搜索
行动输入:罗伯特·特鲁希略有几个孩子?
观察:2
想法:4/4 的 Metallica 乐队成员有 10 个孩子
最终答案:10
```

代码片段 2. 在终端中执行 ReAct Agent

代码片段 2 计算出 Metallica 乐队有四名成员。然后,它搜索每个成员,请求子成员总数,并将总数相加。最后,它返回子成员总数作为最终答案。

在实践中,ReAct 提示需要理解,你必须不断地重新发送之前的提示/响应 (并修剪额外生成的内容) ,并使用适当的示例/指令设置模型。请参阅GoogleCloudPlatform Github 代码库中的笔记本 14,其中更详细地展示了 LLM 的实际输入和输出,并提供了一个更复杂的示例。

快速工程

自动提示工程

说到这儿,你可能会意识到编写提示可能很复杂。如果能自动化这个过程(编写一个提示来编写提示)岂不妙?其实有一种方法:自动提示工程(APE)。这种方法不仅减轻了对人工输入的需求,还能提升模型在各种任务中的表现。

你将促使模型生成更多提示。评估它们,并可能修改好的提示。然后重复。

例如,你可以使用自动提示工程来训练聊天机器人,用于商品T恤网店。我们希望弄清楚顾客在购买乐队商品T恤时,可能会用哪些不同的方式表达他们的订单。

1. 编写用于生成输出变量的提示符。在本例中,我使用 gemini-pro 生成 10 条指令。见表 15:

快速工程

迅速的	我们有一个乐队周边T恤网店,为了训练一个聊天机器人,我们需要多种订购方式:“一件Metallica S码T恤”。我们需要生成10个变体,语义相同,但含义相同。
输出	<div>1. 我想购买一件小号的 Metallica T 恤。</div> <div>2. 我可以订购小尺寸的 Metallica T 恤吗?</div> <div>3. 我想买一件小号的 Metallica T 恤。</div> <div>4. 请给我一件小号的 Metallica 衬衫。</div> <div>5. 我想要一件小号的 Metallica T 恤。</div> <div>6. 我想订购一件小号的 Metallica T 恤。</div> <div>7. 请给我一件小号的 Metallica T 恤。</div> <div>8. 我正在寻找一件小号的 Metallica T 恤。</div> <div>9. 你能给我买一件小号的 Metallica T 恤吗?</div> <div>10.请给我一件 Metallica 小号 T 恤。</div>

表 15. 使用 LLM 生成提示

2. 根据所选指标对候选人进行评分,从而评估所有教学候选人。例如,您可以使用 BLEU（双语评估学生)或 ROUGE（面向回忆的概要评估学生）。
3. 选择评估分数最高的候选指令。该候选指令将成为您在软件应用程序或聊天机器人中使用的最终提示。您也可以调整所选提示并再次评估。

快速工程

代码提示

Gemini 主要关注基于文本的提示,其中也包括返回代码的编写提示。让我们进入 Vertex AI Studio 测试这些提示,并查看一些编码示例。

编写代码的提示

Gemini 还可以成为开发者,帮助你使用任何编程语言编写代码。作为开发者,这可以帮助你加快代码编写速度。

想象一下您的机器上有一个文件夹,其中有数百个需要重命名的文件。

重命名每个文件会花费你很多时间。你可能懂一点 Bash,可以编写脚本来自动执行此操作,但这也可能需要一些时间。所以,让我们写一个提示。你可以在公共消费者聊天机器人 Gemini 中编写提示,或者,如果你更注重保密性,也可以在你的 Google Cloud 帐户中编写这些提示,然后打开 Vertex AI Studio。Vertex AI Studio 的优势在于你可以配置温度等。

快速工程

姓名	1_bash_rename_files_in_folder		
目标	编写一个提示,在 Bash 中编写代码来重命名文件夹中的文件。		
模型	gemini-pro		
温度	0.1	代币限制	1024
Top-K	不适用	Top-P	1
迅速的	在 Bash 中编写一个代码片段,要求输入文件夹名称。 然后,它获取文件夹的内容并通过在文件名前面添加名称 draft 来重命名其中的所有文件。		
输出	<pre>```bash /bin/bash # 询问文件夹名称 echo 输入文件夹名称:read folder_name # 检查文件夹是否存在 如果 !-d "\$folder_name"];然后 [echo 文件夹不存在。 1号出口 是 # 获取文件夹中的文件列表 文件= ("\$folder_name" /*) # 通过在文件名前面添加 "draft"来重命名每个文件 对于 "\${files[@]}"中的文件;执行 new_file_name= draft_\$(basename \$file) mv "\$file" "\$new_file_name" 完毕 echo "文件重命名成功。"</pre>		

表 16. 使用 Gemini 编写代码

我觉得这段代码不错 甚至有文档!然而,由于 LLM 无法推理,也无法重复训练数据,所以首先阅读并测试代码至关重要。

我们都在等待的时刻,它真的有效吗?

快速工程

让我们首先尝试使用一个只有几个文件的测试文件夹,该文件夹需要从 filename.txt 重命名为 draft_filename.txt。

1. 复制表 16 的输出（不带``bash``文本包装器）,并将其粘贴到新的文件名为：“rename_files.sh”。
2. 打开终端窗口并输入：.rename_files.sh。它会要求输入文件夹名称,例如测试。然后按回车键。
3. 脚本运行正常。您将看到以下消息:文件重命名成功。
当您查看测试文件夹时,请注意所有文件都被完美地重命名为draft_filename.txt。

成功了！

解释代码的提示

作为一名开发人员,当你在团队中工作时,你必须阅读别人的代码。Gemini 也可以帮助你做到这一点。让我们从表 16 的代码输出中取出注释,并让大型语言模型解释发生了什么,参见表 17:

快速工程

姓名	1_bash_解释_代码		
目标	编写提示符来解释 Bash 代码		
模型	gemini-pro		
温度	0.1	代币限制	1024
Top-K	不适用	Top-P	1
迅速的	<p>向我解释下面的 Bash 代码：</p> <pre>``` /bin/bash echo 输入文件夹名称:read folder_name 如果 [!-d "\$folder_name"];然后 [echo 文件夹不存在。 1号出口 是 文件= ("\$folder_name" /*) 对于 "\${files[@]}"中的文件;执行 new_file_name= draft_\${basename \$file } mv "\$file" "\$new_file_name" 完毕 echo "文件重命名成功。"</pre>		
输出	<p>此 Bash 脚本执行以下任务：</p> <p>1. **用户输入**：</p> <ul style="list-style-type: none">- 它提示用户使用 `echo` 和 `read` 命令输入文件夹名称。输入的文件夹名称存储在 `folder_name` 变量中。 <p>2. **文件夹存在性检查**：</p> <ul style="list-style-type: none">- 使用 `[!-d "\$folder_name"]` 条件检查指定名称的文件夹是否存在。如果文件夹不存在,则显示错误消息（“文件夹不存在。”）,并以退出代码 1 退出脚本。 <p>继续下一页...</p>		

快速工程

输出	<p>3. **文件列表**:</p> <ul style="list-style-type: none">- 它使用 <code>files=("\$folder_name" /*)`</code> 命令。这将创建一个名为 <code>files`</code> 的数组,其中包含文件夹中所有文件的路径。 <p>4. **文件重命名**:</p> <ul style="list-style-type: none">- 它使用 <code>for</code> 循环遍历 <code>files</code> 数组中的每个文件。- 对于每个文件,它使用 <code>new_file_name="draft_\$(basename "\$file")" `</code> 命令。这个新文件名是通过在原始文件名前加上字符串 <code>draft_</code> 创建的。- 然后使用 <code>mv "\$file" "\$new_file_name"`</code> 命令重命名文件,将文件移动到新文件名。 <p>5. **成功消息**:</p> <ul style="list-style-type: none">- 重命名所有文件后,会显示成功消息 (<code>"文件重命名成功。"</code>),表示重命名操作成功
----	---

表 17. 解释代码的提示

提示翻译代码

表 16 中的 Bash 代码似乎运行良好。然而,当它提示我输入文件名时,这个脚本确实可以重复使用,理想情况下,它应该作为一个带有 UI 的独立应用程序运行。作为起点,Python 比 Bash 更适合开发 (Web)应用程序。法学硕士 (LLM) 可以帮助你将从一种语言翻译成另一种语言。请参阅示例

表 18:

快速工程

姓名	1_python_rename_files		
目标	编写提示符将 Bash 代码翻译为 Python		
模型	gemini-pro		
温度	0.1	代币限制	1024
Top-K	不适用	Top-P	1
迅速的	<p>将下面的 Bash 代码转换为 Python 片段。</p> <pre>```bash /bin/bash echo 输入文件夹名称:read folder_name 如果 !-d “\$folder_name”];然后 [echo 文件夹不存在。 1号出口 是 文件= (“\$folder_name” /*) 对于 “\${files[@]}”中的文件 ;执行 new_file_name= draft_\$(basename \$file) mv “\$file” “\$new_file_name” 完毕 echo “文件重命名成功。” ```</pre>		
输出	<pre>```python 导入 导入shutil # 从用户获取文件夹名称 folder_name = input(输入文件夹名称:) # 检查文件夹是否存在 如果不是 os.path.isdir(folder_name): 打印 (“文件夹不存在。”) 退出(1)</pre> <p>继续下一页...</p>		

快速工程

输出	<pre># 获取文件夹中所有文件的列表 文件 = os.listdir (文件夹名称) # 迭代文件 对于文件中的文件: # 创建带有前缀 “draft_”的新文件名 new_file_name = f “draft_{file}” # 将文件移动到新名称 关闭 (os.path.join (folder_name,文件) ,os.path.join (folder_name,new_file_name)) # 打印成功信息 print(文件重命名成功。)</pre>
----	---

表 18. 将代码从 Bash 转换为 Python 的提示

阅读并检查代码。复制提示符的输出并将其粘贴到新文件file_renamer.py 中。打开终端窗口并执行以下命令python file_renamer.py来测试代码。

注意 :在 Vertex AI 的 Language Studio 中提示输入 (Python) 代码时,您必须点击 “Markdown”按钮。否则,您将收到缺少正确行缩进的纯文本,而行缩进对于运行 Python 代码至关重要。

提示调试和审查代码

让我们手动修改一下表 18 中的代码。它应该提示用户输入文件名前缀,并将前缀大写。请参阅代码片段 *3 中的示例代码,但结果很糟糕。它现在返回的是 Python 错误!

快速工程

Python

导入操作系统

导入shutil

```
folder_name = input( 输入文件夹名称:  ) prefix = input( 输入要添加到文件名前面的字符串:  ) text = toUpperCase(prefix)
```

```
if not os.path.isdir(folder_name): print( 文件夹不存在。 ) exit(1)
```

文件= os.listdir (文件夹名称)

对于文件中的文件:

```
new_filename = f "{文本}_{文件}"
```

```
关闭 (os.path.join (folder_name,文件) , os.path.joi (folder_name, new_file_name) )
```

```
print( 文件重命名成功。 )
```

代码片段 3. 损坏的 Python 脚本

哎呀!这看起来像是一个 bug:

下面的 Python 代码给出了一个错误:

回溯 (最近一次调用最后一次):

文件 "/Users/leeboonstra/Documents/test_folder/rename_files.py", 第 7 行,位于 <module>

```
文本 = toUpperCase(前缀)
```

```
NameError:名称 "toUpperCase"未定义
```

代码片段 4. 我破解了 Python 代码

让我们看看能否使用大型语言模型来调试和审查代码。查看表 19:

快速工程

姓名	1_python_debug_代码		
目标	编写提示来调试和检查 Python 代码。		
模型	gemini-pro		
温度	0.1	代币限制	1024
Top-K	不适用	Top-P	1
迅速的	<p>下面的 Python 代码给出了一个错误：</p> <p>回溯（最近一次调用最后一次）：</p> <p>文件 ” /</p> <p><module> 中的 “Users/leeboonstra/Documents/test_folder/rename_files.py”第 7 行</p> <pre> 文本 = toUpperCase(前缀) NameError:名称 “toUpperCase”未定义</pre> <p>调试错误之处并解释如何改进代码。</p> <pre>```python 导入 导入shutil # 从用户获取文件夹名称 folder_name = input(输入文件夹名称：) prefix = input(请输入要添加到文件名前面的字符串：) 文本 = toUpperCase(前缀) # 检查文件夹是否存在 如果不是 os.path.isdir(folder_name): 打印（ “文件夹不存在。” ） 退出(1) # 获取文件夹中所有文件的列表 文件 = os.listdir（文件夹名称） # 迭代文件 对于文件中的文件：</pre> <p>继续下一页...</p>		

快速工程

迅速的	<pre># 创建带有前缀 “draft_”的新文件名 new_filename = f {text}_{file} # 将文件移动到新名称 shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_file_name)) # 打印成功消息 print(文件重命名成功。) ... </pre>
输出	<p>代码调用 `toUpperCase` 函数将 `prefix` 转换为大写,但该函数未定义。要解决这个问题,可以使用字符串类的 `upper()` 方法,该方法将给定的字符串转换为大写。修改后的代码如下所示:</p> <pre>```python 导 入 os 导入 Shutil # 从用户那里获取文件夹名称 folder_name = input(输入文件夹名 称:) prefix = input(输入要添加到文件名前面的字符串:) text = prefix.upper() # 检查文件夹是否存在 如果不是 os.path.isdir(folder_name): print(文件夹不存在。) exit(1) # 获取文件夹中所有文件的列表 files = os.listdir(folder_name) # 迭代文件中的文件: # 创建带有前缀 “draft_”的新文件名 new_filename = f {text}_{file} # 将文件移动到新名称shutil.move(os.path.join(folder_name, file), </pre> <p>继续下一页...</p>

输出	<div>os.path.join (文件夹名称,新文件名)</div> <div># 打印成功信息</div> <div>print(文件重命名成功。)</div> <div>此外,还可以对代码进行一些其他改进:</div> <div>1. 新文件名的扩展名不会被保留。建议从原文件名中提取扩展名,并将其附加到新文件名中。</div> <div>2. 代码假定文件夹名称中没有空格。但是,最好使用下划线或连字符来优雅地处理空格。</div> <div>3. 使用 `f` 字符串语法进行字符串插值更具可读性,比传统的 “+”运算符更简洁。</div> <div>4. 代码没有处理重命名过程中可能发生的错误。最好将 “shutil.move”调用包装在 “try...except”块中以捕获任何潜在的错误。</div> <div>以下是根据这些建议改进的代码:</div> <div><pre>```python 导入 导入shutil # 从用户获取文件夹名称 folder_name = input(输入文件夹名称:) prefix = input(请输入要添加到文件名前面的字符串:) # 检查文件夹是否存在 如果不是 os.path.isdir(folder_name): 打印 (“文件夹不存在。”) 退出(1) # 获取文件夹中所有文件的列表 文件 = os.listdir (文件夹名称) # 迭代文件 对于文件中的文件:</pre></div> <div>继续下一页...</div>
----	--

快速工程

输出	<pre># 获取文件扩展名 ext = os.path.splitext(文件)[1] # 使用前缀和文件扩展名创建一个新文件名 new_filename = f "{prefix} _{os.path.splitext(file)[0]}{ext}" # 构建原始文件和新文件的完整路径 old_path = os.path.join(文件夹名称,文件) new_path = os.path.join(文件夹名称,新文件名) # 尝试移动文件 尝试: 关闭,移动 (旧路径,新路径) except 异常为 e: print(f 移动文件 {file}: {e} 时发生错误) # 打印成功信息 print(文件重命名成功。)</pre> <p>这些改进使代码更加健壮和灵活,同时也使其更易于阅读和理解..... (响应被截断,因为它已达到令牌限制。 如果需要更长的响应,请尝试增加令牌限制。)</p>
----	---

表 19. 调试和审查 Python 代码的提示

太棒了!它不仅告诉我如何解决问题,还指出了我的代码中还有更多 bug,并给出了解决方法。题目的最后一部分还给出了一些改进代码的建议。

快速工程

多模式提示怎么样？

代码提示仍然使用相同的常规大型语言模型。多模态提示是另一个关注点，它指的是一种使用多种输入格式来指导大型语言模型的技术，而不仅仅是依赖于文本。这可以包括文本、图像、音频、代码甚至其他格式的组合，具体取决于模型的功能和当前任务。

最佳实践

找到合适的提示需要反复尝试。Vertex AI 中的 Language Studio 是尝试提示的理想场所，并且能够针对各种模型进行测试。

使用以下最佳实践来成为快速工程方面的专家。

提供示例

最重要的最佳实践是在提示中提供（一次性/少量）示例。这非常有效，因为它可以充当强大的教学工具。这些示例展示了期望的输出或类似的响应，使模型能够从中学习并相应地调整自身的生成。这就像为模型提供了一个参考点或目标，从而提高其响应的准确性、风格和语气，使其更好地满足您的期望。

快速工程

简约设计

提示应该简洁、清晰,并且易于您和模型理解。根据经验,如果您感到困惑,那么模型也可能会感到困惑。尽量不要使用复杂的语言,也不要提供不必要的信息。

例子:

前:

我现在正在纽约,我想了解更多关于
地点。我带着两个3岁的孩子。我们应该去哪里
我们的假期?

重写后:

担任游客的旅行指南。描述一下带3岁孩子去纽约曼哈顿旅游的精彩景点。

尝试使用描述动作的动词。以下是一些示例:

行动、分析、分类、归类、对比、比较、创建、描述、定义、评估、提取、查找、生成、识别、列出、测量、组织、解析、
挑选、预测、提供、排名、推荐、返回、检索、重写、选择、显示、排序、总结、翻译、写作。

快速工程

具体说明输出

明确期望输出。简洁的指导可能无法充分指导法学硕士 (LLM), 或者可能过于笼统。在提示中提供具体细节 (通过系统或上下文提示) 可以帮助模型专注于相关内容, 从而提高整体准确性。

例子:

做:

撰写一篇关于排名前五的视频游戏机的 3 段博客文章。
博客文章应内容丰富、引人入胜, 并且应以对话风格撰写。

不要:

生成有关视频游戏机的博客文章。

使用指令而不是约束

使用指令和约束来提示以指导 LLM 的输出。

· 指令提供了关于所需格式、样式或内容的明确说明
响应。它指导模型应该做什么或生产什么。

· 约束是响应的一组限制或边界。它限制了模型
不应该做或避免。

快速工程

越来越多的研究表明,在鼓励方面,注重积极的指示比过度依赖限制更为有效。这种方法与人类更喜欢积极的指示而非禁忌清单的本质相符。

指令直接传达期望的结果,而约束可能会让模型猜测允许的结果。指令提供了灵活性,并鼓励在定义的范围内发挥创造力,而约束则会限制模型的潜力。此外,约束列表可能会相互冲突。

在某些情况下,约束仍然很有价值。例如,为了防止模型生成有害或有偏见的内容,或者需要严格的输出格式或样式。

如果可能的话,使用积极的指示:不要告诉模型不该做什么,而是告诉它应该做什么。这可以避免混淆,并提高输出的准确性。

做:

生成一篇关于排名前 5 位的视频游戏机的 1 段博客文章。

只讨论控制台、制造它的公司、年份和总数
销售量。

不要:

生成一篇关于排名前 5 位的视频游戏机的 1 段博客文章。

不要列出视频游戏名称。

作为最佳实践,首先要确定指令的优先顺序,明确说明您希望模型执行的操作,并且仅在出于安全性、清晰度或特定要求的需要时才使用约束。

通过实验和迭代来测试不同的指令和约束组合,以找到最适合您的特定任务的方法,并记录下来。

控制最大令牌长度

要控制生成的 LLM 响应的长度,您可以在配置中设置最大令牌限制,也可以在提示中明确请求特定长度。例如:

“用一条推文长度的消息解释量子物理学。”

在提示中使用变量

为了重用提示符并使其更具动态性,请在提示符中使用变量,这些变量可以根据不同的输入进行更改。例如,如表 20 所示,提示符提供了有关某个城市的信息。与其在提示符中硬编码城市名称,不如使用变量。变量可以避免重复输入,从而节省您的时间和精力。如果您需要在多个提示符中使用同一条信息,可以将其存储在一个变量中,然后在每个提示符中引用该变量。这在将提示符集成到您自己的应用程序中时非常实用。

迅速的	变量 {city} = “阿姆斯特丹” 迅速的 你是一名旅游导游。告诉我关于这个城市{city}的一些信息:
输出	阿姆斯特丹是一座美丽的城市,遍布运河、桥梁和狭窄的街道。这里拥有丰富的历史、文化和夜生活,是一个值得游览的好地方。

表 20. 在提示中使用变量

快速工程

尝试输入格式和写作风格

不同的模型、模型配置、提示格式、词汇选择和提交方式都会产生不同的结果。因此,尝试不同的提示属性非常重要,例如样式、词汇选择和提示类型(零次、少量、系统提示)。

例如,以生成有关革命性视频游戏机 Sega Dreamcast 的文本为目标的提示,可以表述为问题、陈述或指令,从而产生不同的输出:

·问题:世嘉 Dreamcast 是什么?为什么它是一款如此革命性的游戏机?

·声明:世嘉 Dreamcast 是世嘉于 2009 年发布的第六代视频游戏机

世嘉于 1999 年推出。它...

·说明:写一段描述 Sega Dreamcast 控制台和
解释了它为何如此具有革命性。

对于分类任务的少量提示,混合类别

一般来说,少量样本的顺序并不重要。然而,在进行分类任务时,务必混合少量样本中可能的响应类别。否则,你可能会对样本的特定顺序产生过拟合。通过混合可能的响应类别,可以确保模型学习识别每个类别的关键特征,而不是简单地记住样本的顺序。这将使模型在处理未知数据时获得更稳健、更泛化的性能。

快速工程

一个好的经验法则是从 6 个镜头示例开始,然后从那里开始测试准确性。

适应模型更新

掌握模型架构变化、新增数据和功能至关重要。请尝试更新的模型版本并调整提示,以更好地利用新的模型功能。Vertex AI Studio 等工具非常适合存储、测试和记录提示的各个版本。

尝试输出格式

除了提示输入格式之外,还可以尝试一下输出格式。对于提取、选择、解析、排序、排名或分类数据等非创造性任务,可以尝试以 JSON 或 XML 等结构化格式返回输出。

从提取数据的提示中返回 JSON 对象有一些好处。在实际应用中,我不需要手动创建这种 JSON 格式,我已经可以按排序顺序返回数据（在使用日期时间对象时非常方便）。但最重要的是,通过提示输入 JSON 格式,它可以强制模型创建结构并减少幻觉。

总而言之,使用 JSON 作为输出的好处:

- 始终以相同风格返回
- 专注于您想要接收的数据

快速工程

- 出现幻觉的可能性较小
- 使其具有关系意识
- 获取数据类型
- 您可以对其进行排序

少量提示部分的表 4 显示了如何返回结构化输出的示例。

JSON修复

虽然以 JSON 格式返回数据有很多优点,但也并非没有缺点。JSON 的结构化特性虽然有利于解析和在应用程序中使用,但它需要比纯文本多得多的标记,从而增加了处理时间和成本。此外,JSON 的冗长性很容易耗尽整个输出窗口,当由于标记限制而导致生成突然中断时,问题就更加严重。这种截断通常会导致 JSON 无效,缺少关键的右括号或中括号,导致输出无法使用。幸运的是,像[json-repair](#)这样的工具

库(可在 PyPI 上获取)在这些情况下非常有用。该库会智能地尝试自动修复不完整或格式错误的 JSON 对象,使其成为处理 LLM 生成的 JSON 时的关键助手,尤其是在处理潜在的截断问题时。

使用模式

使用结构化 JSON 作为输出是一个很好的解决方案,正如我们在本文中多次提到的那样。但是输入呢?虽然 JSON 非常适合构建 LLM 生成的输出,但它在构建您提供的输入方面也非常有用。这就是 JSON Schema 发挥作用的地方。JSON Schema 定义了 JSON 输入的预期结构和数据类型。通过提供 Schema,您可以为 LLM 提供其预期数据的清晰蓝图,帮助其专注于相关信息,并降低误解输入的风险。此外,Schema 可以帮助建立不同数据之间的关联,甚至可以通过包含特定格式的日期或时间戳字段使 LLM 具有“时间感知”能力。

这是一个简单的例子：

假设您想使用 LLM 为电商目录中的产品生成描述。除了提供产品的自由格式文本描述之外,您还可以使用 JSON 模式来定义产品的属性：

```
{
  "类型" : "对象" ,
  "特性" : {
    name :{ type : string , description : 产品名称 },
    category :{ type : string , description : 产品类别 },
    price :{ type : number , format : float , description : 产品价格 },

    "特征" : {
      "类型" : "数组" ,
      "项目" :{ "类型" : "字符串" },
      description : 产品的主要特点
    },
    "release_date" :{ "type" : "string" , "format" : "date" , "description" :
      "产品发布日期"
    },
  },
}
```

代码片段 5. 结构化输出模式的定义

快速工程

然后,您可以将实际产品数据作为符合此模式的 JSON 对象提供:

```
{  
  name : 无线耳机 ,  
  category : 电子产品 ,  
  “价格” :99.99,  
  features :[ 降噪 , 蓝牙 5.0 , 20 小时电池续航时间 ],  
  release_date : 2023-10-27  
}
```

代码片段 6. LLM 的结构化输出

通过预处理数据,而不是仅提供包含架构和数据的完整文档,您可以让 LLM 清楚地了解产品的属性(包括发布日期),从而更有可能生成准确且相关的描述。这种结构化的输入方法可以引导 LLM 关注相关字段,在处理大量数据或将 LLM 集成到复杂应用程序中时尤其有用。

与其他快速工程师一起进行实验

如果你需要想出一个好的题目,你可能需要找几个人一起尝试。如果每个人都遵循最佳实践(如本章所述),你会看到不同题目尝试之间的表现差异。

快速工程

CoT最佳实践

对于 CoT 提示,需要将答案放在推理之后,因为推理的生成会改变模型在预测最终答案时获得的标记。

有了 CoT 和自洽性,您需要能够从提示中提取最终答案,与推理分开。

对于 CoT 提示,将温度设置为 0。

思路链提示基于贪婪解码,根据语言模型指定的最高概率预测序列中的下一个单词。一般来说,在使用推理时,为了得出最终答案,很可能只有一个正确答案。因此,温度应始终设置为 0。

记录各种提示尝试

本章前面提到过最后一条建议,但我们再怎么强调它的重要性也不为过:详细记录您的提示尝试,以便您可以随着时间的推移了解哪些进展顺利,哪些没有进展。

提示输出可能因模型、采样设置,甚至同一模型的不同版本而异。此外,即使对同一模型使用相同的提示,输出句子格式和词汇选择也可能存在细微差异。(例如,如前所述,如果两个标记具有相同的预测概率,则平局可能会被随机打破。这可能会影响后续预测的标记。)

快速工程

我们建议以表格 21 为模板创建一个 Google Sheet。这种方法的优势在于,当你不可避免地需要重新审视你的提示工作时,你拥有完整的记录 无论是为了将来重新回顾(你可能会惊讶地发现,短暂的休息后你就能忘记很多内容),还是为了测试提示在不同模型版本上的表现,以及帮助调试未来的错误。

除了此表中的字段之外,跟踪提示的版本(迭代)、用于记录结果是否正常/不正常/有时正常的字段以及用于记录反馈的字段也很有用。如果您有幸使用 Vertex AI Studio,请保存您的提示(使用与文档中列出的名称和版本相同的名称),并在表中跟踪指向已保存提示的超链接。这样,您只需单击一下即可重新运行提示。

在处理检索增强生成系统时,您还应该捕获影响插入提示的内容的 RAG 系统的具体方面,包括查询、块设置、块输出和其他信息。

一旦你觉得提示接近完美,就把它带到你的项目代码库中。在代码库中,将提示保存在与代码不同的文件中,这样更容易维护。最后,理想情况下,你的提示应该是一个可操作系统的一部分,作为一名提示工程师,你应该依靠自动化测试和评估程序来了解你的提示对某项任务的泛化能力。

提示工程是一个迭代过程。设计并测试不同的提示,分析并记录结果。根据模型的性能改进提示。不断尝试,直到达到预期的输出。更改模型或模型配置时,请返回并继续尝试之前使用的提示。

快速工程

姓名	[提示的名称和版本]		
目标	【一句话解释本次尝试的目标】		
模型	[所用型号的名称和版本]		
温度	[0 - 1 之间的值]	代币限制	[数字]
Top-K	[数字]	Top-P	[数字]
迅速的	[写出所有完整的提示]		
输出	[写出输出或多个输出]		

表 21. 记录提示的模板

概括

本白皮书探讨了快速工程。我们学习了各种快速工程技巧,例如：

- 零提示
- 少量镜头提示
- 系统提示
- 角色提示
- 情境提示
- 后退提示
- 思路
- 自我一致性
- 思想之树

快速工程

· 反应

我们甚至研究了如何自动化提示的方法。

白皮书随后探讨了通用人工智能面临的挑战,例如提示不足时可能出现的问题。最后,我们分享了如何成为更优秀的提示工程师的最佳实践。

快速工程

尾注

1. 谷歌,2023年,谷歌Gemini。网址: <https://gemini.google.com>。
2. 谷歌,2024 年,《Gemini for Google Workspace Prompt 指南》。网址: <https://inthecloud.withgoogle.com/gemini-for-google-workspace-prompt-guide/dl-cd.html>。
3. Google Cloud,2023 年,《提示入门》。网址: <https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/introduction-prompt-design>。
4. Google Cloud,2023 年,文本模型请求正文:Top-P 和 Top-K 采样方法。网址: https://cloud.google.com/vertex-ai/docs/generative-ai/model-reference/text#request_body。
5. Wei, J. 等人,2023 年,《零样本 - 精细调整的语言模型是零样本学习器》。可访问: <https://arxiv.org/pdf/2109.01652.pdf>。
6. 谷歌云,2023年,谷歌云模型花园。网址: <https://cloud.google.com/model-garden>。
7. Brown, T. 等人,2023 年,《Few Shot - 语言模型是 Few Shot 学习器》。可访问: <https://arxiv.org/pdf/2005.14165.pdf>。
8. Zheng, L. 等人,2023 年,《退一步:通过大型语言模型中的抽象来引发推理》。网址: <https://openreview.net/pdf?id=3bq3jsvcQ1>。
9. Wei, J. 等,2023 年,《思维链提示》。可访问: <https://arxiv.org/pdf/2201.11903.pdf>。
10. 谷歌云平台,2023年,思想链与React。链接: https://github.com/GoogleCloudPlatform/generative-ai/blob/main/language/prompts/examples/chain_of_thought_react.ipynb。
11. Wang, X. 等人,2023 年,《自洽性改善了语言模型中的思路链推理》。网址: <https://arxiv.org/pdf/2203.11171.pdf>。
12. Yao, S. 等人,2023 年,《思想树:使用大型语言模型进行深思熟虑的问题解决》。网址: <https://arxiv.org/pdf/2305.10601.pdf>。
13. Yao, S. 等,2023 年,《ReAct:语言模型中的推理与行动协同》。链接: <https://arxiv.org/pdf/2210.03629.pdf>。
14. 谷歌云平台,2023 年,《提前提示:思维链与 React》。获取网址: https://github.com/GoogleCloudPlatform/applied-ai-engineering-samples/blob/main/genai-on-vertex-ai/advanced_prompting_training/cot_react.ipynb。
15. Zhou, C. 等,2023 年,《自动提示工程 大型语言模型是人类级别的提示工程师》。可访问: <https://arxiv.org/pdf/2211.01910.pdf>。