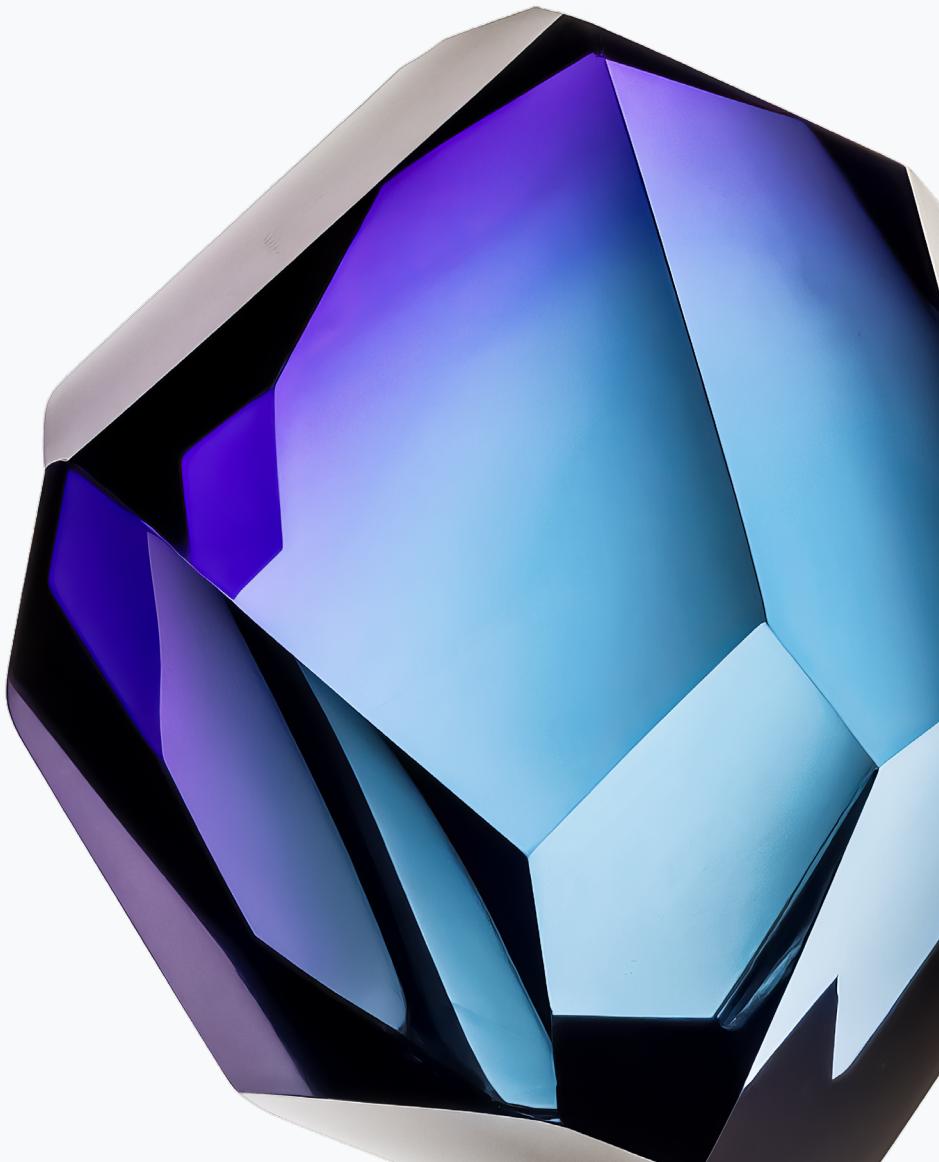


# 提示工 程

作者：Lee Boonstra



谷歌

## 致谢

### 内容贡献者

Michael Sherman

Yuan Cao

Erick Armbrust

Anant Nawalgaria

Antonio Gulli

Simone Cammel

### 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

### 技术文档撰写人

乔伊·海梅克

### 设计师

迈克尔·兰宁

# 目录

介绍	6
提示工程	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
整合所有内容	11
提示技巧	13
通用提示/零样本	13
单样本和少样本	15
系统、上下文和角色提示	18
系统提示	19
角色提示	21
上下文提示	23

后退提示	25
思维链 (CoT)	29
自我一致性	32
思维树 (ToT)	36
ReAct (推理 & 行动)	37
自动提示工程	40
代码提示	42
用于编写代码的提示	42
用于解释代码的提示	44
用于翻译代码的提示	46
用于调试和审查代码的提示	48
多模态提示怎么样？	54
最佳实践	54
提供示例	54
设计时要简洁	55
明确输出内容	56
使用指令而非约束	56
控制最大令牌长度	58
在提示中使用变量	58
尝试不同的输入格式和写作风格	59
对于分类任务的少量样本提示，混合类别	59
适应模型更新	60
尝试不同的输出格式	60

JSON修复	61
使用模式	62
与其他提示工程师一起实验	63
CoT 最佳实践	64
记录各种提示尝试	64
总结	66
尾注	68

你不需要成为数据  
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（有时伴随着其他形式的提示，如图像提示）是模型用来预测特定输出的输入。你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。然而，编写最有效的提示可能很复杂。提示的许多方面都会影响其效果：你使用的模型、模型的训练数据、模型配置、你的措辞、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。不充分的提示可能导致含糊不清、不准确的响应，并可能阻碍模型提供有意义的输出的能力。

当你与 Gemini 聊天机器人聊天时，你基本上是在编写提示，然而本白皮书侧重于在 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，你将可以访问配置，例如温度等。

本白皮书详细讨论了提示工程。我们将研究各种提示技术，以帮助你入门，并分享技巧和最佳实践，以成为提示专家。我们还将讨论你在编写提示时可能面临的一些挑战。

## 提示工程

记住 LLM 的工作原理；它是一个预测引擎。该模型将顺序文本作为输入，然后根据其训练数据预测下一个 token 应该是什么。LLM 被操作化以一遍又一遍地执行此操作，将先前预测的 token 添加到顺序文本的末尾，以预测下一个 token。下一个 token 预测基于先前 token 中的内容与 LLM 在训练期间所见内容之间的关系。

当你编写提示时，你试图设置 LLM 以预测正确的 token 序列。提示工程是设计高质量提示的过程，这些提示指导 LLM 产生准确的输出。此过程涉及调整以找到最佳提示，优化提示长度，并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 的上下文中，提示是提供给模型的输入，用于生成响应或预测。

这些提示可以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。

请随时参考谷歌的提示指南<sup>1</sup>，其中包含简单有效的提示示例。

在提示工程中，您将首先选择一个模型。无论您是在 Vertex AI 中使用 Gemini 语言模型，还是使用 GPT、Claude 或像 Gemma 或 LLaMA 这样的开源模型，都可能需要针对您的特定模型优化提示。

除了提示之外，您还需要调整 LLM 的各种配置。

## LLM 输出配置

一旦您选择了模型，您将需要确定模型配置。大多数 LLM 都带有各种配置选项，用于控制 LLM 的输出。有效的提示工程需要针对您的任务以最佳方式设置这些配置。

### 输出长度

一个重要的配置设置是在响应中生成的令牌数量。

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

减少 LLM 的输出长度并不会使 LLM 在其创建的输出中在文体或文本上更加简洁，它只会导致 LLM 在达到限制后停止预测更多令牌。如果您的需求需要较短的输出长度，您可能还需要设计您的提示以适应。

输出长度限制对于某些 LLM 提示技术尤其重要，例如 ReAct，在这种技术中，LLM 会在您想要的响应之后继续发出无用的令牌。

请注意，生成更多的令牌需要LLM进行更多的计算，从而导致更高的能源消耗和可能更慢的响应时间，进而导致更高的成本。

## 采样控制

LLM并非正式地预测单个令牌。相反，LLM预测下一个令牌可能是什么的概率，LLM词汇表中的每个令牌都获得一个概率。然后对这些令牌概率进行采样，以确定下一个生成的令牌将是什么。

温度、top-K和top-P是决定如何处理预测的令牌概率以选择单个输出令牌的最常见配置设置。

## 温度

温度控制令牌选择中的随机程度。较低的温度适用于期望更具确定性响应的提示，而较高的温度可能导致更多样化或意想不到的结果。温度为0（贪婪解码）是

确定性的：始终选择最高概率的令牌（但请注意，如果两个令牌具有相同的最高预测概率，具体取决于如何实现平局打破您可能不会总是以温度0获得相同的输出）。

接近最大值的温度往往会产生更随机的输出。并且随着温度越来越高，所有令牌都变得同样有可能成为下一个预测的令牌。

Gemini 的温度控制可以理解为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 ( $T$ )，强调以高确定性选择的单个首选温度。较高的 Gemini 温度设置类似于较高的 softmax 温度，使得所选设置周围更广泛的温度范围更易于接受。这种增加的不确定性适应了不需要严格、精确温度的场景，例如在尝试创意输出时。

## Top-K 和 top-P

Top-K 和 top-P（也称为核心采样）是 LLM 中使用的两种采样设置，用于限制预测的下一个 token 来自于预测概率最高的 token。与温度类似，这些采样设置控制着

生成文本的随机性和多样性。

- Top-K 采样从模型预测的分布中选择最有可能的  $K$  个 token。top-K 越高，模型输出的创造性和多样性就越高；top-K 越低，模型输出的限制性和事实性就越强。top-K 为 1 相当于贪婪解码。

- Top-P 采样选择累积概率不超过某个值 (P) 的前几个 token。P 的取值范围从 0 (贪婪解码) 到 1 (LLM 词汇表中的所有 token)。

选择 top-K 和 top-P 的最佳方式是同时尝试这两种方法 (或将两者结合使用) , 看看哪一种能产生你想要的结果。

## 整合在一起

选择 top-K、top-P、温度以及要生成的 token 数量, 取决于具体的应用和期望的结果, 并且这些设置都会相互影响。同样重要的是, 要确保你了解你选择的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 (如在 Vertex Studio 中) , 则满足 top-K 和 top-P 标准的 token 是下一个预测 token 的候选对象, 然后应用温度来从通过 top-K 和 top-P 标准的 token 中进行采样。如果只有 top-K 或 top-P 可用, 则行为相同, 但仅使用一个 top-K 或 P 设置。

如果温度不可用, 那么无论哪些 tokens 符合 top-K 和/或 top-P 标准, 都会被随机选择以生成单个下一个预测 token。

在一种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。

- 如果将温度设置为 0，则 top-K 和 top-P 变得无关紧要——最可能的 token 成为下一个预测的 token。如果将温度设置得非常高（高于 1——通常在 10 以上），则温度变得无关紧要，并且无论哪些 tokens 通过 top-K 和/或 top-P 标准，都会被随机采样以选择下一个预测的 token。
- 如果将 top-K 设置为 1，则温度和 top-P 变得无关紧要。只有一个 token 通过 top-K 标准，并且该 token 是下一个预测的 token。如果将 top-K 设置得非常高，比如设置为 LLM 词汇表的大小，那么任何具有非零概率成为下一个 token 的 token 都将满足 top-K 标准，并且不会选择任何一个。
- 如果将 top-P 设置为 0（或一个非常小的值），那么大多数 LLM 采样实现将只考虑最可能的 token 来满足 top-P 标准，从而使温度和 top-K 无关紧要。如果将 top-P 设置为 1，那么任何具有非零概率成为下一个 token 的 token 都将满足 top-P 标准，并且不会选择任何一个。

作为一个通用的起点，温度为 0.2，top-P 为 0.95，top-K 为 30 将为您提供相对连贯的结果，这些结果可能具有创造性，但不会过度。如果您想要特别有创意的结果，请尝试从温度 0.9、top-P 0.99 和 top-K 40 开始。如果您想要不太有创意的结果，请尝试从温度 0.1、top-P 0.9 和 top-K 20 开始。

最后，如果您的任务总是有一个正确的答案（例如，回答一个数学问题），请从温度 0 开始。

**注意:**随着更多的自由度（更高的温度、top-K、top-P 和输出 tokens），LLM 可能会生成不太相关的文本。

**警告:**您是否见过以大量填充词结尾的回复？这也被称为“重复循环错误”，这是大型语言模型中常见的问题，即模型陷入循环，重复生成相同的（填充）词、短语或句子结构，通常因不适当的温度和 top-k/而加剧

top-p 设置。这种情况可能发生在低温度和高温度设置下，尽管原因不同。在低温下，模型变得过于确定性，严格坚持最高概率路径，如果该路径重新访问先前生成的内容，可能会导致循环。文本。相反，在高温下，模型的输出变得过度随机，增加一个随机选择的词或短语偶然导致返回的概率回到之前的状态，由于大量可用选项而产生循环。在这两种情况下，模型的采样过程都会“卡住”，导致单调且无用的输出，直到填满输出窗口。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

大型语言模型（LLM）经过调整以遵循指令，并接受大量数据的训练，因此它们可以理解提示并生成答案。但大型语言模型并非完美；您的提示文本越清晰，大型语言模型预测下一个可能的文本就越好。此外，利用大型语言模型训练方式和工作原理的特定技术将帮助您从大型语言模型中获得相关的结果。

既然我们了解了什么是提示工程以及它需要什么，让我们深入研究一些最重要的提示技巧的例子。

### 通用提示/零样本

一个零样本提示是最简单的提示类型。它只提供任务的描述以及一些供LLM入门的文本。这个输入可以是任何东西：一个问题，一个故事的开头，或者指令。零样本这个名称代表“没有例子”。

让我们在 Vertex AI 中使用 Vertex AI Studio（用于语言），它提供了一个测试提示的场所。在表 1 中，您将看到一个零样本提示示例，用于对电影评论进行分类。

下面使用的表格格式是记录提示的好方法。您的提示在最终进入代码库之前可能会经历多次迭代，因此以一种有条理、结构化的方式跟踪您的提示工程工作非常重要。关于这种表格格式、跟踪提示工程工作的重要性以及提示开发过程的更多信息，请参见本章后面的“最佳实践”部分（“记录各种提示尝试”）。

模型温度应设置为较低的数字，因为不需要创造力，并且我们使用 `gemini-pro` 默认的 `top-K` 和 `top-P` 值，这实际上禁用了这两个设置（请参阅上面的“LLM 输出配置”）。注意生成的输出。单词 `disturbing` 和 `masterpiece` 应该使预测更加复杂，因为这两个词都用在同一个句子中。

名称	1_1_movie_classification		
目标	将电影评论分类为正面、中性或负面。		
模型	gemini-pro		
温度	0.1	令牌限制	5
Top-K	不适用	Top-P	1
提示	<p>将电影评论分类为正面、中性或负面。</p> <p>评论：“她”是一项令人不安的研究，揭示了如果人工智能持续进化且不受控制，人类将走向何方。我希望有更多像这部杰作一样的电影。</p> <p>情感：</p>		
输出	积极		

表1. 零样本提示的示例

当零样本不起作用时，你可以在提示中提供演示或示例，这会产生“单样本”和“少样本”提示。  
通用提示/零样本

## 单样本和少样本

在为AI模型创建提示时，提供示例很有帮助。这些示例可以帮助模型理解你在要求什么。当你希望引导模型输出某种特定的结构或模式时，示例尤其有用。

一个**单样本提示**，提供一个单一示例，因此得名单样本。其思想是模型有一个可以模仿的示例，以便最好地完成任务。

一个**少样本提示**<sup>7</sup> 提供 多个 示例给模型。这种方法向模型展示了它需要遵循的模式。这个想法与单样本类似，但所需模式的多个示例增加了模型遵循该模式的机会。

少样本提示所需的示例数量取决于几个因素，包括任务的复杂性、示例的质量以及你正在使用的生成式AI（gen AI）模型的能力。一般来说，你应该使用至少三到五个示例进行少样本提示。但是，对于更复杂的任务，你可能需要使用更多示例，或者由于模型的输入长度限制，你可能需要使用更少的示例。

表2显示了一个少样本提示示例，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求。

目标	将披萨订单解析为 JSON		
模型	gemini-pro		
温度	0.1	令牌限制	250
Top-K	不适用	Top-P	1
提示	<p>将客户的披萨订单解析为有效的 JSON：</p> <p>示例：</p> <p>我想要一个小号的披萨，配有奶酪、番茄酱和意大利辣香肠。</p> <p>JSON 响应：</p> <pre>``` {   "size": "small",   "type": "normal",   "ingredients": [["cheese", "tomato sauce", "peperoni"]] } ``` </pre>		
	下页继续...		

提示	<p>示例：</p> <p>我可以要一个大披萨，配番茄酱、罗勒和马苏里拉奶酪吗？</p> <pre>{   "size": "large",   "type": "normal",   "ingredients": [["tomato sauce", "bacon", "mozzarella"]] }</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": "large",   "type": "half-half",   "ingredients": [["cheese", "mozzarella"], ["tomato sauce", "ham", "pineapple"]] } ``` </pre>

表 2. 少样本提示示例

当您为提示选择示例时，请使用与您想要执行的任务相关的示例。这些例子应该是多样化的、高质量的，并且写得很好。一个小小的错误可能会混淆模型，并导致不希望的输出。

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

## 系统、上下文和角色提示

系统、上下文和角色提示都是用于指导大型语言模型生成文本的技术，但它们侧重于不同的方面：

- **系统提示**为语言模型设置整体上下文和目的。它定义了模型应该做什么的“大局”，例如翻译语言、对评论进行分类等。
- **上下文提示**提供与当前对话或任务相关的具体细节或背景信息。它帮助模型理解所提问题的细微之处，并相应地调整响应。
- **角色提示**为语言模型分配一个特定的角色或身份。  
这有助于模型生成与分配的角色及其相关的知识和行为一致的响应。

系统、上下文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以具有上下文。

但是，每种类型的提示都有略微不同的主要目的：

- **系统提示**：定义模型的基本能力和总体目的。
- **上下文提示**：提供即时、特定于任务的信息来指导响应。  
它高度特定于当前的任务或输入，这是动态的。
- **角色提示**：构建模型的输出风格和声音。它增加了一层特殊性和个性。

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

让我们深入了解这三种不同的提示。

## 系统提示

表3包含一个系统提示，我在其中指定了关于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，并且我指定了更高的令牌限制。然而，由于我对如何返回输出的明确指示，模型没有返回额外的文本。

目标	将电影评论分类为正面、中性或负面。		
模型	gemini-pro		
温度	1	令牌限制	5
Top-K	40	Top-P	0.8
提示	<p>将电影评论分类为正面、中性或负面。仅以大写形式返回标签。</p> <p>评论：“她”是一项令人不安的研究，揭示了如果人工智能被允许持续发展且不受控制，人类将走向何方。它太令人不安了，我看不下去。</p> <p>情感：</p>		
输出	负面		

表3. 系统提示的示例

系统提示可以用于生成满足特定要求的输出。名称“系统提示”实际上代表“向系统提供额外任务”。例如，您可以使用系统提示生成与特定编程语言兼容的代码片段，或者您可以使用系统提示返回某种结构。请查看表4，在那里我以JSON格式返回输出。

目标	将电影评论分类为积极、中立或消极，返回JSON。		
模型	gemini-pro		
温度	1	令牌限制	1024
Top-K	40	Top-P	0.8
提示	<p>将电影评论分类为积极、中立或消极。返回有效的JSON：</p> <p>评论：“她”是一项令人不安的研究，揭示了如果人工智能被允许持续发展且不受控制，人类将走向何方。它太令人不安了，我看不下去。</p> <p>模式： ```</p> <p>电影： { "情感": 字符串 "积极"   "消极"   "中立", "名称": 字符串 } 电影评论： { "movie_reviews": [电影] } ```</p> <p>JSON 响应：</p>		
输出	``` { "movie_reviews": [ { "sentiment": "负面", "name": "她" } ] } ```		

表 4. 使用 JSON 格式进行系统提示的示例

从提取数据的提示中返回 JSON 对象有一些好处。在一个实际应用中，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（在处理日期时间对象时非常方便），但最重要的是，通过提示使用 JSON 格式，它迫使模型创建一个结构并限制幻觉。

系统提示对于安全性和毒性也可能非常有用。要控制输出，只需在提示中添加一行，例如：‘你的回答应该具有尊重性。’

## 角色提示

角色提示是提示工程中的一种技术，它涉及为 gen AI 模型分配一个特定的角色。这可以帮助模型生成更相关和信息更丰富的输出，因为模型可以根据分配给它的特定角色来设计其响应。

例如，你可以角色扮演提示一个通用人工智能模型，让它成为图书编辑、幼儿园老师或励志演说家。一旦模型被分配了一个角色，你就可以给它提供特定于该角色的提示。例如，你可以提示一位老师创建一个他们可以复习的课程计划。请看表5，其中模型扮演的是旅行指南的角色。

目标	充当旅行指南，并提供 3 个旅行建议		
模型	gemini-pro		
温度	1	令牌限制	1024
Top-K	40	Top-P	0.8
提示	<p>我希望你充当旅行指南。我会给你写信，告诉你我的位置，然后你建议我附近可以参观的 3 个地方。在某些情况下，我还会告诉你我想参观的地方类型。</p> <p>我的建议："我在阿姆斯特丹，我只想参观博物馆。"</p> <p>旅行建议：</p>		
输出	<ol style="list-style-type: none"> <li>1. 荷兰国家博物馆：世界闻名的博物馆，以伦勃朗的《夜巡》等荷兰杰作而闻名。不要错过令人难以置信的绘画和文物收藏，这些展品展示了荷兰的艺术和历史。</li> <li>2. 梵高博物馆：沉浸在文森特·梵高的迷人作品中。该博物馆收藏了他最多的绘画和素描作品，包括《星夜》和《向日葵》。</li> <li>3. 阿姆斯特丹市立博物馆：探索来自世界各地的现代和当代艺术。博物馆的藏品包括毕加索、康定斯基和里特维尔德的作品，馆址位于一座引人注目的现代建筑中。</li> </ol>		

表 5. 角色提示示例

上面的例子展示了扮演旅行社代理人的例子。当您将角色更改为地理老师时，您会注意到您将收到不同的回复。

为 AI 模型定义角色视角，可以为其提供语气、风格和专注的蓝图  
您正在寻找的专业知识，以提高您的

输出的质量、相关性和有效性。

以下是一些我发现有效的风格，您可以从中选择：

对抗性的、描述性的、直接的、正式的、幽默的、有影响力的、非正式的、鼓舞人心的、有说服力的

让我们在表 6 中更改我们的提示，以包含幽默和鼓舞人心的风格。

提示	<p>我希望你扮演一名旅行指南。我会写信告诉你我的位置，你将以幽默的风格建议我附近 3 个值得参观的地方。</p> <p>我的建议：“我在曼哈顿。”</p> <p>旅行建议：</p>
输出	<ol style="list-style-type: none"><li>1. 看哪，帝国心态：登上帝国大厦令人眩晕的高度，沐浴在曼哈顿天际线的荣耀中。准备好感受自己像站在大苹果之巅的金刚，只不过少了那根巨猿尺寸的香蕉。</li><li>2. 在MoMA尽情享受艺术的熏陶：在现代艺术博物馆（MoMA）释放你内心的艺术爱好者。凝视那些让你困惑的杰作，让你质疑你的火柴人画作是否具有任何艺术价值。<p>在第五大道尽情购物：在标志性的第五大道上尽情享受购物的乐趣。准备好迎接价格冲击吧，在那些会让你的钱包哭泣的设计师精品店里橱窗购物（或者真的购物）。但是，嘿，你身处曼哈顿，所以不妨拥抱这种奢华吧！</p></li><li>3. 表6. 以幽默的语气和风格进行角色提示的示例</li></ol>

#### 情境提示

为一篇关于复古游戏的博客推荐文章

目标	背景：你正在为一个关于80年代复古街机视频游戏的博客写作。		
模型	gemini-pro		
温度	1	令牌限制	1024
Top-K	40	Top-P	0.8
提示	<p>建议3个可以写文章的主题，并用几行文字 描述这篇文章应该包含的内容。</p> <p>Suggest 3 topics to write an article about with a few lines of description of what this article should contain.</p>		
输出	<p>1. **街机柜体设计的演变：** 本文将探讨街机柜体设计的演变，从20世纪70年代早期的木质和金属柜体到20世纪80年代及以后的时尚霓虹灯设计。 **来自过去的冲击：80年代标志性街机游戏：**。本文将回顾20世纪80年代一些最具标志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、流行的机制和持久的魅力。</p> <hr/> <p>2. 表7. 上下文提示示例</p> <p>3. 后退提示</p>		

此模型使用了来自公开数据集的训练数据，但没有使用任何敏感或私人信息。这可能包括在训练过程中使用过的某些技术。这些技术可能在某些情况下对模型的性能产生负面影响。

通过关注一般原则而不是具体细节，它可以帮助减轻LLM响应中的偏差，后退提示。

is prompted directly.

It can help to mitigate biases in LLM responses, by focusing on general principles instead of specific details, step-back prompting.

让我们来看看这些例子，以了解后退提示如何改进结果。在将传统提示（表 8）与后退提示（表 9）进行比较之前，让我们先回顾一下传统提示：

目标	为一个第一人称射击游戏关卡编写一个故事情节。		
模型	gemini-pro		
温度	1	令牌限制	1024
Top-K	40	Top-P	0.8
提示	为一个新的第一人称射击游戏关卡编写一段故事情节，该关卡具有挑战性和吸引力。		
输出	<p>关卡开始时，玩家的小队在一个密集的城市区域遭到一个无情的敌对派系的伏击。玩家必须在迷宫般的后巷和破旧的建筑物中战斗，利用隐身和精确度来消灭敌军并避免被发现。玩家必须利用他们的环境来发挥优势，设置陷阱，利用掩护，并做出经过计算的行动才能生存。</p> <p>一路上，他们发现情报表明敌人正在计划一次毁灭性的袭击，玩家必须争分夺秒地收集证据并将其传回总部，赶在敌人实施计划之前。</p>		

表 8. 在我们将传统提示与后退提示进行比较之前的传统提示

当您将温度设置为 1 时，您可能会得到各种关于故事情节的创意写作，但它也相当随机和泛泛而谈。所以让我们退一步：

提示	基于流行的第一人称射击动作游戏，在第一人称射击游戏中，有哪些5个虚构的关键设置有助于具有挑战性和吸引力的关卡故事情节？
输出	<ol style="list-style-type: none"> <li>1. **废弃的军事基地**：一个庞大的、后末日时代的军事综合体爬满了变异士兵和流氓机器人，非常适合挑战枪战。</li> <li>2. **赛博朋克城市**：一个霓虹灯闪烁的未来城市环境，拥有高耸的摩天大楼和密集的巷道，以网络增强的敌人为特色以及黑客机制。</li> <li>3. **外星飞船**：一艘搁浅在地球上的巨大外星飞船，拥有怪异的走廊、零重力区域和需要遭遇的外星生物。</li> <li>4. **僵尸横行的城镇**：一个被成群的侵略性僵尸占领的荒凉城镇，具有激烈的近距离战斗和解谜，以寻找安全通道。</li> <li>5. **水下研究设施**：一个被水淹没的深海实验室，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。</li> </ol>

表 9. 自治性提示的一个例子

是的，这些主题似乎很适合第一人称射击游戏。让我们回到原始提示，但这次我们将后退步骤的答案作为上下文，看看它会返回什么。

提示	<p>上下文：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"> <li>1. **废弃的军事基地**：一个广阔的、后末日军事综合体，爬满了变异的士兵和流氓机器人，非常适合具有挑战性的枪战。</li> <li>2. **赛博朋克城市**：一个霓虹灯闪烁的未来城市环境，拥有高耸的摩天大楼和密集的巷道，以网络增强的敌人和黑客机制为特色。</li> <li>3. **外星飞船**：一艘巨大的外星飞船搁浅在地球上，拥有怪异的走廊、零重力区域和需要遭遇的地球外生物。</li> <li>4. **僵尸肆虐的小镇**：一个荒凉的小镇被成群的具有攻击性的僵尸所占据，以激烈的近距离战斗和解谜为特色，以寻找安全通道。</li> <li>5. **水下研究设施**：一个被水淹没的深海实验室，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。</li> </ol> <p>选择一个主题，为一个第一人称射击电子游戏的新关卡写一个段落的故事线，该关卡具有挑战性和吸引力。</p>
输出	<p>在浑浊深渊的中心，坐落着一个破旧的水下研究设施，它是人类雄心及其灾难性后果的证明。笼罩在黑暗中，伴随着故障机器的嗡嗡声，以及被淹没灵魂的回声所萦绕，该设施现在是一个源自深海的噩梦的滋生地。玩家，一名装备精良的精英海军陆战队员，配备先进的潜水装备和实验性武器，任务是冒险进入这个充满恐怖的水生领域。凭借韧性和足智多谋，他们必须在危险的走廊中穿梭，解决神秘的谜题，并面对潜伏在深处的可怕的海怪。从应对深海的巨大压力到智胜狡猾的水生掠食者，在这个未知的地下世界中的每一刻都在考验着人类的耐力和勇气。</p>

表10. 自我一致性提示的示例

这看起来像一个有趣的电子游戏！通过使用后退提示技术，您可以提高提示的准确性。

## 思维链 (CoT)

思维链 (CoT) 提示是一种通过生成 **中间推理步骤** 来提高 LLM 推理能力的技巧。这有助于 LLM 生成更准确的答案。您可以将其与少量样本提示相结合，以便在更复杂的任务上获得更好的结果，这些任务需要在响应之前进行推理，因为这是零样本思维链的一个挑战。

思维链 (CoT) 有很多优点。首先，它非常有效且无需费力，并且可以与现成的LLM（大型语言模型）很好地配合使用（因此无需进行微调）。通过思维链提示，您还可以获得可解释性，因为您可以从LLM的响应中学习，并了解所遵循的推理步骤。如果出现故障，您将能够识别它。当在不同的LLM版本之间移动时，思维链似乎可以提高稳健性。这意味着您的提示在不同LLM之间的性能漂移应该比您的提示不使用推理链时要小。当然，也有缺点，但它们在某种程度上是直观的。

LLM 的响应包括思维链推理，这意味着更多的输出 tokens，这意味着预测花费更多的钱并且需要更长的时间。

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。

提示	当我 3 岁时，我的伴侣的年龄是我的 3 倍。现在，我 20 岁了。我的伴侣多大了？
输出	63 岁

表 11. 试图解决数学问题的提示示例

哎呀。这显然是错误的答案。事实上，LLM 经常在数学任务中遇到困难，并且可能提供不正确的答案——即使是像乘以两个数字这样简单的任务。这是因为它们是在大量文本上训练的，而数学可能需要不同的方法。所以让我们看看中间推理步骤是否会改善 the 输出。

提示	当我 3 岁时，我的伴侣的年龄是我的 3 倍。现在，我 20 岁了。我的伴侣多大了？让我们一步一步地思考。
输出	<ol style="list-style-type: none"> <li>1. 让我们假设我目前的年龄是“x”岁。</li> <li>2. 当我 3 岁时，我的年龄是 3 岁。</li> <li>3. 我当时的伴侣的年龄是我年龄的3倍，这意味着我的伴侣的年龄是<math>3 * 3 = 9</math>岁。</li> <li>4. 现在，我20岁了，这意味着自从我3岁以来，我的年龄增长了<math>20 - 3 = 17</math>岁。</li> <li>5. 既然我3岁时，我的伴侣9岁，而且我的年龄增长了17岁，那么我的伴侣的年龄肯定也增长了17岁。</li> <li>6. 因此，我的伴侣现在的年龄是<math>9 + 17 = 26</math>岁。</li> </ol> <p>所以，我的伴侣26岁。</p>

表12. 思维链提示的一个例子

好了，现在最终答案是正确的。那是因为我们清楚地指示LLM解释每个步骤，而不是仅仅返回一个答案。有趣的是，模型会加上增加的17年。在我看来，我会计算我和我的伴侣之间的年龄差，然后加起来。 $(20+(9-3))$ 。让我们帮助模型更像我一样思考。

表 12 是一个“零样本”思维链的例子。当与单样本或少样本结合使用时，思维链提示可能非常强大，如表 13 所示：

提示	<p>问：我弟弟2岁时，我的年龄是他年龄的两倍。现在我40岁了。我弟弟多大了？让我们一步一步地思考。</p> <p>答：我弟弟2岁时，我的年龄是<math>2 * 2 = 4</math>岁。 这相差2岁，而且我年龄更大。现在我40岁了，所以我弟弟是<math>40 - 2 = 38</math>岁。答案是38。</p> <p>问：我3岁时，我的伴侣的年龄是我的3倍。现在，我20岁了。我的伴侣多大了？让我们一步一步地思考。</p> <p>答：</p>
输出	<p>我3岁时，我的伴侣的年龄是<math>3 * 3 = 9</math>岁。年龄相差6岁，我的伴侣年龄更大。现在我20岁了，所以我的伴侣是<math>20 + 6 = 26</math>岁。答案是26。</p>

表13. 单样本思维链提示的示例

思维链对于各种用例都很有用。考虑代码生成，将请求分解为几个步骤，并将这些步骤映射到特定的代码行。或者在您有某种种子时创建合成数据，例如“产品名为XYZ，编写一个描述，引导模型完成您基于给定标题的产品所做的假设。”通常，任何可以通过“讨论”解决的任务都是思维链的良好候选者。如果您可以解释解决问题的步骤，请尝试思维链。

请参考GoogleCloudPlatform Github存储库中托管的笔记本<sup>10</sup>，其中将详细介绍CoT提示：

在本章的最佳实践部分，我们将学习一些特定于思维链提示的最佳实践。

## 自治性

虽然大型语言模型在各种自然语言处理任务中表现出了令人印象深刻的成功，但它们的推理能力通常被视为一种局限性，无法仅通过增加模型大小来克服。正如我们在之前的思维链提示部分所学到的，可以提示模型生成像人类解决问题一样的推理步骤。然而，CoT 使用一种简单的“贪婪解码”策略，限制了其有效性。自治性 结合了抽样和多数投票，以生成不同的推理路径并选择最一致的答案。它提高了大型语言模型生成的响应的准确性和连贯性。

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. 生成不同的推理路径：为大型语言模型提供相同的提示多次。高温设置鼓励模型生成不同的推理路径和对问题的看法。
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类为重要或不重要。零样本思维链提示将被发送到大型语言模型多次，以查看每次提交后响应是否不同。请注意友好的语气、用词以及电子邮件中使用的讽刺意味。所有这些都可能欺骗大型语言模型。

提示	<p>EMAIL: ``` 嗨， 我看到你用WordPress搭建了你的网站。这是一个很棒的开源内容管理系统。我过去也用过。它有很多很棒的用户插件，而且设置起来也很容易。</p> <p>我确实注意到了联系表单中的一个错误，当你选择姓名栏时会发生。请看我输入姓名栏的截图。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。我很喜欢阅读它。你可以随意保留网站上的这个错误，因为它给了我更多有趣的东西可以阅读。</p> <p>干杯， 哈里·黑客。 ```</p> <p>将以上邮件分类为重要或不重要。让我们逐步思考并解释原因。</p> <p style="text-align: right;">下页继续...</p>
----	---

<p>输出 尝试 1</p>	<p><b>**步骤 1：确定电子邮件的目的。**</b> 这封邮件的目的是通知收件人他们网站联系表单中的一个错误。发件人还提到他们喜欢阅读该网站，并且暂时会保留这个错误。</p> <p><b>**步骤 2：评估该错误的潜在影响。**</b> 联系表单中的错误可能允许攻击者在网站上执行任意的JavaScript 代码。这可能导致攻击者能够控制网站、窃取敏感信息或破坏网站。</p> <p><b>**步骤 3：考虑发件人的可信度。**</b> 这封邮件的发件人声称自己是一个名叫哈利的黑客。然而，没有办法验证发件人的身份。发件人可能是一个合法的黑客，试图帮助收件人，但也有可能发件人是一个恶意行为者，试图欺骗收件人点击一个恶意链接或下载一个恶意文件。</p> <p><b>**结论：重要**</b> 根据该错误的潜在影响和发件人的可信度，这封邮件应被分类为<b>**重要**</b>。收件人应立即采取行动修复该错误，保护他们的网站免受攻击。</p>
<p>输出 尝试 2</p>	<p><b>**解释：**</b></p> <ol style="list-style-type: none"><li><b>**缺乏紧迫感：**</b> 这封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，并以陈述的语气呈现信息。</li><li><b>**非关键错误报告：**</b> 联系表格中报告的错误被描述为有趣但并不关键。电子邮件发送者甚至建议网站所有者可以保留该错误。</li><li><b>**缺乏个人影响：**</b> 这封邮件没有表明对网站所有者的任何直接或个人的影响。它只是提到发件人喜欢阅读该网站。</li></ol> <p style="text-align: right;"><a href="#">下页继续...</a></p>

输出 尝试 2	<p>4. **缺少行动请求：** 这封邮件没有明确要求网站所有者采取任何行动。它没有要求修复错误，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一个观察结果，并表达对网站的兴趣，而不是提出严重的担忧或请求帮助。 **结论：不重要**</p> <p>尝试 3</p>
输出 ——	<p>**步骤 1：确定电子邮件的目的。** 试图帮助收件人的黑客，但也可能发件人是</p> <p>**步骤 2：评估该错误的潜在影响。**联系表单中的错误可能允许攻击者在网站上执行任意的JavaScript 代码。这可能导致攻击者能够控制网站、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。** 这封邮件的发件人声称自己是一个名叫哈利的黑客。然而，没有办法验证发件人的身份。发件人可能是一个合法的 hacker who is trying to help the recipient, but it is also possible that the sender is 一个恶意行为者，试图利用该漏洞为自己谋取利益。</p> <p>**结论：** 基于以上因素，该邮件应被归类为**重要**。 联系表单中的漏洞是一个严重的安全风险，并且发件人的可信度未知。收件人应立即采取行动修复该漏洞，并且</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。

根据您使用的模型和温度配置，它可能返回“重要”或“不重要”。

通过生成许多思维链，并采用最常出现的答案（“重要”），我们可以从大语言模型中获得更一致的正确答案。

这个例子展示了自洽性提示如何通过考虑多个角度并选择最一致的答案来提高大型语言模型（LLM）响应的准确性。

## 思维树（ToT）

现在我们已经熟悉了思维链和自洽性提示，让我们回顾一下思维树（ToT）。<sup>12</sup>它概括了CoT提示的概念，因为它允许LLM同时探索多个不同的推理路径，而不仅仅是遵循单一的线性思维链。如图1所示。

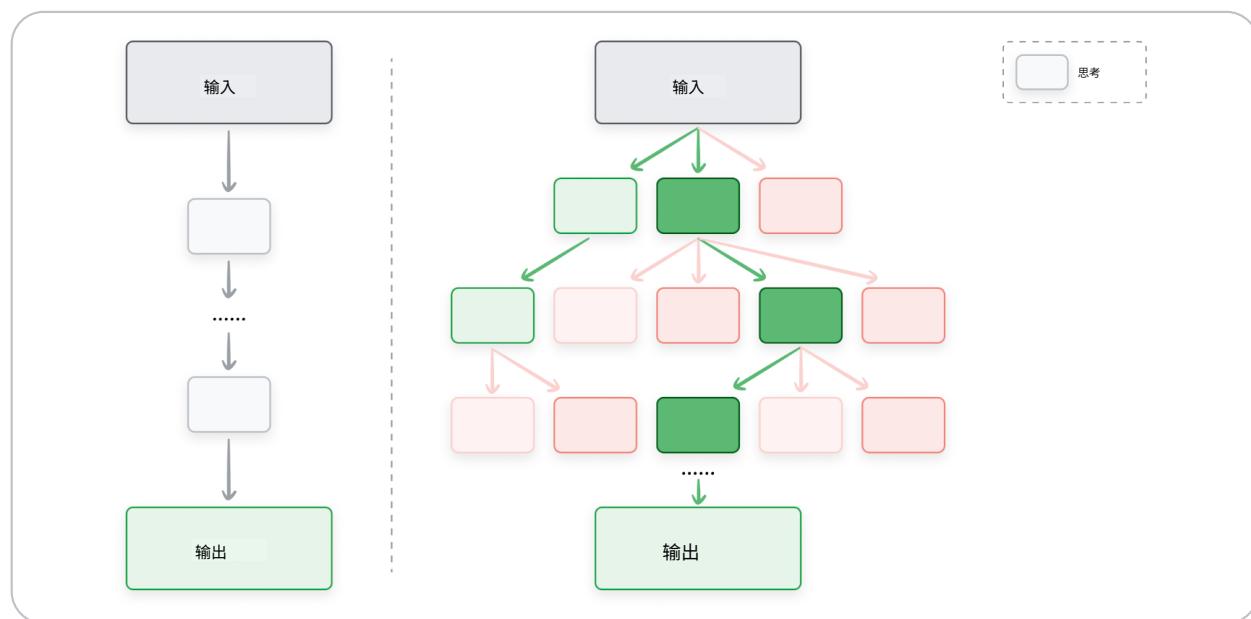


图1. 左侧为思维链提示的可视化，右侧为思维树提示的可视化

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，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。

有一个很棒的笔记本，更详细地介绍了基于论文“大型语言模型引导的思维树”的思维树（To T）。<sup>9</sup>

## ReAct（推理 & 行动）

推理和行动（ReAct）[10]<sup>13</sup>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）来解决复杂的任务，从而允许LLM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模仿了人类在现实世界中的运作方式，因为我们会进行口头推理，并且可以采取行动来获取信息。在各种领域中，ReAct的表现优于其他提示工程方法。

ReAct提示的工作原理是将推理和行动结合到一个思考-行动循环中。LLM首先对问题进行推理，并生成行动计划。然后，它执行计划中的行动并观察结果。LLM随后使用观察结果来更新其推理并生成新的行动计划。此过程一直持续到LLM找到问题的解决方案。

为了看到这一点，你需要编写一些代码。在代码片段 1 中，我使用 Python 的 langchain 框架，以及 VertexAI (google-cloud-aiplatform) 和 google-search-results pip 包。

要运行此示例，你必须从 <https://serpapi.com/manage-api-key> 创建一个（免费的）SerpAPI 密钥，并设置一个环境变量 SERPAPI\_API\_KEY。

接下来，让我们编写一些 Python 代码，让 LLM 来计算：Metallica 乐队中有多少位著名的父亲拥有孩子。

### Python

```
from langchain.agents import load_tools
from langchain.agents import initialize_agent
from langchain.agents import AgentType
from langchain.llms import VertexAI

prompt = "Metallica 乐队成员有多少个孩子？"

llm = VertexAI(temperature=0.1)
tools = load_tools(["serpapi"], llm=llm)

agent = initialize_agent(tools, llm,
                        agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True)
agent.run(prompt)
```

代码片段 1. 使用 LangChain 和 VertexAI 创建一个 ReAct 代理

代码片段 2 显示了结果。请注意，ReAct 进行了五个搜索的链式操作。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，并将想法链接到下一次搜索。

```
> 进入新的 AgentExecutor 链...
Metallica 有 4 名成员。
行动：搜索
行动输入：James Hetfield 有几个孩子？
观察：三个孩子
思考：1/4 Metallica 乐队成员有 3 个孩子
行动：搜索
行动输入：Lars Ulrich 有几个孩子？
观察：3
思考：2/4 Metallica 乐队成员有 6 个孩子
行动：搜索
行动输入：Kirk Hammett 有几个孩子？
观察：Hammett 自 1998 年以来一直与他的第二任妻子 Lani 结婚。
他们有两个儿子，安吉尔（生于2006年9月29日）和文森佐（生于2008年6月28日）。
思考：3/4的Metallica乐队成员有8个孩子
行动：搜索
行动输入：罗伯特·特鲁希略有几个孩子？
观察：2
思考：4/4的Metallica乐队成员有10个孩子
最终答案：10
```

## 片段2. 在终端中执行ReAct代理

代码片段2计算出Metallica乐队有四名成员。然后它搜索每位乐队成员，请求孩子的总数并加总。最后，它返回孩子的总数作为最终答案。

在实践中，ReAct提示需要理解，您必须不断地重新发送之前的提示/响应（并修剪掉多余的生成内容），并设置具有适当示例/说明的模型。请参考GoogleCloudPlatform Github存储库中托管的笔记本<sup>14</sup>，其中更详细地展示了实际的LLM输入和输出，并提供了一个更详细的示例。

## 自动提示工程

在这一点上，您可能会意识到编写提示可能很复杂。如果能够自动化这个过程（编写一个提示来编写提示）岂不是很好？嗯，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，而且还提高了模型在各种任务中的性能。

你将提示一个模型生成更多的提示。评估它们，可能会修改好的提示，然后重复这个过程。

例如，你可以使用自动提示工程，这将有助于训练一个用于商品 T 恤网店的聊天机器人。我们想要弄清楚客户可以用哪些不同的方式来表达他们购买乐队商品 T 恤的订单。

1. 编写提示，这将生成输出变体。在这个例子中，我使用 `gemini-pro` 来生成 10 条指令。见表 15：

提示	我们有一个乐队商品 T 恤网店，为了训练一个聊天机器人，我们需要各种订购方式：“一件 Metallica T 恤 S 码”。生成 10 个变体，具有相同的语义，但保持相同的含义。
输出	<ol style="list-style-type: none"> <li>1. 我想购买一件小号的 Metallica T 恤。</li> <li>2. 我可以订购一件小号的 Metallica T 恤吗？</li> <li>3. 我想买一件小号的 Metallica T 恤。</li> <li>4. 一件 Metallica 衬衫，小号，请。</li> <li>5. 我想要一件小号的 Metallica T 恤。</li> <li>6. 我想订购一件小号的 Metallica T 恤。</li> <li>7. 请来一件小号 Metallica T 恤。</li> <li>8. 我在找一件小号 Metallica T 恤。</li> <li>9. 能帮我拿一件小号 Metallica T 恤吗？</li> <li>10. 请来一件 Metallica 小号 T 恤。</li> </ol>

表 15. 使用 LLM 生成提示

2. 通过基于所选的指标对候选对象进行评分来评估所有指令候选对象。  
例如，您可以使用 BLEU（双语评估替补）或 ROUGE（面向召回的要点理解评估）。
3. 选择评估得分最高的指令候选者。此候选者将成为您可以在软件应用程序或聊天机器人中使用的最终提示。您也可以调整选择的提示并再次评估。

## 代码提示

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

### 编写代码的提示

Gemini 也可以成为一名开发者，并帮助你使用任何你选择的编程语言编写代码。作为一名开发者，这可以帮助你加快编写代码的过程。

想象一下你的机器上有一个文件夹，里面有数百个文件需要重命名。

重命名每个文件会花费你大量时间。你了解一点 Bash，并且可以编写一个脚本来自动化这个过程，但这可能也需要一段时间。所以让我们编写一个提示。你可以在公共消费者聊天机器人 Gemini 中编写提示，或者如果你更关心保密性，你可以在你的 Google Cloud 帐户中编写这些提示并打开 Vertex AI Studio。Vertex AI Studio 的优势在于你可以配置温度等。

名称	1_bash_rename_files_in_folder		
目标	编写一个提示，用于编写 Bash 代码来重命名文件夹中的文件。		
模型	gemini-pro		
温度	0.1	令牌限制	1024
Top-K	不适用	Top-P	1
提示	<p>编写一个 Bash 代码片段，该代码片段要求输入文件夹名称。</p> <p>然后，它获取文件夹的内容，并通过在文件名前面加上“draft”来重命名里面的所有文件。</p>		
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 if [ ! -d "\$folder_name" ]; then echo "文件夹不存在。" exit 1 fi  # 获取文件夹中的文件列表 files=( "\$folder_name"/* )  # 通过在文件名前面加上“draft”来重命名每个文件 for file in \${files[@]}; do new_file_name="draft_\$(basename \$file)" mv "\$file" "\$new_file_name" done  echo "文件已成功重命名。" ``` </pre>		

表 16. 使用 Gemini 编写代码

在我看来，这似乎是不错的代码——甚至还有文档！然而，由于大型语言模型无法进行推理，并且会重复训练数据，因此务必先阅读并测试你的代码。

我们都在等待的时刻，它真的有效吗？

让我们先用一个测试文件夹来试试，里面只有几个文件，需要将文件名从 `filename.txt` 重命名为 `draft_filename.txt`。

1. 复制表 16 中的输出（不包括 `bash` 文本包装器），并将其粘贴到一个新的文件中，命名为：“`rename_files.sh`”。
2. 打开一个终端窗口，输入：`./rename_files.sh`。它会要求输入一个文件夹名称，例如 `test`，然后按回车键。
3. 脚本似乎运行良好。您将看到消息：**Files renamed successfully.**（文件已成功重命名。）  
当您查看测试文件夹时，请注意所有文件都已完美地重命名为 `draft_filename.txt`。

成功了！

## 解释代码的提示

作为一名开发者，当你在团队中工作时，你必须阅读别人的代码。Gemini 也可以在这方面帮助你。让我们采用表 16 中的代码输出，移除注释，并要求大型语言模型解释发生了什么，见表 17：

名称	1_bash_explain_the_code		
目标	编写一个提示来解释 Bash 代码		
模型	gemini-pro		
温度	0.1	令牌限制	1024
Top-K	不适用	Top-P	1
提示	<p>请向我解释以下 Bash 代码：</p> <pre>``` #!/bin/bash  echo "输入文件夹名称：" read folder_name  if [ ! -d "\$folder_name" ]; then echo "文件夹不存在。" exit 1 fi  files=( "\$folder_name"/* )  for file in "\${files[@]}"; do new_file_name="draft_\$(basename "\$file")" mv "\$file" "\$new_file_name" done  echo "文件已成功重命名。" ``` </pre>		
输出	<p>这个 Bash 脚本执行以下任务：</p> <ol style="list-style-type: none"> <li><b>用户输入</b>: <ul style="list-style-type: none"> <li>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。</li> </ul> </li> <li><b>文件夹存在性检查</b>: <ul style="list-style-type: none"> <li>它使用 `[ ! -d "\$folder_name" ]` 条件检查具有指定名称的文件夹是否存在。如果文件夹不存在，它会显示一条错误消息（“文件夹不存在。”），并以退出代码 1 退出脚本。</li> </ul> </li> </ol>		
	下页继续...		

输出	<p>3. **文件列表**:</p> <ul style="list-style-type: none"> <li>- 它使用 `files=( "\$folder_name"/* )` 命令列出指定文件夹内的所有文件。这将创建一个名为 `files` 的数组，其中包含文件夹中所有文件的路径。</li> </ul> <p>4. **文件重命名**:</p> <ul style="list-style-type: none"> <li>- 它使用 `for` 循环遍历 `files` 数组中的每个文件。</li> <li>- 对于每个文件，它使用 `new_file_name="draft_\${basename}"` 命令生成一个新的文件名。这个新的文件名是通过在原始文件名前面加上字符串“draft_”来创建的。</li> <li>- 然后，它使用 `mv "\$file" "\$new_file_name"` 命令重命名文件，该命令将文件移动到新的文件名。</li> </ul> <p>5. **成功消息**:</p> <ul style="list-style-type: none"> <li>- 重命名所有文件后，它会显示一条成功消息（“文件已成功重命名。”），以表明重命名操作已成功</li> </ul>
----	--

表17. 用于解释代码的提示

## 用于翻译代码的提示

表16中的bash代码似乎工作正常。然而，如果这个脚本能够提示我输入文件名，那么它将真正具有可重用性，理想情况下，它应该作为一个独立的应用程序与an UI一起工作。作为一个起点，Python将比Bash更适合（Web）应用程序。LLM可以帮助将代码从一种语言翻译成另一种语言。请参见表18中的示例：

名称	1_python_rename_files		
目标	编写一个提示，将Bash代码翻译成Python		
模型	gemini-pro		
温度	0.1	令牌限制	1024
Top-K	不适用	Top-P	1
提示	<p>将下面的Bash代码翻译成Python代码片段。</p> <pre>```bash #!/bin/bash  echo "输入文件夹名称：" read folder_name  if [ ! -d "\$folder_name" ]; then echo "文件夹不存在。" exit 1 fi  files=( "\$folder_name"/* )  for file in "\${files[@]}"; do new_file_name="draft_\$(basename "\$file")" mv "\$file" "\$new_file_name" done  echo "文件已成功重命名。" ``` </pre>		
输出	<pre>```python import os import shutil  # 从用户处获取文件夹名称 folder_name = input("输入文件夹名称：")  # 检查文件夹是否存在 if not os.path.isdir(folder_name):     print("文件夹不存在。")     exit(1) ``` </pre>		
	下页继续...		

输出	<pre># 获取文件夹中所有文件的列表 files = os.listdir(folder_name)  # 遍历文件 for file in files:     # 创建一个带有前缀 "draft_" 的新文件名     new_file_name = f"draft_{file}"      # 将文件移动到新名称     shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_file_name))      # 打印成功消息     print("文件已成功重命名。")     ``</pre>
----	---

表 18. 用于将代码从 Bash 翻译为 Python 的提示

阅读并审查代码。从提示中复制输出并将其粘贴到新文件中：

file\_renamer.py 。通过打开终端窗口测试代码，并执行以下命令 **python file\_renamer.py**。

**注意：**当在 Vertex AI 的 Language Studio 中提示（Python）代码时，您必须单击“Mark down”按钮。否则，您将收到缺少正确行缩进的纯文本，这对于运行 Python 代码非常重要。

## 用于调试和审查代码的提示

让我们手动编写一些对表 18 代码的编辑。它应该提示用户输入文件名的前缀，并将此前缀以大写字符写入。请参阅代码片段 \*3 中的示例代码，但真是太糟糕了。它现在返回 Python 错误！

## Python

```
import os
import shutil

folder_name = input("Enter the folder name: ")
prefix = input("Enter the string to prepend to the filename: ")
text = toUpperCase(prefix)

如果非 os.path.isdir  foldername ：
    打印 "文件夹不存在。"
    退出

files = os.listdir(folder_name)

for file in files:
    newfilename      f"{text}_{file}"

    file }"shutil.move os.path.join  foldername ,  file ,
    os.path.join  foldername ,  newfile

    name )打印 "文件重命名成功。"
```

代码片段 3. 一个损坏的 Python 脚本

哎呀！这看起来像个错误：

下面的 Python 代码给出了一个错误：

```
Traceback (最近一次调用 last):
  File "/Users/leeboonstra/Documents/test_folder/rename_files.py", line 7,
in <module>
    text = toUpperCase(prefix)
NameError: name 'toUpperCase' is not defined
```

代码片段 4. 我破坏了 Python 代码

让我们看看是否可以要求大型语言模型来调试和审查代码。请看表 19：

名称	1_python_debug_code		
目标	编写一个提示来调试和审查 Python 代码。		
模型	gemini-pro		
温度	0.1	令牌限制	1024
Top-K	不适用	Top-P	1
提示	<p>下面的 Python 代码给出了一个错误：</p> <p>Traceback (最近一次调用 last):</p> <pre>文件 "/" Users/leeboonstra/Documents/test_folder/ rename_files.py", 第 7 行, 在 &lt;module&gt; 中     text = toUpperCase(prefix) NameError: name 'toUpperCase' is not defined</pre> <p>调试哪里出错了，并解释如何改进代码。</p> <pre>```python import os import shutil  # 从用户处获取文件夹名称 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 = toUpperCase(prefix)  # 检查文件夹是否存在 if not os.path.isdir(folder_name):     print("文件夹不存在。")     退出(1)  # 获取文件夹中所有文件的列表 files = os.listdir(folder_name)  # 遍历文件 for file in files:</pre>		

下页继续...

提示	<pre> # 创建一个以 "draft_" 为前缀的新文件名 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("文件已成功重命名。") ``` </pre>
输出	<p>代码调用 `toUpperCase` 函数将 `prefix` 转换为大写，但是该函数未定义。要解决此问题，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 import os import shutil  # 从用户处获取文件夹名称 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 if not 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)  # 遍历文件 for file in files:     # 创建一个以 "draft_" 为前缀的新文件名     new_filename = f"{text}_{file}"      # 将文件移动到新名称     shutil.move(os.path.join(folder_name, file), ``` </pre> <p style="text-align: right;">下页继续...</p>

输出	<pre>os.path.join(folder_name, new_filename))  # 打印成功消息 print("文件已成功重命名。") ```  此外，还可以对代码进行一些其他的改进： <ol style="list-style-type: none"> <li>1. 新文件名的文件扩展名未保留。最好从原始文件名中提取扩展名，并将其附加到新文件名。</li> <li>2. 该代码假设文件夹名称中没有空格。然而，处理空格时，最好优雅地将其替换为下划线或连字符。</li> <li>3. 使用 `f` 字符串语法进行字符串插值比传统的 `+` 运算符更具可读性和简洁。</li> <li>4. 该代码没有处理在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。</li> </ol> <p>以下是包含这些建议的改进代码：</p> <pre>```python import os import shutil # 从用户获取文件夹名称 folder_name = input("输入文件夹名称: ") prefix = input("输入要添加到文件名的字符串: ") # 检查文件夹是否存在 if not 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) # 遍历文件 for file in files:</pre> </pre>
----	---

下页继续...

输出	<pre># 获取文件扩展名 ext = os.path.splitext(file)[1]  # 创建一个带有前缀和文件扩展名的新文件名 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(folder_name, file) new_path = os.path.join(folder_name, new_filename)  # 尝试移动文件 try:     shutil.move(old_path, new_path)  except Exception as e:     print(f"移动文件时发生错误 {file}: {e}")  # 打印成功消息 print("文件已成功重命名。") ```  这些改进使代码更健壮、更灵活，同时也使其更易于阅读和理解.....（响应已被截断，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。）</pre>
----	--

表 19. 用于调试和审查 Python 代码的提示

太棒了。它不仅告诉我如何解决问题，还发现我的代码有更多错误以及如何解决这些错误。提示的最后一部分给出了改进代码的建议。

## 多模态提示怎么样？

代码提示仍然使用相同的常规大型语言模型。多模态提示是一个单独的问题，它指的是一种使用多种输入格式来引导大型语言模型的技术，而不是仅仅依赖于文本。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手头的任务。

## 最佳实践

找到合适的提示需要不断尝试。Vertex AI 中的 Language Studio 是一个完美的地方，可以用来试验你的提示，并且能够针对各种模型进行测试。

使用以下最佳实践，成为提示工程方面的专家。

### 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，因为它是一种强大的教学工具。这些示例展示了所需的输出或类似的响应，使模型能够从中学习并调整其自身的生成，从而更好地满足您的期望。这就像给模型一个参考点或目标，提高其响应的准确性、风格和语气，从而更好地匹配您的期望。

## 设计时要简洁

提示应该简洁、清晰，并且易于您和模型理解。作为经验法则，如果它已经让您感到困惑，那么它也可能让模型感到困惑。尽量不要使用复杂的语言，也不要提供不必要的信息。

示例：

之前：

我现在正在纽约旅游，想了解更多关于好去处的信息。我和两个3岁的孩子在一起。我们假期应该去哪里？

重写后：

充当游客的旅行指南。描述在纽约曼哈顿与3岁儿童一起游览的好地方。

尝试使用描述动作的动词。这是一组示例：

行动，分析，分类，归类，对比，比较，创造，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，排序，推荐，返回，检索，重写，选择，展示，排序，总结，翻译，写作。

## 具体说明输出

具体说明所需的输出。简洁的指令可能无法充分引导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。

示例：

执行：

生成一篇关于排名前五的视频游戏机的三段式博客文章。

这篇博客文章应该具有信息性和吸引力，并且应该以对话式的风格撰写。

不要：

生成一篇关于视频游戏机的博客文章。

## 使用指令而非约束

指令和约束在提示中用于指导大型语言模型（LLM）的输出。

- 一个**指令**提供了关于所需格式、风格或响应内容的明确指示。它指导模型应该做什么或产生什么。
- 一个**约束**是对响应的一组限制或边界。它限制了模型不应该做什么或避免什么。

越来越多的研究表明，在提示中关注积极的指令可能比过度依赖约束更有效。这种方法与人类更喜欢积极指令而不是不该做什么的清单的方式相一致。

指令直接传达期望的结果，而约束可能会让模型猜测什么是允许的。它提供了灵活性，并在定义的边界内鼓励创造力，而约束可能会限制模型的潜力。此外，约束列表可能会相互冲突。

约束在某些情况下仍然有价值。为了防止模型生成有害或有偏见的内容，或者当需要严格的输出格式或风格时。

如果可能，使用积极的指令：不要告诉模型不要做什么，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。

执行：

生成一篇关于排名前五的视频游戏机的博客文章，共一段。  
只讨论游戏机、制造公司、年份和总销量。

不要：

生成一篇关于排名前五的视频游戏机的博客文章，共一段。  
不要列出视频游戏名称。

作为最佳实践，首先要优先考虑指令，清楚地说明你希望模型做什么，只有在为了安全、清晰或特定要求时才使用约束。

进行实验和迭代，以测试指令和约束的不同组合，找到最适合你特定任务的方法，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，你可以在配置中设置最大令牌限制，或者在提示中明确请求特定长度。例如：

“用一条推特的长度解释量子物理。”

## 在提示中使用变量

为了重用提示并使其更具动态性，可以在提示中使用变量，这些变量可以针对不同的输入进行更改。例如，如表 20 所示，一个提示给出了关于一个城市的事。与其在提示中硬编码城市名称，不如使用变量。变量可以节省您的时间和精力，避免重复。如果您需要在多个提示中使用相同的信息，您可以将其存储在一个变量中，然后在每个提示中引用该变量。当将提示集成到您自己的应用程序中时，这非常有意义。

提示	变量 {city} = "阿姆斯特丹" 提示 你是一名旅游指南。告诉我一个关于{city}城市的事
输出	阿姆斯特丹是一座美丽的城市，到处都是运河、桥梁和狭窄的街道。这里历史悠久、文化底蕴深厚、夜生活丰富，是一个值得游览的好地方。

表 20. 在提示中使用变量

## 尝试不同的输入格式和写作风格

不同的模型、模型配置、提示格式、措辞和提交方式可能会产生不同的结果。因此，尝试提示属性（如风格、措辞和提示类型（零样本、少样本、系统提示））非常重要。

例如，一个旨在生成关于革命性视频游戏

主机世嘉Dreamcast文本的提示，可以被表述为一个**问题**、一个**陈述**或一个**指令**，从而产生不同的输出：

- **问题**：什么是世嘉Dreamcast，为什么它是一款如此具有革命性的游戏机？
- **陈述**：世嘉Dreamcast是世嘉于1999年发布的一款第六代视频游戏机。它...
- **指令**：写一个段落，描述世嘉Dreamcast游戏机，并解释为什么它如此具有革命性。

## 对于分类任务的少量样本提示，混合类

一般来说，你的少量样本示例的顺序应该不太重要。然而，在进行分类任务时，请确保你在少量样本示例中混合了可能的响应类别。这是因为你可能会过度拟合到示例的特定顺序。通过混合可能的响应类别，你可以确保模型正在学习识别每个类别的关键特征，而不仅仅是简单地记住示例的顺序。这将导致在未见过的数据上获得更稳健和更具泛化性的性能。

一个好的经验法则是从6个少量样本示例开始，并从那里开始测试准确性

◦

## 适应模型更新

对你来说，掌握模型架构的变化、添加的数据和功能非常重要。尝试更新的模型版本，并调整你的提示，以更好地利用新的模型功能。像Vertex AI Studio这样的工具非常适合存储、测试和记录你的提示的各种版本。

## 尝试输出格式

除了提示输入格式外，还可以考虑尝试输出格式。对于非创意任务，如提取、选择、解析、排序、排名或分类数据，尝试以结构化格式（如JSON或XML）返回你的输出。

从提取数据的提示中返回 JSON 对象有一些好处。在一个实际应用中，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（在处理日期时间对象时非常方便），但最重要的是，通过提示使用 JSON 格式，它迫使模型创建一个结构并限制幻觉。

总而言之，使用 JSON 作为输出的好处：

- 始终以相同的样式返回
- 专注于您想要接收的数据

- 减少产生幻觉的可能性
- 使其具有关系感知能力
- 您可以获得数据类型
- 您可以对其进行排序

few-shot prompting 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 格式返回数据提供了许多优势，但并非没有缺点。JSON 的结构化特性，虽然有利于在应用程序中进行解析和使用，但比纯文本需要更多的 token，从而导致处理时间增加和成本更高。此外，JSON 的冗长性很容易消耗整个输出窗口，当由于 token 限制而突然中断生成时，这会变得特别麻烦。这种截断通常会导致无效的 JSON，缺少关键的结束大括号或方括号，从而使输出无法使用。幸运的是，像 json-repair 库（可在 PyPI 上找到）这样的工具在这些情况下非常宝贵。这个库智能地尝试自动修复不完整或格式错误的 JSON 对象，使其成为使用 LLM 生成的 JSON 的关键盟友，尤其是在处理潜在的截断问题时。

## 使用模式

正如我们在本文中多次看到的那样，使用结构化的 JSON 作为输出是一个很好的解决方案。但是输入呢？虽然 JSON 非常适合构建 LLM 生成的输出，但它对于构建您提供的输入也非常有用。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。通过提供一个模式，您就为 LLM 提供了一个清晰的蓝图，说明它应该期望的数据，帮助它将其注意力集中在相关信息上，并降低误解输入的风险。此外，模式可以帮助建立不同数据片段之间的关系，甚至可以通过包含具有特定格式的日期或时间戳字段来使 LLM“感知时间”。

这是一个简单的例子：

假设您想使用 LLM 为电子商务目录中的产品生成描述。您可以不只是提供产品的自由文本描述，而是可以使用 JSON 模式来定义产品的属性：

```
{  
  "type": "object",  
  "properties": {  
    "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" },  
  
    "features": {  
      "type": "array",  
      "items": { "type": "string" },  
      "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

然后，您可以提供符合此模式的实际产品数据作为 JSON 对象：

```
{  
  "name": "无线耳机",  
  "category": "电子产品",  
  "price": 99.99,  
  "features": ["噪音消除", "蓝牙 5.0", "20 小时电池续航时间"],  
  "release_date": "2023-10-27"  
}
```

代码片段 6. 来自 LLM 的结构化输出

通过预处理您的数据，并且不提供完整的文档，而是同时提供模式和数据，您可以使 LLM 清楚地了解产品的属性，包括其发布日期，从而使其更有可能生成准确且相关的描述。这种结构化的输入方法，将 LLM 的注意力引导到相关字段，在处理大量数据或将 LLM 集成到复杂的应用程序中时尤其有价值。

## 与其他提示工程师一起试验

如果您必须尝试提出一个好的提示，您可能希望找到多人来进行尝试。当每个人都遵循最佳实践（如本章所列）时，您将看到所有不同提示尝试之间的性能差异。

## CoT 最佳实践

对于思维链 (CoT) 提示，需要在推理之后给出答案，因为推理的生成会改变模型在预测最终答案时获得的令牌。

使用思维链 (CoT) 和自我一致性，你需要能够从你的提示中提取最终答案，并与推理分开。

对于思维链 (CoT) 提示，将温度设置为0。

思维链提示基于贪婪解码，根据语言模型分配的最高概率预测序列中的下一个词。一般来说，当使用推理来得出最终答案时，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

最后一个技巧在本章前面已经提到过，但我们再怎么强调它的重要性也不为过：完整地记录你的提示尝试，这样你就可以随着时间的推移学习哪些有效，哪些无效。

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，即使对于同一模型的相同提示，输出语句的格式和措辞也可能出现细微差异。（例如，如前所述，如果两个令牌具有相同的预测概率，则可能会随机打破平局。这可能会影响后续预测的令牌。）

我们建议创建一个以表21为模板的Google Sheet。这种方法的优点是，当你不可避免地需要重新审视你的提示工作时，你有一个完整的记录——无论是为了将来重新拾起它（你会惊讶于仅仅短暂休息后你会忘记多少），还是为了测试提示在不同模型版本上的性能，以及帮助调试未来的错误。

除了此表中的字段外，跟踪提示的版本（迭代）、一个用于捕获结果是否OK/NOT OK/SOMETIMES OK的字段以及一个用于捕获反馈的字段也很有帮助。如果你足够幸运能够使用Vertex AI Studio，请保存你的提示（使用与你的文档中列出的相同的名称和版本），并在表中跟踪指向已保存提示的超链接。这样，你始终只需单击一下即可重新运行你的提示。

在处理检索增强生成系统时，您还应该捕获RAG系统的特定方面，这些方面会影响插入到提示中的内容，包括查询、分块设置、分块输出和其他信息。

一旦您觉得提示接近完美，就将其应用到您的项目代码库中。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，您的提示是运营系统的一部分，作为提示工程师，您应该依赖自动化测试和评估程序来了解您的提示对任务的泛化程度。

提示工程是一个迭代过程。设计和测试不同的提示，分析，并记录结果。根据模型的性能改进您的提示。不断实验，直到获得所需的输出。当您更改模型或模型配置时，返回并继续试验以前使用的提示。

名称	[您的提示的名称和版本]		
目标	[用一句话解释本次尝试的目标]		
模型	[使用的模型的名称和版本]		
温度	[0到1之间的值]	令牌限制	[数字]
Top-K	[数字]	Top-P	[数字]
提示	[写出所有完整的提示]		
输出	[写出输出或多个输出]		

表 21. 用于记录提示的模板

## 摘要

本白皮书讨论了提示工程。我们学习了各种提示技术，例如：

- 零样本提示
- 少样本提示
- 系统提示
- 角色提示
- 
- 
- 
- 思维链
- 自我一致性
- 思维树

- ReAct

我们甚至研究了如何自动化你的提示。

然后，该白皮书讨论了通用人工智能的挑战，例如当您的提示不充分时可能出现的问题。最后，我们总结了如何成为一名更好的提示工程师的最佳实践。

## 尾注

1. Google, 2023, Gemini by Google。可在以下网址获取：[谷歌云](https://www.google.com/search?q=Gemini+by+Google) / [Gemini by Google](https://www.google.com/search?q=Gemini+by+Google)。
2. Google, 2024, Gemini for Google Workspace Prompt Guide。可在以下网址获取：[谷歌云](https://www.google.com/search?q=Gemini+for+Google+Workspace+Prompt+Guide) / [Gemini for Google Workspace Prompt Guide](https://www.google.com/search?q=Gemini+for+Google+Workspace+Prompt+Guide)。
3. Google Cloud, 2023, Introduction to Prompting。可在以下网址获取：[谷歌云](https://cloud.google.com/introduction-to-prompting) / [Introduction to Prompting](https://cloud.google.com/introduction-to-prompting)。
4. Google Cloud, 2023, Text Model Request Body: Top-P & top-K sampling methods。可在以下网址获取：[谷歌云](https://cloud.google.com/text-model-request-body-top-p-top-k-sampling-methods) / [Text Model Request Body: Top-P & top-K sampling methods](https://cloud.google.com/text-model-request-body-top-p-top-k-sampling-methods)。
5. Wei, J., et al., 2023, Zero Shot - Fine Tuned language models are zero shot learners。可在以下网址获取：[arXiv](https://arxiv.org/pdf/2109.01652.pdf) / [Zero Shot - Fine Tuned language models are zero shot learners](https://arxiv.org/pdf/2109.01652.pdf)。
6. 谷歌云, 2023, 谷歌云模型花园。可在以下网址获取：[谷歌云](https://cloud.google.com/models) / [谷歌云模型花园](https://cloud.google.com/models)。
7. Brown, T. 等人, 2023, 《少量样本 - 语言模型是少量样本学习者》。可在以下网址获取：[arXiv](https://www.semanticscience.org/paper/2023/06/01/0001) / [少量样本 - 语言模型是少量样本学习者](https://www.semanticscience.org/paper/2023/06/01/0001)。
8. Zheng, L. 等人, 2023, 《后退一步：通过大型语言模型中的抽象来唤起推理》。可在以下网址获取：[arXiv](https://arxiv.org/pdf/2301.00101.pdf) / [后退一步：通过大型语言模型中的抽象来唤起推理](https://arxiv.org/pdf/2301.00101.pdf)。
9. Wei, J. 等人, 2023, 《思维链提示》。可在以下网址获取：[arXiv](https://arxiv.org/pdf/2301.00101.pdf) / [思维链提示](https://arxiv.org/pdf/2301.00101.pdf)。
10. 谷歌云平台, 2023, 《思维链和 React》。可在以下网址获取：[谷歌云](https://cloud.google.com/prompting/react) / [思维链和 React](https://cloud.google.com/prompting/react)。
11. 王, X., 等, 2023年, 自洽性提高语言模型中的思维链推理。可在以下网址获取：[arXiv](https://arxiv.org/pdf/2301.00101.pdf) / [自洽性提高语言模型中的思维链推理](https://arxiv.org/pdf/2301.00101.pdf)。
12. 姚, S., 等, 2023年, “思维树：使用大型语言模型进行审慎的问题解决”。可在以下网址获取：[arXiv](https://arxiv.org/pdf/2301.00101.pdf) / [思维树：使用大型语言模型进行审慎的问题解决](https://arxiv.org/pdf/2301.00101.pdf)。
13. 姚, S., 等, 2023年, “ReAct：在语言模型中协同推理和行动”。可在以下网址获取：[arXiv](https://arxiv.org/pdf/2301.00101.pdf) / [ReAct：在语言模型中协同推理和行动](https://arxiv.org/pdf/2301.00101.pdf)。
14. Google Cloud Platform, 2023年, “高级提示：思维链和React”。可在以下网址获取：[github](https://github.com/GoogleCloudPlatform/applied-ai-engineering-samples/blob/main/genai-on-vertex-ai/advanced_prompting_training/cot_react.ipynb) / [高级提示：思维链和React](https://github.com/GoogleCloudPlatform/applied-ai-engineering-samples/blob/main/genai-on-vertex-ai/advanced_prompting_training/cot_react.ipynb)。
15. Zhou, C. 等人, 2023, 《自动提示工程——大型语言模型是人类水平的提示工程师》。可在以下网址获取：[arXiv](https://arxiv.org/pdf/2211.01910.pdf) / [自动提示工程——大型语言模型是人类水平的提示工程师](https://arxiv.org/pdf/2211.01910.pdf)。