Prompt Engineering：一本即将发布的专业著作

《Prompt Engineering》是由Lee Boonstra撰写，计划于2025年2月由Google出版。这本书聚焦于提示工程领域，集结了多位业界专家的贡献与指导。

参与本书的作者和贡献者包括：Michael Sherman、Yuan Cao、Erick Armbrust、Anant Nawalgaría、Antonio Gulli、Simone Cammel。书中还有Antonio Gulli、Anant Nawalgaría和Grace Mollison等人的协助和指导意见。此外，Joey Haymaker和Michael Lanning也参与了相关工作。

本书将为读者带来前沿的提示工程知识与实践经验，适合希望深入理解Prompt Engineering及其实际应用的技术人员和研究者。

人人皆可写 Prompt：Prompt Engineering 简介

在大语言模型（LLM）的世界里，Prompt（提示词）就是你与模型交流的桥梁。或许你以为写 Prompt 需要数据科学家或机器学习工程师的技能，其实不然——每个人都可以写出专属于自己的 Prompt。

虽然人人可写，但要写出高效、精准的 Prompt 并不简单。影响 Prompt 效果的因素有很多，比如所选用的模型、模型训练时用到的数据、模型配置参数、你的用词、表达风格、结构和上下文等。这些细节都会影响模型输出质量。因此，Prompt Engineering 是一个需要不断试验和优化的过程。

如果 Prompt 设计不当，模型可能给出含糊或不准确的答复，甚至无法输出有意义的内容。所以，想写好 Prompt，需要关注细节，并在实践中不断调整和提升。

Prompt Engineering：让大模型更懂你

大语言模型（LLM）本质上是一个预测引擎。它接收一段文本作为输入，然后根据之前的训练数据预测下一个最合适的词或符号，并将其不断添加到文本末尾，逐步生成完整输出。你写的每一个 Prompt（提示词），其实就是在引导模型一步步地完成你期望的内容。

Prompt engineering，就是设计高质量 Prompt 的过程，目的在于让 LLM 给出更准确的输出。这个过程中，你需要不断尝试和调整——选择合适的模型，优化 Prompt 的长度，评估用词、结构和表达风格是否符合任务需求。每次微调，都有可能让模型的表现更贴合你的目标。

不只是写一句话那么简单，Prompt engineering 更像是一门需要用心打磨的艺术。只有通过持续优化，才能让大模型真正理解你的意图，给出理想的答案。

LLM Output Configuration：调教大模型生成内容的关键参数

在使用大语言模型（LLM）时，模型的输出效果不仅取决于你的 Prompt，模型本身的输出配置同样关键。合理设置这些参数，可以让模型的表现更贴合你的需求，无论是追求严谨、准确，还是希望获得更有创意的内容。

输出长度（Output length）

输出长度决定了模型在每次响应时能生成多少 token。生成更多 token 意味着更高的计算消耗、更长的等待时间和更高的成本。如果你只需要简短的回答，需在 Prompt 设计和输

出长度参数上双管齐下。对于一些高级提示技巧（如 ReAct），合理限制输出长度尤为重要，以防模型生成无用或重复内容。

采样控制（Sampling controls）

LLM 通过对每一个可能的下一个 token 进行概率预测，再采样出实际输出。采样过程涉及几个核心参数：

- **Temperature**：控制输出的随机性。温度低时（如 0），模型更倾向于选择最确定的答案；温度高时，输出更加多样和富有创意。极端设置（过高或过低）可能导致无意义或重复输出。
- **Top-K**：限定每次只从概率最高的 K 个 token 中采样。K 越大，输出越丰富，K=1 时则完全确定性输出。
- **Top-P（Nucleus sampling）**：从累计概率不超过 P 的 token 集合中采样。P 越大，参与采样的 token 越多，输出越多样。

参数组合与调优

这些参数可以组合使用，共同影响最终输出。比如在 Vertex Studio 中，通常是先用 top-K 和 top-P 筛选候选 token，再应用 temperature 进行采样。不同的场景需要不同的参数组合：

- 想要结果更有创造力：温度 0.9，top-P 0.99，top-K 40
- 需要结果更准确少偏差：温度 0.1，top-P 0.9，top-K 20
- 只有唯一正确答案时（如解数学题）：温度 0

需要注意，参数设置过于极端，可能导致其他参数失效。比如 temperature 设为 0，top-K 和 top-P 就不再起作用；top-K 设为 1 也会让 temperature 和 top-P 失效。

常见问题与优化

错误的参数设置容易让模型陷入“重复循环”——一直输出无意义的重复内容。这通常是温度或 top-K/top-P 设置不合理导致的。最佳实践是针对任务特点，不断微调参数，找到创意和准确性之间的平衡点。

总之，灵活配置 LLM 的输出参数，是释放大模型能力、获得理想结果的关键一步。

Prompting Techniques：让大模型更好理解你的意图

在与大语言模型（LLM）交互时，Prompt 的设计方式直接影响模型的理解和输出效果。以下是使用 LLM 时常见且有效的几种 Prompting 技巧：

1. 零样本（Zero-shot）Prompting

零样本 Prompt 是最基础的方式，只需要描述任务和相关文本，不提供任何示例。比如直接让模型判断一段影评的情感倾向。零样本适合模型已经具备相关知识、无需复杂推理的场景。通常，任务描述越清晰、指令越明确，模型表现越好。

示例（情感分类）：

```
Classify movie reviews as POSITIVE, NEUTRAL or NEGATIVE. Review:
"Her" is a disturbing study revealing the direction humanity is
```

```
headed if AI is allowed to keep evolving, unchecked. I wish there were more movies like this masterpiece. Sentiment:
```

2. 单样本与少样本（One-shot & Few-shot）Prompting

当零样本效果不佳时，可以通过加入一到多个示例，帮助模型更好理解任务和期望输出格式。单样本（One-shot）只提供一个示例，少样本（Few-shot）则给出多个。对于复杂、结构化的任务，提供高质量且多样化的示例非常重要，能显著提升输出准确性和一致性。一般建议少样本示例数量为 3-5 个，复杂任务可适当增加，但需注意模型输入长度限制。

示例（将披萨订单解析为 JSON）：

``` Parse a customer's pizza order into valid JSON.

EXAMPLE: I want a small pizza with cheese, tomato sauce, and pepperoni. JSON Response: { "size": "small", "type": "normal", "ingredients": ["cheese", "tomato sauce", "pepperoni"] }

EXAMPLE: Can I get a large pizza with tomato sauce, basil and mozzarella { "size": "large", "type": "normal", "ingredients": ["tomato sauce", "basil", "mozzarella"] }

Now, I would like a large pizza, with the first half cheese and mozzarella. And the other tomato sauce, ham and pineapple. JSON Response: ```

高质量的示例要覆盖常见情况和边缘情况，示例内容要相关、规范，避免歧义或错误。

**3. 系统、上下文与角色 Prompting**

- **系统（System）Prompt**：定义模型的整体任务和目标，是“大背景”。如指定模型用于翻译、分类等。
- **上下文（Contextual）Prompt**：提供当前任务的具体背景信息，帮助模型理解细节和语境，使输出更贴合实际场景。
- **角色（Role）Prompt**：赋予模型特定身份或角色，让其以特定风格、语气输出内容。例如让模型扮演医生、老师等。

三者常常结合使用，通过合理设计，让大模型输出更符合预期。

**总结**

Prompting 技巧的核心在于明确、具体、结构化地表达你的需求。通过选择合适的 Prompt 类型和示例，结合系统、上下文和角色设定，可以极大提升 LLM 的理解力和输出质量。不断迭代和优化，是获得最佳效果的关键。

**深入理解大模型的 System、Contextual 和 Role Prompting 技巧**

在与大语言模型（LLM）互动时，如何设计 Prompt 极大影响模型的输出效果。除了常见的任务描述和示例外，System、Contextual 和 Role Prompting 是三种非常实用的高级提示技巧。下面将为你解析它们的核心原理与最佳实践。

---

**System Prompting：为模型设定“大局观”**

System prompting 用于为模型设定整体任务目标和输出规范，即告诉模型“你该做什么，以及怎么做”。通过系统提示，可以定义输出的结构、风格或格式。例如，你可以要求模型
```

“只返回情感标签，且全部大写”，或“以 JSON 格式输出结果”。系统提示非常适合需要高一致性、结构化输出的场景，比如数据抽取、代码生成或安全要求较高的应用。同时，适当调整参数如 temperature、token limit 也能进一步控制输出的创造性和长度。

示例：

```
Classify movie reviews as positive, neutral or negative. Only
return the label in uppercase. Review: "Her" is a disturbing
study revealing the direction humanity is headed if AI is allowed
to keep evolving, unchecked. It's so disturbing I couldn't watch
it. Sentiment:
```

输出：

```
NEGATIVE
```

Contextual Prompting：提供“上下文”，让输出更贴合实际

Contextual prompting 强调为模型提供具体的背景信息或场景细节。这样能让模型更准确地理解你的需求，生成更符合实际场景的内容。例如，在让模型为复古游戏博客推荐选题时，直接说明“你正在为80年代街机游戏写博客”，模型会据此给出高度相关的选题建议。Contextual prompting 能有效提升模型的理解力和响应相关性，尤其适合需要模型结合具体情境或动态任务的场合。

示例：

```
Context: You are writing for a blog about retro 80's arcade video
games. Suggest 3 topics to write an article about with a few
lines of description of what this article should contain.
```

Role Prompting：让模型“扮演角色”，定制输出风格与专长

Role prompting 是给模型分配一个具体身份或角色，让它以该角色的视角和风格进行回应。无论是扮演旅行顾问、老师、编辑还是演讲者，模型都会根据角色调整语言、内容和专业度。通过角色设定，不仅能改善输出的相关性和专业性，还能定制语气，如幽默、正式、鼓舞人心等，让体验更加个性化。

示例：

```
I want you to act as a travel guide. I will write to you about my
location and you will suggest 3 places to visit near me. My
suggestion: "I am in Amsterdam and I want to visit only museums."
```

输出：

1. Rijksmuseum: 世界著名的荷兰艺术与历史博物馆。
2. Van Gogh Museum: 收藏梵高画作的必访艺术殿堂。
3. Stedelijk Museum Amsterdam: 现代与当代艺术的风向标。

总结

System、Contextual 和 Role Prompting 能让你根据需求灵活控制大模型的输出——无论是结构、内容、风格还是专业度。合理运用这些技巧，将极大提升 LLM 在实际业务或创意场景中的表现力和可用性。

Step-back Prompting：让大模型更善于推理和创作的技巧

在与大语言模型互动时，如何让它给出更准确、更有洞察力的答案？Step-back prompting（后退提示）是一种值得尝试的方法。这种技巧鼓励模型不要直接解决具体问题，而是先思考与任务相关的更一般性问题，再将这个答案作为新提示的一部分，最终聚焦于具体问题的解决。

Step-back Prompting 的核心原理

Step-back prompting 的关键在于“先广后深”。通过引导模型先思考更广泛、底层的原则或环境，可以激活模型深层的知识和推理能力，然后再聚焦于实际任务。这种方式可以提升输出的准确性和创造力，帮助模型跳出直接回答的局限，生成更有深度和逻辑性的内容。

实际应用举例

假设你要让模型为第一人称射击游戏关卡编写故事情节。直接提问时，模型可能给出内容较为普通和随机的回答。但如果你采用 step-back prompting，先让模型总结出“令人印象深刻的游戏关卡常见的五大主题”，比如：

1. 废弃的军事基地
2. 赛博朋克城市
3. 外星飞船
4. 僵尸肆虐的小镇
5. 水下研究设施

接着，挑选其中一个主题，如“水下研究设施”，再让模型基于这个主题编写故事情节。这样生成的内容会更丰富、更贴合玩家体验——例如描绘玩家在幽暗海底实验室中与怪物周旋、解谜生存的紧张氛围。

Step-back Prompting 的优势

- 激发模型的背景知识和全局思考能力
- 生成更具创造性和逻辑性的内容
- 有助于减少模型的偏见，聚焦于原理而非细节
- 适用于需要创新、分析或结构化思考的复杂任务

结语

Step-back prompting 是一种高效提升大模型输出质量的技巧。通过让模型“先退一步”，先思考一般性原则，再深入到具体任务，不仅能带来更有深度的答案，也能让 AI 的创作更加精彩纷呈。如果你希望让大模型为你提供更优质的内容，不妨尝试这种方法！

Chain of Thought (CoT)：让大模型像人一样“思考”的提示技巧

在使用大语言模型（LLM）解答复杂问题时，我们常常遇到模型给出错误或不完整答案的情况。Chain of Thought（CoT）提示方法，就是为了解决这一难题。它通过引导模型一步步推理，让答案更准确、更有逻辑性。

什么是 Chain of Thought (CoT)?

Chain of Thought（思维链）是一种让大模型在回答问题前，先输出中间推理步骤的提示技术。这种方法尤其适用于需要推理、分析或多步计算的问题，比如数学题、代码生成、逻辑推理等。通过“让我们一步步来思考”等指令，模型会像人类一样，按步骤分解问题，最终给出答案。

实际应用举例

以一道年龄推理题为例：

- 传统提示：
  “我3岁时，伴侣比我大3倍。现在我20岁，伴侣多大？”
  直接让模型回答，可能会得到错误答案（如 63 岁）。

- Chain of Thought 提示：
  “我3岁时，伴侣比我大3倍。现在我20岁，伴侣多大？让我们一步步思考。”
  模型输出：

- 3岁时伴侣是3×3=9岁。
- 现在我20岁，年龄增长了17年。
- 伴侣也增长17年，现在是9+17=26岁。
- 答案：26岁。

通过分步骤推理，模型更容易“走对路”，也让用户更容易理解整个思考过程。

Chain of Thought 的优势

- **提升准确率**：通过分步推理，减少直接给错答案的概率。
- **结果可解释**：每一步推理都有据可查，便于发现和修正错误。
- **适用范围广**：不仅适用于数学题，还适用于代码生成、数据合成等需要“讲逻辑”的任务。
- **模型迁移更稳健**：在不同模型版本间，表现更一致。

进阶实践：Self-consistency

为了进一步提升准确率，可以多次提交同样的 CoT 提示，让模型用不同的推理路径解答，然后采用出现次数最多的答案，提升结果的可靠性。

注意事项

- CoT 会让输出变长，消耗更多的计算资源和时间。
- 对于非常简单的问题，可能不需要使用 CoT。

结语

Chain of Thought（CoT）让大模型不再“拍脑袋”给答案，而是像人类一样思考和推理。只需在提示中鼓励模型“逐步分析”，你就能获得更准确、更有逻辑的答案。这一技巧，值得每一位 LLM 用户掌握和应用。

Self-consistency：提升大模型推理准确率的进阶技巧

在使用大语言模型（LLM）进行推理类任务时，即使采用了 Chain of Thought（CoT）分步思考提示，模型依然可能因为“贪婪解码”策略而导致推理路径单一、答案不够稳定。Self-consistency（自洽性）是一种进一步提升模型输出准确性与一致性的实用方法。

Self-consistency 的原理

Self-consistency 通过多次生成不同的推理路径，再用“少数服从多数”的方式，选出最常见的答案作为最终结果。这种方法结合了采样和多数票原则，能够提升模型输出的准确率和连贯性。

其核心流程包括：

1. **生成多样的推理路径**
   对同一问题多次向 LLM 提示，引导其用不同的思路进行推理（通常通过调高 temperature 参数实现）。
2. **提取每次生成的最终答案**
   对每个推理流程的结论进行汇总。
3. **选择最常见的答案**
   用多数票原则，选出出现频率最高的答案作为最终结果。

实际应用场景

以邮件分类为例，如果一封邮件内容模糊、语气带有幽默或讽刺，不同推理链可能得出“重要”或“不重要”两种结论。通过 Self-consistency，多次让模型独立推理，再统计哪种分类更常见，可以有效避免偶发性错误，提升判断的稳定性和正确性。

优缺点分析

- **优势：**
- 提高推理类任务的准确性与一致性
- 能发现并修正单一推理路径的偶然错误

- 适用于推理不确定性较高或主观性较强的复杂任务

- **不足：**

- 计算成本高，需要多次生成和汇总
- 处理速度变慢，适合对准确率要求高的场景

结语

Self-consistency 是对 Chain of Thought 的有力补充。它让大模型在面临复杂或模棱两可的问题时，能够像人一样“群策群力”，选出最有说服力的答案。如果你追求推理结果的高准确率，一定要尝试这一技巧！

Tree of Thoughts（ToT）：让大模型推理像树一样分叉生长

在大语言模型的推理任务中，传统的 Chain of Thought（CoT）方法是让模型按顺序一步步推理，像走一条单行道，从输入到输出只经历一条推理路径。而 Tree of Thoughts（ToT）则进一步扩展了这种思考方式，让模型可以像树一样探索多条推理分支，提升处理复杂任务的能力。

Tree of Thoughts 的核心原理

Tree of Thoughts（ToT）通过维持一个“思维树”，让每一步的推理都可以分叉出多条不同的可能路径。每一个“思维节点”都代表着一段连贯的语言序列，即某一步的思考结果。模型可以从一个节点出发，探索多种推理方向，不同的分支会形成一棵树，最终多条路径会在输出处汇合。

这种方式，相比于单一线性的推理链，允许模型“并行”地考虑各种可能性，适合需要深入探索和多角度分析的复杂问题。

ToT 的优势与应用场景

- **多路径探索**：同时考虑多条推理路线，而不是只走一条直线。
- **更强的复杂任务能力**：适合那些需要发散思维、权衡多种可能性的场景。
- **灵活的推理结构**：结构上像一棵树，分支与合并让思考更加灵活多样。

实际中，ToT 可以应用于需要综合多种证据、路径依赖性强的问题，例如复杂的决策制定、多步推理题目、策略规划等。

总结

Tree of Thoughts（ToT）让大模型的推理方式从“单线思考”进化为“树状探索”，极大增强了模型应对复杂任务时的多样性和深度。对于需要广泛思考、反复权衡的场景，ToT 是一种值得尝试的高级提示工程方法。

ReAct（Reason & Act）：让大模型能思考也能行动的新范式

在大语言模型的复杂任务处理中，ReAct（Reason & Act）是一种结合“推理”与“行动”的新型提示工程方法。它不仅让模型能用自然语言推理，还能主动调用外部工具（如搜索引擎、代码解释器等）来获取信息，实现“边想边做”。

ReAct 的核心思路

ReAct 模型通过“思考-行动循环”来解决问题。具体流程如下：

1. **推理**：模型首先分析任务，生成下一步的行动计划。
2. **行动**：根据计划，模型执行具体操作，比如发起搜索请求、调用API等。
3. **观察**：模型收集行动结果，把新信息作为输入。

4. **再推理，再行动**：模型用新获得的信息继续推理，制定新的计划，并执行，直到任务完成。

这种循环可以持续多轮，直到模型获得最终答案。

实际应用举例

以“统计乐队 Metallica 成员子女数量”为例，ReAct 代理会：

- 首先判断有几位乐队成员。
- 针对每位成员，发起搜索查询他们有几个孩子。
- 每次得到结果后，更新累计总数。
- 最终返回总子女数作为答案。

这一过程展现了模型如何智能地分步行动和推理，像人类一样逐步获取信息并整合出最终结果。

ReAct 的优势

- 能处理需要外部知识或实时信息的复杂问题。
- 模型推理和行动交替，提升解题的灵活性和准确性。
- 模拟人类思维与行动的方式，适应多领域实际场景。

总结

ReAct 方法让大语言模型不再只是“坐而论道”，而是能“边思考边行动”，主动探索、查找和整合信息解决复杂问题。这种方式大幅提升了模型的实用性和智能水平，是推动智能体应用发展的重要一步。

Automatic Prompt Engineering（APE）：让模型自动生成高质量提示词

在大语言模型的实际应用中，手动编写高质量的 prompt（提示词）可能是一项复杂且耗时的工作。Automatic Prompt Engineering（自动提示词工程，简称 APE）为此提供了解决方案：让模型自动生成、评估和优化 prompt，从而提升任务表现，同时减少人工干预。

APE 的基本流程

1. **生成多样化 prompt**
   首先，给模型一个指令，让它针对某个任务生成多种不同表达方式的 prompt。例如，为训练一个乐队周边 T 恤店的聊天机器人，模型可以将“我要一件 Metallica 乐队 S 码 T 恤”用 10 种不同说法表达出来。

2. **评估和筛选 prompt**
   对生成的 prompt 进行自动化评价，可以采用 BLEU、ROUGE 等指标，对这些候选 prompt 进行打分，衡量表达的多样性和质量。

3. **选择和优化**
   选出得分最高的 prompt 作为最终应用的提示词，也可以对其继续微调并重新评估，反复迭代，进一步提升效果。

实际案例

以乐队 T 恤下单为例，模型自动生成了如下 10 种表达方式：

1. I'd like to purchase a Metallica t-shirt in size small.
2. Can I order a small-sized Metallica t-shirt?
3. I'd like to buy one Metallica t-shirt in small.
4. One Metallica shirt, size small, please.
5. I want one Metallica t-shirt in size small.
6. I'd like to place an order for one Metallica t-shirt in small.
7. Small Metallica t-shirt, one please.
8. I'm looking for one Metallica t-shirt in small.
9. Can you get me one Metallica t-shirt in small?
10. One Metallica small t-shirt, please.

通过自动化流程，可以高效获得多样且优质的 prompt，极大简化了训练和部署聊天机器人的过程。

总结

Automatic Prompt Engineering 让提示词的生成和优化由模型自动完成，显著提升了开发效率和模型性能。通过自动生成、评估和筛选，开发者可以快速获得适合实际业务场景的高质量 prompt，推动了智能应用的落地。

Code Prompting：用大模型轻松生成、解释、翻译与调试代码

在现代开发过程中，如何高效生成和优化代码成为开发者关注的核心问题。Code prompting，借助大语言模型（如 Gemini），为开发者提供了全新的自动化编程体验，让代码生成、解释、翻译和调试变得前所未有的简单和高效。

---

1. 代码生成与自动化

通过简单的文字描述，开发者可以让大模型生成所需的代码。例如，如果需要一个脚本批量重命名文件，只需提出需求："写一个 Bash 脚本，询问文件夹名，并将所有文件名加上 draft 前缀。" 模型会自动生成脚本，包括用户输入、文件夹校验、遍历重命名等逻辑。实际测试表明，生成的 Bash 脚本可直接运行，实现批量文件重命名，极大提高了开发效率。

---

2. 代码解释与协作

在团队协作或阅读他人代码时，理解代码逻辑可能成为障碍。此时，只需将代码粘贴给大模型，模型便能按步骤详细解释每一行代码的作用，如输入处理、文件校验、循环重命名及结果提示。这样，开发者可以快速掌握代码结构，提升协作效率。

---

3. 代码翻译与迁移

将 Bash 脚本迁移到 Python 或其他语言，模型同样可以胜任。开发者只需提供原始脚本和目标语言要求，模型即可输出等价的 Python 代码，包括用户输入、文件校验、循环处理及文件重命名等操作。这样，跨语言迁移变得轻松快捷，适用于更多复杂场景。

---

4. 代码调试与优化

在实际开发中，代码难免出现错误。将出错的代码和报错信息提交给大模型，模型不仅能准确定位错误原因（如不正确的函数名），还会给出修改建议并返回优化后的代码。此外，模型还会提出进一步改进建议，比如处理文件扩展名、兼容含空格文件夹名、捕获异常等，使代码更健壮、易读、可维护。

总结

Code prompting 让开发者能够用自然语言与大模型对话，轻松实现代码的生成、解释、翻译和调试。不仅提高了开发效率，还降低了编程门槛，让每个人都能专注于更有创造力的工作。随着模型能力的提升，Code prompting 正在重塑开发流程，赋能新一代智能编程体验。

Prompt Engineering 最佳实践指南

在大语言模型的使用过程中，如何高效、高质量地与模型对话，往往取决于你的提示（prompt）设计是否科学合理。以下是实用的 prompt engineering 最佳实践，助你用更少的试错、更高的效率，获得理想的模型输出。

---

1. 提供示例，提升效果

在提示中加入一到数个示例（one shot / few shot），能显著提升模型理解和输出的准确性。示例为模型提供了参照标准，有助于其模仿你的表达风格、数据结构和预期内容。就像在教学时给出范例，模型会更好地“学会”你的需求。

---

2. 设计简洁明了的提示

优秀的提示应当简短、直接、易于理解。如果你自己都觉得提示太复杂，模型也很可能理解有误。避免不必要的信息，采用动词直接描述所需动作，比如“生成”、“分析”、“归类”、“总结”、“翻译”等，明确表达你的目标。

---

3. 明确期望输出，具体而非泛泛

清晰地描述你想要的输出格式、风格和内容。越具体的要求，模型越容易聚焦并生成准确结果。例如：“生成一篇三段式的博客，内容涵盖五大游戏主机，风格要有趣且具对话感。”比单纯“生成一篇博客”效果更好。

---

4. 优先使用积极指令，必要时添加约束

以“应该做什么”为主导，优先给出正向的、具体的操作指令，而非仅仅罗列禁止行为。正向指令更易让模型理解和发挥创造力。当需要防止有害内容或强制输出格式时，再适当加入约束条件。

---

5. 控制输出长度与格式

可通过设置最大 token 数，或在提示中直接说明输出字数/段落数来控制响应长度。例如：“请用推文长度解释量子物理。”同时，针对结构化任务，要求输出为 JSON、XML 等格式，有助于后续处理和减少幻觉。

---

6. 善用变量，提升可复用性

在提示中用变量（如 {city}）替代具体内容，使提示模板可灵活复用于不同输入，有效节省时间和降低维护成本，尤其适合集成到应用系统中。

---

7. 多样化尝试输入与输出格式

不同的表达方式、词汇选择、提示结构（如零样本、少样本、系统设定）都会影响模型表现。建议多做尝试，优化至最佳组合。对于分类等任务，混排 few-shot 示例的类别，避免模型过拟合顺序。

---

8. 跟进模型更新，及时调整提示

模型架构与功能会不断升级。建议关注新特性、适配新版本并调整提示内容。使用 Vertex AI Studio 等工具可便于测试、存档和管理不同版本的提示。

---

9. 输出结构化数据时优先 JSON，并做好修复准备

要求模型输出 JSON 等结构化数据，有助于统一风格、简化解析及减少幻觉。但需注意 JSON 输出更占用 token，易因截断而变得无效。可借助如 json-repair 等库自动修复不完整的 JSON，保证数据可用性。

通过遵循这些最佳实践，你可以大幅提升与大语言模型的交互质量，获得更精确、高效、可控的智能输出。持续优化提示内容，善于总结复用，将为你的 AI 应用开发带来长期价值。

JSON Repair：让结构化输出更可靠

在使用大语言模型（LLM）时，很多人喜欢让模型输出 JSON 格式的数据。这样不仅能减少幻觉，还能让数据具备类型、关系和排序等优势，非常适合后续的自动化处理和解析。

但 JSON 输出也有不小的挑战。由于 JSON 结构本身较为冗长，占用的 token 比纯文本多，容易导致生成被截断。一旦输出被截断，就会出现缺失括号、缺少字段等无效或损坏的 JSON，无法直接被程序识别和使用。

此时，json-repair 等工具就显得尤为重要。json-repair 可以自动修复不完整或格式错误的 JSON 输出，智能补全丢失的括号、字段等，让模型生成的 JSON 数据恢复可用。这对于处理大模型生成的结构化数据，特别是在 token 限制导致输出截断时，是非常实用的解决方案。

借助 json-repair，你可以安心让模型输出结构化 JSON 数据，极大提升数据处理的自动化和可靠性。

利用 JSON Schema 让大语言模型输入输出更智能

在与大语言模型（LLM）交互时，结构化的 JSON 输出已经成为提升数据可用性和自动化处理效率的首选方式。但其实，JSON 不仅适合用来组织输出，同样适合用来规范模型的输入。这里，JSON Schema 就派上了大用场。

JSON Schema 可以明确规定输入数据的结构和数据类型，相当于为 LLM 提供了一份“说明书”。这样，模型能准确理解哪些字段最重要、每个字段需要什么类型的数据，从而聚焦于关键信息，减少误解和偏差。比如，你可以在 schema 中加入时间戳或日期字段，还能建立字段之间的关系，让模型具备“时间感知”和“关系感知”能力。

举个例子：如果你想让 LLM 生成电商商品的描述，与其只给一段自由文本，不如用 JSON Schema 规定商品名、类别、价格、功能列表、发布时间等字段。这样不仅让输入更有条理，还能确保模型输出内容更精准、更相关。

通过这种结构化输入的方式，无论是处理大量数据，还是对接复杂的业务系统，LLM 都能更高效、更可靠地完成任务。

和其他提示工程师一起实验，提升提示效果

在设计高质量的提示（prompt）时，协同实验尤为重要。如果你正在尝试构建一个优秀的提示，不妨邀请其他提示工程师一同参与。每个人都可以根据本章提到的最佳实践，独立提出自己的提示方案。

通过这种方式，你会发现即便遵循相同的原则，不同的提示设计依然会带来表现上的差异。这为找到更优的提示结构和表达方式提供了宝贵机会。

团队共同实验，不仅能集思广益，还能帮助你更快识别有效的提示方式，从而提升模型的整体表现和应用效果。

记录每一次提示实验，让优化更高效

在进行提示工程（prompt engineering）时，详细记录每一次的提示尝试至关重要。这样做不仅能帮助你追踪不同提示的表现，还能为未来的优化和问题排查提供宝贵参考。

建议使用如 Google Sheet 的表格模板，记录每个提示的名称、目标、所用模型及其版本、参数设置（如 temperature、token limit 等）、完整的提示内容和输出结果。此外，还应追踪每次迭代、结果判定（OK/NOT OK/SOMETIMES OK）以及相关反馈。如果使用了专门的平台，比如 Vertex AI Studio，可以添加链接，方便随时复现历史实验。

对于涉及检索增强生成（RAG）系统的场景，还应记录影响提示内容的具体参数和配置。等到提示趋于理想后，建议将其保存到项目代码库中，与代码分离，便于维护和更新。同时，配合自动化测试和评估机制，持续验证提示的泛化效果。

提示工程是一个持续迭代的过程：不断尝试、分析与完善，直至获得满意的结果。每当模型或配置有变，都要回头复查并调整已有的提示。系统化的记录，是实现高效迭代和团队协作的关键。

高效使用CoT（Chain of Thought）提示的最佳实践

在应用CoT（Chain of Thought）提示工程时，有几个关键要点需要重点关注：

首先，始终将最终答案放在推理过程之后。原因在于，推理的生成会影响模型在预测最终答案时接收到的token序列，只有在完整推理后再给出答案，才能获得更准确的输出。

另外，为了确保答案能够被提取，建议在你的提示中将最终答案与推理过程区分开。例如，在答案前加上明显标识，方便自动化工具或人工快速定位最终结果。

在参数设置方面，务必将temperature设置为0。由于CoT提示依赖贪婪解码——即每一步都选择概率最高的下一个词——而且通常推理问题只有一个标准答案，因此无需引入随机性，temperature为0可以最大程度保证输出的一致性和准确性。

按照这些实践方法设计和优化你的CoT提示，可以显著提升模型的推理质量与答案准确率。

Prompt Engineering：多样化提示技术一览

在探索Prompt Engineering的过程中，我们学习了多种实用的提示技术，每种方法都有其独特的应用场景和优势：

- 零样本提示（Zero prompting）：无需示例，直接让模型完成任务，适用于模型具备较强泛化能力的场景。
- 少样本提示（Few shot prompting）：通过提供少量示例，引导模型理解任务要求，提升输出准确性。
- 系统提示（System prompting）：设定系统级别的指令或规则，帮助模型在更大范围内把控输出风格和内容。
- 角色提示（Role prompting）：为模型指定特定角色或身份，使其输出更贴合预设人物或专业视角。
- 上下文提示（Contextual prompting）：利用丰富的上下文信息，为模型提供更全面的参考，提升推理和生成质量。
- 反思提示（Step-back prompting）：引导模型跳出当前思路，从更高层次重新审视和解决问题。
- 连锁思维（Chain of thought）：鼓励模型分步推理，将复杂任务拆解为多个环节，逐步接近最终答案。
- 自洽性（Self consistency）：通过多次生成和比对答案，提升推理结果的稳定性和可靠性。
- 思维树（Tree of thoughts）：让模型并行探索不同思路，最终择优选择最佳解决方案。

这些提示技术的灵活运用，有助于充分发挥大模型的能力，提升任务完成的效果与质量。

Prompt Engineering 领域权威资源推荐

在深入研究Prompt Engineering的过程中，有许多权威文献与实践指南值得参考。以下是一些重要的研究论文和官方文档，涵盖了主流提示技术、采样方法及应用实践：

- Google Gemini及Google Workspace Prompt Guide，提供了最新的产品信息与官方操作指南。
- Google Cloud的Prompt设计与采样方法文档，详细讲解了如Top-P和Top-K等生成参数设置方式。
- 零样本学习（Zero Shot）和少样本学习（Few Shot）的开创性论文，解析了大模型在有限数据下的泛化能力。
- “Take a Step Back”与“Chain of Thought Prompting”等论文，系统介绍了抽象推理、分步推理等高阶提示技术。
- “Self Consistency”、“Tree of Thoughts”与“ReAct”等创新方法，推动了复杂问题求解和推理一致性的提升。
- “Automatic Prompt Engineering”则探讨了大模型自动优化提示的前沿方向。

这些文献为Prompt Engineering提供了扎实的理论基础和丰富的实操经验，是学习和提升提示工程能力的重要参考资料。