大模型提示词技巧Prompt Engineering，看这一篇就够了

原创 腾讯程序员 腾讯技术工程 2025年09月12日 17:37 广东

文末抽奖~

文末抽奖~

文末抽奖~

作者：rocky

> 让模型听话，按照要求思考，关键就在Prompt。

前言

你在写prompt时候，是不是总觉得大模型它不听话。要么答非所问、要么一堆废话。扒开思考过程仔细阅读时而觉得它聪明绝顶，时而又觉得它愚蠢至极。明明已经对了怎么又推理到错的地方去了，明明在提示词中提醒过了不要这么思考它怎么就瞎想了。这也许就是每一个Prompt Engineer的困扰。怎么能让模型按照要求去思考。

长提示词到底应该怎么写，有没有方法可以一次命中，找到那个终极的提示词。 答案是否定的，一篇成功的长提示词总是要经历初始版本、调优、测试、再调优。不过这个过程中有规律可循，有方法可套。 以下就是被提示词反复捶打，经历无数痛苦经历后总结的一套提示词写作方案，保你可以得到满意的长提示词，让模型听话。

结构

也许你小某书、某站看过了各种提示词结构，比如：CRISE，BROKE，ICIO等等，这些框架当然是有很大价值的，在非精准类问题(精准类问题：数据查询分析，非精准类问题：文本解析，写作、翻译等)或者非复杂性问题上没有问题。在复杂性高的精准性问题上就没有那么有效了。我们这边一直探索的是大模型在数据分析场景的应用，对准确性要求极高，覆盖的场景也非常的复杂，经过探索和尝试，总结出来一套形式有效的提示词结构：

角色/任务 + 核心原则 + 上下文处理 + CoT(Chain of Thoughts) + 输出规范 + Few-Shot

还需要适当添加要求和限制，下面会以实战经验来讲解每一个模块应该怎么写。

写在前面

模型是接收Prompt的主体，同时也是写Prompt的高手，在初始版本和调优过程中也可以起到关键作用。

借助模型生成初始版本Prompt

1. 准备query和期望输出的结果30条
2. 准备上下文输出，和文本结构介绍
3. 清晰描述模型要实现的目标以及输出的提示词框架

将以上内容给到大模型，可以快速得到初始版的提示词，比自己动手写第一版要有效的多。

```
# Role: 你是一个高级Prompt Engineer
# Task: 基于输出信息，高质量输出Prompt
# 核心规则: 按照框架要求撰写Prompt

# 上下文处理
1. 输出输入示例全部在<context></context>标签内，文件类型为csv，第一列为输入query，第二列为预期输出结果
2. 维度知识库全部在<context1></context1>标签内，文件类型为txt，不同维度以&&&&隔开，维度层级以树状结构呈现

# 要求
1. prompt框架: 角色/任务 + 核心原则 + 上下文处理 + 执行步骤(体现思考过程) + 输出提示 + 生成案例

<context>文档1</context>
<context1>文档2</context1>
```

借助模型优化Prompt

1. 准备测试集以及当前prompt生成的结果
2. 添加准确结果和备注，备注描述生成错误结果的原因

```
# Role: 你是一个高级Prompt Engineer
# Task: 优化当前Prompt

# 上下文处理
1. 当前prompt<context></context>标签内，文件是md格式
2. 当前prompt测试结果在<context1></context1>标签内，文件为csv格式，第一类为用户query，第二列为当前为当前答案，第三列为正确答案，第四列为当前答案得分，第五列为错误原因描述
3. 维度知识库全部在<context2></context2>标签内，文件类型为txt，不同维度以&&&&隔开，维度层级以树状结构呈现

# 执行步骤
1. 仔细阅读当前prompt测试结果，分析错误原因
2. 仔细分析维度知识库和当前prompt定位产生错误的原因
3. 根据错误原因和文档编写新的prompt

<context>文档1</context>
<context1>文档2</context1>
<context2>文档3</context2>
```

使用模型初始化和优化可以解决基础问题，真正的优化还是要靠自己。

Prompt格式

md或者json，我选择md格式。

不仅仅是因为md格式比较好看，主要是为你md格式结构清晰，撰写方便，而且拓展性很好。总结下来md是比较好的选择。

json格式虽然结构清晰，但是扩展性太差，写的太长了容易把自己搞晕，慎重选择。

Prompt模块

不同模块承担不同的作用，复杂程度不同需要的模块也不同。

角色&任务

角色辅助，讲清楚任务。 此部份在prompt最前面，是最高指令，告诉模型它是谁，要干嘛。

角色：模型本身是具备各领域知识能力的，解决当前具体问题需要调用模型哪方面的能力，是通过角色定位完成的。 你是一名牙科医生，你是一名数据分析师、你是一名川菜厨师等让模型从一个杂学家变成专业领域的科学家。

任务：一句话讲清楚模型要干嘛，数据分析师可以写sql查询数据、可以使用python分析数据、可以数据可视化，也可以写分析报告。

角色和任务约束模型调用某方面能力完成一个具体的事情。

核心原则

核心原则可以一开始就输出，也可以在调优过程中生成。 可以理解为模型执行任务时要遵守的最高原则，纲领性质的要求。所以核心准则不能多，3条以内，超过3条很容易就失效了。

比如在生成sql的prompt中，为了保证生成的sql可以查询出数据，就得有以下核心原则。

```
# 核心原则:
1. 必须基于表结构生成SQL，其中的表明和字段名必须来自表结构信息
2. SQL中非时间筛选条件必须基于维度解析、客户解析
```

比如在做分词提取时，我们的分词倾向性也可以写在核心原则内

```
## 核心原则
1. **宁可多输出，不要漏输出** ： 当不确定某个词是否为有效维度时，倾向于输出该词
2. **复合词完整性优先** ： 优先识别完整的复合词，避免过度拆分
3. **修饰词谨慎标记** ：修饰词谨慎标记。且修饰词不可单独输出，必须和有效维度词成对输出
```

一开始实现某个任务时，核心原则可能还没有，在优化过程中有些问题在提示词主体中总是解决不了，可以考虑在核心原则中优化。对于模型来源核心原则会被考虑的权重是比较高的，仅次于角色和任务。

上下文处理

当前Context Engineering概念比Prompt Engineering更加流行，一句话概括就是让上下文以恰当的格式出现在恰当的位置，知识库可以包括：多论对话的长短记忆、知识库rag结果、提示词、工作流上游输出等。能让上下文发挥最大作用，就必须把上下文讲清楚，放对位置。

上下文模块组织原则：

1. 上下文内容比较长，最好放到最后，以免打断提示词

2. 上下文结构讲清楚，合适和组织形式影响token数量也会影响性能(不展开讲)

3. 上下文在任务中承担的作用和价值

举例：在生成sql环节，上下文输入较多，具体组织形式如下：

```
# 数据预处理
## **表结构**  生成的SQL必须基于表结构，表结构信息全部在<context></context>XML标签内，主要包括表元数据和表周期(月表或者日表，决定时间的格式)
## **维度解析** 用户Query维度解析结果，维度解析结果全部在<context1></context1>XML标签内，决定生成SQL的维度条件和指标（格式：维度名称1=维度值1,维度名称2=维度值2，指标=指标名称）
## **客户解析** ：用户Query客户解析结果，全部在<context2></context2>XML标签内，其中的gid或者cid决定生成SQL的集团或者客户筛选条件
## **用户Query** ：全部在<context3></context3>XML标签内
## **表当前日期** ：{{#1754381512719.structured_output#}} 其中table_month是月表的表当前月，table_day是日表的表当前日
```

上下文输入：一般放在提示词结尾处：

```
<context>
{{#context#}}
</context>
<context1>
{{#1753157521153.output#}}
</context1>
<context2>
{{#1754893243111.output#}}
</context2>
<context3>
{{#sys.query#}}
</context3>
```

特别注意：上下文的结构和形式的优化一般适合提示词的优化协同的，二者同步优化才能达到最好的效果。

CoT(Chain of Thoughts)

CoT

CoT本来是提示词的一种框架，是针对逻辑比较强的任务场景提出的。就是要提醒或者约束模型按照要求思考，以提升准确率。

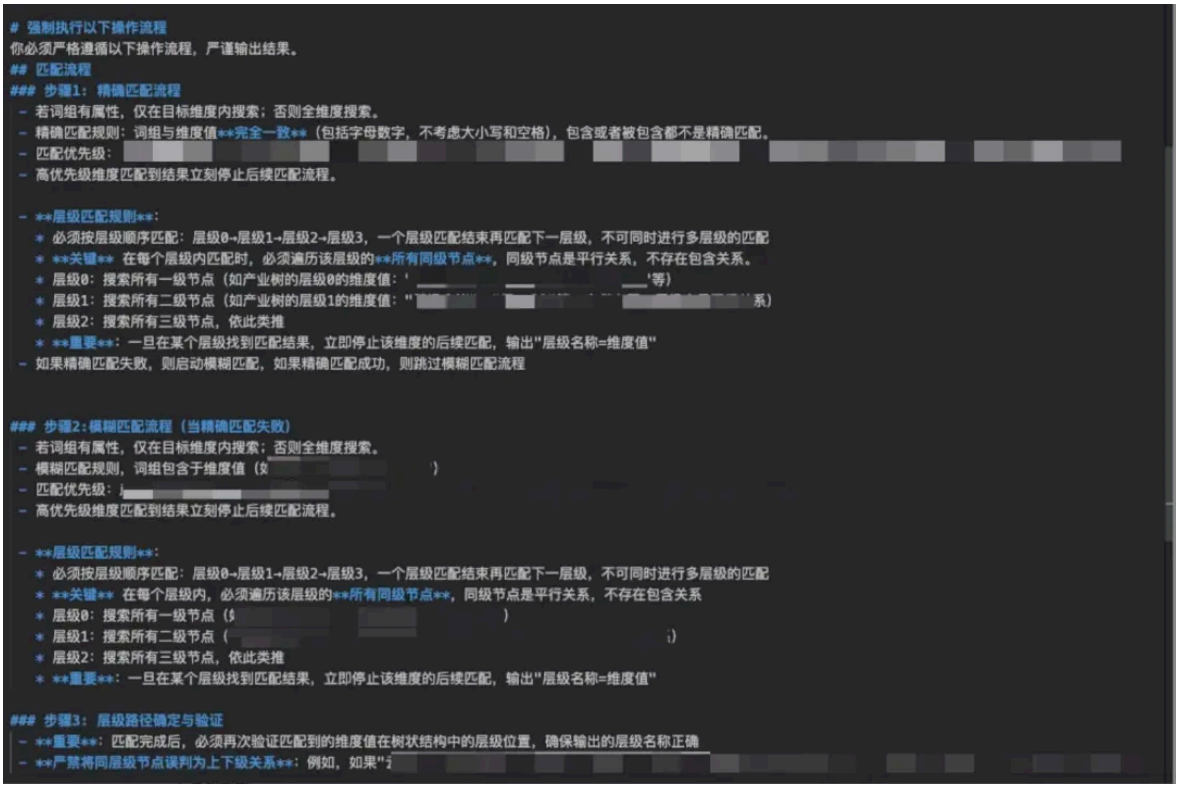
举个经典例子：小明有5个苹果，3个梨。妈妈拿走2个苹果，爸爸给了1个梨，小明拿1个梨和姐姐换了1个苹果，请问最终小明有几个苹果几个梨。

提示词1: 请回答最终小明有几个苹果几个梨； 这时候答案很有可能是错的。

提示词2: 第一步：将小明每次获取、失去、交换所有物品作为一个节点，奖整个过程按照节点切分成不同的计算任务 第二步：计算每一个节点结束后小明所有物品的数量 第三步：计算出最终的结果后复盘是否准确；这个时候模型就是一步一步计算结果，更容易得到正确的答案。

在复杂场景下，CoT，也可以理解为执行流程或者说思考过程，可以作为整个prompt一部份，模型在充分理解任务和上下文之后，再按照CoT步骤执行拆解任务，往往可以让模型按照要求执行，听话程度高出很多。我们的经验是可以提升准确了20个percent。

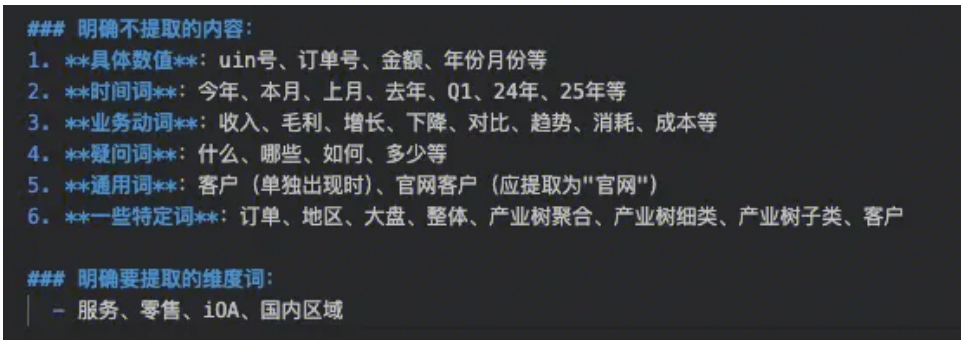
示例如下：维度解析



要求和限制

要求和限制，看是什么级别，可以写在CoT模块内，也可以单独一个模块，因地制宜即可。

要求和限制一般是任务中需要特殊强调、特殊处理的逻辑，建议二者分开写。举例：



特殊逻辑表达

在写prompt中有些逻辑用文字特别难以准确表达，有时候准确表达出来需要上百字，对于模型准确理解就更难了。 这个时候可以考虑使用伪代码来表达，模型理解起来既快又准。

比如，收入月报每月定稿时间13日，如何根据当前时间取出月表的最新时间，并考核时间的格式。

````
1. **table_month 表当前月计算逻辑**:
```python
if current_date.day < 13:
    table_month = current_date.month - 2   # 注意：日期小于13，月份要减2
else:
    table_month =current_date.month - 1

return table_month.strftime(%Y%m01)   # 格式化输出表当前月
```

2. **table_day 表当前日计算逻辑**:
```python
table_day = current_date.day - 1

return table_month.strftime(%Y%m%d)   # 格式化输出表当期日
````

输出规范

模型太爱表达了，它往往不会只输出你想要的内容，总是输出很多自己的思考过程或者考虑的因素，以表达自己的聪明。又或者是不按照要求的格式输出，对输出的规范要求必不可少，一些平台可以实现结构化输出，不过结构化输出的基础是要模型能输出结构清晰的结构。

输出规范一般包含两部分内容：

1. 期望输出的内容和结构
2. 禁止输出的内容和结构

举例如下：

```
## 输出格式
只输出提取到的维度词，多个维度词用逗号分隔。如果没有有效维度词，输出"无"。
```

```
## 结果输出
- 一次词组只输出一个结果
- 输出格式："词组 -- 层级名称=维度值"
- 结果净化：禁止输出推理过程，直接输出结果
- 无匹配时：输出无维度。
```

Few-Shot

提升准确率非常有效的手段，就好比一个应届生，你让他去看一个文件，然后按照文件要求做事，很难理解到位。 如果你再提供一两个例子，基本上聪明的同学就能很好的完成任务。模型当然属于聪明的同学这一类。示例一定要按照上述CoT的过程来写，二者一致则能让模型最大限度的按照既定的要求思考。

举例如下：
```

```
# 案例：
用户Query："重点产品收入同比是多少？"

黑话解读：
自上而下逐个取出黑话，严格遵守匹配规则进行匹配，
- EMR （匹配失败）
- ES（匹配失败）
- WeData （匹配失败）
- 重点产品 （匹配成功） 输出 重点产品 -- 二级产业树聚合=计算,存储,网络，且停止后续匹配
```

写在最后

不同的模型、不同的场景也许写prompt的细节不尽相同，但整体的框架是相通的。按照这个框架，人人都可以写出满意的Promt！ 以上分享来自腾讯CSIG磐石数据中心。在数据洪流时代，企业不缺数据，缺的是从数据中洞见未来的能力。CSIG磐石数据中心以领先的AI数据分析引擎，为您打造“会思考、能决策”的智能数据中枢！

参与抽奖方式：

点击下面卡片关注俺们公众号

然后在后台回复“912”即可

周日下午6点准时开奖

如果没中奖也没关系，每周都会抽奖~

记得设为星标⭐

修改于2025年09月23日