# OpenAI官方Prompt工程操作指南详解

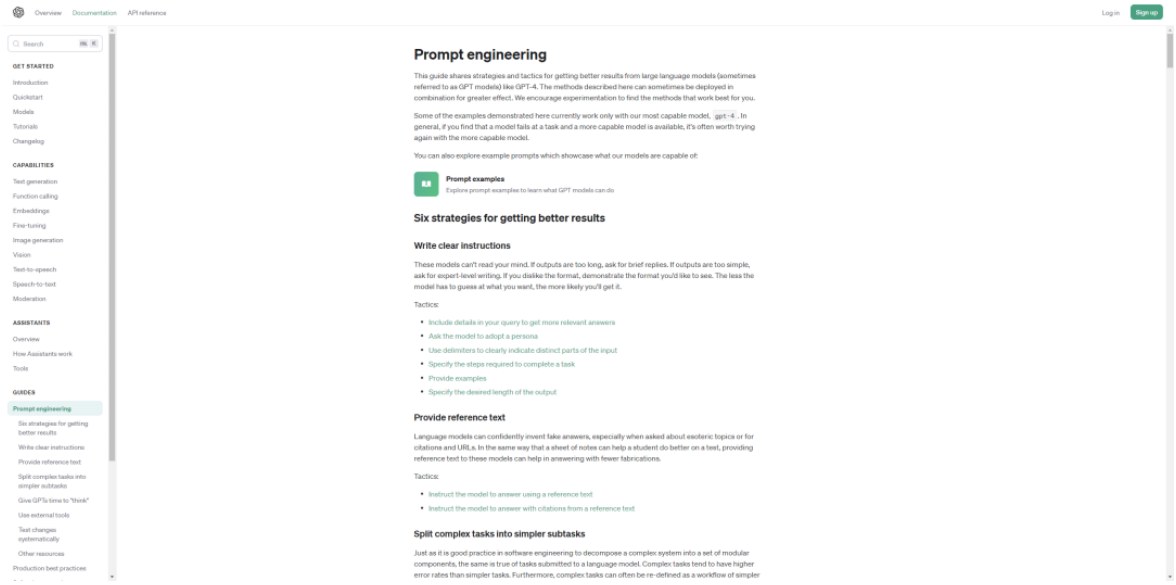
**前言：其实Prompt从用户的角度出发，更多是关于如何用好大模型，而非开发大模型。并且技术难度也不高，所以作为选学，有兴趣的学习者可以看一下。**

其实一直有很多人问我，Prompt要怎么写效果才好，有没有模板。

我每次都会说，能清晰的表达你的想法，才是最重要的，各种技巧都是其次。但是，我还是希望发给他们一些靠谱的文档。

但是，网上各种所谓的Prompt框架、教程，真的乱七八糟，让人头都大。

直到前两天，12月15号，OpenAI在他们的文档里上线了Prompt engineering，也就是提示词工程指南，至此，终于算是有了一个权威且有效的Prompt工程标准文档。




我花了20分钟看完了后，其实挺会心一笑的，整篇指南简洁、明确、高效，写的非常棒。

OpenAI提到6条大的原则，分别是：

**1. Write clear instructions（写出清晰的指令）**

**2.Provide reference text（提供参考文本）**

**3. Split complex tasks into simpler subtasks（将复杂的任务拆分为更简单的子任务）**

**4. Give the model time to "think"（给模型时间“思考”）**

**5. Use external tools（使用外部工具）**

**6. Test changes systematically（系统地测试变更）**

我用这篇文章，来通俗易懂的给大家聊一下具体的原则和例子，第六条可以不看，对普通用户没啥大用。最后我会再放一张脑图，没空看的可以收藏一下文章，然后滑到最后

去保存脑图。

我觉得可以信我，市面上99%的Prompt框架和技巧，都不如这一篇文章有用。

## 一. 写出清晰的指令

这个其实就是我天天说的，任何Prompt技巧都不如清晰的表达你的需求，这就像人与人沟通一样，话都说不明白，怎么能让对面理解你呢？一味的靠抄Prompt模板，其实不是长久之计。

所以，写出清晰的指令，是核心中的核心。

如何写出清晰的指令，OpenAI给出了6条小技巧：

### 1. 把话说详细

尽量多的提供任何重要的详细信息和上下文，说白了，就是把话说明白一点，不要一个太笼统。

比如：不要说：“总结会议记录”

而是说：“用一个段落总结会议记录。然后写下演讲者的 Markdown 列表以及他们的每个要点。最后，列出发言人建议的后续步骤或行动项目（如果有）。”

### 2. 让模型充当某个角色

你可以把大模型想象成一个演员，你要告诉他让他演什么角色，他就会更专业更明确，一个道理。

比如：充当一个喜欢讲笑话的喜剧演员，每当我当我请求帮助写一些东西时，你会回复一份文档，其中每个段落至少包含一个笑话或有趣的评论。

### 3. 使用分隔符清楚地指示输入的不同部分

三引号、XML 标签、节标题等分隔符可以帮助划分要区别对待的文本节。可以帮助大模型更好的理解文本内容。我最喜欢用"""把内容框起来。

比如：用50个字符总结由三引号分隔的文本。"""在此插入文字"""

### 4. 指定完成任务所需的步骤

有些任务能拆就拆，最好指定为一系列步骤。明确地写出这些步骤可以使模型更容易去实现它们。

比如：使用以下分步说明来响应用户输入。

步骤1 - 用户将为您提供三引号中的文本。用一个句子总结这段文字，并加上前缀“Summary:”。

步骤2 - 将步骤1中的摘要翻译成西班牙语，并添加前缀“翻译：”。

### 5. 提供例子

也就是经典的少样本提示，few-shot prompt，先扔给大模型例子，让大模型按你的例子来输出。

比如：按这句话的风格来写XX文章："""落霞与孤鹜齐飞，秋水共长天一色。渔舟唱晚，响穷彭蠡之滨"""

### 6. 指定所输出长度

可以要求模型生成给定目标长度的输出。目标输出长度可以根据单词、句子、段落、要点等的计数来指定。中文效果不明显，同时你给定的长度只是个大概，多少个字这种肯定会不精准，但是像多少段这种效果就比较好。

比如：用两个段落、100个字符概括由三引号分隔的文本。"""在此插入文字"""

## 二. 提供参考文本

给大模型文本或者文档，能大幅度的降低大模型胡说八道的概率。其实就是把大模型当知识库来用。

### 1. 让模型使用参考文本作答

知识库的经典用法，让大模型使用我们提供的信息来组成其答案。

比如：使用提供的由三重引号引起来的文章来回答问题。如果在文章中找不到答案，请写“我找不到答案”。

"""<在此插入文档>"""

"""<在此插入文档>"""

问题：<在此插入问题>

### 2. 让模型通过引用参考文本来回答

如果已经给了文本，则可以直接要求模型通过引用所提供文档中的段落来为其答案添加引用。可以提高正确性，增加可验证性。

比如：您将获得一份由三重引号和一个问题分隔的文档。您的任务是仅使用提供的文档回答问题，并引用用于回答问题的文档段落。如果文档不包含回答此问题所需的信息，则只需写：“信息不足”。如果提供了问题的答案，则必须附有引文注释。使用以下格式引用相关段落（{“引用”：...}）。

"""<在此插入文档>"""

问题：<在此插入问题>

## 三. 将复杂的任务拆分为更简单的子任务

其实跟人类一样，你作为Leader，让下属一次性去做一个非常大的事，出错的概率是很大的，很多大项目也是这样，你甚至无从下手。所以经常我们在工作中，都说的是要拆，拆各种细节、子任务、子目标等等。大模型也是同样的道理。

把复杂的任务给拆给更为简单的子任务，大模型会有更好的表现。

### 1. 使用意图分类来识别与用户查询最相关的指令

意图识别是一个很经典的例子。比如在客服场景中，用户问了一个问题“我断网了咋整”，你让大模型直接回复其实是挺蛋疼的，但是这时候就可以拆，先拆大分类下的意图识别，再回答具体的问题。

比如还是“我断网了咋整”这个问题：

步骤1，先判断问题类别：

```
我们将向您提供客户服务查询。将每个查询分为主要类别和次要类别。提供 json 格式的输出，其中包含以下键：主要和次要。

主要类别：计费、技术支持、帐户管理或一般查询。

计费二级类别：
- 取消订阅或升级
- 添加付款方式
- 收费说明
- 对收费提出争议

技术支持二级分类：
- 故障排除
- 设备兼容性
- 软件更新

账户管理二级分类：
- 重设密码
- 更新个人信息
- 关闭账户
- 账户安全

一般查询二级类别：
- 产品信息
- 价钱
- 反馈
- 与人类交谈
```

现在，大模型根据步骤1，知道“我断网了咋整”是属于技术支持中的故障排除了，我们就可以再继续步骤2：

```
您将收到需要在技术支持环境中进行故障排除的客户服务查询。通过以下方式帮助用户：

- 请他们检查所有进出路由器的电缆是否已连接。请注意，随着时间的推移，电缆松动是很常见的。
- 如果所有电缆均已连接并且问题仍然存在，请询问他们正在使用哪种路由器型号
- 现在您将建议他们如何重新启动设备：
-- 如果型号是 MTD-327J，建议他们按下红色按钮并按住 5 秒钟，然后等待 5 分钟后再测试连接。
-- 如果型号是 MTD-327S，建议他们拔下并重新插入，然后等待 5 分钟再测试连接。
- 如果客户的问题在重新启动设备并等待 5 分钟后仍然存在，请通过输出 {“IT 支持请求”} 将他们连接到 IT 支持。
- 如果用户开始询问与此主题无关的问题，请确认他们是否想结束当前有关故障排除的聊天，并根据以下方案对他们的请求进行分类：

<在此处插入上面的主要/次要分类方案>
```

这时候，用户的“我断网了咋整”就能得到非常有效的回答了。

**2. 对于需要很长对话的对话应用，总结或过滤之前的对话**

这个技巧偏开发者。普通用户可以跳过。

因为模型具有固定的上下文长度，因此用户和助手之间的对话无法无限期地继续。

解决此问题有多种解决方法，第一个是总结对话中的历史记录。一旦输入的大小达到预定的阈值长度，这可能会触发总结部分对话的查询，并且先前对话的摘要可以作为系统消息的一部分包括在内。或者，可以在整个对话过程中在后台异步总结之前的对话。

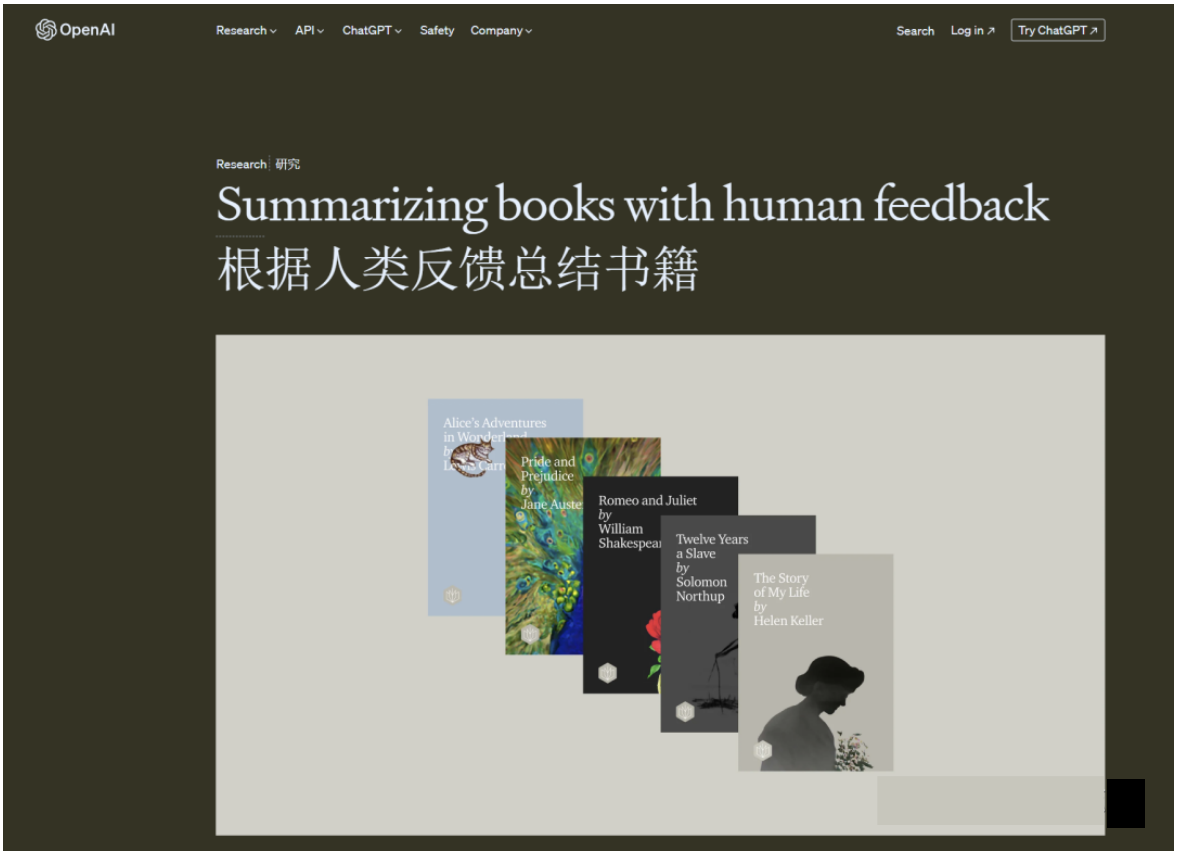
这两种方法都行，或者还可以把过去的所有聊天记录存成向量库，后续跟用户对话的时候动态查询嵌入，也可以。

**3. 分段总结长文档并递归构建完整总结**

同样偏开发者。普通用户可以跳过。

其实就是总结几百页PDF文档的原理，比如让大模型总结一本书，肯定是超Token上限了嘛，所以可以使用一系列查询来总结文档的每个部分。章节摘要可以连接和总结，生成摘要的摘要。这个过程可以递归地进行，直到总结整个文档。 OpenAI 在之前的研究中已经使用 GPT-3 的变体研究了这种总结书籍的过程的有效性。

详细的可以看这篇文档：https://openai.com/research/summarizing-books



## 四. 给模型时间“思考”

think step by step（一步步思考）这个神级提示词的源头。其实也就是链式思考（CoT），Chain-of-Thought Prompting，非常非常有用的一个策略。

还是跟人一样，我直接问你12314992*177881等于多少你肯定也懵逼，但是我要是给你时间让你一步步计算，学过小学数学的我觉得都能算出来对吧。

OpenAI在CoT的基础上，又详细给出了3个技巧：

### 1. 让模型在急于得出结论之前找出自己的解决方案

比如你扔个数学题给大模型，你让他判断对或者不对，你会发现结果很随机，一会对或者不对，但是如果你先让他自己做一遍，再去判断对与不对，结果就会准非常多了。

比如你可以说：首先制定自己的问题解决方案。然后将你的解决方案与学生的解决方案进行比较，并评估学生的解决方案是否正确。在你自己完成问题之前，不要决定学生的解决方案是否正确。

### 2. 使用内心独白来隐藏模型的推理过程

非常有意思的一个技巧，你可能会问不是说一步一步思考把推理过程放出来效果会更好嘛。

你说的对，但是这条技巧是面对开发者的，对于某些应用程序，大模型用于得出最终答案的推理过程不适合与用户共享。例如，在辅导应用程序中，我们可能希望鼓励学生得出自己的答案，但模型关于学生解决方案的推理过程可能会向学生揭示答案。

所以就有了这么一个内心独白的技巧。内心独白的想法是让模型将原本对用户隐藏的部分输出放入结构化格式中，以便于解析它们。然后，在向用户呈现输出之前，将解析输出并且仅使部分输出可见。

比如：

步骤 1 - 首先找出你自己的问题解决方案。不要依赖学生的解决方案，因为它可能是不正确的。将这一步的所有工作用三引号 (""") 括起来。

第 2 步 - 将您的解决方案与学生的解决方案进行比较，并评估学生的解决方案是否正确。将这一步的所有工作用三引号 (""") 括起来。

第 3 步 - 如果学生犯了错误，请确定在不泄露答案的情况下可以给学生什么提示。将这一步的所有工作用三引号 (""") 括起来。

步骤 4 - 如果学生犯了错误，请向学生提供上一步的提示（在三重引号之外）。不要写“步骤 4 - ...”，而写“提示：”。

Problem Statement: <insert problem statement>

Student Solution: <insert student solution>

---

问题陈述：<插入问题陈述>

学生解决方案：<插入学生解决方案>

接下来，我们可以让模型使用所有可用信息来评估学生解决方案的正确性。

将您的解决方案与学生的解决方案进行比较，并评估学生的解决方案是否正确。

问题陈述：“”“<插入问题陈述>”“”

您的解决方案：“”“<插入模型生成的解决方案>”“”

学生的解决方案：“”“<插入学生的解决方案>”“”

最后，我们可以让大模型使用自己的分析来以乐于助人的导师的角色构建回复。

你是一名数学导师。如果学生犯了错误，请以不透露答案的方式向学生提供提示。如果学生没有犯错，只需给他们一个鼓励性的评论。

问题陈述：“”“<插入问题陈述>”“”

您的解决方案：“”“<插入模型生成的解决方案>”“”

学生的解决方案：“”“<插入学生的解决方案>”“”

分析：“”“<插入上一步生成的模型分析>”“”

用多次跟API通讯的方式，同时隐藏模型的推理过程，来完成一次学生的辅导方案对话。

**3. 询问模型在之前的过程中是否遗漏了什么内容**

这个技巧在长文本问答中常用，比如我们给了一个文档，要让大模型模型来列出与一个特定问题相关的信息。如果源文档很大，模型通常会过早停止并且无法列出所有相关信息。在这种情况下，通过使用后续的promtp让模型查找之前传递中错过的任何相关信息，通常可以获得更好的性能。

比如我让他根据我的文档，给我列出这个问题在文档中的相关片段：“北京烤鸭到底好吃在哪”，然后让他用JSON格式输出

[{"相关片段"： "..."},

在输出停止以后，我们可以再问一句：

还有更多相关片段吗？注意不要重复摘录。还要确保相关片段包含解释它们所需的所有相关上下文 - 换句话说，不要提取缺少重要上下文的小片段。

## 五. 使用外部工具

大模型并不是万能的，很多东西吧，大模型的效果并没有那么好，比如数学、比如一些实时问题等等，所以需要一些外部工具来帮助处理。

换句话说，如果第三方工具能稳定的获得结果，那其实并不需要大模型去做什么，或者只让大模型做一个答案组装类的工作就够了。

### 1. 使用基于嵌入的搜索实现高效的知识检索

绝大部分知识库的原理，检索增强生成 (RAG)，Retrieval Augmented Generation，比如我问如何评价马上要上映的电影《海王2》，你让大模型自己去答肯定就废了，它是静态的，根本不知道《海王2》要上映了，所以需要先去联网进行查询，查完以后把一堆资料灌回来，让大模型自己根据自己查到的这些资料进行回答。这是动态的信息。

但是也有静态的知识库，就是用的向量匹配的方式，常见步骤：加载文件 -> 读取文本 -> 文本分割 -> 文本向量化 -> 问句向量化 -> 在文本向量中匹配出与问句向量最相似的top k个 -> 匹配出的文本作为上下文和问题一起添加到prompt中 -> 提交给大模型生成回答。

就是这么玩的。

### 2. 使用代码执行来进行更准确的计算或调用外部API

都知道大模型自己的计算能力垃圾，所以OpenAI建议，如果遇到需要计算的东西，最好让大模型写一段计算的Python代码，毕竟Python最计算题很成熟了。

比如：求以下多项式的所有实值根：3*x**5 - 5*x**4 - 3*x**3 - 7*x - 10。您需要通过将 Python 代码括在三个反引号中来编写和执行，例如"""代码放在这里"""。用它来执行计算。

当然，都用Python了，你也可以把自己的API文档复制给它，让大模型知道该如何写代码调用你的API。

**3. 给模型提供特定的功能**

很偏开发者的一个技巧，普通用户可以直接跳过。

简而言之，你可以通过 API 请求，传递一系列特定的函数描述。告诉模型哪些函数是可用的，以及这些函数的参数应该是什么样的。然后模型模可以生成相应的函数参数，这些参数随后会以 JSON 格式通过 API 返回。

你都拿到JSON数组了，跟数据库可以做多少交互相信也不用我多说了吧，做数据查询、数据处理等等，啥玩意都行。

处理完以后再返回一个JSON数组给大模型，让大模型变成人类语言输出给用户，完事。

## 六. 系统地测试变更

可以直接跳过此节，对于普通用户几乎没用。

主要是帮助开发者判断更改Prompt（例如新指令或新设计）是否使系统变得更好或更差。毕竟大部分时间的样本量都比较小，很难区分真正有改进还是纯粹的运气。

所以，OpenAI建议搞个评估程序，用来判断优化系统的设计是否有效。

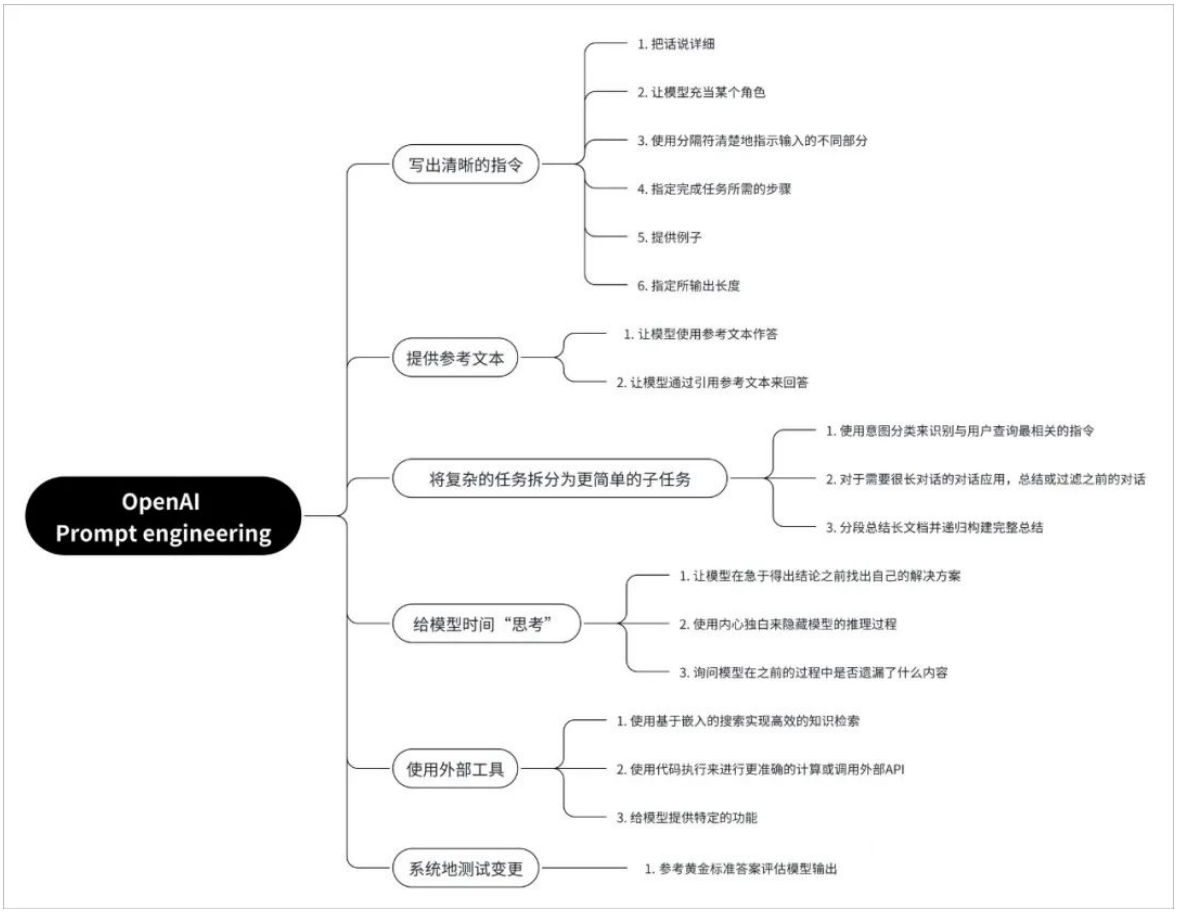
这块我就不细说了，有兴趣的或者正在开发自己的AI应用的，可以自己去看看：

https://platform.openai.com/docs/guides/prompt-engineering/strategy-test-changes-systematically

## 写在最后

OpenAI这个Prompt engineering写的相当详细了，我真的绝对，比市面上太多太多的框架和课程都要好。

为了方便大家偶尔复习，我也做了一张脑图，可以跟文章结合着看。




说实话，我一直认为，Prompt这个东西，就是日常表达能力的一个映射。

日常中跟人沟通能力很强的人，其实不知道这些技巧，一样能跟大模型协同的很好。

表达清楚自己的需求，与人协同顺利，才是最重要的啊。

突然想起了我们公司的一条价值观：

**“布阵清晰、传球舒服、接球靠谱”**

我深以为然。

与人、与AI，其实本质没有什么不同。