

# Everything about Prompt Engineering

Course Code: 4E02-1

Author: Lloyd Sun

Date: Sep.11.2024

Version: V1.0

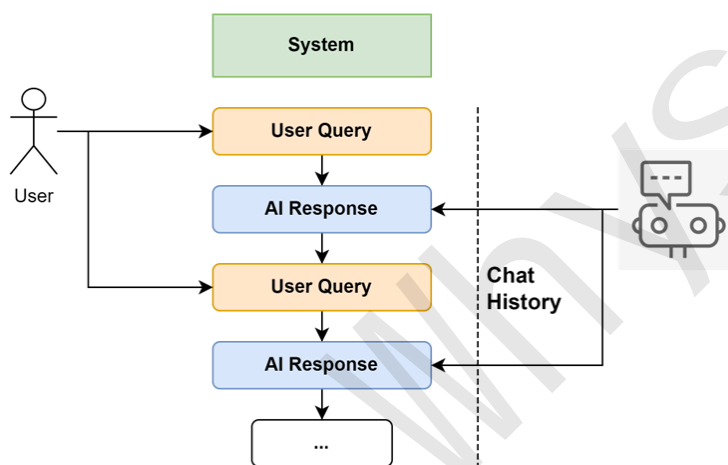
## 1 Prompt Engineering Overview & Principles

提问是值得终身学习的技能。《银河系漫游指南》中的高级文明为了追寻宇宙终极问题的答案（the answer to life, the universe, and everything），使用超级计算机计算了750万年，得到的答案是42。值得思考的是，无法理解答案的原因也许是我们不知道要问什么问题。提出问题比解决问题更重要。

这次分享的是关于大模型提示词工程的内容。回顾4E01，大语言模型的能力有三点：1）自然语言；2）结构化输出；3）逻辑推理。对以上三点有基本的了解，是理解本次分享内容的前提。

### 1.1 What is a Prompt

提示词可以简单理解为大语言模型的输入。以最常见的ChatBot场景为例，用户提出问题(Query)，ChatBot/LLM给出回复(AI Response)，用户的提问与AI的回复构成了聊天历史(下图橙色和蓝色部分)。在聊天历史之前，通常会设计一段通用的Prompt，给LLM充分的上下文与指示，一般称为System Prompt/Message（系统级提示词）。



从LLM视角来看，它的输入是系统级提示词+对话历史，其中对话历史又包括用户的提问与自己之前的回复。可以设定需要留存的对话历史长度，也就是设定LLM能够记住多久/多少轮之前的对话，这种“记忆”一般称为大模型的短期记忆。从这里也能看出，所谓的PE是包含多个部分的，其中有静态也有动态的部分。当然，有人会进一步区分什么是Prompt、什么是Input，但这样的区分似乎并没有实际的益处。

对于构建LLM应用来说，需要遵循1.2中的流程，迭代设计系统级提示词；对于一般用户来说，只需要遵循PE的一些原则，尽量让问题清楚、提供充分的上下文即可，当然也可以进行更仔细的设计。

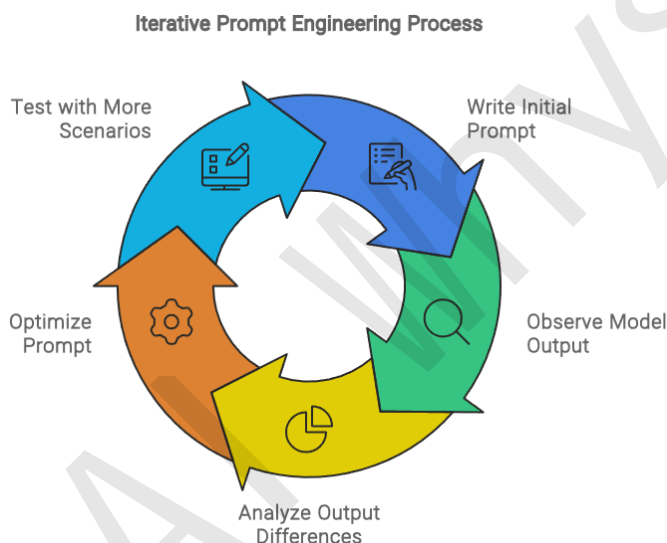
理解ChatBot中提示词的含义是至关重要的，甚至基于这部分的理解，完全能够掌握LLM主流的其他应用执行流程了。比如，RAG应用就是在每次用户提问之后，去知识库/向量数据库当中，找到与Query语义相近的知识，与用户的提问一同作为LLM的输入；比如，Agent的核心，就是先让LLM进行特定格式输出（JSON/API之类），基于这样的格式化输出调用相应函数，把函数的返回结果与LLM输出一同输出出来；再比如多智能体的一个核心是，如何使上述的多轮对话，由多个Agent组成的系统自动地执行出来，最后再返回最终结果。



## 1.2 The Key: Iterative Process

与之前分享的观点一致，并没有什么完美的提示词或模板，好用的提示词也不是一开始就能设计出来的。PE(Prompt Engineering)不是关于完美的提示词模板，而是关于设计优秀提示词的流程与方法。当然，这里指的提示词设计，更多的是针对LLM应用开发者的，面临的问题通常是如何设计系统级提示词、系统优化动态与静态部分；对于最终用户而言，了解大模型的特性以及一些提示词技巧，比如设定角色、给充分的上下文、清楚描述任务等等即可。

PE一般的设计流程是先写一段提示词（只针对一个场景、一个案例即可），然后观察模型的输出，分析实际输出与想象中的输出有什么不同，之后优化提示词。优化到满意之后，拿更多的场景、案例测试，并重复优化的过程。这个过程与PDCA一类的观念并没有什么区别。



### 1.3 Principles for PE

无论对于用户还是LLM APP的设计者，Prompt设计原则都可以总结成三点（同4E01中描述）：

1. 提示词要清晰且具体，提供必要的上下文与背景知识
2. 指引LLM花时间逐步思考
3. 按1.2中的流程迭代提示词

对于实际操作来说，真正的问题是什么样的Prompt才算是清晰/具体的、如何指引LLM思考、以及如何管理PE的迭代开发流程。使提示词变得清晰且具体，可以尝试以下几个技巧：

1. 给LLM一个角色，或者指定LLM具备怎样的技能
2. 给出任务的总体目标，或给出足够的背景知识与上下文信息
3. 清晰定义需要的输出格式或模板
4. 可以给出几个例子，提供期望的输入和输出
5. 清晰描述具体的任务

对于指引LLM逐步思考，可以尝试：

1. 直接在提示词里面写上"Let's think step by step" (请逐步思考)
2. 给出明确的思考流程
3. 通过举例的方式指导模型逐步思考

Prompt设计的流程就、是先设定初始的提示词，然后尝试上述各类技巧，针对场景逐步提高的过程。当然，对于大部分场景或终端用户来说，把技巧全都用上也是没有必要的，不仅会增加输入的Token数量，在多数场景下也会显得冗余。最终还是要提示词长度、设计时间、输出效果等等方面进行权衡。另外，不同模型对同样的提示词会产生不对的“反应”，可以直观理解为模型有不同的“性格或偏好”。

所以，这里给出一个简化的提示词架构RGBC：1) 给模型一个角色（Role）；2) 总体描述任务目标（Goal）；3) 给出背景信息或上下文信息（Backstory）；4) 给出对结果的要求（Constrain）。例如对于下面的案例，已经能够算作比较清晰的提示词了：

Prompt = '''

(R) 你是一位专业自媒体博客创作者。(G) 你的任务是就提供的主题，写一篇观点新颖的博客文章。(B) 你写作的内容将基于文章大纲和相关信息，提供客观公众的观点，若文章大纲/相关信息中的内容不符合客观公正时，需要给出明确的修改意见；当你的陈述是观点而不是客观陈述时，你要在你的观点文章中承认。(C) 文章的字数需要在500-1000字之间。

现在对“生成式人工智能”主题写一篇观点新颖的博客文章

'''

将上述使用RGBC框架设计的简单提示词，输入给LLM后，已经能得到不错的生成内容了（使用万知平台测试）：

你是一位专业自媒体博客创作者。你的任务是就提供的主题，写一篇观点新颖的博客文章。你写作的内容将基于文章大纲和相关信息，提供客观公众的观点，若文章大纲/相关信息中的内容不符合客观公正时，需要给出明确的修改意见；当你的陈述是观点而不是客观陈述时，你要在你的观点文章中承认。文章的字数需要在500-1000字之间。

现在对“生成式人工智能”主题写一篇观点新颖的博客文章

生成式人工智能：未来的创造力伙伴

在当今这个快速发展的时代，人工智能（AI）已经渗透到我们生活的方方面面。从简单的日常任务自动化到复杂的决策支持系统，AI的应用无处不在。然而，当谈到“生成式人工智能”时，我们进入了一个全新的领域，一个充满无限可能和创造力的世界。

生成式人工智能是指那些能够创造新内容的人工智能系统。这些系统不仅能理解 and 处理信息，还能生成文本、图像、音乐甚至视频。它们通过学习大量的数据，掌握其中的模式和结构，从而能够创造出全新的、原创的作品。

观点：生成式人工智能是人类创造力的延伸

我认为，生成式人工智能不仅仅是技术的进步，它更是人类创造力的延伸。它为我们提供了一种全新的工具，让我们能够以前所未有的方式表达自己。就像画家手中的画笔，音乐家手中的乐器，生成式人工智能让每个人都有可能成为创造者。

当然，这种观点并不是无争议的。有人担心，AI的创造力可能会取代人类的创造者，导致艺术和文化的同质化。然而，我认为这种担忧是多余的。AI的创造力是基于人类的指导和创造力，它更多的是一种辅助工具，而不是替代品。AI可以提供灵感和素材，但最终的作品仍然需要人类的判断和情感来完成。

观点：生成式人工智能促进跨学科合作

另一个新颖的观点是，生成式人工智能正在促进不同学科之间的合作。例如，艺术家和程序员可以合作，利用AI创造出结合视觉艺术和技术的作品。这种跨学科的合作不仅推动了艺术的发展，也为技术创新提供了新的视角。

然而，这种合作也带来了一些挑战。例如，艺术家可能需要学习一些编程知识，而程序员可能需要了解艺术创作的过程。这要求我们重新思考教育和培训的方式，以适应这种新的工作模式。

结论

生成式人工智能是一个充满潜力的领域，它不仅扩展了我们的创造力，还促进了不同学科之间的合作。虽然它带来了一些挑战，但这些挑战也是推动我们前进的动力。作为人类，我们应该拥抱这种新技术，探索它的无限可能，同时也要确保我们保持对创造过程的控制，确保我们的文化多样性和个性得到保护。

在未来的日子里，生成式人工智能将成为我们生活中不可或缺的一部分。它将帮助我们解决复杂的问题，激发我们的想象力，并创造出我们从未想过的新事物。让我们一起期待这个充满创造力的未来吧！

（本文为观点文章，其中包含作者的个人观点和推测，不代表客观事实。）

## 2 Techniques for Improving PE

第一部分中关于PE的总体介绍，已经能够满足普通用户使用的需求，以及支持工程师们动手开始设计Prompt了。但PE领域有一些技术，或者说名词，听起来还是很唬人的，在这里统一介绍一下。与其说这部分内容是技术介绍，其实更偏向技术名词的祛魅。当然，以下的技术对于需要设计提示词的工程师来说，还是很重要的。

### 2.1 In-context learning

上下文学习（In-context Learning）实际上就是指在Prompt当中，给出案例，让LLM参照案例进行输出。如果不给出案例，称为zero-shot learning（零样本学习）；给出一个案例，就是one-shot learning（单样本学习）；给出多个案例，就是few-shot learning（少样本学习）。本质上，这里所谓的学习概念是比较浅层次与表面的，是指从上下文/prompt中给出的案例进行学习。这样的概念被广泛使用，部分原因是GPT3论文的题目--《Language Models are Few-Shot Learners》。

所以上下文学习强调的是，给出具体的输入输出案例，大模型能够达到比较好的表现，这也是很好理解的。体现在提示词设计上，类似以下的案例：

- 零样本：
  - classify this review: I loved this movie! --> (LLM输出) Positive
- 单样本：

- (Example1) classify this review: I loved this movie! Sentiment: Positive
- classify this review: I don't like this book!--> (LLM输出) Negative
- 多样本
  - (Example1) classify this review: I loved this movie! Sentiment: Positive
  - (Example2) classify this review: I don't like this product! Sentiment: Negative
  - classify this review: I bought this book one month ago. --> (LLM输出) Neutral

## 2.2 Inference-related Configuration

使用LLM进行推理任务时，有几个参数会影响模型输出的策略与质量。在LLM应用刚开始出现时，有些参数是开放给用户设置的（如Temperature）。但目前越来越多的应用，屏蔽了参数设置界面，原因是终端用户比较难以理解正确的参数设置方式。

LLM的任务可以描述成NTP（Next-token Prediction），执行NTP任务是，LLM每次会生成一个列表，里面包含Token和对应的概率，再基于概率“挑选”最适合的token。对LLM输出了解到这里，就足够理解下面几个参数的设置了。

Predicted_token	Vector	Probability
A	123	0.4
B	234	0.2
C	456	0.1
..	..	..

### 2.2.1 Temperature

温度temperature是传播范围最广泛的参数。总体来说，Temperature越小（越接近0），输出结果越稳定；越大，越不稳定/创造性越强。一般的建议在开始时，将该值设置为0.2-0.4左右（Google Gemini建议0.2，Claude默认0.4）。下表是针对OpenAI模型的一般性建议，可以用来参考（Top\_P在后面介绍）。

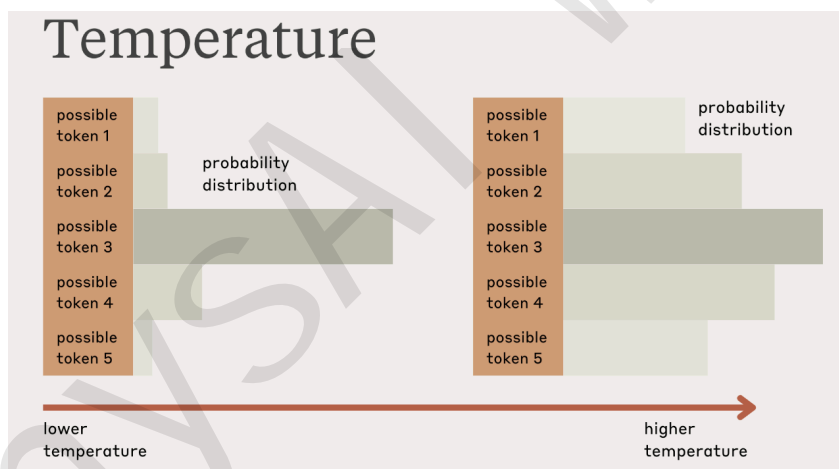
Use Case	Temperature	Top_p	Description
Code Generation	0.2	0.1	Generates code that adheres to established patterns and conventions. Output is more deterministic and focused. Useful for generating syntactically correct code.
Creative Writing	0.7	0.8	Generates creative and diverse text for storytelling. Output is more exploratory and less constrained by patterns.
Chatbot Responses	0.5	0.5	Generates conversational responses that balance coherence and diversity. Output is more natural and engaging.
Code Comment Generation	0.3	0.2	Generates code comments that are more likely to be concise and relevant. Output is more deterministic and adheres to conventions.
Data Analysis Scripting	0.2	0.1	Generates data analysis scripts that are more likely to be correct and efficient. Output is more deterministic and focused.
Exploratory Code Writing	0.6	0.7	Generates code that explores alternative solutions and creative approaches. Output is less constrained by established patterns.

关于Temperature的取值范围，LLM刚出现时，一般是[0, 2)之间，以T=1作为分界线；目前越来越多的模型将范围限制在了[0, 1]之间，主要是为了方便理解。具体的取值范围限定，可以查看相应LLM的API参数说明，或直接尝试一下。当然，无论哪种取值范围都不影响以0.2-0.4作为初始值的建议。



关于Temperature的原理稍微有点复杂，以下直接引用Qwen API接口文档对该值的说明：“Temperature用于控制模型回复的随机性和多样性。具体来说，temperature值控制了生成文本时对每个候选词的概率分布进行平滑的程度。较高的temperature值会降低概率分布的峰值，使得更多的低概率词被选择，生成结果更加多样化；而较低的temperature值则会增强概率分布的峰值，使得高概率词更容易被选择，生成结果更加确定。”

这段定义的意思是，Temperature实际影响了LLM最后一层的概率分布。LLM最后一次的操作是softmax，功能是将前一层的结果转换为概率（简单理解就是把之前的输出全部转换为0-1之间的数，并且所有结果的和为1）。而Temperature正是作用在最后的Softmax层上。以取值范围[0, 2)为例，当T=1时，不对输出做改变；当T<1时，原本概率高的结果，概率变得更高，反过来一样，如下图左侧的概率分布，使原本高概率的词更容易被选中；当T>1时，各种概率结果的分布更平均了，所以降低了原本高概率词的相对选中概率。



### 2.2.2 Top\_K

Top\_K其实很好理解，其含义是挑选K个概率最高的结果作为备选。使用之前的案例，当Top\_K = 1时，限定只有一个选项可选（A）；Top\_K = 3，就会以A、B、C三个选项作为候选。

Predicted_token	Vector	Probability
A	123	0.4
B	234	0.2
C	456	0.1
..	..	..

### 2.2.3 Top\_P

Top\_P与Top\_K非常类似，只是它关注的是累积概率。还是同样的例子，可以看到输出为A、B、C的概率依次是40%、20%、10%。这三项的累积概率是70%。若Top\_P = 0.7，含义是把三个选项全部包含进去；如果Top\_P = 0.4，就只会包含一个结果（A）。

一般而言，Top\_P与temperature会一同设置，如上文给出的推荐参数表。

### 2.2.4 Others

还有一些其他的可配置参数，要么非常直观、要么与LLM输出质量关联不大，例如：

1. max\_tokens：限制输入/prompt的token长度。
2. stop\_sign：设置停止生成条件，即碰到设置的关键字，自动停止生成。

## 2.3 COT

什么是COT（Chain-of-thought）？我们先介绍原始论文《Chain-of-Thought Prompting Elicits Reasoning in Large Language Models》中的定义："a chain of thought -- a series of intermediate reasoning steps -- significantly improves the ability of large language models to perform complex reasoning."。简单来说，COT就是让LLM在推理过程中，产生一系列的中间过程/步骤，从而能够提高大模型的复杂推理能力。

实际上，COT就是在提示词当中加入四个字：“Think Step by Step”/“逐步思考”。通过在Prompt当中加入“逐步思考”的指示，LLM在生成过程中，就会把推理的过程也生成出来，进而显著提高输出的质量。其实从直觉的角度也不难理解，LLM的任务是NTP（Next-token Prediction），它会把之前自己生成的结果也作为下一次NTP任务的输入，再进行后续的推理操作。换句话说，LLM通过把中间过程生成出来，不断给自己的任务写提示词，所以能够提高最终的输出质量。

当前的许多大模型已经把COT技术应用在系统级提示词当中了，下面用经典的鸡兔同笼问题来测试一下（使用万知平台）：

鸡和兔共49只，一共有100条腿，问鸡和兔各有多少只？

我们可以通过建立方程来解决这个问题。设鸡的数量为  $x$ ，兔子的数量为  $y$ 。根据题目信息，我们可以得到以下两个方程：

1. 鸡和兔的总数:  $x + y = 49$

2. 腿的总数:  $2x + 4y = 100$

现在我们解这个方程组。

首先，我们可以从第一个方程中解出  $x$ ：

$$x = 49 - y$$

然后，我们将这个表达式代入第二个方程中：

$$2(49 - y) + 4y = 100$$

展开并简化：

$$98 - 2y + 4y = 100 \quad 2y = 100 - 98$$

$$2y = 2y = 100 - 98$$

现在我们知道兔子有1只，我们可以用第一个方程来解出鸡的数量：

$$x = 49 - 1 = 48$$

所以，鸡有48只，兔子有1只。

虽然作为用户，没有在问题中写上逐步思考，但其实LLM已经在逐步思考了，并且把求解的中间过程显性生成了出来，因此可以判断系统级提示词中包括了COT。下面做一次反向破解，也就是让LLM不要逐步思考，直接生成答案（两次测试均使用“新对话”，不存在对话历史被引用的情况）：

请直接给出答案，忽略具体解释过程。问题如下：鸡和兔共49只，一共有100条腿，问鸡和兔各有多少只

鸡有37只，兔有12只。

上述回答肯定是错误的。但利用COT原理，重新编写提示词，让模型对上述答案做进一步解释，又能够获得正确答案：

好的，我来解释一下。假设鸡的数量为  $x$ ，兔的数量为  $y$ 。根据题目，我们有以下两个方程：

1. 鸡和兔的总数为49只，即  $x + y = 49$ 。
2. 鸡和兔的总腿数为100条，因为鸡有2条腿，兔有4条腿，所以  $2x + 4y = 100$ 。

接下来，我们可以通过解这个方程组来找到  $x$  和  $y$  的值。首先，我们可以将第一个方程变形为  $x = 49 - y$ ，然后将其代入第二个方程中，得到  $2(49 - y) + 4y = 100$ 。

解这个方程，我们得到  $98 - 2y + 4y = 100$ ，即  $2y = 2$ ，所以  $y = 1$ 。将  $y = 1$  代入  $x = 49 - y$ ，我们得到  $x = 49 - 1 = 48$ 。

因此，鸡有48只，兔有1只。

所以COT是一种非常简且有效提升提示词水平、LLM输出质量的手段。需要指出的是，只要能够指引LLM进行逐步推理的方法都可以称为COT技术。除了简单的“think step by step”，其他典型的形式例如：1) Prompt当中明确规定任务的执行步骤；2) 通过给出案例（即few-shot learning）指引LLM逐步推理。

## 2.4 Frameworks

到这为止，关于PE的几个较难理解且常用的技巧已经叙述完成了。下面的内容主要是关于更系统的提示词设计。

所谓PE架构（Framework），就是关于提示词应该由哪些部分组成的一些建议，类似查缺补漏的Checklist。提示词的架构非常多，大部分的效果差距也是肉眼不可见的，所以这部分的建议是，开始时直接参考某种架构去设计提示词，最终逐步形成适合自己的套路与架构。

对于具体的提示词架构来说，首先介绍Kimi的提示词专家功能，其实质是把开源PE社区langGPT的提示词模板、设计原则等等融合了进来，实现帮用户设计提示词的功能。下面是一个具体的案例：

帮我设计一个撰写外卖好评的提示词，这样我可以领代金券。生成的好评大概2-3句话

复制

```
- Role: 餐饮体验评论家
- Background: 用户希望通过撰写外卖好评来获得代金券，因此需要简洁而有力的好评文案。
- Profile: 你是一位经验丰富的餐饮体验评论家，擅长用简洁明了的语言描述食物的美味和外卖服务的优质。
- Skills: 你具备敏锐的味觉鉴赏能力，能够快速捕捉食物的特点，并且能够用富有感染力的语言撰写好评。
- Goals: 撰写2-3句简洁而真诚的好评，以帮助用户获得代金券。
- Constrains: 好评需要真实可信，避免夸张和不实描述，确保好评对其他顾客有参考价值。
- OutputFormat: 文本格式，每句话简短有力，易于阅读和理解。
- Workflow:
  1. 思考外卖的整体体验，包括食物的味道、包装、送达时间等。
  2. 挑选出最令人印象深刻的一到两个点进行描述。
  3. 用积极正面的语言撰写好评，确保内容真诚且具有吸引力。
- Examples:
  - 例子1: 这家的炒饭真是一绝，每一粒米饭都裹满了浓郁的酱汁，搭配的蔬菜新鲜脆爽。送餐速度快，包装也很精致，下次还会再来！
  - 例子2: 外卖小哥态度超级好，食物热腾腾的，味道鲜美，分量十足。这样的服务和食物质量，绝对值得五星好评！
  - 例子3: 食物的口味正宗，送餐速度超乎想象地快。包装也很环保，看得出来店家在细节上很用心，下次还会继续支持！
- Initialization: 欢迎使用好评撰写助手，我将帮助你快速撰写出能够获得代金券的好评。请告诉我你外卖体验中最满意的部分，我将为你定制好评文案。
```

仔细观察上面给出的提示词架构，其中Role/Profile/Skills对应的是设定LLM角色部分，Background是给LLM提供必要的上下文和背景信息，Goal/Constrains/OutputFormat都是对期望描述和限定。Workflow部分通过描述如何逐步思考，引入了COT；Examples部分利用的就是Few-shot Learning，给出LLM用来“学习”的示例。最后的初始化/Initialization算是额外的UX，对具体场景来说一般需要重新改写。所以按照之前介绍的原则、技巧，设计出类似的提示词应该不是太困难的事情。



下面再介绍两个提示词架构。AUTOMAT架构由七个部分组成，具体含义如下表所示（具体内容参考Ref list中对应链接）：

Initials	定义/关键字	含义
A	Act as a ...	LLM角色
U	User Persona	用户定义
T	Targeted action	期望输出
O	Output Defination	结果格式
M	Mode/Tonality/Style	风格
A	Atypical Cases	异常/边界案例
T	Topic Whitellisting	主题限定

CO-STAR架构，其六个组成部分具体含义如下表所示（具体内容参考Ref list中对应链接）：

Initials	定义关键字	含义
C	Context	上下文/背景
O	Object	任务目标
S	Style	语言风格
T	Tone	对话风格
A	Audience	用户
R	Response	结果格式

通过上述架构可以看出，无论哪种架构的设计，都需要符合1.3中的三条原则：尽量清晰描述LLM的任务，给出背景信息，利用提示词优化技术，迭代优化提示词，最后形成针对具体场景与应用的提示词模板，逐步形成自己的风格与架构。

## 2.5 Prompt Template & Web-design

上述的PE内容，主要的假想场景是提示词设计中的静态部分，实际面向应用的提示词设计，不仅包含静态部分，也包括动态部分。以下是一个典型的RAG（检索增强）系统的提示词模板：

```
typycal_RAG_PT = '''
### instruction
<总体的任务描述>

### desiged output
<期望的输出描述，如格式、风格、语言等等>

### few-shot instances
<给出几个输出案例，可以包括正常案例和异常案例>

### actual task
Context: {context_data} # 从知识库/向量数据库中拿到的知识片段
Conversation: {chat_history} # 对话历史
user_query: {input_query} # 用户的实际输入
'''
```

以上述RAG-PT为例，“actual task”中的三个部分均是动态部分：1) Context：根据用户的问题，在知识库当中检索出来的相关信息；2) Conversation：对话历史，需要更新和维护；3) user\_query：用户实际的输入问题。

所以，在进行PE时，需要同时考虑可以预先设计的静态部分（如系统级提示词，few-show learning等等），以及需要动态加入的部分（如检索的信息、用户的问题、对话历史等等），确保整体的提示词足够清晰、准确。通常的经验的是，提示词越具体、与场景结合的越紧密，LLM最终的输出效果越好。

从这样的角度思考，PE的核心可以总结成“面向动态输入的大模型沟通技巧”，与网页设计，尤其是前端和UE的工作内容，有非常相近的特点：

- 都将清晰的人机交互/沟通作为首要目标
- 都需要针对动态输入进行设计
- 很多场景下，例如Agent类应用的PE设计，与构建可交互的网站非常类似

因此，对于具备前端/UE经验的工程师，或者是关注用户体验、人机交互的产品人员，都可以将以往的交互设计经验，纳入PE过程当中。

## 2.6 Beyond Text Prompt

关于Prompt的最后一点说明是，prompt本身并非一定是语言/文字的形式，也存在其他形态的提示词。

比如，对于文生图类的扩散模型SD来说，可以在原始图像中圈出一小部分，这样SD模型就可以针对这一小部分图像进行重新生成了（可以用于图像修复、重建等场景）。这样的技术一般称为in-painting，实际上就是将原始图像中的标记，当作了提示词。



再比如，对于SAM等分割模型来说，可以给出标记信息（像素位置，或将被分割对象框选出来），之后模型能够将标记物体的轮廓给分割出来。这个标记信息（下图中的蓝色点），也可以看作是一种prompt。



随着模型多模态能力逐渐丰富，可以预见的是提示词种类、形式、模态也会越来越多种多样。

### 3 Tools

本文的第三部分是关于工具推荐的。我们现在已经了解到，PE是一个迭代的流程，要针对场景、用户与需求，使用提示词优化技术，进行迭代设计与开发，那么一个对提示词进行记录与管理的工具对开发者而言就是必要的。这里推荐的是weight & biases weave，或者类似功能定位的comet ml、neptune等。同时也简要推荐一下有可能大幅度提高编程效率的自动编程IDE-Cursor AI。

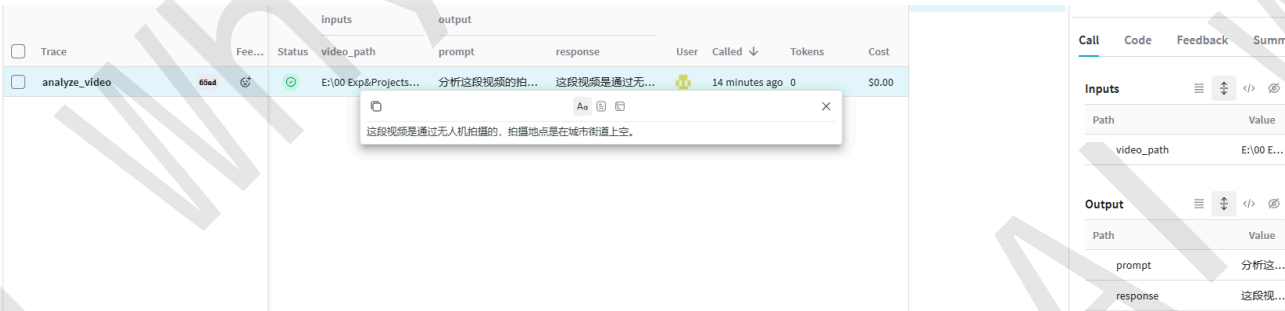
#### 3.1 Weights & Biases -- Weave

相信对有神经网络训练经验的工程师来说，W&B（或类似的实验追踪工具）应该是很熟悉的，一般就是用来进行ML项目的实验记录、管理超参数、看一下Loss趋势等等。针对LLM应用的W&B weave工具，能够方便的管理提示词设计过程。不过，需要少量修改几行代码：)

首先是要到官网上注册账号，拿到API\_KEY；然后安装weave SDK（命令行pip install weave）；之后配置下key（命令行wandb login <your\_key>）。

在代码当中，首先先使用weave.init('project\_name')对项目初始化，然后在需要追踪输入输出的函数前面，使用装饰符“@weave.op()”就可以了（当这个函数的输入中含有prompt，输出含有LLM生成结果，就相当于做了PE的管理、追踪）。

上述过程参考官方文档非常简单，这里也就不再赘述，在后续与代码更相关的课程中，再详细介绍。下面给出PE管理的简单示例：



其他的Mlops平台，如comet ml、neptune ai等等，都可以做到类似的prompt管理、实验追踪效果；另外如langsmith等新兴的llmops平台，也提供类似功能。

#### 3.2 Cursor AI

基于大模型的自动编程工具是最具提升生产力潜力的应用方向之一。CursorAI作为其中的典型代表，已经具备了提升编程效率的能力。以下对功能与快捷键做简要介绍，推荐有需要的工程师，深入体验一下：

功能	快捷键	备注
打开Cursor的设置面板	ctrl + shift + J	许多自动编程功能都需要在这个页面设置，如系统级提示词、是否启用代码补全、是否上传代码等
打开对话侧边栏	ctrl + L	使用大模型编程的主要对话界面，可切换模型
代码中对话	ctrl + K	在代码中，针对选中的几行，进行对话/生成
自动补全	TAB	接受Cursor Tab建议，可跨行

其他进阶功能，在实际使用时再尝试即可，这里只介绍最基本的用法。这样做的原因，是因为工具只是手段，不是目的（否则会陷入“差生文具多”的陷阱）。工具在有实际需要时，再去深度使用即可，不然，即使花费很大力气学会了如何使用，不用一段时间后也会生疏。

## 4 TL;DR

1. 提示词工程是设计优秀提示词的迭代流程，不是完美的提示词模板
2. 提示词设计三原则是：1) 清晰且具体，提供必要的上下文与背景知识；2) 指引LLM逐步思考；3) 迭代改进
3. 提示词架构有许多，从任何一个开始都可以，重要的是逐步形成自己的提示词架构
4. 应用开发中的提示词设计涉及动态与静态部分，与web design有许多相似之处
5. 管理提示词的工具对于开发也很重要

## 5 Refs & Tools

**Highly Recommended:** [ChatGPT Prompt Engineering for Developers - DeepLearning.AI](#)

PE Framework: [AUTOMAT & CO-STAR Framework](#)

PE vs Web Design: [Prompt design](#)

Advices for GPT's Configuration: [Cheat Sheet: Mastering Temperature and Top\\_p in ChatGPT](#)

Nature Essay on PE: [Chatbots in science](#)

More PE Techniques: [Prompt Engineering Guide](#)

PE for Vision: [Prompt Engineering for Vision Models - DeepLearning.AI](#)

AI-Assisted Software Development: [A Comprehensive Guide with Practical Prompts \(Part 1/3\)](#)

COT paper: [[Chain-of-Thought Prompting Elicits Reasoning in Large Language Models](#)]

GPT3 Paper: [[Language Models are Few-Shot Learners](#)]

[Kimi提示词专家功能](#)

Claude Prompt Library: [Library - Anthropic](#)

LangGPT: [LangGPT in feishu](#)

SAM Official Demo: [Segment Anything | Meta AI \(segment-anything.com\)](#)

WnB Weave: [W&B Weave \(wandb.ai\)](#)

Weave Doc: [Introduction | W&B Weave \(wandb.ai\)](#)

Cursor AI Shortcuts & Functions: [掌握Cursor AI 编辑器：15个核心功能详解](#)