面向开发者的 LLM 入门课程

简介

LLM 正在逐步改变人们的生活，而对于开发者，如何基于 LLM 提供的 API 快速、便捷地开发一些具备更强能力、集成LLM 的应用，来便捷地实现一些更新颖、更实用的能力，是一个急需学习的重要能力。

由吴恩达老师与 OpenAI 合作推出的大模型系列教程，从大模型时代开发者的基础技能出发，深入浅出地介绍了如何基于大模型 API、LangChain 架构快速开发结合大模型强大能力的应用。其中，《Prompt Engineering for Developers》教程面向入门 LLM 的开发者，深入浅出地介绍了对于开发者，如何构造 Prompt 并基于 OpenAI 提供的 API 实现包括总结、推断、转换等多种常用功能，是入门 LLM 开发的经典教程；《Building Systems with the ChatGPT API》教程面向想要基于 LLM 开发应用程序的开发者，简洁有效而又系统全面地介绍了如何基于 ChatGPT API 打造完整的对话系统；《LangChain for LLM Application Development》教程结合经典大模型开源框架 LangChain，介绍了如何基于 LangChain 框架开发具备实用功能、能力全面的应用程序，《LangChain Chat With Your Data》教程则在此基础上进一步介绍了如何使用 LangChain 架构结合个人私有数据开发个性化大模型应用。

上述教程非常适用于开发者学习以开启基于 LLM 实际搭建应用程序之路。因此，我们将该系列课程翻译为中文，并复现其范例代码，也为其中一个视频增加了中文字幕，支持国内中文学习者直接使用，以帮助中文学习者更好地学习 LLM 开发；我们也同时实现了效果大致相当的中文 Prompt，支持学习者感受中文语境下 LLM 的学习使用，对比掌握多语言语境下的 Prompt 设计与 LLM 开发。未来，我们也将加入更多 Prompt 高级技巧，以丰富本课程内容，帮助开发者掌握更多、更巧妙的 Prompt 技能。

受众

适用于所有具备基础 Python 能力，想要入门 LLM 的开发者。

亮点

《ChatGPT Prompt Engineering for Developers》、《Building Systems with the ChatGPT API》、《LangChain for LLM Application Development》、《LangChain Chat with Your Data》等教程作为由吴恩达老师与 OpenAI 联合推出的官方教程，在可预见的未来会成为 LLM 的重要入门教程，但是目前还只支持英文版且国内访问受限，打造中文版且国内流畅访问的教程具有重要意义；同时，GPT 对中文、英文具有不同的理解能力，本教程在多次对比、实验之后确定了效果大致相当的中文 Prompt，支持学习者研究如何提升 ChatGPT 在中文语境下的理解与生成能力。

内容大纲

一、面向开发者的提示工程

注：吴恩达《ChatGPT Prompt Engineering for Developers》课程中文版

目录：

7. 文本扩展 Expanding @邹雨衡
8. 聊天机器人 Chatbot @长琴
9. 总结 @长琴

附1 使用 ChatGLM 进行学习 @宋志学

二、搭建基于 ChatGPT 的问答系统

注：吴恩达《Building Systems with the ChatGPT API》课程中文版

目录：

1. 简介 Introduction @Sarai
2. 模型，范式和 token Language Models, the Chat Format and Tokens @仲泰
3. 检查输入-分类 Classification @诸世纪
4. 检查输入-监督 Moderation @诸世纪
5. 思维链推理 Chain of Thought Reasoning @万礼行
6. 提示链 Chaining Prompts @万礼行
7. 检查输入 Check Outputs @仲泰
8. 评估（端到端系统）Evaluation @邹雨衡
9. 评估（简单问答）Evaluation-part1 @陈志宏、邹雨衡
10. 评估（复杂问答）Evaluation-part2 @邹雨衡
11. 总结 Conclusion @Sarai

三、使用 LangChain 开发应用程序

注：吴恩达《LangChain for LLM Application Development》课程中文版

目录：

1. 简介 Introduction @Sarai
2. 模型，提示和解析器 Models, Prompts and Output Parsers @Joye
3. 存储 Memory @徐虎
4. 模型链 Chains @徐虎
5. 基于文档的问答 Question and Answer @苟晓攀
6. 评估 Evaluation @苟晓攀
7. 代理 Agent @Joye
8. 总结 Conclusion @Sarai

四、使用 LangChain 访问个人数据

注：吴恩达《LangChain Chat with Your Data》课程中文版

目录：

1. 简介 Introduction @Joye
2. 加载文档 Document Loading @Joye
3. 文档切割 Document Splitting @苟晓攀

4. 向量数据库与词向量 Vectorstores and Embeddings @刘伟鸿、仲泰
5. 检索 Retrieval @刘伟鸿
6. 问答 Question Answering @邹雨衡
7. 聊天 Chat @高立业
8. 总结 Summary @高立业

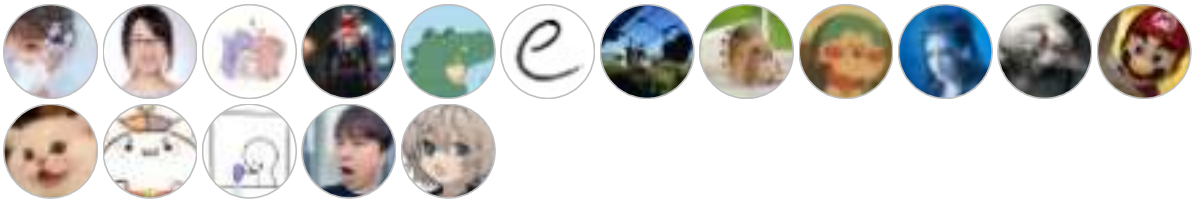
致谢

核心贡献者

- 邹雨衡-项目负责人（Datawhale成员-对外经济贸易大学研究生）
- 长琴-项目发起人（内容创作者-Datawhale成员-AI算法工程师）
- 玉琳-项目发起人（内容创作者-Datawhale成员）
- 徐虎-教程编撰成员（内容创作者）
- Joye-教程编撰成员（内容创作者-数据科学家）
- 刘伟鸿-教程编撰成员（内容创作者-江南大学非全研究生）
- 高立业（内容创作者-DataWhale成员-算法工程师）
- Zhang Yixin（内容创作者-IT爱好者）
- 万礼行（内容创作者-视频翻译者）
- 仲泰（内容创作者-Datawhale成员）
- 魂兮（内容创作者-前端工程师）
- 诸世纪（内容创作者-算法工程师）
- 宋志学（内容创作者-Datawhale成员）
- Sarai（内容创作者-AI应用爱好者）

其他

1. 特别感谢 @Sm1les、@LSGOMYP 对本项目的帮助与支持；
2. 感谢 GithubDaily 提供的双语字幕；
3. 如果有任何想法可以联系我们 DataWhale 也欢迎大家多多提出 issue；
4. 特别感谢以下为教程做出贡献的同学！



Made with contrib.rocks.

关注我们

扫描下方二维码关注公众号：Datawhale

Datawhale 是一个专注于数据科学与 AI 领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。微信搜索公众号Datawhale可以加入我们。

LICENSE

前言

亲爱的读者朋友：

您好！欢迎阅读这本《面向开发者的 LLM 入门教程》。

最近，以GPT-4为代表的大规模预训练语言模型备受关注。这些模型拥有数十亿到千亿参数，通过学习大规模文本语料库，获得了十分强大的语言理解和生成能力。与此同时，OpenAI等公司推出的API服务，使得访问这些模型变得前所未有的便捷。

那么如何运用这些强大的预训练模型开发实用的应用呢?本书汇聚了斯坦福大学的吴恩达老师与OpenAI合作打造的大语言模型（LLM）系列经典课程，从模型原理到应用落地，全方位介绍大模型的开发技能。

本书首先介绍 Prompt Engineering 的方法，提示是连接用户与模型的桥梁，优化提示对模型效果至关重要。通过案例，读者可以学习文本总结、推理、转换等基础NLP任务的Prompt设计技巧。

然后，本书指导读者基于 ChatGPT 提供的 API 开发一个完整的、全面的智能问答系统，包括使用大语言模型的基本规范，通过分类与监督评估输入，通过思维链推理及链式提示处理输入，检查并评估系统输出等，介绍了基于大模型开发的新范式，值得每一个有志于使用大模型开发应用程序的开发者学习。

通过对LLM或大型语言模型给出提示(prompt)，现在可以比以往更快地开发AI应用程序，但是一个应用程序可能需要进行多轮提示以及解析输出。在此过程有很多胶水代码需要编写，基于此需求，哈里森·蔡斯 (Harrison Chase) 创建了一个用于构建大模型应用程序的开源框架 LangChain ，使开发过程变得更加丝滑。

在Langchain部分，读者将会学习如何结合框架 LangChain 使用 ChatGPT API 来搭建基于 LLM 的应用程序，帮助开发者学习使用 LangChain 的一些技巧，包括：模型、提示和解析器，应用程序所需要用到的存储，搭建模型链，基于文档的问答系统，评估与代理等。

当前主流的大规模预训练语言模型，如ChatGPT（训练知识截止到2021年9月）等，主要依赖的是通用的训练数据集，而未能有效利用用户自身的数据。这成为模型回答问题的一个重要局限。具体来说，这类模型无法使用用户的私有数据，比如个人信息、公司内部数据等，来生成个性化的回复。它们也无法获得用户最新的实时数据，而只能停留在预训练数据集的时间点。这导致模型对许多需要结合用户情况的问题无法给出满意答案。如果能赋予语言模型直接访问用户自有数据的能力，并让模型能够实时吸收用户最新产生的数据，则其回答质量将能大幅提升。

最后，本书重点探讨了如何使用 LangChain 来整合自己的私有数据，包括：加载并切割本地文档；向量数据库与词向量；检索回答；基于私有数据的问答与聊天等。

可以说，本书涵盖大模型应用开发的方方面面，相信通过本书的学习，即便您没有丰富编程经验，也可以顺利入门大模型，开发出有实用价值的AI产品。让我们共同推进这一具有革命性的新兴技术领域吧！

如果你在学习中遇到任何问题，也欢迎随时与我们交流。

祝您的大模型之旅愉快而顺利！

环境配置

本章介绍了阅读本教程所需环境的配置方法，包括 Python、Jupyter Notebook、OpenAI API key、相关库来运行本书所需的代码。

请注意，以下环境配置有的只需一次配置（如 Python、Jupyter Notebook等），有的需要在每次复现代码时配置（如 OpenAI API key 的配置等）。

一、安装Anaconda

由于官网安装较慢，我们可以通过清华源镜像来安装Anaconda

Anaconda3-2023.07-1-Windows-x86_64.exe	893.8 MiB	2023-07-14 04:38
Anaconda3-2023.07-1-MacOSX-x86_64.sh	595.4 MiB	2023-07-14 04:38
Anaconda3-2023.07-1-MacOSX-x86_64.pkg	593.8 MiB	2023-07-14 04:38
Anaconda3-2023.07-1-MacOSX-arm64.sh	629.9 MiB	2023-07-14 04:37
Anaconda3-2023.07-1-Linux-x86_64.sh	1010.4 MiB	2023-07-14 04:37
Anaconda3-2023.07-1-Linux-ppc64le.sh	468.7 MiB	2023-07-14 04:37
Anaconda3-2023.07-1-MacOSX-arm64.pkg	628.1 MiB	2023-07-14 04:37
Anaconda3-2023.07-1-Linux-s390x.sh	336.1 MiB	2023-07-14 04:37
Anaconda3-2023.07-1-Linux-aarch64.sh	711.9 MiB	2023-07-14 04:37

选择对应的版本下载安装即可。

如果已安装Anaconda，则可以跳过以下步骤。

- 如果我们使用Window系统，可以下载`Anaconda3-2023.07-1-Windows-x86_64.exe`安装包直接安装即可。
- 如果我们使用MacOS系统
    1. Intel芯片：可以下载`Anaconda3-2023.07-1-MacOSX-x86_64.sh`
    2. Apple芯片：可以下载`Anaconda3-2023.07-1-MacOSX-arm64.sh`
       并执行以下操作：

```
# 以Intel处理器为例，文件名可能会更改
sh Anaconda3-2023.07-1-MacOSX-x86_64.sh -b
```

接下来，初始化终端Shell，以便我们可以直接运行conda。

```
~/anaconda3/bin/conda init
```

现在关闭并重新打开当前的shell，我们会发现在命令行的前面多了一个`(base)`,这是anaconda的一个基础`python`环境。下面我们使用以下命令来创建一个新的环境：

```
# 创建一个名为chatgpt且python版本为3.9的环境
conda create --name chatgpt python=3.9 -y
```

创建完成后，现在我们来激活 chatgpt 环境：

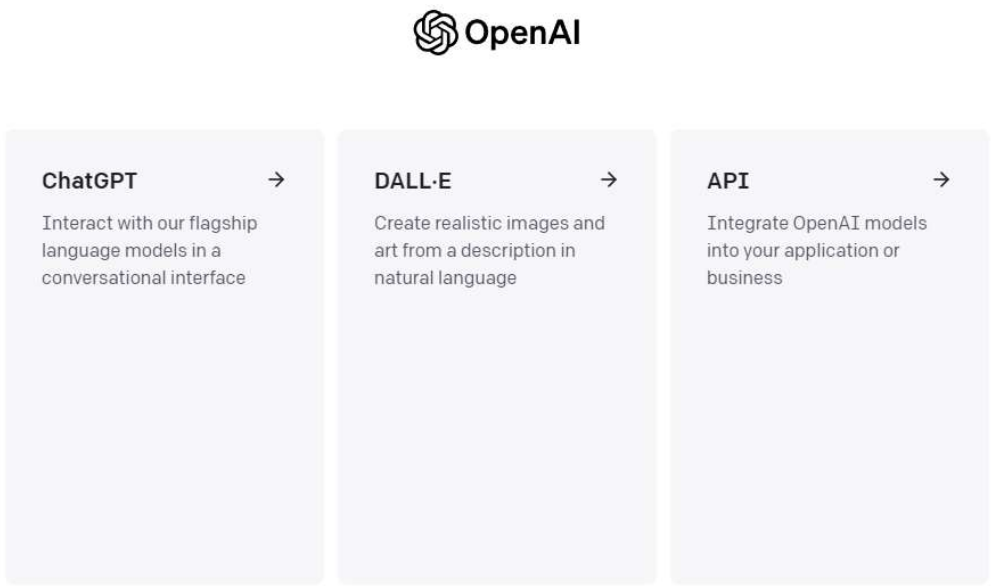
```
conda activate chatgpt
```

二、安装本书需要用到的python库

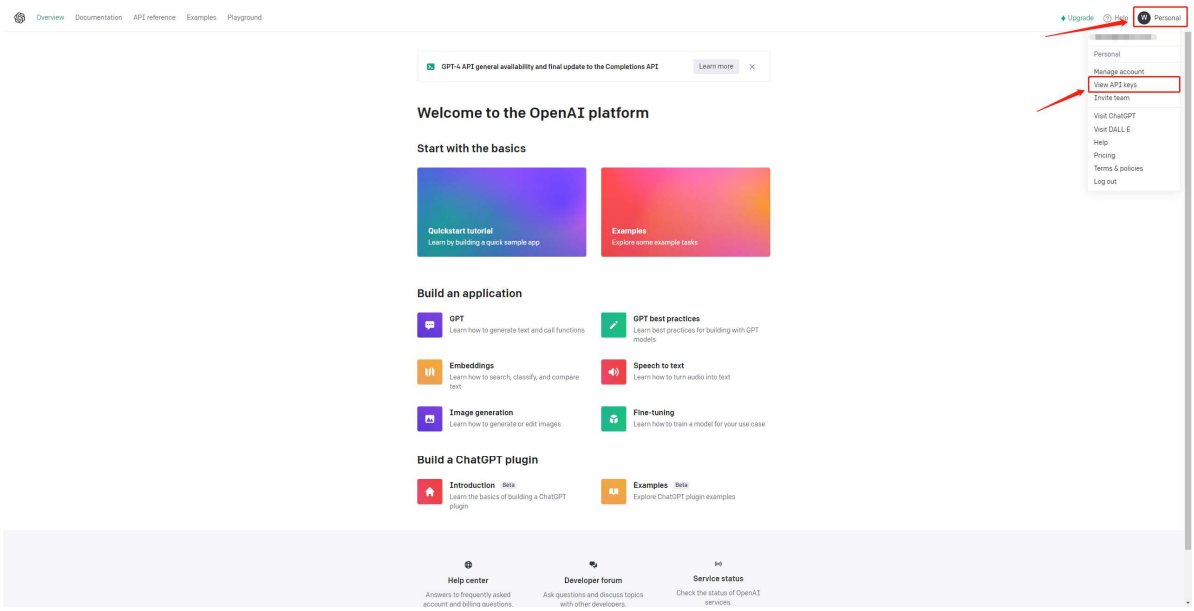
```
!pip install -q python-dotenv
!pip install -q openai
!pip install -q langchain
```

三、获取并配置OpenAI API key

在获取OpenAI API key之前我们需要openai官网中注册一个账号。这里假设我们已经有了openai账号，先在openai官网登录，登录后如下图所示：



我们选择`API`，然后点击右上角的头像，选择`View API keys`，如下图所示：



点击`Create new secret key`按钮创建OpenAI API key，我们将创建好的OpenAI API key复制以此形式`OPENAI_API_KEY="sk-..."`保存到`.env`文件中，并将`.env`文件保存在项目根目录下。# TODO:放到哪个固定位置待确认

下面是读取`.env`文件的代码

```
import os
import openai
from dotenv import load_dotenv, find_dotenv

# 读取本地/项目的环境变量。

# find_dotenv()寻找并定位.env文件的路径
# load_dotenv()读取该.env文件，并将其中的环境变量加载到当前的运行环境中
# 如果你设置的是全局的环境变量，这行代码则没有任何作用。
_ = load_dotenv(find_dotenv())

# 获取环境变量 OPENAI_API_KEY
openai.api_key = os.environ['OPENAI_API_KEY']
```

将读取`.env`文件的代码封装成函数供每一章节直接调用获取在OpenAI API key。

```
import os
from dotenv import load_dotenv, find_dotenv
def get_openai_key():
    _ = load_dotenv(find_dotenv())
    return os.environ['OPENAI_API_KEY']

openai.api_key = get_openai_key()
```

第一部分 面向开发者的提示工程

Prompt，提示，最初是 NLP 研究者为下游任务设计出来的一种任务专属的输入形式或模板，在 ChatGPT 引发大语言模型新时代之后，Prompt 即成为与大模型交互输入的代称。即我们一般**将给大模型的输入称为 Prompt，将大模型返回的输出称为 Completion。**

随着 ChatGPT 等 LLM（大语言模型）的出现，自然语言处理的范式正在由 Pretrain-Finetune（预训练-微调）向 Prompt Engineering（提示工程）演变。对于具有较强自然语言理解、生成能力，能够实现多样化任务处理的 LLM 来说，一个合理的 Prompt 设计极大地决定了其能力的上限与下限。**Prompt Engineering，即是针对特定任务构造能充分发挥大模型能力的 Prompt 的技巧。**要充分、高效地使用 LLM，Prompt Engineering 是必不可少的技能。

LLM 正在逐步改变人们的生活，而对于开发者，如何基于 LLM 提供的 API 快速、便捷地开发一些具备更强能力、集成LLM 的应用，来便捷地实现一些更新颖、更实用的能力，是一个急需学习的重要能力。要高效地基于 API 开发集成 LLM 的应用，首要便是学会如何合理、高效地使用 LLM，即如何构建 Prompt Engineering。第一部分 面向开发者的提示工程，源于由吴恩达老师与 OpenAI 合作推出的 《ChatGPT Prompt Engineering for Developers》教程，其面向入门 LLM 的开发者，深入浅出地介绍了对于开发者，**如何构造 Prompt 并基于 OpenAI 提供的 API 实现包括总结、推断、转换等多种常用功能**，是入门 LLM 开发的第一步。对于想要入门 LLM 的开发者，你需要充分掌握本部分的 Prompt Engineering 技巧，并能基于上述技巧实现个性化定制功能。

本部分的主要内容包括：书写 Prompt 的原则与技巧；文本总结（如总结用户评论）；文本推断（如情感分类、主题提取）；文本转换（如翻译、自动纠错）；扩展（如书写邮件）等。

目录：

第一章 简介

欢迎来到**面向开发者的提示工程**部分，本部分内容基于**吴恩达老师的《Prompt Engineering for Developer》课程**进行编写。《Prompt Engineering for Developer》课程是由**吴恩达老师**与 OpenAI 技术团队成员 **Isa Fulford** 老师合作授课，Isa 老师曾开发过受欢迎的 ChatGPT 检索插件，并且在教授 LLM （Large Language Model， 大语言模型）技术在产品中的应用方面做出了很大贡献。她还参与编写了教授人们使用 Prompt 的 OpenAI cookbook。我们希望通过本模块的学习，与大家分享使用提示词开发 LLM 应用的最佳实践和技巧。

网络上有许多关于提示词（Prompt， 本教程中将保留该术语）设计的材料，例如《30 prompts everyone has to know》之类的文章，这些文章主要集中在 **ChatGPT 的 Web 界面上**，许多人在使用它执行特定的、通常是一次性的任务。但我们认为，对于开发人员，**大语言模型（LLM） 的更强大功能是能通过 API 接口调用，从而快速构建软件应用程序。**实际上，我们了解到 DeepLearning.AI 的姊妹公司 AI Fund 的团队一直在与许多初创公司合作，将这些技术应用于诸多应用程序上。很兴奋能看到 LLM API 能够让开发人员非常快速地构建应用程序。

在本模块，我们将与读者分享提升大语言模型应用效果的各种技巧和最佳实践。书中内容涵盖广泛，包括软件开发提示词设计、文本总结、推理、转换、扩展以及构建聊天机器人等语言模型典型应用场景。我们衷心希望该课程能激发读者的想象力，开发出更出色的语言模型应用。

随着 LLM 的发展，其大致可以分为两种类型，后续称为**基础 LLM** 和**指令微调（Instruction Tuned）LLM。基础LLM**是基于文本训练数据，训练出预测下一个单词能力的模型。其通常通过在互联网和其他来源的大量数据上训练，来确定紧接着出现的最可能的词。例如，如果你以“从前，有一只独角兽”作为 Prompt ，基础 LLM 可能会继续预测“她与独角兽朋友共同生活在一片神奇森林中”。但是，如果你以“法国的首都是什么”为 Prompt ，则基础 LLM 可能会根据互联网上的文章，将回答预测为“法国最大的城市是什么？法国的人口是多少？”，因为互联网上的文章很可能是有关法国国家的问答题目列表。

与基础语言模型不同，**指令微调 LLM** 通过专门的训练，可以更好地理解并遵循指令。举个例子，当询问“法国的首都是什么？”时，这类模型很可能直接回答“法国的首都是巴黎”。指令微调 LLM 的训练通常基于预训练语言模型，先在大规模文本数据上进行**预训练**，掌握语言的基本规律。在此基础上进行进一步的训练与**微调（finetune）**，输入是指令，输出是对这些指令的正确回复。有时还会采用**RLHF（reinforcement learning from human feedback，人类反馈强化学习）**技术，根据人类对模型输出的反馈进一步增强模型遵循指令的能力。通过这种受控的训练过程。指令微调 LLM 可以生成对指令高度敏感、更安全可靠的输出，较少无关和损害性内容。因此。许多实际应用已经转向使用这类大语言模型。

因此，本课程将重点介绍针对指令微调 LLM 的最佳实践，我们也建议您将其用于大多数使用场景。当您使用指令微调 LLM 时，您可以类比为向另一个人提供指令（假设他很聪明但不知道您任务的具体细节）。因此，当 LLM 无法正常工作时，有时是因为指令不够清晰。例如，如果您想问“请为我写一些关于阿兰·图灵(Alan Turing)的东西”，在此基础上清楚表明您希望文本专注于他的科学工作、个人生活、历史角色或其他方面可能会更有帮助。另外您还可以指定回答的语调， 来更加满足您的需求，可选项包括*专业记者写作*，或者*向朋友写的随笔*等。

如果你将 LLM 视为一名新毕业的大学生，要求他完成这个任务，你甚至可以提前指定他们应该阅读哪些文本片段来写关于阿兰·图灵的文本，这样能够帮助这位新毕业的大学生更好地完成这项任务。本书的下一章将详细阐释提示词设计的两个关键原则：**清晰明确**和**给予充足思考时间。**

第二章 提示原则

如何去使用 Prompt，以充分发挥 LLM 的性能？首先我们需要知道设计 Prompt 的原则，它们是每一个开发者设计 Prompt 所必须知道的基础概念。本章讨论了设计高效 Prompt 的两个关键原则：**编写清晰、具体的指令**和**给予模型充足思考时间**。掌握这两点，对创建可靠的语言模型交互尤为重要。

首先，Prompt 需要清晰明确地表达需求，提供充足上下文，使语言模型准确理解我们的意图，就像向一个外星人详细解释人类世界一样。过于简略的 Prompt 往往使模型难以把握所要完成的具体任务。

其次，让语言模型有充足时间推理也极为关键。就像人类解题一样，匆忙得出的结论多有失误。因此 Prompt 应加入逐步推理的要求，给模型留出充分思考时间，这样生成的结果才更准确可靠。

如果 Prompt 在这两点上都作了优化，语言模型就能够尽可能发挥潜力，完成复杂的推理和生成任务。掌握这些 Prompt 设计原则，是开发者取得语言模型应用成功的重要一步。

一、原则一 编写清晰、具体的指令

亲爱的读者，在与语言模型交互时，您需要牢记一点:以**清晰、具体**的方式表达您的需求。假设您面前坐着一位来自外星球的新朋友，其对人类语言和常识都一无所知。在这种情况下，您需要把想表达的意图讲得非常明确，不要有任何歧义。同样的，在提供 Prompt 的时候，也要以足够详细和容易理解的方式，把您的需求与上下文说清楚。

并不是说 Prompt 就必须非常短小简洁。事实上，在许多情况下，更长、更复杂的 Prompt 反而会让语言模型更容易抓住关键点，给出符合预期的回复。原因在于，复杂的 Prompt 提供了更丰富的上下文和细节，让模型可以更准确地把握所需的操作和响应方式。

所以，记住用清晰、详尽的语言表达 Prompt，就像在给外星人讲解人类世界一样，*“Adding more context helps the model understand you better.”*。

从该原则出发，我们提供几个设计 Prompt 的技巧。

1.1 使用分隔符清晰地表示输入的不同部分

在编写 Prompt 时，我们可以使用各种标点符号作为“分隔符”，将不同的文本部分区分开来。

分隔符就像是 Prompt 中的墙，将不同的指令、上下文、输入隔开，避免意外的混淆。你可以选择用 `` ```，"""，< >，<tag> </tag>，: `` 等做分隔符，只要能明确起到隔断作用即可。

使用分隔符尤其重要的是可以防止 **提示词注入（Prompt Rejection）**。什么是提示词注入？就是用户输入的文本可能包含与你的预设 Prompt 相冲突的内容，如果不加分隔，这些输入就可能“注入”并操纵语言模型，导致模型产生毫无关联的乱七八糟的输出。

在以下的例子中，我们给出一段话并要求 GPT 进行总结，在该示例中我们使用 ``` 来作为分隔符。

```
from tool import get_completion

text = f"""
您应该提供尽可能清晰、具体的指示，以表达您希望模型执行的任务。\
这将引导模型朝向所需的输出，并降低收到无关或不正确响应的可能性。\
不要将写清晰的提示词与写简短的提示词混淆。\
在许多情况下，更长的提示词可以为模型提供更多的清晰度和上下文信息，从而导致更详细和相关的输出。
"""
# 需要总结的文本内容
prompt = f"""
把用三个反引号括起来的文本总结成一句话。
```

```
```{text}```
"""
# 指令内容，使用 ``` 来分隔指令和待总结的内容
response = get_completion(prompt)
print(response)
```

为了获得所需的输出，您应该提供清晰、具体的指示，避免与简短的提示词混淆，并使用更长的提示词来提供更多的清晰度和上下文信息。

1.2 寻求结构化的输出

有时候我们需要语言模型给我们一些**结构化的输出**，而不仅仅是连续的文本。

什么是结构化输出呢？就是按照某种格式组织的内容，例如JSON、HTML等。这种输出非常适合在代码中进一步解析和处理。例如，您可以在 Python 中将其读入字典或列表中。

在以下示例中，我们要求 GPT 生成三本书的标题、作者和类别，并要求 GPT 以 JSON 的格式返回给我们，为便于解析，我们指定了 Json 的键。

```
prompt = f"""
请生成包括书名、作者和类别的三本虚构的、非真实存在的中文书籍清单，\
并以 JSON 格式提供，其中包含以下键:book_id、title、author、genre。
"""
response = get_completion(prompt)
print(response)
```

```
{
  "books": [
    {
      "book_id": 1,
      "title": "迷失的时光",
      "author": "张三",
      "genre": "科幻"
    },
    {
      "book_id": 2,
      "title": "幻境之门",
      "author": "李四",
      "genre": "奇幻"
    },
    {
      "book_id": 3,
      "title": "虚拟现实",
      "author": "王五",
      "genre": "科幻"
    }
  ]
}
```
```

1.3 要求模型检查是否满足条件

如果任务包含不一定能满足的假设（条件），我们可以告诉模型先检查这些假设，如果不满足，则会指出并停止执行后续的完整流程。您还可以考虑可能出现的边缘情况及模型的应对，以避免意外的结果或错误发生。

在如下示例中，我们将分别给模型两段文本，分别是制作茶的步骤以及一段没有明确步骤的文本。我们将要求模型判断其是否包含一系列指令，如果包含则按照给定格式重新编写指令，不包含则回答“未提供步骤”。

```
# 满足条件的输入（text中提供了步骤）
text_1 = f"""
泡一杯茶很容易。首先，需要把水烧开。\
在等待期间，拿一个杯子并把茶包放进去。\
一旦水足够热，就把它倒在茶包上。\
等待一会儿，让茶叶浸泡。几分钟后，取出茶包。\
如果您愿意，可以加一些糖或牛奶调味。\
就这样，您可以享受一杯美味的茶了。
"""
prompt = f"""
您将获得由三个引号括起来的文本。\
如果它包含一系列的指令，则需要按照以下格式重新编写这些指令：

第一步 - ...
第二步 - …
…
第N步 - …

如果文本中不包含一系列的指令，则直接写“未提供步骤”。"
\"\"\"{text_1}\"\"\"
"""
response = get_completion(prompt)
print("Text 1 的总结:")
print(response)
```

```
Text 1 的总结:
第一步 - 把水烧开。
第二步 - 拿一个杯子并把茶包放进去。
第三步 - 把烧开的水倒在茶包上。
第四步 - 等待几分钟，让茶叶浸泡。
第五步 - 取出茶包。
第六步 - 如果需要，加入糖或牛奶调味。
第七步 - 就这样，您可以享受一杯美味的茶了。
```

上述示例中，模型可以很好地识别一系列的指令并进行输出。在接下来一个示例中，我们将提供给模型没有预期指令的输入，模型将判断未提供步骤。

```
# 不满足条件的输入（text中未提供预期指令）
text_2 = f"""
今天阳光明媚，鸟儿在歌唱。\
这是一个去公园散步的美好日子。\
鲜花盛开，树枝在微风中轻轻摇曳。\
人们外出享受着这美好的天气，有些人在野餐，有些人在玩游戏或者在草地上放松。\
这是一个完美的日子，可以在户外度过并欣赏大自然的美景。
```

```
"""
prompt = f"""
您将获得由三个引号括起来的文本。\
如果它包含一系列的指令，则需要按照以下格式重新编写这些指令：

第一步 - ...
第二步 - …
…
第N步 - …

如果文本中不包含一系列的指令，则直接写“未提供步骤”。"
\"\"\"{text_2}\"\"\"
"""
response = get_completion(prompt)
print("Text 2 的总结:")
print(response)
```

```
Text 2 的总结:
未提供步骤。
```

1.4 提供少量示例

"Few-shot" prompting，即在要求模型执行实际任务之前，给模型一两个已完成的样例，让模型了解我们的要求和期望的输出样式。

例如，在以下的样例中，我们先给了一个祖孙对话样例，然后要求模型用同样的隐喻风格回答关于“韧性”的问题。这就是一个少样本样例，它能帮助模型快速抓住我们要的语调和风格。

利用少样本样例，我们可以轻松“预热”语言模型，让它为新的任务做好准备。这是一个让模型快速上手新任务的有效策略。

```
prompt = f"""
您的任务是以一致的风格回答问题。

<孩子>: 请教我何为耐心。

<祖父母>: 挖出最深峡谷的河流源于一处不起眼的泉眼；最宏伟的交响乐从单一的音符开始；最复杂的挂毯以一根孤独的线开始编织。

<孩子>: 请教我何为韧性。
"""
response = get_completion(prompt)
print(response)
```

```
<祖父母>: 韧性是一种坚持不懈的品质，就像一棵顽强的树在风雨中屹立不倒。它是面对困难和挑战时不屈不挠的精神，能够适应变化和克服逆境。韧性是一种内在的力量，让我们能够坚持追求目标，即使面临困难和挫折也能坚持不懈地努力。
```

二、原则二 给模型时间去思考

在设计 Prompt 时，给予语言模型充足的推理时间非常重要。语言模型与人类一样，需要时间来思考并解决复杂问题。如果让语言模型匆忙给出结论，其结果很可能不准确。例如，若要语言模型推断一本书的主题，仅提供简单的书名和一句简介是不足够的。这就像让一个人在极短时间内解决困难的数学题，错误在所难免。

相反，我们应通过 Prompt 指引语言模型进行深入思考。可以要求其先列出对问题的各种看法，说明推理依据，然后再得出最终结论。在 Prompt 中添加逐步推理的要求，能让语言模型投入更多时间逻辑思维，输出结果也将更可靠准确。

综上所述，给予语言模型充足的推理时间，是 Prompt Engineering 中一个非常重要的设计原则。这将大大提高语言模型处理复杂问题的效果，也是构建高质量 Prompt 的关键之处。开发者应注意给模型留出思考空间，以发挥语言模型的最大潜力。

2.1 指定完成任务所需的步骤

接下来我们将通过给定一个复杂任务，给出完成该任务的一系列步骤，来展示这一策略的效果。

首先我们描述了杰克和吉尔的故事，并给出提示词执行以下操作：首先，用一句话概括三个反引号限定的文本。第二，将摘要翻译成英语。第三，在英语摘要中列出每个名称。第四，输出包含以下键的 JSON 对象：英语摘要和人名个数。要求输出以换行符分隔。

```
text = f"""
在一个迷人的村庄里，兄妹杰克和吉尔出发去一个山顶井里打水。\
他们一边唱着欢乐的歌，一边往上爬，\
然而不幸降临——杰克绊了一块石头，从山上滚了下来，吉尔紧随其后。\
虽然略有些摔伤，但他们还是回到了温馨的家中。\
尽管出了这样的意外，他们的冒险精神依然没有减弱，继续充满愉悦地探索。
"""
# example 1
prompt_1 = f"""
执行以下操作：
1-用一句话概括下面用三个反引号括起来的文本。
2-将摘要翻译成英语。
3-在英语摘要中列出每个人名。
4-输出一个 JSON 对象，其中包含以下键：english_summary，num_names。

请用换行符分隔您的答案。

Text:
```{text}```
"""
response = get_completion(prompt_1)
print("prompt 1:")
print(response)
```
```

```
prompt 1:
1-两个兄妹在山上打水时发生意外，但最终平安回家。
2-In a charming village, siblings Jack and Jill set off to fetch water from a well on top of a hill. While singing joyfully, they climbed up, but unfortunately, Jack tripped on a stone and rolled down the hill, with Jill following closely behind. Despite some minor injuries, they made it back to their cozy home. Despite the mishap, their adventurous spirit remained undiminished as they continued to explore with delight.
3-Jack, Jill
4-{"english_summary": "In a charming village, siblings Jack and Jill set off to fetch water from a well on top of a hill. While singing joyfully, they climbed up, but unfortunately, Jack tripped on a stone and rolled down the hill, with Jill following closely behind. Despite some minor injuries, they made it back to their cozy home. Despite the mishap, their adventurous spirit remained undiminished as they continued to explore with delight.", "num_names": 2}
```

上述输出仍然存在一定问题，例如，键"姓名"会被替换为法语（译注：在英文原版中，要求从英语翻译到法语，对应指令第三步的输出为 'Noms:'，为Name的法语，这种行为难以预测，并可能为导出带来困难）

因此，我们将Prompt加以改进，该 Prompt 前半部分不变，同时**确切指定了输出的格式。**

```
prompt_2 = f"""
1-用一句话概括下面用<>括起来的文本。
2-将摘要翻译成英语。
3-在英语摘要中列出每个名称。
4-输出一个 JSON 对象，其中包含以下键：English_summary，num_names。

请使用以下格式：
文本：<要总结的文本>
摘要：<摘要>
翻译：<摘要的翻译>
名称：<英语摘要中的名称列表>
输出 JSON：<带有 English_summary 和 num_names 的 JSON>

Text: <{text}>
"""
response = get_completion(prompt_2)
print("\nprompt 2:")
print(response)
```

```
prompt 2:
Summary: 在一个迷人的村庄里，兄妹杰克和吉尔在山顶井里打水时发生了意外，但他们的冒险精神依然没有减弱，继续充满愉悦地探索。

Translation: In a charming village, siblings Jack and Jill set off to fetch water from a well on top of a hill. Unfortunately, Jack tripped on a rock and tumbled down the hill, with Jill following closely behind. Despite some minor injuries, they made it back home safely. Despite the mishap, their adventurous spirit remained strong as they continued to explore joyfully.

Names: Jack, Jill

JSON Output: {"English_summary": "In a charming village, siblings Jack and Jill set off to fetch water from a well on top of a hill. Unfortunately, Jack tripped on a rock and tumbled down the hill, with Jill following closely behind. Despite some minor injuries, they made it back home safely. Despite the mishap, their adventurous spirit remained strong as they continued to explore joyfully.", "num_names": 2}
```

2.2 指导模型在下结论之前找出一个自己的解法

在设计 Prompt 时，我们还可以通过明确指导语言模型进行自主思考，来获得更好的效果。

举个例子，假设我们要语言模型判断一个数学问题的解答是否正确。仅仅提供问题和解答是不够的，语言模型可能会匆忙做出错误判断。

相反，我们可以在 Prompt 中先要求语言模型自己尝试解决这个问题，思考出自己的解法，然后再与提供的解答进行对比，判断正确性。这种先让语言模型自主思考的方式，能帮助它更深入理解问题，做出更准确的判断。

接下来我们会给出一个问题和一份来自学生的解答，要求模型判断解答是否正确：

```
prompt = f"""
判断学生的解决方案是否正确。

问题：
我正在建造一个太阳能发电站，需要帮助计算财务。

    土地费用为 100美元/平方英尺
    我可以以 250美元/平方英尺的价格购买太阳能电池板
    我已经谈判好了维护合同，每年需要支付固定的10万美元，并额外支付每平方英尺10美元
    作为平方英尺数的函数，首年运营的总费用是多少。

学生的解决方案：
设x为发电站的大小，单位为平方英尺。
费用：

    土地费用：100x
    太阳能电池板费用：250x
    维护费用：100,000美元+100x
    总费用：100x+250x+100,000美元+100x=450x+100,000美元
"""
response = get_completion(prompt)
print(response)
```

学生的解决方案是正确的。他正确地计算了土地费用、太阳能电池板费用和维护费用，并将它们相加得到了总费用。

但是注意，学生的解决方案实际上是错误的。（*维护费用项100x应为10x，总费用450x应为360x*）

我们可以通过指导模型先自行找出一个解法来解决这个问题。

在接下来这个 Prompt 中，我们要求模型先自行解决这个问题，再根据自己的解法与学生的解法进行对比，从而判断学生的解法是否正确。同时，我们给定了输出的格式要求。通过拆分任务、明确步骤，让模型有更多时间思考，有时可以获得更准确的结果。在这个例子中，学生的答案是错误的，但如果我们没有先让模型自己计算，那么可能会被误导以为学生是正确的。

```
prompt = f"""
请判断学生的解决方案是否正确，请通过如下步骤解决这个问题：

步骤：

    首先，自己解决问题。
    然后将您的解决方案与学生的解决方案进行比较，对比计算得到的总费用与学生计算的总费用是否一致，
并评估学生的解决方案是否正确。
    在自己完成问题之前，请勿决定学生的解决方案是否正确。

使用以下格式：

    问题：问题文本
    学生的解决方案：学生的解决方案文本
    实际解决方案和步骤：实际解决方案和步骤文本
    学生计算的总费用：学生计算得到的总费用
    实际计算的总费用：实际计算出的总费用
    学生计算的费用和实际计算的费用是否相同：是或否
    学生的解决方案和实际解决方案是否相同：是或否
    学生的成绩：正确或不正确

问题：

    我正在建造一个太阳能发电站，需要帮助计算财务。
    - 土地费用为每平方英尺100美元
    - 我可以以每平方英尺250美元的价格购买太阳能电池板
    - 我已经谈判好了维护合同，每年需要支付固定的10万美元，并额外支付每平方英尺10美元;

    作为平方英尺数的函数，首年运营的总费用是多少。

学生的解决方案：

    设x为发电站的大小，单位为平方英尺。
    费用：
    1. 土地费用：100x美元
    2. 太阳能电池板费用：250x美元
    3. 维护费用：100,000+100x=10万美元+10x美元
    总费用：100x美元+250x美元+10万美元+100x美元=450x+10万美元

实际解决方案和步骤：
"""
response = get_completion(prompt)
```

```
print(response)
```

实际解决方案和步骤：

1. 土地费用：每平方英尺100美元，所以总费用为100x美元。
2. 太阳能电池板费用：每平方英尺250美元，所以总费用为250x美元。
3. 维护费用：固定费用为10万美元，额外费用为每平方英尺10美元，所以总费用为10万美元+10x美元。
4. 总费用：将上述三项费用相加，得到总费用为100x美元+250x美元+10万美元+10x美元=360x+10万美元。

学生计算的总费用：450x+10万美元
实际计算的总费用：360x+10万美元
学生计算的费用和实际计算的费用是否相同：否
学生的解决方案和实际解决方案是否相同：否
学生的成绩：不正确

三、局限性

开发大模型相关应用时请务必铭记：

虚假知识：模型偶尔会生成一些看似真实实则编造的知识

在开发与应用语言模型时，需要注意它们可能生成虚假信息的风险。尽管模型经过大规模预训练，掌握了丰富知识，但它实际上并没有完全记住所见的信息，难以准确判断自己的知识边界，可能做出错误推断。若让语言模型描述一个不存在的产品,它可能会自行构造出似是而非的细节。这被称为“幻觉”(Hallucination)，是语言模型的一大缺陷。

如下示例展示了大模型的幻觉。我们要求告诉我们华为公司生产的 *GT Watch 运动手表* 产品的信息，事实上，这个公司是真实存在的，但产品是编造的，而模型一本正经地提供了它编造的知识，而且迷惑性很强。

```
prompt = f"""
告诉我华为公司生产的GT Watch运动手表的相关信息
"""
response = get_completion(prompt)
print(response)
```

华为公司生产的GT Watch运动手表是一款智能手表，具有多种功能和特点。以下是相关信息：

1. 设计和外观：GT Watch采用圆形表盘设计，具有精致的外观和高质量的材料制造。它有多种颜色和表带选择，可以根据个人喜好进行定制。

2. 显示屏：GT Watch配备了1.39英寸的AMOLED显示屏，具有高清分辨率和良好的可视性。用户可以通过触摸屏幕进行操作和导航。

3. 运动追踪：GT Watch具有全天候的运动追踪功能，可以监测用户的步数、跑步距离、卡路里消耗和心率等数据。它还支持多种运动模式，如跑步、骑行、游泳等。

4. 健康监测：GT Watch可以监测用户的心率、血氧饱和度和睡眠质量等健康指标。它还提供健康建议和提醒，帮助用户保持良好的健康状态。

5. 通知和连接：GT Watch可以与用户的手机进行连接，通过蓝牙技术实现通知推送和电话提醒。用户可以在手表上查看短信、电话和社交媒体通知，无需拿出手机。

6. 长续航时间：GT Watch具有较长的续航时间，一次充电可以使用数天。它还支持快速充电技术，可以在短时间内充满电。

7. 其他功能：GT Watch还具有其他功能，如天气预报、闹钟、计时器、计步器等。它还支持NFC支付和音乐控制等便利功能。

总体而言，华为GT Watch是一款功能强大、外观精致的智能运动手表，适合那些注重健康和运动的用户使用。

语言模型生成虚假信息的“幻觉”问题，是使用与开发语言模型时需要高度关注的风险。由于幻觉信息往往令人无法辨别真伪，开发者必须警惕并尽量避免它的产生。

目前 OpenAI 等公司正在积极研究解决语言模型的幻觉问题。在技术得以进一步改进之前，开发者可以通过Prompt设计减少幻觉发生的可能。例如，可以先让语言模型直接引用文本中的原句，然后再进行解答。这可以追踪信息来源，降低虚假内容的风险。

综上，语言模型的幻觉问题事关应用的可靠性与安全性。开发者有必要认识到这一缺陷（注：截至2023年7月），并采取Prompt优化等措施予以缓解，以开发出更加可信赖的语言模型应用。这也将是未来语言模型进化的重要方向之一。

注意:

关于反斜杠使用的说明：在本教程中，我们使用反斜杠 \ 来使文本适应屏幕大小以提高阅读体验，而没有用换行符 \n 。GPT-3 并不受换行符（newline characters）的影响，但在您调用其他大模型时，需额外考虑换行符是否会影响模型性能。

四、英文原版 Prompt

1.1 使用分隔符清晰地表示输入的不同部分

```
text = f"""
You should express what you want a model to do by \
providing instructions that are as clear and \
specific as you can possibly make them. \
This will guide the model towards the desired output, \
and reduce the chances of receiving irrelevant \
or incorrect responses. Don't confuse writing a \
clear prompt with writing a short prompt. \
In many cases, longer prompts provide more clarity \
and context for the model, which can lead to \
more detailed and relevant outputs.
"""
prompt = f"""
Summarize the text delimited by triple backticks \
into a single sentence.
```{text}```
"""
response = get_completion(prompt)
print(response)
```

```
To guide a model towards the desired output and reduce irrelevant or incorrect
responses, it is important to provide clear and specific instructions, which can
be achieved through longer prompts that offer more clarity and context.
```

1.2 寻求结构化的输出
```