

3 Prompt 工程

随着模型训练数据规模和参数数量的持续增长，大语言模型突破了泛化瓶颈，并涌现出了强大的指令跟随能力。泛化能力的增强使得模型能够处理和理解多种未知任务，而指令跟随能力的提升则确保了模型能够准确响应人类的指令。两种能力的结合，使得我们能够通过精心编写的指令输入，即 **Prompt**，来引导模型适应各种下游任务，从而避免了传统微调方法所带来的高昂计算成本。**Prompt 工程**，作为一门专注于如何编写这些有效指令的技术，成为了连接模型与任务需求之间的桥梁。它不仅要求对模型有深入的理解，还需要对任务目标有精准的把握。通过 **Prompt 工程**，我们能够最大化地发挥大语言模型的潜力，使其在多样化的应用场景中发挥出卓越的性能。本章将深入探讨 **Prompt 工程** 的概念、方法及作用，并介绍上下文学习、思维链等技术，以及 **Prompt 工程** 的相关应用。

* 本书持续更新，GIT Hub 链接为：<https://github.com/ZJU-LLMs/Foundations-of-LLMs>。

3.1 Prompt 工程简介

传统的自然语言处理研究遵循“预训练-微调-预测”范式，即先在大规模语料库上作预训练，然后在下游任务上微调，最后在微调后的模型上进行预测。然而，随着语言模型在规模和能力上的显著提升，一种新的范式——“预训练-提示预测”应运而生，即在预训练模型的基础上，通过精心设计 Prompt 引导大模型直接适应下游任务，而无需进行繁琐微调，如图 3.1 所示。在这一过程中，Prompt 的设计将对模型性能产生深远影响。这种专注于如何编写 Prompt 的技术，被称为 **Prompt 工程**。在本节中，我们将深入介绍 Prompt 工程的定义及其相关概念，探讨其在自然语言处理领域中的重要性和应用。

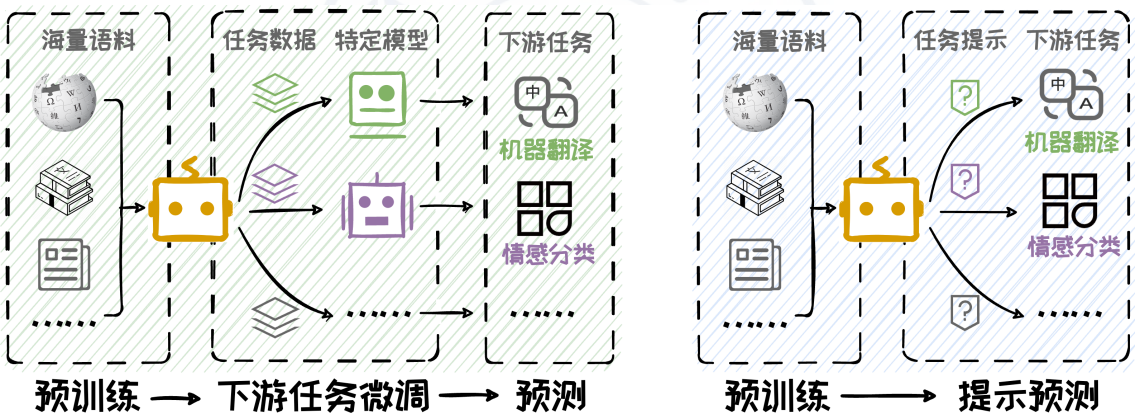


图 3.1: “预训练-微调-预测”范式与“预训练-提示预测”范式对比。

3.1.1 Prompt 的定义

Prompt 是指用于指导生成式人工智能模型执行特定任务的输入指令¹，这些指令通常以自然语言文本的形式出现。Prompt 的核心目的是清晰地描述模型应该执行的任务，以引导模型生成特定的文本、图像、音频等内容。如图 3.2 所示，通过

¹https://en.wikipedia.org/wiki/Prompt_engineering



图 3.2: 几种常见的 Prompt 例子。

精心设计的 Prompt，模型能够实现多样化的功能。例如，通过提供明确的情感分类指令，模型能够准确地对文本进行情感分析；通过特定主题的创作指令，模型能够生成富有创意的诗歌。此外，在多模态模型的应用场景中，Prompt 还可以包含画面描述，从而指导模型生成相应的视觉作品。

Prompt 的应用范围广泛，不仅限于文本到文本的任务。由于本书主要关注语言模型，本章节将聚焦于**文本生成模型**，并深入探讨如何通过精心设计的 Prompt 来引导模型生成符合特定任务要求的文本输出。

3.1.2 Prompt 工程的定义

Prompt 工程 (Prompt Engineering)，又称提示工程，是指设计和优化用于与生成式人工智能模型交互的 Prompt 的过程²。这种技术的核心在于，将新任务通过 Prompt 构建为模型在预训练阶段已经熟悉的形式，利用模型固有的泛化能力来执行新的任务，而无需在额外的特定任务上进行训练。Prompt 工程的成功依赖于对

²https://en.wikipedia.org/wiki/Prompt_engineering

预训练模型的深入理解，以及对任务需求的精确把握。通过构造合适的 Prompt 输入给大语言模型，大语言模型能够帮助我们完成各种任务 [47]。

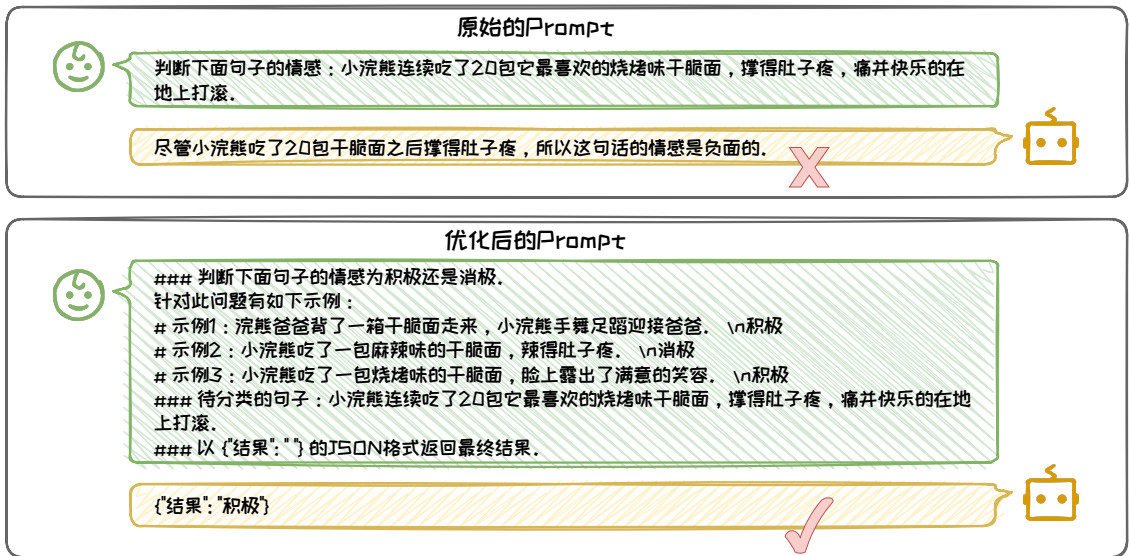


图 3.3: Prompt 工程技术应用前后的效果对比。

如图 3.3所示，通过 Prompt 工程的优化，原始 Prompt 被改写为更加全面、规范的形式。优化后的 Prompt 能够显著提升模型生成回答的质量。因此，在与大语言模型互动过程中构建优质且全面的 Prompt 至关重要，它直接决定了能否获得有价值的输出。经过良好设计的 Prompt 通常由任务说明、上下文、问题、输出格式四个基本元素组成：

- **任务说明**——向模型明确提出具体的任务要求。任务说明应当清晰、直接，并尽可能详细地描述期望模型完成的任务。
- **上下文**——向模型提供的任务相关背景信息，用以增强其对任务的理解以及提供解决任务的思路。上下文可以包括特定的知识前提、目标受众的背景、相关任务的示例，或任何有助于模型更好地理解任务的信息。

- **问题**——向模型描述用户的具体问题或需要处理的信息。这部分应直接涉及用户的查询或任务，为模型提供一个明确的起点。问题可以是显式的提问，也可以是隐式的陈述句，用以表达用户的潜在疑问。
- **输出格式**——期望模型给出的回答的展示形式。这包括输出的格式，以及任何特定的细节要求，如简洁性或详细程度。例如，可以指示模型以 JSON 格式输出结果。

Prompt 的四个基本元素——任务说明、上下文、问题和输出格式，对于大语言模型生成的效果具有显著影响。这些元素的精心设计和组合构成了 Prompt 工程的核心。在此基础上，Prompt 工程包括多种技巧和技术，如上下文学习（In-Context Learning）和思维链（Chain of Thought）等。这些技巧和技术的结合使用，可以显著提升 Prompt 的质量，进而有效地引导模型生成更符合特定任务需求的输出。具体关于上下文学习的内容将在 3.2 节中讨论，思维链的内容将在 3.3 节中讨论，而 Prompt 的使用技巧将在 3.4 节中详细探讨。

然而，随着 Prompt 内容的丰富和复杂化，输入到模型中的 Prompt 长度也随之增加，这不可避免地导致了模型推理速度的减慢和推理成本的上升。因此，在追求模型性能的同时，如何有效控制和优化 Prompt 的长度，成为了一个亟待解决的问题，需要在确保模型性能不受影响的前提下，尽可能压缩输入到大型模型中的 Prompt 长度。为此，LLMLingua [11] 提出了一种创新的由粗到细的 Prompt 压缩方法，该方法能够在不牺牲语义完整性的情况下，将 Prompt 内容压缩至原来的二十分之一，同时几乎不损失模型的性能。此外，随着 RAG 技术的兴起，模型需要处理的上下文信息量大幅增加。FIT-RAG [22] 技术通过高效压缩检索出的内容，成功将上下文长度缩短至原来的 50% 左右，同时保持了性能的稳定，为处理大规模上下文信息提供了有效的解决方案。

3.1.3 Prompt 分词向量化

在构建合适的 Prompt 之后，用户将其输入到大语言模型中，以期得到满意的生成结果。但是，语言模型无法直接理解文本。在 Prompt 进入大模型之前，需要将它拆分成一个 Token 的序列，其中 Token 是**承载语义的最小单元**，标识具体某个词，并且每个 Token 由 Token ID 唯一标识。将文本转化为 Token 的过程称之为**分词** (Tokenization)，如图 3.4所示，对于“小浣熊吃干脆面”这样一句话，经过分词处理之后，会变成一个 Token 序列，每个 Token 有对应的 Token ID。

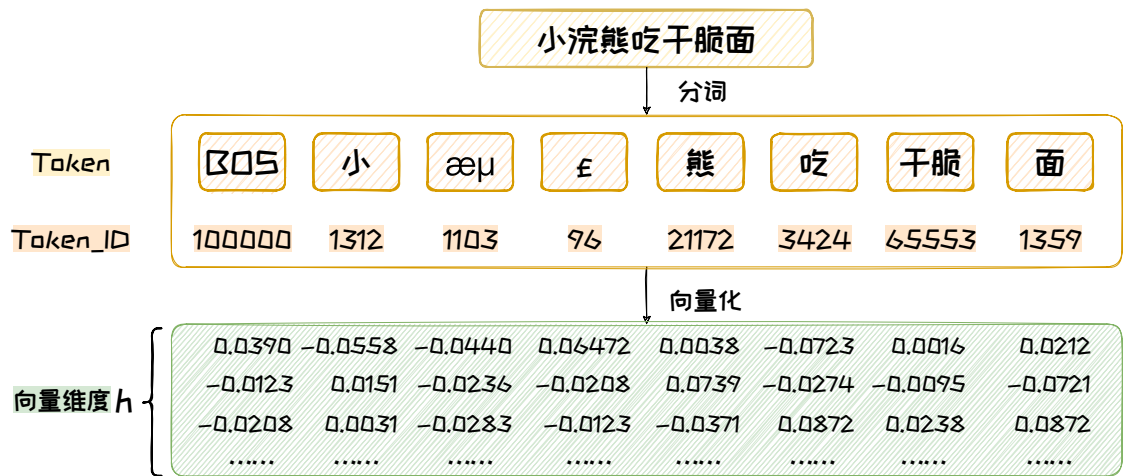


图 3.4: 分词与嵌入的过程，以 DeepSeek-V2-Chat 模型的分词器为例 [6]。

值得注意的是，一句话可能存在多种拆分方式。例如，上述句子可拆分为“小，浣熊，吃干，脆面”。然而，这种拆分方式可能导致语义混乱、不清晰。因此，分词过程颇具挑战性，我们需要精心设计分词方法。为实现有效分词，首先需构建一个包含大语言模型所能识别的所有 Token 的**词表**，并依据该词表进行句子拆分。

在构建大语言模型的词表时，分词器依赖于分词算法，如 BBPE [34]、BPE [8] 和 WordPiece [29] 等，这些算法通过分析语料库中的**词频**等信息来划分 Token。本文将**以 BBPE (Byte-Level Byte Pair Encoding) 算法为例**，阐述其分词过程。BBPE 算法的流程主要包括以下四个步骤：

- **1. 初始化词表：**首先，将所有字符按照其底层编码拆分为若干字节，并将这些单字节编码作为初始词表的 Token。
- **2. 统计词频：**接下来，统计词表中所有 Token 对（即相邻 Token 的组合）的出现频率。在初始阶段，Token 对即为相邻字节的组合。
- **3. 合并高频 Token 对：**然后，选择出现频率最高的 Token 对，将其合并成一个新的 Token 并加入词表。
- **4. 迭代合并：**重复步骤 2 和步骤 3，不断迭代合并，直至达到预设的词表大小或达到指定的合并次数。

我们可以通过设定迭代次数等方法来决定分词粒度，不同的分词粒度导致我们得到不同大小的词表。若**词表过小**，例如仅包含 256 个 Token 表示单字节，能够组合出所有 GBK 编码的汉字和英文，但会导致形态相近但意义迥异的词汇在模型中难以区分，限制了模型承载丰富语义的能力，并大幅增加生成的序列长度。相反，若**词表过大**，虽然能涵盖更多长尾词汇，但可能导致对这些词汇的学习不够深入，且难以有效捕捉同一单词不同形态之间的关联。因此，在构建词表时，需在涵盖广泛词汇与保持语义精细度之间找到恰当的平衡点，以确保大语言模型既能学习到丰富的词汇知识，又能准确理解和生成具有复杂语义的文本。

具体来说，如图 3.4 所示，为了有助于模型更准确地理解词义，同时减少生成常用词所需的 Token 数量，词表中收录了语料库中**高频**出现的词语或短语，形成独立的 Token。例如，“干脆”一词在词表中以一个 Token 来表示。为了优化 Token 空间并压缩词表大小，构建词表时会包含了一些特殊的 Token，这些 Token 既能单独表示语义，也能通过两两组合，表示语料库中**低频**出现的生僻字。例如，“浣”字使用两个 Token，“æµ”和“£”，来表示。通过这种处理方式，词表既能涵盖常见的高频词汇，又能通过 Token 组合灵活表达各类稀有字符。由此可见，词表构建在一定程度上受到“先验知识”的影响，这些知识源自人类语料库的积累与沉淀。

每个大语言模型都有自己的分词器，分词器维护一个词表，能够对文本进行分词。分词器的质量对模型的性能有着直接的影响。一个优秀的分词器不仅能显著提升模型对文本的理解能力，还能够提高模型的处理速度，减少计算资源的消耗。一个好的分词器应当具备以下特点：首先，它能够准确地识别出文本中的关键词和短语，从而帮助模型更好地捕捉语义信息；其次，分词器的效率直接影响到模型的训练和推理速度，一个高效的分词器能够实现对文本 Token 的优化压缩，进而显著缩短模型在处理数据时所需的时间。

表 3.1: 模型分词器对比表

模型	词表大小	中文分词效率 (字 / Token)	英文分词效率 (词 / Token)
LLaMA1	32000	0.6588	0.6891
LLaMA2	32000	0.6588	0.6891
LLaMA3	128256	1.0996	0.7870
DeepSeek-V1	100016	1.2915	0.7625
DeepSeek-V2	100002	1.2915	0.7625
GPT-3.5 & GPT-4	100256	0.7723	0.7867
GPT-3	50,257	0.4858	0.7522
Qwen-1.5	151646	1.2989	0.7865
StarCoder	49152	0.9344	0.6513

在常用的开源模型中，不同模型采用了不同的分词器，这些分词器具有各自的特点和性能。它们的质量受到多种因素的影响，包括词表的大小、分词的效率等属性。表 3.1 是对常见开源大语言模型的分词器的对比分析，其中中文语料库节选自《朱自清散文》³，英文语料库来自莫泊桑短篇小说《项链》⁴。我们可以观察到，像 DeepSeek [6]、Qwen [41] 这类中文开源大语言模型，对中文分词进行了优化，平均每个 Token 能够表示 1.3 个字（每个字仅需 0.7 个 Token 即可表示），一些常用词语和成语甚至可以直接用一个 Token 来表示。相比之下，以英文为主要语料的模型，如 GPT-4、LLaMA 系列，对中文的支持度较弱，分词效率不高。在英文

³https://www.sohu.com/a/746456997_120075260.

⁴<https://americanliterature.com/author/guy-de-maupassant/short-story/the-diamond-necklace>

中，由于存在“ly”、“ist”等后缀 Token，一个英文单词通常需要用 1 个及以上的 Token 来表示。单个 Token 承载更多的语义，模型在表达同样的文本时，只需要输出更少的 Token，显著提升了推理效率。

通过这种对比，我们可以清晰地看到不同模型分词器在处理不同语言时的效率，这对于选择合适的模型和优化模型性能具有重要的指导意义。

在完成分词之后，这些 Token 随后会经过模型的嵌入矩阵 (Embedding Matrix) 处理，转化为固定大小的**表征向量**。这些向量序列被直接输入到模型中，供模型理解和处理。在模型生成阶段，模型会根据输入的向量序列计算出词表中每个词的概率分布。模型从这些概率分布中选择并输出对应的 Token，这些 Token 再被转换为相应的文本内容。

上述通过分词技术将文本分割成 Token，再将 Token 转化为特征向量，在高维空间中表征这些文本的处理流程，使得语言模型能够捕捉文本的深层语义结构，并有效地处理和学习各种语言结构，从简单的词汇到复杂的句式和语境。

3.1.4 Prompt 工程的意义

Prompt 工程提供了一种高效且灵活的途径来执行自然语言处理任务。它允许我们无需对模型进行微调，便能有效地完成既定任务，避免微调带来的巨大开销。通过精心设计的 Prompt，我们能够激发大型语言模型的内在潜力，使其在**垂域任务**、**数据增强**、**智能代理**等多个领域发挥出卓越的性能。

1. 垂域任务

应用 Prompt 工程来引导大语言模型完成垂直领域任务，可以避免针对每个任务进行特定微调。不仅可以避免微调模型的高昂计算成本，还可以减少对标注数据的依赖。使得大语言模型可以更好的应用在垂直领域任务中。例如，在 Text-to-SQL 任务中，我们可以应用 Prompt 工程的技巧，引导大语言模型根据用户输入的文本直接生成高质量的 SQL 查询，而无需进行有监督微调。基于 GPT 模型的 Prompt

工程方法在 Spider [44] 榜单上取得了突破性成绩，超越了传统的微调方法。此外，在知识密集型任务问答领域 MMLU [9] 等多个领域，基于 Prompt 工程的方法也取得最佳效果。

2. 数据增强

应用 Prompt 工程通过大语言模型来进行数据增强，不仅能够提升现有数据集的质量，还能够生成新的高质量数据。这些数据可以用于训练和优化其它模型，以将大语言模型的能力以合成数据的方法“蒸馏”到其他模型上。例如，我们可以引导 ChatGPT 模型生成包含丰富推理步骤的数据集，用于增强金融领域 Text-to-SQL 模型的推理能力 [45]。此外，通过精心设计的提示，还能生成包含复杂指令的数据集，如 Alpaca [32] 和 Evol-Instruct [21]。将这些合成的数据集用于微调参数量较小的模型，可以其在保持较小模型尺寸和低计算成本的同时，接近大型模型的性能。

3. 智能代理

应用 Prompt 工程可以将大语言模型构建为智能代理 (Intelligent Agent, IA)⁵。智能代理，又叫做智能体，能够感知环境、自主采取行动以实现目标，并通过学习或获取知识来提高其性能。在智能代理进行感知环境、采取行动、学习知识的过程中，都离不开 Prompt 工程。例如，斯坦福大学利用 GPT-4 模拟了一个虚拟西部小镇 [25]，多个基于 GPT-4 的智能体在其中生活和互动，他们根据自己的角色和目标自主行动，进行交流，解决问题，并推动小镇的发展。整个虚拟西部小镇的运转都是由 Prompt 工程驱动的。

本节探讨了 Prompt 的概念，Prompt 工程的概念以及意义，揭示了 Prompt 在大语言模型应用中的关键作用和广阔潜力。接下来，我们将进一步拓展这一主题：第3.2节将探索上下文学习的相关内容，揭示其在提升模型理解和响应能力中的作用；第3.3节将详细介绍思维链提示方法及其变种，展示如何通过这些方法增强模

⁵https://en.wikipedia.org/wiki/Intelligent_agent

型的逻辑推理和问题解决能力；第3.4节将分享构建有效 Prompt 的技巧，指导读者如何设计 Prompt 以激发模型生成更优质的内容；第3.5节将具体展示 Prompt 工程在大语言模型中的实际应用，通过实例阐释其不同场景下的应用策略和效果。

3.2 上下文学习

随着模型训练数据规模和参数数量的持续扩大，大语言模型涌现出了上下文学习（In-Context Learning, ICL）能力。其使得语言模型能够通过给定的任务说明或示例等信息来掌握处理新任务的能力。引入上下文学习，我们不再需要针对某个任务训练一个模型或者在预训练模型上进行费时费力的微调，就可以快速适应一些下游任务。这使得用户可以仅仅通过页面或者 API 的方式即可利用大语言模型来解决下游任务，为“语言模型即服务”（LLM as a Service）模式奠定了坚实的基础。本节从上下文学习的定义，演示示例选择，和影响其性能的因素，对上下文学习展开介绍。

3.2.1 上下文学习的定义

上下文学习（In-Context Learning, ICL）[2] 是一种通过构造特定的 Prompt，来使得语言模型理解并学习下游任务的范式，这些特定的 Prompt 中可以包含**演示示例**，**任务说明**等元素。上下文学习实现的关键在于如何设计有效的 Prompt，以引导模型理解任务的上下文和目标。通常，这些 Prompt 会包含任务说明以及一系列的示例，模型能够从这些上下文信息中学习任务的逻辑和规则，从而在没有额外训练的情况下，生成符合任务要求的输出。基于以上优点，上下文学习被广泛应用于解决垂域任务，数据增强，智能代理等应用中。

在上下文学习中，Prompt 通常包含几个与待解决任务相关的演示示例，以展示任务输入与预期输出。这些示例按照特定顺序组成上下文，并与问题拼接共同

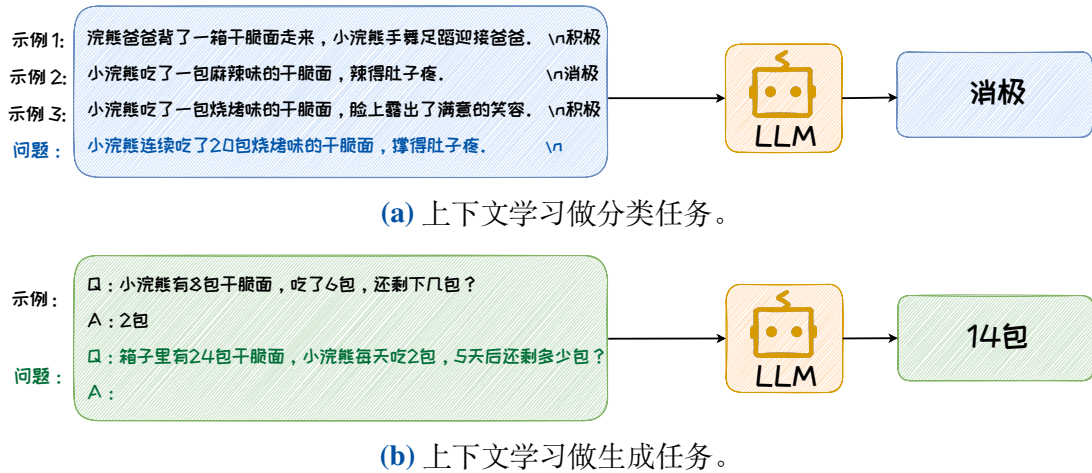


图 3.5: 上下文学习示例。

组成 Prompt 输入给大语言模型。大语言模型从上下文中学习任务范式，同时利用模型自身的能力对任务进行作答。在图 3.5 的例子 (a) 中，模型被用于文本情感分类任务，给定一段文本，模型能够判断其情感倾向，识别出文本表达的是积极还是消极情绪。图 3.5 的例子 (b) 则展示了数学运算任务，模型会仿照示例的形式，直接给出对应的运算结果。按照示例数量的不同，上下文学习可以呈现出多种形式：**零样本 (Zero-shot)** 上下文学习、**单样本 (One-shot)** 上下文学习和**少样本 (Few-shot)** 上下文学习 [2]，如图 3.6 所示。

- **零样本上下文学习**：在此种学习方式下，仅需向模型提供任务说明，而无需提供任何示例。其具有强大的场景泛化能力。但零样本学习的性能完全依赖于大语言模型的能力，并且在处理任务时可能表现欠佳。
- **单样本上下文学习**：这种方式仅需为模型提供一个示例，贴合人类“举一反三”的学习模式。不过，单样本学习的效果强依赖于示例相对于任务的代表性。
- **少样本上下文学习**：这种方法通过为模型提供少量的示例（通常为几个至十几个），显著提升模型在特定任务上的表现。但在，示例的增加会显著增加大语言模型推理时的计算成本。示例的代表性和多样性也将影响其生成效果。

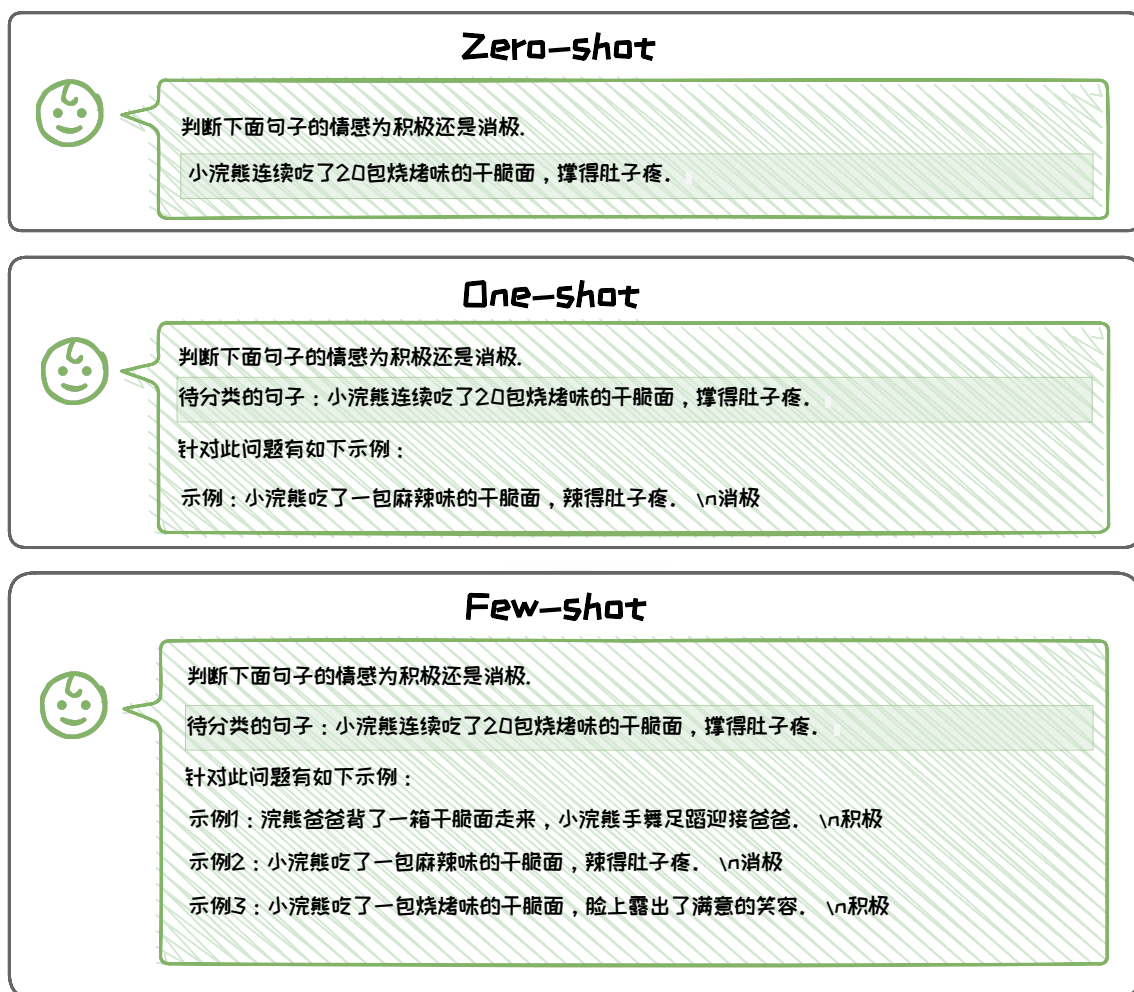


图 3.6: 三种上下文学习形式的示例。

尽管上下文学习在许多任务中表现出色，但它为何奏效仍然是一个重要的研究问题。对此，斯坦福大学的一项研究 [40] 提供了一种解释——“将上下文学习视为隐式贝叶斯推理”。在大语言模型的预训练阶段，模型从大量文本中学习潜在的概念。当运用上下文学习进行推理时，大语言模型借助演示示例来“锚定”其在预训练期间所习得的相关概念，从而进行上下文学习，并对问题进行预测。以图 3.5 为例，模型之所以能给出正确答案，是因为模型在预训练时已经学习到与情感相关的概念，比如积极情感的表现形式（满意的笑容，手舞足蹈……）、句法结构和句法关系等等，当模型在推理时，借助“浣熊吃了一包麻辣味的干脆面，辣得肚子疼。 \n 消极”等示例“锚定”到情感等相关概念，并基于这些概念，给出问题答案。

3.2.2 演示示例选择

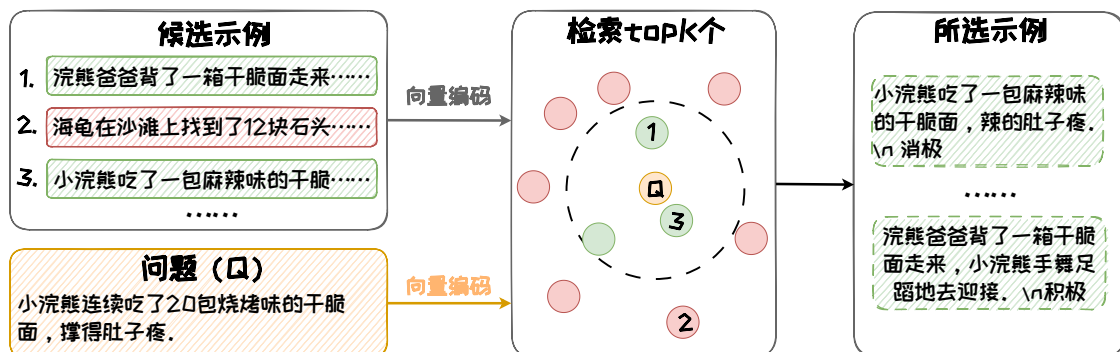
在上下文学习中，演示示例在引导大语言模型理解任务中扮演着重要作用，其内容和质量直接影响着学习效果。因此，合理选择演示示例对提升上下文学习性能至关重要。演示示例选择主要依靠相似性和多样性 [20]：

- **相似性**是指精心挑选出与待解决问题最为相近的示例。相似性可以从多个层面进行度量，如语言层面的相似性（包括关键字匹配或语义相似度匹配）、结构相似性等等。通过选取相似的示例，能够为模型提供与待解决问题接近的参照，使大语言模型更易理解该问题。
- **多样性**则要求所选的示例涵盖尽量广的内容，扩大演示示例对待解决问题的覆盖范围。多样化的示例能够帮助模型从不同的角度去理解任务，增强其应对各种问题的能力。

除了相似性和多样性，在某些任务中，还需对任务相关的因素进行考虑。本节主要关注相似性和多样性两个因素，并探讨基于相似性和多样性如何从大量候选示例中选择出合适的演示示例。接下来对基于相似性和多样性的三类示例选择方法 [20] 展开介绍：

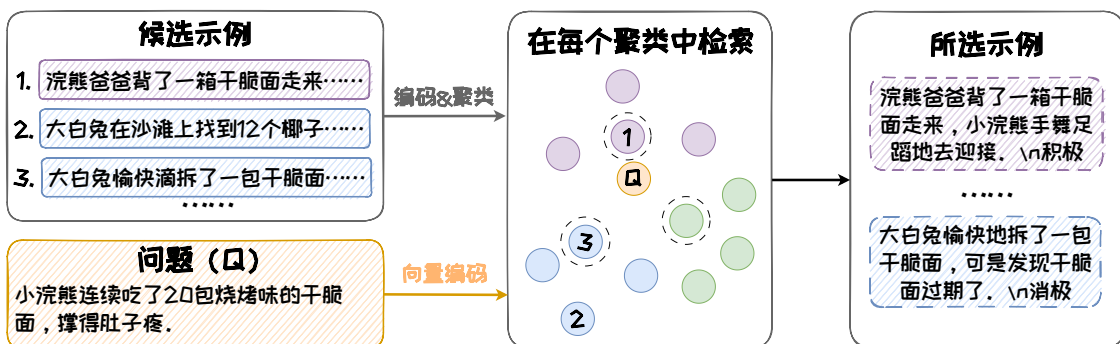
1. 直接检索

给定一组候选示例，直接检索的方法依据候选示例与待解决问题间的相似性对候选示例进行排序，然后选取排名靠前的 K 个示例。直接检索的代表性方法是 KATE [17]。如图 3.7 所示，KATE 利用 RoBERTa 对待解决问题和候选示例（移除标签）进行编码。然后通过计算解决问题编码和候选示例编码间的向量余弦相似度对二者的相似度进行评分。基于此评分，选择评分最高的 K 个示例作为上下文学习的演示示例。直接检索的方法简单易操作，是目前应用广泛的示例选择策略之一。但是，其未对示例的多样性进行考虑，选择出的示例可能趋向**同质化**。



2. 聚类检索

为缓解直接检索中存在的样例趋同的问题，聚类检索方法采用先聚类后检索的方法来保证检索结果的多样性。其先把所有候选示例划分为 K 个簇，然后从每个簇中选取最为相似的一个示例。这样可以有效避免选择出多个相似的示例，从而保障了多样性。Self-Prompting [15] 是其中的代表性方法。如图 3.8 所示，Self-Prompting 首先将候选示例和待解决问题编码成向量形式，接着运用 K-Means 算法把示例集合聚为 K 个簇。依照问题与示例之间的余弦相似度，从每个簇中选取与问题最相似的示例，由此得到 K 个示例。虽然聚类检索方法提高了示例的多样性，但因为有些簇与问题可能并不相似，导致选择的示例的相似性可能不够高。



3. 迭代检索

直接检索和聚类检索在相似性和多样性之间往往顾此失彼。为了**兼顾相似性多样性**，迭代检索策略应运而生。迭代检索首先挑选与问题高度相似的示例，随后在迭代过程中，结合当前问题和已选示例，动态选择下一个示例，从而确保所选示例的相似性和多样性。**RetICL** [28] 是迭代检索的代表性方法，如图 3.9 所示。**RetICL** 根据当前问题初始化基于 **LSTM** [10] 的检索器内部状态，并选择一个示例。接着根据当前问题和所选示例集更新检索器内部状态，并选择下一个示例。这一过程不断迭代，直到得到 k 个示例。尽管迭代检索在计算上相对复杂，但其能够生成更优的示例集，在复杂任务中展现出更好的适应性和灵活性。

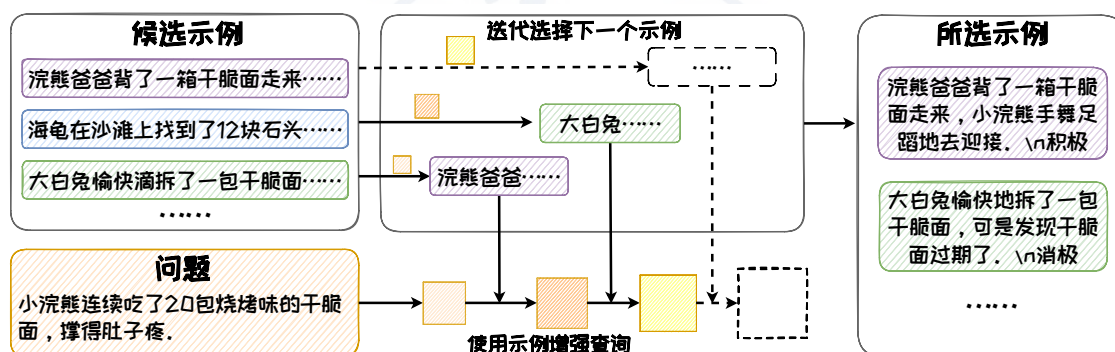


图 3.9: 迭代检索。

除了在示例选择策略上有所不同外，现有的示例选择方法在检索器的选择和设计上亦呈现出差异。一些方法采用现成的检索器，而另一些方法则为了追求更卓越的性能，选择在特定语料库上对检索器进行微调。关于检索器的深入介绍及其分类，将在第六章 6.8 中进行详尽阐述。

3.2.3 性能影响因素

通过精心设计示例选择策略，上下文学习的效果可以得到显著提升。但是，除了示例选择以外，上下文学习的性能仍受到多种因素的共同影响。这些因素涉及**预训练数据**、**预训练模型**，以及**演示示例**等多个方面 [48]。本小节将讨论这些关键因素如何影响上下文学习的效果。

1. 预训练数据的影响

预训练数据是上下文学习能力的来源，深刻影响着上下文学习的性能。对预训练数据而言，以下三方面因素是影响上下文学习性能的关键：

- **领域丰富度**：预训练数据的所覆盖的领域的丰富度直接影响模型的领域泛化能力。在丰富的跨领域语料库进行预训练的模型具备更稳定的上下文学习能力。而单一领域的语料库可能限制模型的适应性，即使该领域与目标任务高度相关，也无法保证最佳的上下文学习性能 [31]。
- **任务多样性**：预训练数据中的任务多样性是提升上下文学习性能的重要因素。多样化的任务类型有助于模型学习到更广泛的知识 and 技能，增强其任务泛化能力，从而在面对新任务时表现也更为出色 [26]。
- **训练数据的分布特性**：训练数据中存在突发性分布和罕见类别时，能够增强模型的上下文学习能力。突发性分布使得“文本-标签映射”的数据模式在整个训练过程中得以反复出现，罕见类别增强模型处理少见或新颖输入的能力，从而提升模型上下文学习的表现 [4]。

2. 预训练模型的影响

预训练模型对上下文学习的性能影响主要体现在**模型参数规模**上。当模型参数达到一定的规模时，上下文学习才能得以涌现。通常，模型的参数数量需达到亿级别及以上。常见的拥有上下文学习能力的模型中，规模最小的是来自阿里巴巴通义实验室的 Qwen2-0.5B 模型，具有 5 亿参数。一般而言，模型的规模越大，其上下文学习性能也越强 [39]。此外，模型的架构和训练策略也是影响上下文学习性能的重要因素。

3. 演示示例的影响

在 3.2.2 节中，我们已经讨论了演示示例选择对上下文学习的重要性。接下来，我们将继续探讨**演示示例的格式、输入-标签映射、示例数量及顺序**对上下文学习的影响。

示例格式。不同的任务对于示例格式的要求不同。如图3.5中展示的用上下文学习做情感分类的例子，我们只需要给出输入以及输出。但是，一些复杂推理任务，例如算术、代码等，仅仅给出输入和输出不足以让模型掌握其中的推理过程。在这种情况下，以思维链的形式构造示例，即在输入和输出之间添加中间推理步骤，能帮助模型逐步推理并学习到更复杂的映射关系，思维链将在3.3详细介绍。

输入-输出映射的正确性。对于一个包含输入、输出的示例，大语言模型旨在从中学习到输入-输出映射，以完成目标任务。如果给定示例中的输入-输出映射是错误的，势必会对上下文学习的效果产生影响。例如在情感分类任务中，将积极的文本错误地标记为消极，模型可能会学习到这种错误的映射方式，从而在下游任务中产生错误。对输入-输出映射错误的敏感性与模型规模大小有关 [14, 43]。较大的模型在上下文学习中对输入-输出映射的正确性表现出更高的敏感性 [14]。相比之下，较小的模型输入-输出映射错误的敏感性较弱 [24, 39]。

演示示例的数量和顺序。增加演示示例的数量通常能够提升上下文学习性能，但随着示例数量的增多，性能提升的速率会逐渐减缓 [23]。此外，相较于分类任务，生成任务更能从增加的示例数量中获益 [16]。此外，演示示例的顺序同样是影响上下文学习性能的关键因素，不同示例顺序下的模型表现存在显著差异。最优的示例顺序具有模型依赖性，即对某一模型有效的示例顺序未必适用于其他模型 [19]。同时，示例顺序的选择也受到所使用数据集的特定特性影响 [17]。

除上述因素外，Prompt 中的任务说明的质量也直接影响上下文学习的效果。清晰、明确的任务说明能够为模型提供明确指导，从而提升上下文学习的性能 [43]。因此，在设计演示示例和任务说明时，应综合考虑这些因素，以优化模型的表现。

本节介绍了上下文学习，区分了零样本、单样本和少样本三种形式。探讨了直接检索、聚类检索和迭代检索三种示例选择策略，分析其优缺点及代表性方法。最后从预训练数据、模型本身及演示示例等方面分析影响上下文学习性能的因素。

3.3 思维链

随着语言模型参数规模的持续扩张，其可以更好的捕捉语言特征和结构，从而在语义分析、文本分类、机器翻译等自然语言处理任务中的表现显著增强。但是，在面对算术求解、常识判断和符号推理等需要复杂推理能力的任务时，模型参数规模的增长并未带来预期的性能突破，这种现象被称作“Flat Scaling Curves” [38]。这表明，仅靠模型规模的扩大不足以解决所有问题，我们需要探索新的方法以提升模型的推理能力和智能水平。人类在解决复杂问题时，通常会逐步构建推理路径以导出最终答案。基于这一理念，一种创新的 Prompt 范式——**思维链提示** (Chain-of-Thought, CoT) [38] 被用于引导模型进行逐步推理。CoT 可以显著提升大语言模型处理复杂任务中的表现，从而突破“Flat Scaling Curves”的限制，激发大语言模型的内在推理潜能。

3.3.1 思维链提示的定义

思维链提示 (Chain-of-Thought, CoT) [38] 通过模拟人类解决复杂问题时的思考过程，引导大语言模型在生成答案的过程中引入一系列的**中间推理步骤**。这种方法不仅能够显著提升模型在推理任务上的表现，而且还能够揭示模型在处理复杂问题时的内部逻辑和推理路径。

CoT 方法的核心是构造**合适的 Prompt** 以触发大语言模型**一步一步生成推理路径**，并生成最终答案。早期方法在构造 Prompt 时，加入少量包含推理过程的样本示例 (Few-Shot Demonstrations) [38]，来引导模型一步一步生成答案。在这些示例中，研究者精心编写在相关问题上的推理过程，供模型模仿、学习。这种方法使得模型能够从这些示例中学习如何生成推理步骤，一步步输出答案。图 3.10 展示了一个用于求解数学问题的 CoT 形式的 Prompt 的例子。其中，样例给出了与待求

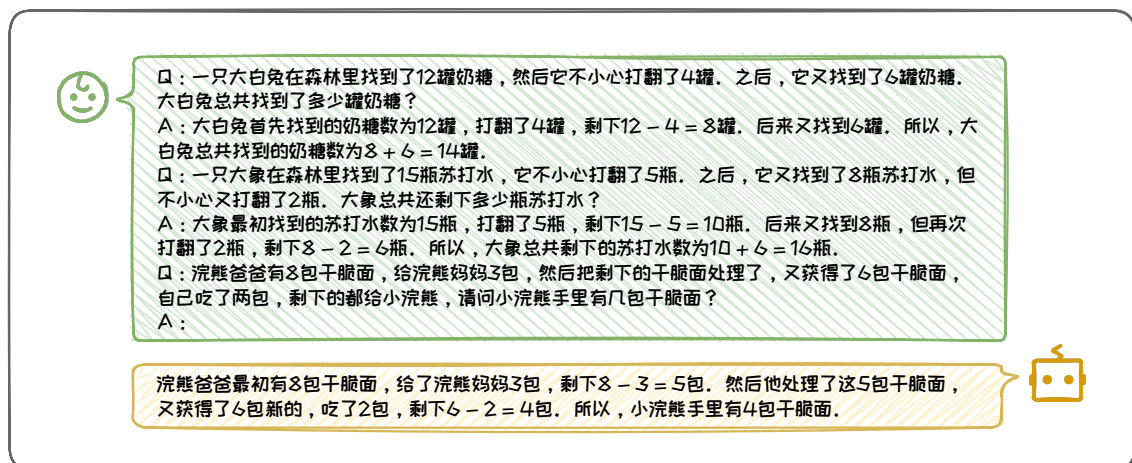


图 3.10: 包含少量样本示例的 CoT 提示示例。

解问题相关的数学问题的解题步骤作为参考, 大语言模型会模仿此样例对复杂的数学计算一步一步进行求解。通过引入 CoT, 大语言模型在解算术求解、常识判断和符号推理等复杂问题上性能显著提升。并且在 CoT 的加持下, 大语言模型处理复杂问题的能力随着模型参数规模的变大而增强。

在 CoT 核心思想的指引下, 衍生出了一系列的扩展的方法。这些扩展的方法按照其推理方式的不同, 可以归纳为三种模式: **按部就班**、**三思后行**和**集思广益**。这几种模式的对比如图 3.11所示。

- **按部就班**。在按部就班模式中, 模型一步接着一步地进行推理, 推理路径形成了一条逻辑连贯的链条。在这种模式下, 模型像是在遵循一条预设的逻辑路径, “按部就班”的一步步向前。这种模式以 CoT [38]、Zero-Shot CoT [12]、Auto-CoT [46] 等方法为代表。
- **三思后行**。在三思后行模式中, 模型每一步都停下来估当前的情况, 然后从多个推理方向中选择出下一步的行进方向。在这种模式下, 模型像是在探索一片未知的森林, 模型在每一步都会停下来评估周围的环境, “三思后行”以找出最佳推理路径。这种模式以 ToT [42]、GoT [1] 等方法为代表。

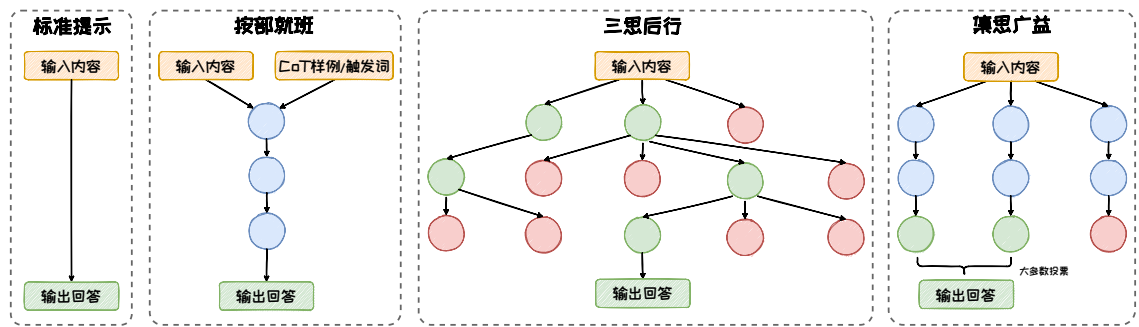


图 3.11: 不同 CoT 结构的对比。

- **集思广益**。在集思广益模式中，模型同时生成多条推理路径并得到多个结果，然后整合这些结果，得到一个更为全面和准确的答案。在这种模式下，模型像是在召开一场智者的会议，每个智者都带来了自己的见解，最终通过讨论和整合，“集思广益”得出一个更优的结论。这一类模式以 Self-Consistency [36] 等方法为代表。

3.3.2 按部就班

按部就班模式强调的是**逻辑的连贯性**和**步骤的顺序性**。在这种模式下，模型一步接着一步的进行推理，最终得到结论。其确保了推理过程的清晰和有序，使得模型的决策过程更加透明和可预测。原始的少样本思维链（CoT）方法就采用了按部就班模式。其通过手工构造几个一步一步推理回答问题的例子作为示例放入 Prompt 中，来引导模型一步一步生成推理步骤，并生成最终的答案。这种方法在提升模型推理能力方面取得了一定的成功，但是需要费时费力地手工编写大量 CoT 示例，并且过度依赖于 CoT 的编写质量。针对这些问题，研究者在原始 CoT 的基础上进行了扩展，本节将介绍 CoT 的两种变体：Zero-Shot CoT 和 Auto-CoT。

1. Zero-Shot CoT

Zero-Shot CoT [12] 通过简单的提示，如 “Let’s think step by step”，引导模型自行生成一条推理链。其无需手工标注的 CoT 示例，减少了对人工示例的依赖，多个推理任务上展现出了与原始少样本 CoT 相媲美甚至更优的性能。

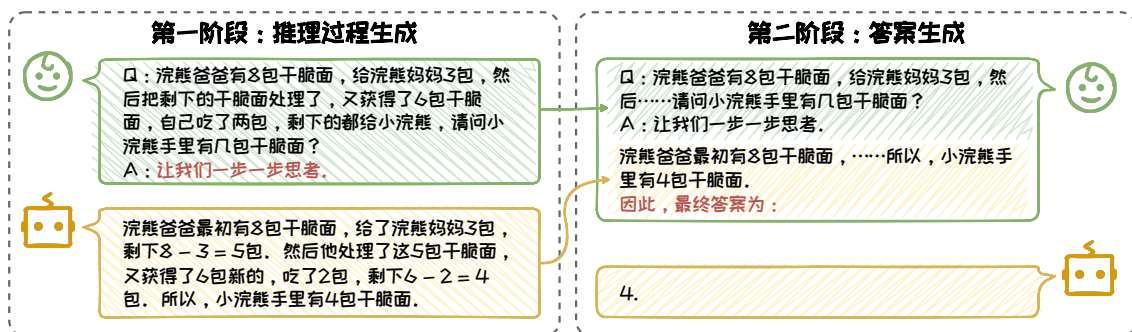


图 3.12: Zero-Shot CoT 提示的流程。

Zero-Shot CoT 整体流程如图 3.12 所示, 它使用两阶段方法来回答问题。首先, 在第一阶段, 在问题后面跟上一句“让我们一步一步思考”或者“Let's think step by step”来作为 CoT 的提示触发词, 来指示大语言模型先生成中间推理步骤, 再生成最后的答案。在第二阶段, 把原始的问题以及第一阶段生成的推理步骤拼接在一起, 在末尾加上一句“Therefore, the answer is”或者“因此, 最终答案为”, 把这些内容输给大语言模型, 让他输出最终的答案。通过这样的方式, 无需人工标注 CoT 数据, 即可激发大语言模型内在的推理能力。大语言模型能够逐步推理出正确的答案, 展现了 Zero-Shot CoT 在提升模型推理能力方面的潜力。

2. Auto CoT

在 Zero-Shot CoT 的基础之上, Auto-CoT [46] 引入与待解决问题相关的问题及其推理链作为示例, 以继续提升 CoT 的效果。相关示例的生成过程是由大语言模型自动完成的, 无需手工标注。Auto-CoT 的流程如图 3.13 所示, 其包含以下步骤:

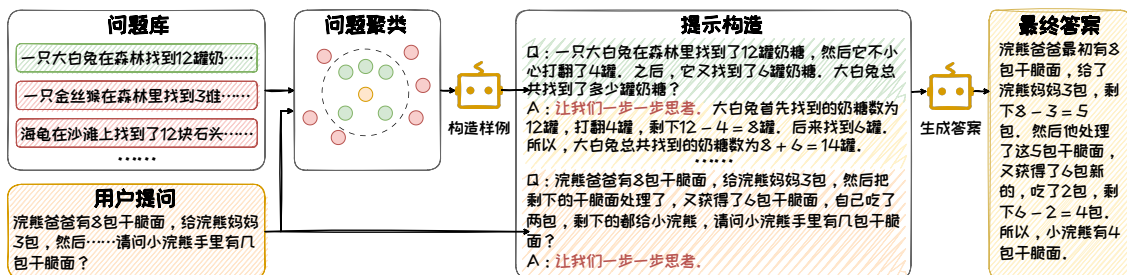


图 3.13: Auto-CoT 提示的流程。

- 利用聚类技术从问题库中筛选出与用户提问位于一个簇中的问题。
- 然后，借助 Zero-Shot CoT 的方式，为筛选出的问题生成推理链，形成示例。这些示例包含了不同问题及其对应的推理内容，可为模型提供不同解题思路，辅助模型做出更为审慎的推理。
- 在这些示例的基础上，Auto-CoT 以“让我们一步一步思考”引导大语言模型生成针对用户问题的推理链和答案。

3.3.3 三思后行

三思后行模式强调的是在决策过程中的融入**审慎和灵活性**。在这种模式下，模型在每一步都会停下来评估当前的情况，判断是否需要调整推理方向。这种模式的核心在于允许模型在遇到困难或不确定性时进行回溯和重新选择，确保决策过程的稳健性和适应性。其模仿了人类在解决问题时，会有一个反复选择回溯的过程。人们会不断从多个候选答案中选择最好的那个，并且如果一条思维路子走不通，就会回溯到最开始的地方，选择另一种思维路子进行下去。基于这种现实生活的观察，研究者在 CoT 的基础上提出了思维树（Tree of Thoughts, ToT）[42]、思维图（Graph of Thoughts, GoT）[1] 等三思后行模式下的 CoT 变体。

ToT 将推理过程构造为一棵思维树，其从以下四个角度对思维树进行构造。

- **拆解**。将复杂问题拆分成多个简单子问题，每个子问题的解答过程对应一个思维过程。思维过程拆解的形式和粒度依任务类别而定，例如在数学推理任务上以一行等式作为一个思维过程，在创意写作任务上以内容提纲作为思维过程。
- **衍生**。模型需要根据当前子问题生成可能的下一步推理方向。衍生有两种模式：样本启发和命令提示。样本启发以多个独立的示例作为上下文，增大衍生空间，适合于创意写作等思维空间宽泛的任务；命令提示在 Prompt 中指明规则和要求，限制衍生空间，适用于 24 点游戏等思维空间受限的任务。

- **评估。**利用模型评估推理节点合理性。根据任务是否便于量化评分，选择投票或打分模式。投票模式中，模型在多节点中选择，依据票数决定保留哪些节点；打分模式中，模型对节点进行评分，依据评分结果决定节点的保留。
- **搜索。**从一个或多个当前状态出发，搜索通往问题解决方案的路径。依据任务特点选择不同搜索算法。可以使用深度优先搜索、广度优先搜索等经典搜索算法，也可以使用 A* 搜索、蒙特卡洛树搜索等启发式搜索算法。

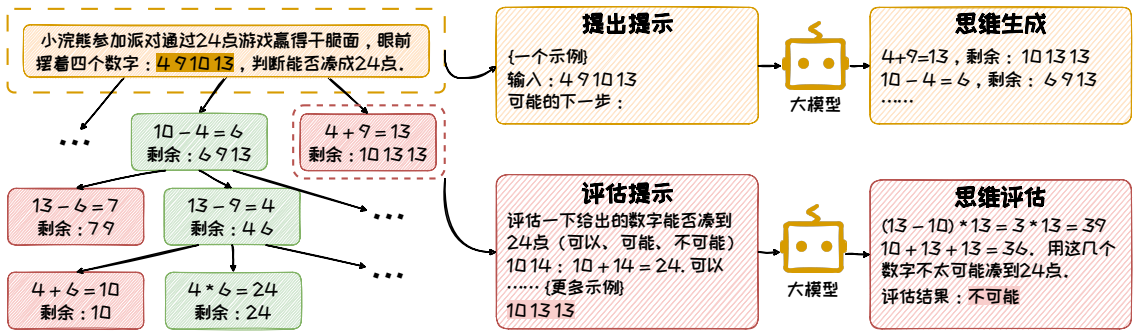


图 3.14: ToT 提示的流程。

图 3.14 通过 24 点游戏展示了一个 ToT 的具体例子。在这个例子中，给定 4 个数字，然后让大语言模型利用加减乘除 (+-*/) 四个运算符来组合这四个数字，使得最终的运算结果为 24。首先，ToT 基于当前所剩下的数字，通过上下文学习让模型选择两个数字作运算，并生成多个方案，在图上表现为思维树的多个子节点。之后以广度优先搜搜的方式遍历每一个子节点，评估当前剩余的数字是否能够凑到 24 点，保留可能凑出 24 点的节点，这一步也是通过上下文学习的方式来实现的。不断重复上述两个步骤，直到得出最终合理的结果。

在 ToT 的基础上，GoT 将树扩展为有向图，以提供了每个思维自我评估修正以及思维聚合的操作。该图中，顶点代表某个问题（初始问题、中间问题、最终问题）的一个解决方案，有向边代表使用“出节点”作为直接输入，构造出思维“入节点”的过程。GoT 相比于 ToT 的核心优势是其思维自我反思，以及思维聚合的能力，能够将来自不同思维路径的知识和信息进行集成，形成综合的解决方案。

3.3.4 集思广益

集思广益模式强调的是通过**汇集多种不同的观点和方法**来优化决策过程。在这种模式下，模型不仅仅依赖于单一的推理路径，而是通过探索多种可能的解决方案，从中选择最优的答案。这种方法借鉴了集体智慧的概念，即通过整合多个独立的思考结果，可以得到更全面、更准确的结论。在集体智慧的启发下，研究者在 CoT 的基础上探讨了如何通过自洽性来增强模型的推理能力，提出了 Self-Consistency [36] 方法。其引入多样性的推理路径并从中选择最一致的答案，从而提高了模型的推理准确性。Self-Consistency 不依赖于特定的 CoT 形式，可以与其他 CoT 方法兼容，共同作用于模型的推理过程。

如图 3.15 所示，Self-Consistency 的实现过程可以分为三个步骤：(1) 在随机采样策略下，使用 CoT 或 Zero-Shot CoT 的方式来引导大语言模型针对待解决问题生成一组多样化的推理路径；(2) 针对大语言模型生成的每个推理内容，收集其最终的答案，并统计每个答案在所有推理路径中出现的频率；(3) 选择出现频率最高的答案作为最终的、最一致的答案。

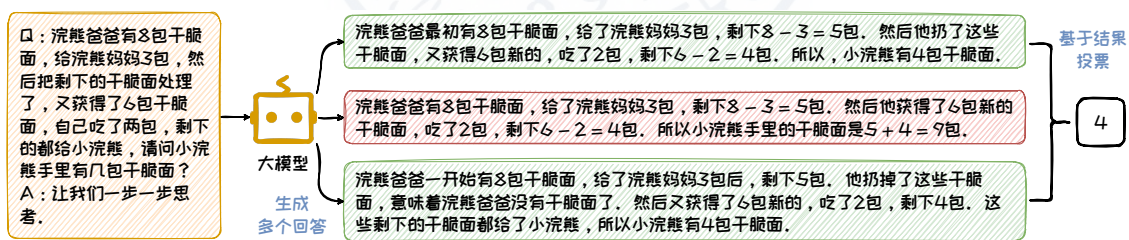


图 3.15: Self-Consistency 的流程。

本节探讨了思维链（Chain-of-Thought, CoT）的思想和模式。CoT 的核心思想是在提示中模拟人类解决问题的思考过程，在 Prompt 中嵌入解决问题的推理过程，从而在无需特定任务微调的情况下显著提升模型在推理任务上的表现。CoT 方法包含了多种模式：按部就班、按部就班以及集思广益。这些模式可以满足不同的推理需求，增强了大语言模型在复杂任务中的推理能力。

3.4 Prompt 技巧

基于上下文学习和思维链等 Prompt 工程技术，本节将进一步探讨可用于进一步提升大语言模型生成质量的 Prompt 技巧，包括合理归纳提问、适时运用思维链 (CoT) 以及巧妙运用心理暗示等。应用本节中介绍的 Prompt 技巧，可以引导模型生成更加精准、符合预期的内容，进一步提升大语言模型在实际应用中的表现。

3.4.1 规范 Prompt 编写

编写规范的 Prompt 是我们与大语言模型进行有效沟通的基础。经典的 Prompt 通常由任务说明，上下文，问题，输出格式等部分中的一个或几个组成。以图 3.16 中这个情感分类的 Prompt 为例。在这个例子中：

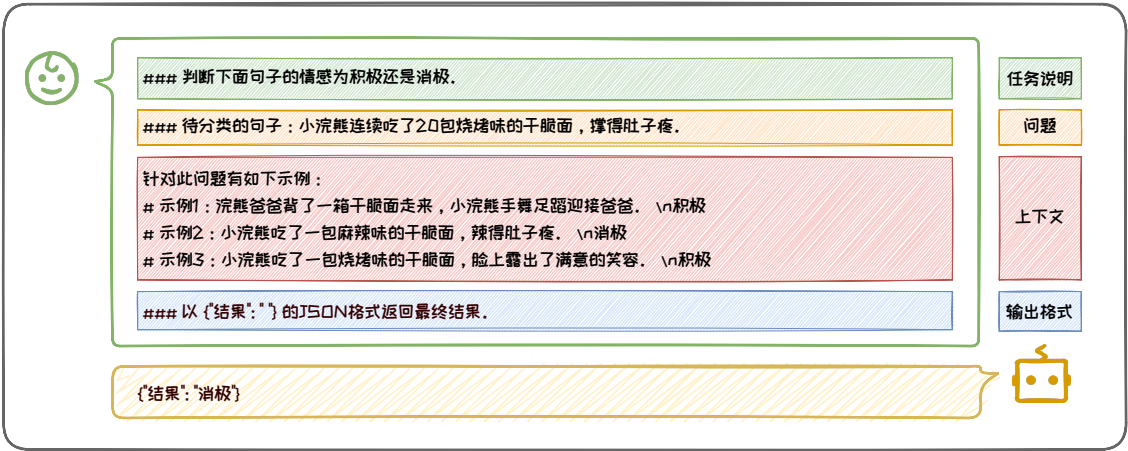


图 3.16: 经典的 Prompt 示例。

- **任务说明**是“### 判断下面句子的情感为积极还是消极。”，它明确了模型需要完成的任务；
- **上下文**是“针对此问题有如下示例：# 示例 1：浣熊爸爸背了一箱干脆面走来，小浣熊手舞足蹈迎接爸爸。 \n积极 \n# 示例 2：小浣熊吃了一包麻辣味的干脆面，辣得肚子疼。 \n消极 \n# 示例 3：小浣熊吃了一包烧辣味的干脆

面，脸上露出了满意的微笑。\\n 积极 \\n”。上下文提供了帮助模型理解和回答问题的示例或背景信息；

- **问题**是“待分类的句子：小浣熊连续吃了 20 包烧烤味的干脆面，撑得肚子疼。”，是用户真正想要模型解决的问题，它可以是一个段落（比如摘要总结任务中被总结的段落），也可以是一个实际的问题（比如问答任务中用户的问题），或者表格等其他类型的输入内容；
- **输出格式**是“以”结果”：“”的 JSON 格式返回最终结果。”，它规范了模型的输出格式。

通过这个例子可以看出，在编写经典 Prompt 的过程中，Prompt 各个组成部分都很重要，它们的规范性，直接影响模型的输出质量。同时，各个组成部分的排版也很重要。接下来，我们将详细介绍经典 Prompt 的规范编写需要满足的要求。

1. 任务说明要明确

明确的任务说明是构建有效 Prompt 的关键要素之一。一个清晰、具体的任务说明能够确保模型准确理解任务要求，并产生符合预期的输出。例如，在情感分类任务中，任务说明“判断下面句子的情感为积极还是消极。”就是一个明确的示例，它清晰地定义了任务类型（情感分类）和分类的具体类别（积极或消极）。相反，模糊或不明确的任务说明可能导致模型误解用户的真实意图，从而产生不符合预期的输出。如图 3.17 所示，“分类下面的句子”这样的任务说明就缺乏具体性，没有明确指出分类的类型和类别，使得模型难以准确执行任务。

为了确保任务说明的明确性，我们需要明确以下几个要点：

- **使用明确的动词**：选择能够清晰表达动作的动词，如“判断”、“分类”、“生成”等，避免使用模糊的动词如“处理”或“操作”。
- **具体的名词**：使用具体的名词来定义任务的输出或目标，例如“积极”和“消极”在情感分类任务中提供了明确的分类标准。

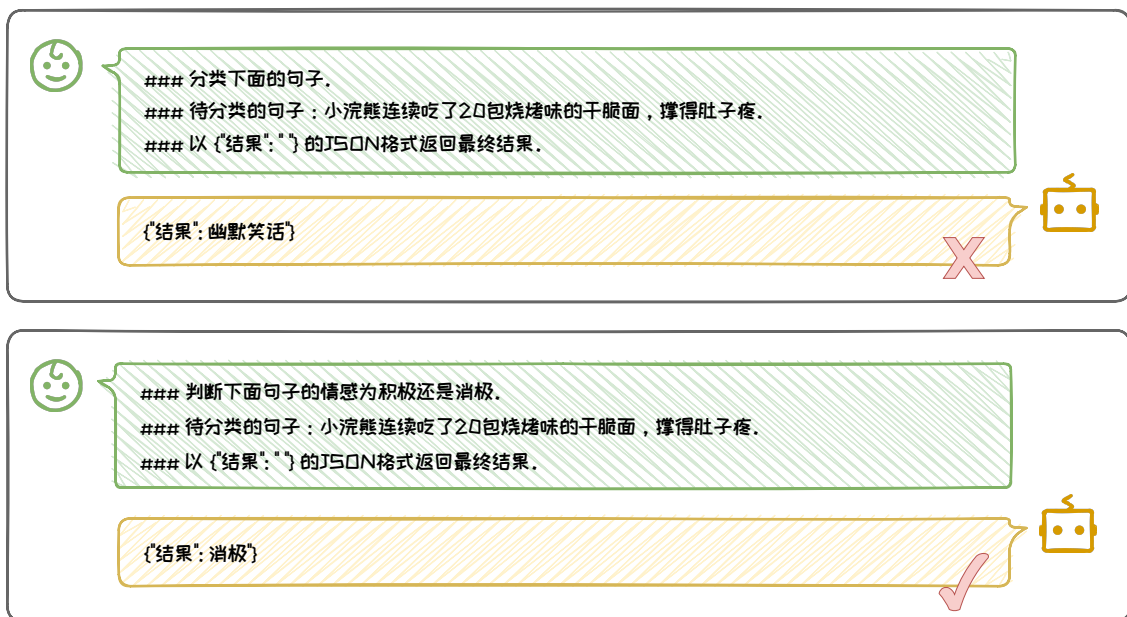


图 3.17: 不同的任务说明对比。

- **简洁明了：**任务说明应简洁且直接，避免冗长或复杂的句子结构，使模型能够快速抓住任务的核心要求。
- **结构化布局：**在较长的 Prompt 中，将任务说明放置在开头和结尾，因为模型通常更关注这些部分的信息 [18]。这种布局有助于确保模型首先和最后接触到的是最关键的任务信息。

通过这些策略，我们可以确保任务说明既清晰又具体，从而帮助模型更好地理解 and 执行任务，最终产生高质量的输出。

2. 上下文丰富且清晰

在 Prompt 设计中，上下文的作用不容忽视，它有时直接决定了模型能否给出正确的答案。一个**丰富且清晰**的上下文能够显著提升模型的理解和回答准确率。上下文的**丰富性**体现在其内容的多样性和相关性。上下文可以包括与问题直接相关的背景信息、具体的演示示例，或是对话的连续性内容。例如，在情感分类任务中，提供具体的示例句子及其对应的情感标签，可以帮助模型更好地理解任务的

具体要求和预期的输出。上下文的清晰性则要求上下文信息必须与问题紧密相关，避免包含冗余或不必要的信息。清晰的上下文应直接指向任务的核心，减少模型在处理信息时的混淆和误解。例如，在问答任务中，上下文应仅包含与问题直接相关的信息，避免引入可能误导模型的无关内容。

在图 3.18 两个上下文设计的例子中，第一个例子的上下文紧密围绕问题，提供了丰富的直接相关信息，没有任何冗余内容。这种设计有助于模型迅速聚焦于关键信息，从而准确回答问题。相比之下，第二个例子的上下文不够丰富，并且单个例子则包含了大量与问题无关的细节，这些冗余信息不仅使上下文显得不明确，还可能加重模型处理信息的负担，导致模型难以准确把握问题的核心，进而影响其回答的准确性。

Figure 3.18 illustrates two different context designs for a sentiment classification task. The top example shows a concise and relevant context that leads to a correct answer, while the bottom example shows a verbose context with irrelevant details that leads to an incorrect answer.

Example 1 (Top):

判断下面句子的情感为积极还是消极。
针对此问题有如下示例：
待分类的句子：小浣熊连续吃了20包最爱的烧味味的干腿面，撑得肚子疼，脸色憔悴。
示例1：浣熊爸爸背了一箱干腿面走来，小浣熊手舞足蹈，非常积极的迎接爸爸。 \n积极
示例2：小浣熊吃了一包烧味味的干腿面，脸上露出了满意的笑容。 \n积极
示例3：小浣熊吃了一包麻辣味的干腿面，味道很棒，但是辣得肚子疼，它感到非常难受。 \n消极
以{"结果": " "} 的JSON格式返回最终结果。

{ "结果": "消极" }

Example 2 (Bottom):

判断下面句子的情感为积极还是消极的。
待分类的句子：小浣熊连续吃了20包最爱的烧味味的干腿面，撑得肚子疼，脸色憔悴。
针对此问题有如下示例：
示例：小浣熊爸爸给小浣熊带来了一箱小浣熊最爱吃的干腿面，小浣熊兴高采烈地去迎接小浣熊爸爸。爸爸给了小浣熊一包麻辣味的干腿面，小浣熊对麻辣味的食物不太适应，但是它看出麻辣味干腿面包装很漂亮，忍不住吃了一包。麻辣味干腿面很好吃，小浣熊脸上露出了幸福的笑容。但是没过一会，它的肚子就开始疼起来了，嘴巴也火辣辣的，脸色变得苍白。 \n小浣熊高兴地迎接爸爸，并且吃了干腿面很幸福，尽管吃了面之后肚子疼，但是总体而言情感还是积极的。
以{"结果": " "} 的JSON形式返回最终结果。

{ "结果": "虽然小浣熊肚子疼，但是它吃到了最爱的烧味干腿面，所以总体而言情感是积极的" }

图 3.18: 不同的上下文对比。

3. 输出格式要规范

规范的输出格式对于确保模型输出的可用性和准确性至关重要。通过指定明确的输出格式，可以使模型的**输出结构化**，便于下游任务直接提取和使用生成内容。常用的输出格式包括 JSON、XML、HTML、Markdown 和 CSV 等，每种格式都有其特定的用途和优势。

例如，在图 3.19 中的 Prompt 例子中，“以 {”结果”: ”}”的 JSON 格式返回最终答案。”明确指定了答案应以 JSON 格式输出，并且以一个简短的例子指名 JSON 中的关键字。这种规范的输出格式不仅使得结果易于解析和处理，还提高了模型输出的准确性和一致性。如果不明确规定输出格式，模型可能会输出非结构化或不规范的结果，这会增加后续处理的复杂性。在第二个例子中，如果模型输出的答案是一个自由格式的文本字符串，那么提取具体信息就需要进行复杂的字符串解析，而不是像 JSON 等结构化格式那样可以直接提取，这就给后续对于结果的处理与使用带来了麻烦。

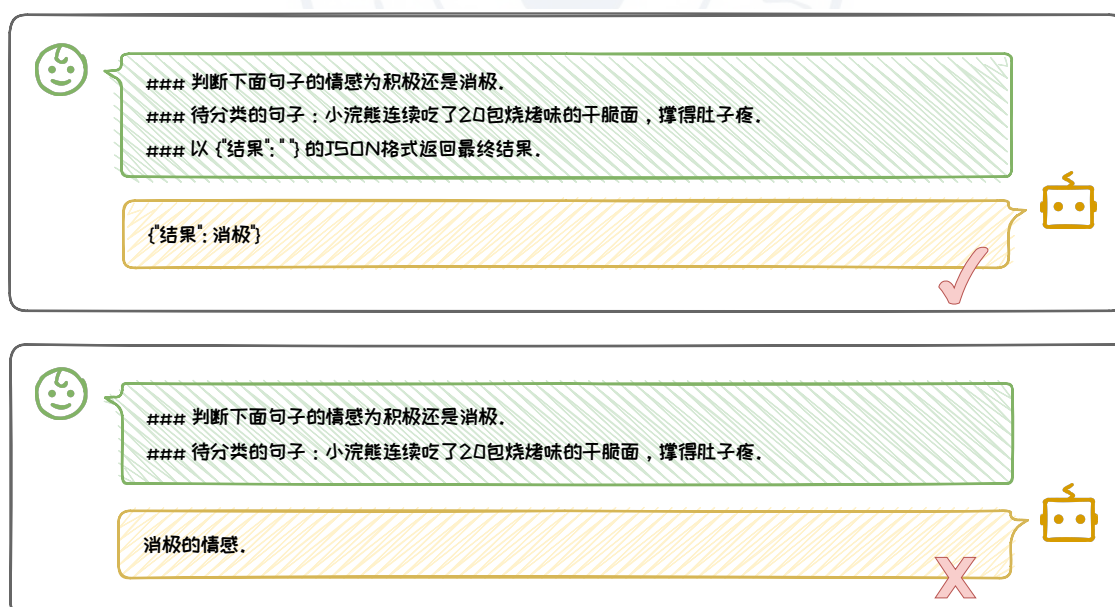


图 3.19: 不同输出格式对比。

为了确保输出格式的规范性，可以采取以下措施：

- **明确指定输出格式**：在 Prompt 中明确指出希望模型使用的输出格式，如“请以 JSON 格式返回结果”，并且选择广泛接受和易于处理的输出格式，如 JSON、CSV 等，易于解析和数据交换。
- **提供输出格式的示例**：在 Prompt 中提供一个输出格式的具体示例，比如在 JSON 中明确指出关键字，帮助模型理解预期的输出结构。

这些措施可以令模型的输出既规范又易于处理，从而提高整个系统的效率和准确性。规范的输出格式不仅简化了数据处理流程，还增强了模型输出的可靠性和一致性，为用户提供了更加流畅和高效的交互体验。

4. 排版要清晰

一个优秀的 Prompt 还必然具备清晰的排版，这对于模型的理解 Prompt 至关重要。清晰的排版有助于模型准确捕捉任务的关键信息，从而提高其执行任务的准确性和效率。相反，复杂的排版可能会导致信息模糊，使模型难以准确理解任务的具体要求 [7]，进而影响输出结果的质量。清晰的排版通常涉及**使用合适的分隔符和格式化技巧**，将 Prompt 的不同组成部分（如任务说明、上下文、问题和输出格式）明确区分开来。在图 3.16 所示的例子中，我们使用“#”和“###”以及换行符有效地将各个部分分隔开，使得每个部分的内容清晰可见，便于模型理解和处理。相反，如果排版混乱，例如在 Prompt 中混合使用不使用任何分隔符，模型可能会混淆不同部分的内容，导致无法准确执行任务。例如，去除图 3.16 中的 Prompt 例子的排版符号与样式，将图 3.20 所示的复杂混乱的 Prompt 输入给大语言模型，模型回答的质量将大幅度下降。

为了确保 Prompt 的排版清晰，可以采取以下措施：

- **使用一致的分隔符**：选择并坚持使用一种或几种分隔符（如“#”、“###”、“—”等），以区分不同的 Prompt 部分。

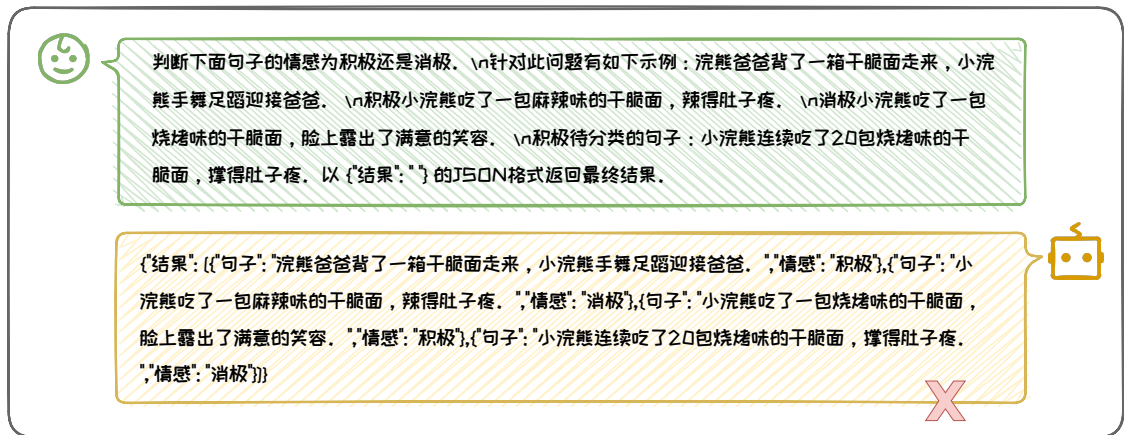


图 3.20: 不清晰的排版。

- **合理使用空白和缩进：**通过增加空白行和适当的缩进，增强 Prompt 的可读性，帮助模型区分不同的内容块。
- **清晰的标题和子标题：**为每个部分提供清晰的标题或子标题，使模型能够快速识别每个部分的主题。

通过这些措施，我们构造既清晰又易于理解的 Prompt 的排版，从而帮助模型更好地执行任务，提升信息处理的效率和准确性。

3.4.2 合理归纳提问

在与大语言模型的交互中，提问的质量直接影响到信息触达的效率和深度。一个精心设计的提问不仅能够明确表达需求，还能引导模型聚焦于问题的核心，从而获得精准且有价值的回答。本节将探索如何通过“合理归纳提问”来提升交互的质量。具体的，我们将重点介绍两个高级提问策略：“复杂问题拆解”和“追问”。这两个策略是提问艺术的精髓，它们不仅帮助我们深入理解问题的本质，还显著提高了我们获取信息和解决问题的效率。通过这些策略，我们能够更精确地引导模型，确保其输出既准确又全面，从而满足我们的需求。

1. 复杂问题拆解

在处理复杂问题时，我们可以将问题分解为**更小、更易于理解**的子问题，并**逐一解决**。在计算机算法设计中，这种策略被称为“分而治之”，其基本理念是通过逐一解决子问题，最终达成解决整个问题的目标。在与大语言模型进行交互时，同样可以借鉴“分而治之”的思想，将复杂问题分解为一系列子问题，并引导模型逐一回答，以确保每个子问题都得到充分考虑和解答。随后，将这些子问题的答案汇总，形成最终的全面回答。这一过程包括两个关键步骤：**分步引导**和**归纳总结**。

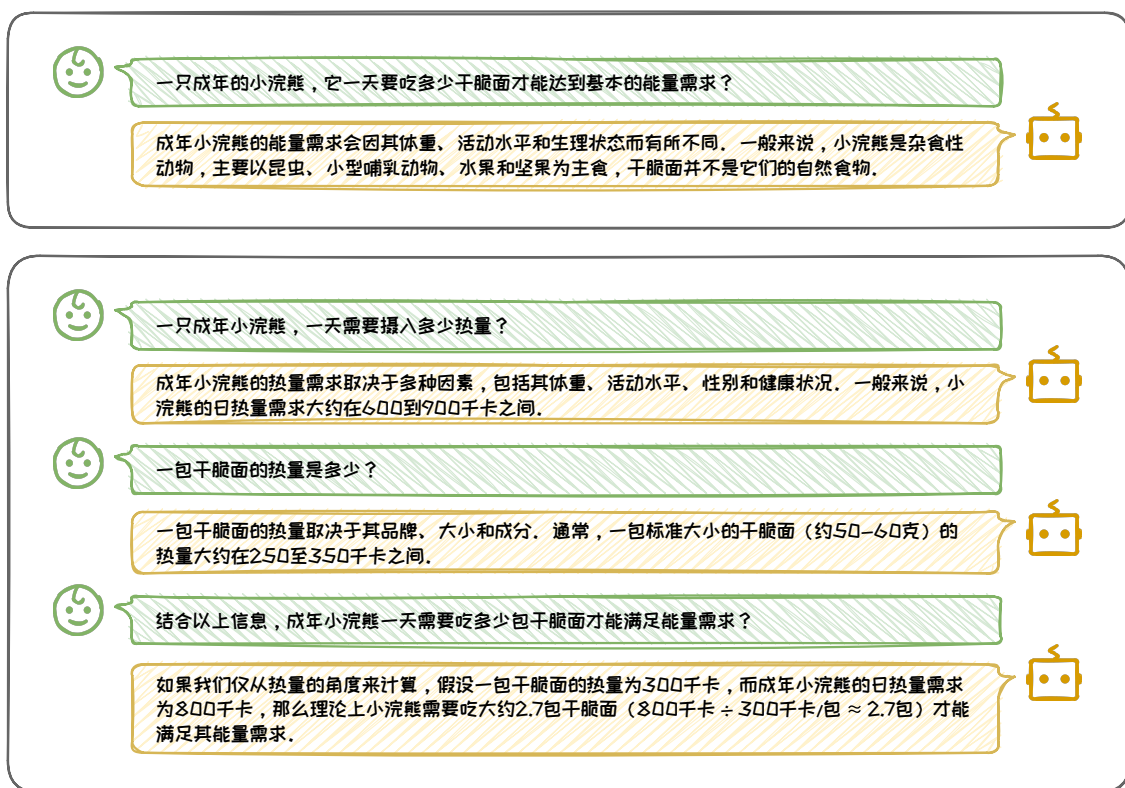


图 3.21: 复杂问题拆解例子对比。

首先，在**分步引导**阶段，我们需将复杂问题细化为多个子问题，并引导模型针对每个子问题进行深入分析和回答。这一步骤旨在确保每个子问题都能得到详尽的解答，从而为后续的归纳总结奠定坚实基础。其次，在**归纳总结**阶段，我们将各个子问题的答案进行汇总，并综合形成最终的全面回答。这一步骤不仅有助于我们全面把握问题的各个方面，还能确保最终答案的准确性和完整性。

如图 3.21 所示。用户提出了一个关于成年小浣熊一天需要吃多少干脆面才能满足能量需求的问题。通过分步引导,我们将这个问题分解为两个关键的小问题:“一只成年小浣熊,一天需要摄入多少热量?”和“一包干脆面的热量是多少?”。模型分别回答了这两个问题,提供了小浣熊的日热量需求和干脆面的热量含量。随后,通过归纳总结的提示“结合以上信息,成年小浣熊一天需要吃多少包干脆面才能满足能量需求?”,我们将这些分散的信息整合起来,计算出小浣熊需要摄入的干脆面数量。这一过程不仅展示了如何通过逐步提问来引导模型提供详细信息,还强调了在解决复杂问题时,系统地分解问题和整合答案的重要性。

这种方法的优势在于它能够帮助用户和模型更有效地处理复杂信息,确保每个细节都被考虑到,并最终形成一个准确和全面的答案。通过分步引导和归纳总结,我们能够有序地解决复杂问题,提供高质量的解答。

2. 追问

通过在对话中进行追问,用户可以引导大语言模型的输出更贴合心意的内容。这种对话形式的交互不仅可以促进更深层次的理解和更为丰富的讨论,而且有助于更精确地表达用户的真实想法,从而更好地指导模型的思考,使其输出更加贴合用户需求。从追问的形式和目的角度来看,追问可以分为三种形式: **深入追问**、**扩展追问**、**反馈追问**。

(1) 深入追问

深入追问的形式是指用户可以根据大语言模型的输出继续发问来深入挖掘**特定话题的深层信息**。这种追问适用于需要对某个概念、现象或过程有详尽解释的场景。通过深入追问,用户能够引导模型提供更为细致和深入的信息,从而加深对特定领域的理解。以图 3.22 为例,用户首先询问了小浣熊是否可以食用干脆面。模型基于干脆面的成分给出了初步的回答,指出其对小浣熊的健康可能有害。随后,用户通过深入追问,探讨了如果去除干脆面中的调味料和添加剂,是否可以安全

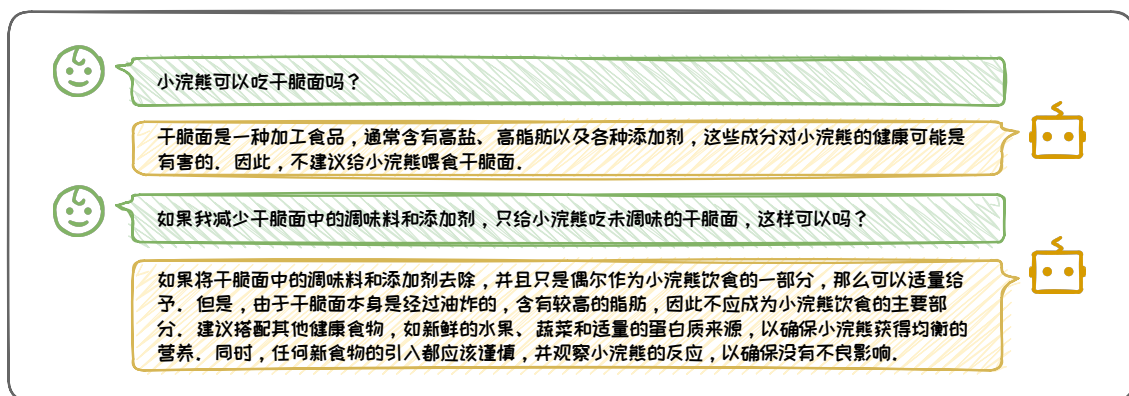


图 3.22: 深入追问示例。

地喂食小浣熊。模型进一步解释了即使去除调味料和添加剂，干脆面仍含有较高的脂肪，因此不应成为小浣熊饮食的主要部分，并建议了如何搭配其他食物以确保营养均衡。这一系列的追问展示了用户如何通过连续提问引导模型提供更为深入的信息。

(2) 扩展追问

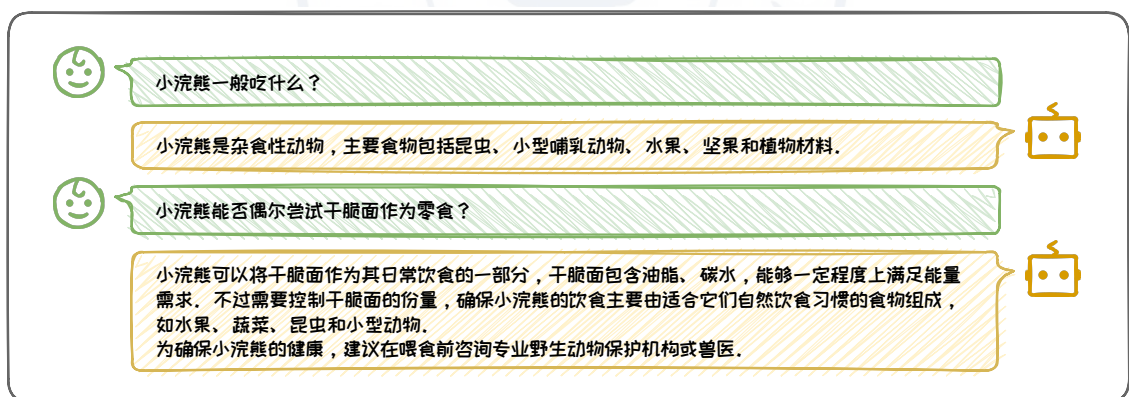


图 3.23: 扩展追问示例。

扩展追问是一种在大语言模型给出回答的基础上，进一步要求模型提供更多相关信息或例子的提问方式，其目的在于**拓宽讨论的广度**，收集更多数据、例证或选项，帮助用户获得更广泛的视角，增加对话题的理解。这种追问特别适用于需要全面了解一个主题的场景。在图 3.23 这个例子中，用户首先询问了小浣熊的常规

饮食，模型提供了小浣熊作为杂食性动物的饮食概况。随后，用户通过扩展追问，探讨了小浣熊是否可以食用干脆面。模型进一步解释了干脆面可以作为小浣熊饮食的一部分，但强调了控制份量和保持饮食多样性的重要性。这一系列的追问不仅展示了用户如何通过提问获取更多关于小浣熊饮食的信息，还强调了在引入新食物时咨询专业意见的必要性。通过扩展追问，用户能够获得更全面的视角。

(3) 反馈追问

反馈追问的形式是在大语言模型的输出不符合预期或存在错误时，提供反馈，指出问题所在，并请求模型进行更正或澄清。其目的在于通过**反馈机制**提升模型的准确性，确保信息的正确性。这种追问允许用户指出模型输出中的具体错误或不足，并请求模型对其进行修正，有助于提高对话质量。在图 3.24 这个例子中，用户首先询问了小浣熊连续食用 20 包干脆面后的应对措施。模型最初的回答建议小浣熊可以适量食用干脆面，但用户通过反馈追问指出了小浣熊出现了不良反应。模型随后修正了回答，指出大量食用干脆面对小浣熊的健康有害，并提供了紧急处理建议。通过反馈追问，用户能够获得更加准确可靠的信息。

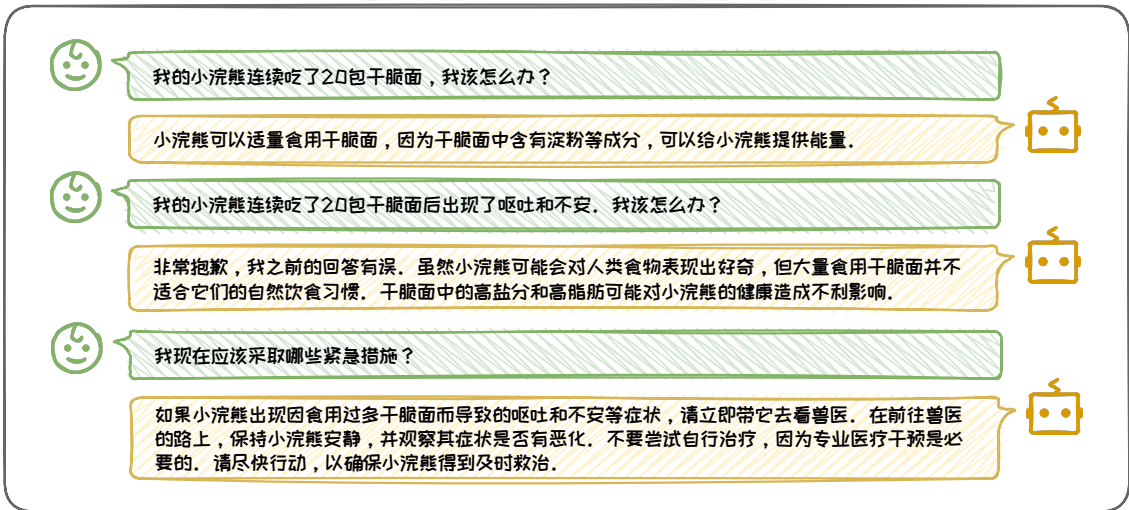


图 3.24: 反馈追问示例。

3.4.3 适时使用 CoT

思维链技术 (Chain of Thought, CoT) [38] 是在处理涉及算术、常识和符号推理等复杂推理的任务时的理想选择。在处理这类任务的过程中, 通过 CoT 引导模型理解和遵循中间步骤, 能够显著提高得出答案的准确率。3.3 节中已经讨论过 CoT 几种经典范式。本节将讨论在何时使用 CoT 以及如何使用 CoT。

1. 何时使用 CoT

在决定何时使用 CoT 时, 需要对**任务类别**、**模型规模**以及**模型能力**三方面因素进行考虑。

在**任务类别**方面, CoT 技术特别适用于需要复杂推理的任务, 如算术、常识和符号推理。在这些任务上, CoT 能够引导大语言模型生成逻辑严密、条理清晰的中间推理步骤, 从而提高正确答案的生成概率, 如图 3.25 所示。然而, 对于情感分类、常识问答等简单问题, 标准的 Prompt 方法已足够有效, 使用 CoT 可能难以提升效果, 反而可能引入不必要的复杂性。

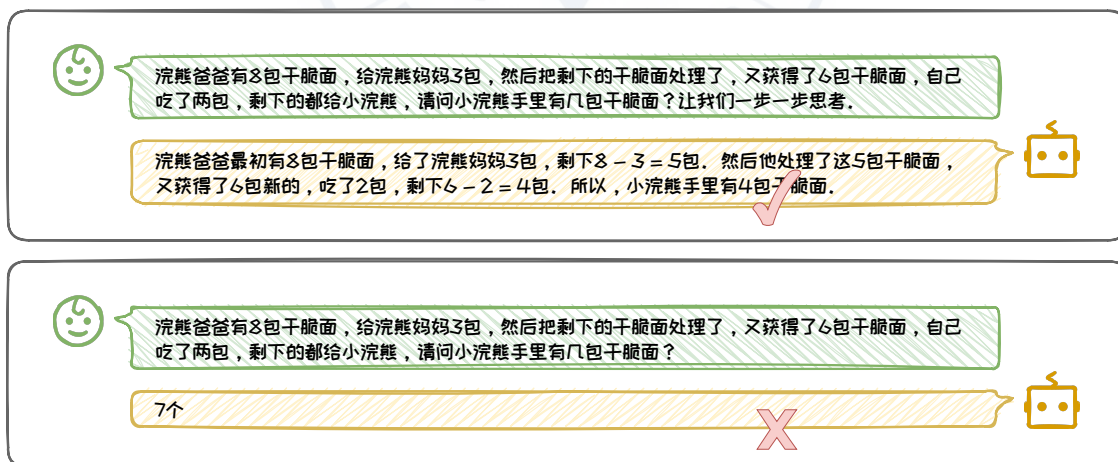


图 3.25: 在推理任务上使用 CoT。

在**模型规模**的考量上, CoT 技术应用于参数量超过千亿的巨型模型时, 能够显著提升其性能, 例如, PaLM [5] 模型和 GPT-3 [3] 模型等模型。然而, 在规模较小

的模型上应用 CoT 技术可能会遭遇挑战，如生成逻辑不连贯的思维链，或导致最终结果的准确性不如直接的标准提示方法，如图 3.26 所示。

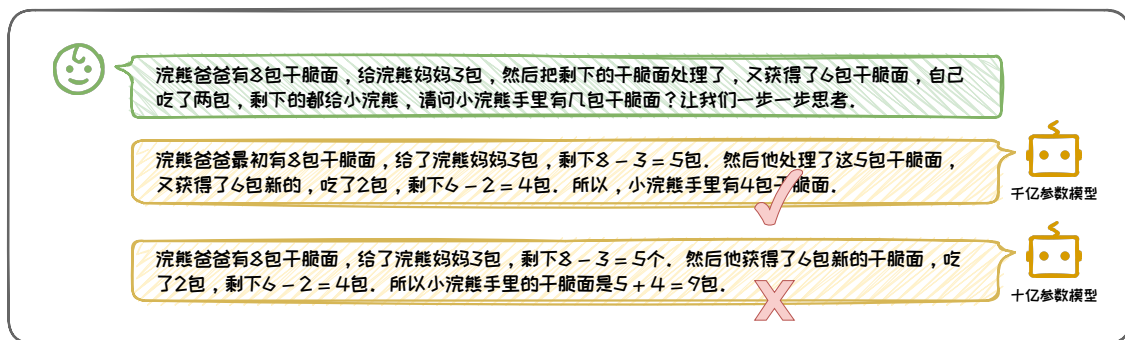


图 3.26: 不同规模模型使用 CoT 对比。

在模型能力的考量上，CoT 是否起效与模型在预训练阶段是否进行过推理方面的指令微调有关。对于那些未经推理方面的指令微调的大语言模型，如早期的 GPT-3、Palm 以及目前开源的基础版本模型，例如 LLaMA2-13B-Base 和 Baichuan2-13B-Base，适当的 CoT 提示能够激发其卓越的 CoT 推理能力；对于已经历过推理方面的指令微调的大语言模型，如 ChatGPT、GPT-4 以及 LLaMA2-13B-Chat 等，即便在没有 CoT 指令的情况下，也能自发生成条理清晰的中间推理步骤。在许多情况下，这些模型在没有 CoT 指令的条件下反而展现出更佳的性能，这表明它们在指令微调过程中可能已经内化了 CoT 指令，使得即便在没有明确 CoT 提示时，仍能隐式遵循 CoT 推理路径。


2. 灵活使用 CoT

灵活使用 CoT 的关键在于根据任务的具体需求和模型的特性来调整 CoT 的使用方式。主要涉及调整 CoT 的详细程度以及使用不同的 CoT 形式两个方面。

- **调整 CoT 的详细程度**：我们可以指定 CoT 输出的详细程度，以适应不同的用户需求，如图 3.27 所示。对于简单的计算问题，在用户不需要推理的中间过程时，我们可以直接给出最终乘法和加法的结果。而对于复杂的计算、推理


问题，或者用户需要理解中间推理过程时，我们需要通过样例进行引导，以使其展示完整的推理步骤。

- **使用不同的 CoT 形式：**我们可以根据不同任务场景，选择不同的 CoT 形式。在不需要特定领域知识，仅需对问题进行逻辑推理和逐步分析时，可以使用 Zero-Shot CoT 或者 Auto CoT 的方式，通过“让我们一步一步思考”这种 CoT 提示触发词，来引导模型以 CoT 的形式回答内容。在处理需要高准确度和可靠性的任务时，可要求模型生成多个回答并提出最终结果，进而运用 Self-Consistency 方法筛选出一致性最强的答案。例如，在编写代码时，模型可生成多个版本，通过 Self-Consistency 方法确保最终选定的代码在逻辑上最为一致。对于包含创意思维的任务，可选用 ToT 和 GoT 方法来在多个可能的思维路径中进行探索和选择。例如，在创作故事时，模型可利用 ToT 或 GoT 探索不同的情节发展路径，并选取最富有趣味性或合理性的发展方向。




示例：金丝猴爸爸得到了10颗奶糖，他给了金丝猴妈妈4颗奶糖，然后把剩下的奶糖分给了其他金丝猴。之后，他又得到了7颗奶糖，自己吃了3颗，剩下的都给了小金丝猴。请问小金丝猴手里有几颗奶糖？回答：(7-3=4) 颗奶糖。

浣熊爸爸有8包干脆面，给浣熊妈妈3包，然后把剩下的干脆面处理了，又获得了6包干脆面，自己吃了两包，剩下的都给小浣熊，请问小浣熊手里有几包干脆面？




(6-2=4) 包干脆面。



金丝猴爸爸得到了10颗奶糖，给金丝猴妈妈4颗奶糖，然后把剩下的奶糖分给了其他金丝猴，又得到了7颗奶糖，自己吃了3颗，剩下的都给小金丝猴。请问小金丝猴手里有几颗奶糖？回答：金丝猴爸爸给金丝猴妈妈后剩下(10-4=6)颗，分给其他金丝猴后没有剩余。再得到7颗奶糖，吃了3颗后剩下(7-3=4)颗，所以小金丝猴手里有4颗奶糖。

浣熊爸爸有8包干脆面，给浣熊妈妈3包，然后把剩下的干脆面处理了，又获得了6包干脆面，自己吃了两包，剩下的都给小浣熊，请问小浣熊手里有几包干脆面？



浣熊爸爸给浣熊妈妈后剩下(8-3=5)包，处理后没有剩余。再获得6包干脆面，吃了2包后剩下(6-2=4)包，所以小浣熊手里有4包干脆面。

图 3.27: 通过 Few-Shot 示例指定 CoT 输出的形式和风格。

3.4.4 善用心理暗示

在硅谷，流传着一句创业金句，“**Fake it till you make it**”（假装它直到你成功）。这句话具体含义为，先吹嘘你的想法，进而吸引资本和人才，最终在实践中努力追赶并实现既定目标。这句话源自一种积极的心理暗示方法：**通过模仿自信和乐观的心态，一个人可以在他们的现实生活中实现这些品质**。这种现象不仅局限于人类的行为，积极的心理暗示也可用于激发大语言模型的潜力。这种心理暗示可以通过角色扮演和情景代入的方式传达给大语言模型。

1. 角色扮演

通过 Prompt 指导大语言模型扮演特定角色能够显著改善其与角色相关的技能。这种技术被称为角色扮演（Role-Playing），它可使大语言模型能够生成更为准确、角色相关的内容。通过为大语言模型设定一个详尽的角色，例如数据科学家、诗人或律师，可以有效地引导模型的输出朝向所需的方向，从而提供更高质量的回答。为了构建一个有效的角色，需要在指令中**包含具体属性、职责、知识和技能**。在设计角色设定提示时，选择对特定任务具有明显优势的角色至关重要。通过

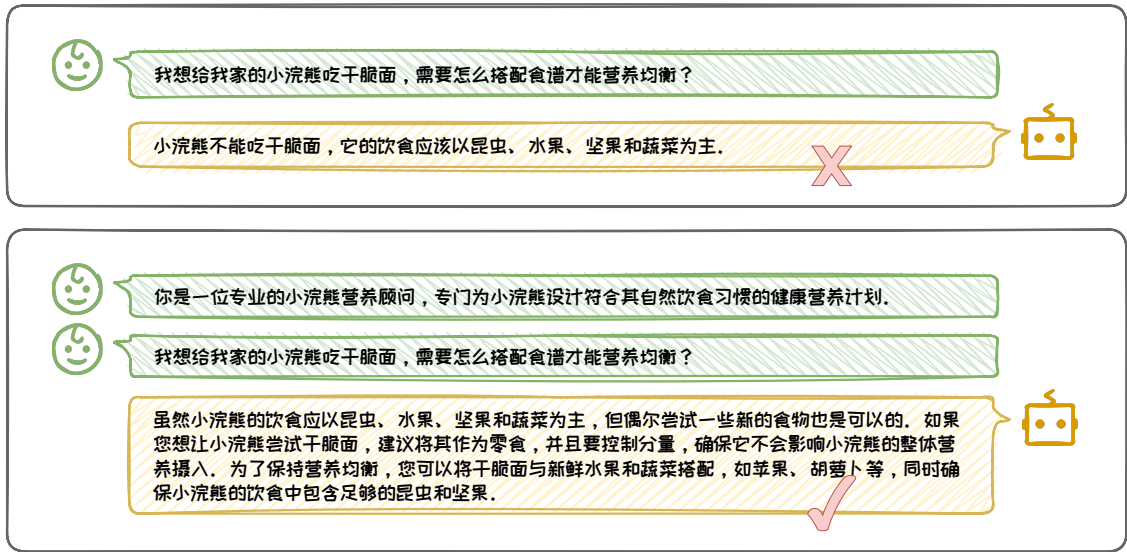


图 3.28: 角色扮演示例。

额外的描述来强调这一优势，通常可以带来更好的结果 [13]。

如图 3.28 所示，通过角色扮演，模型承担了“专业的小浣熊营养顾问”这一角色，展现出了对小浣熊饮食需求的专业知识。在回答用户关于小浣熊尝试干脆面的询问时，模型不仅坚持了小浣熊应以昆虫、水果、坚果和蔬菜为主的自然饮食原则，还巧妙地提出了将干脆面作为零食的建议，并强调了控制分量的重要性。此外，模型还建议将干脆面与新鲜水果和蔬菜搭配，确保小浣熊的饮食中包含足够的昆虫和坚果，以维持营养均衡。这一建议既科学又实用，充分展现了角色扮演在提升模型交互质量和适应用户需求方面的巨大潜力。

2. 情景代入

在日常生活中，一个人容易受到周围环境和交往人群的影响，可能会逐渐采纳周围人的行为和思维方式。这种情景代入现象不仅适用于人，也同样适用于大语言模型。通过将模型置于特定的“情景”或“环境”中，可以影响其生成的文本内容和风格。情景代入指的是将特定情境下所需的专业知识、历史背景等信息嵌入到模型的响应中。

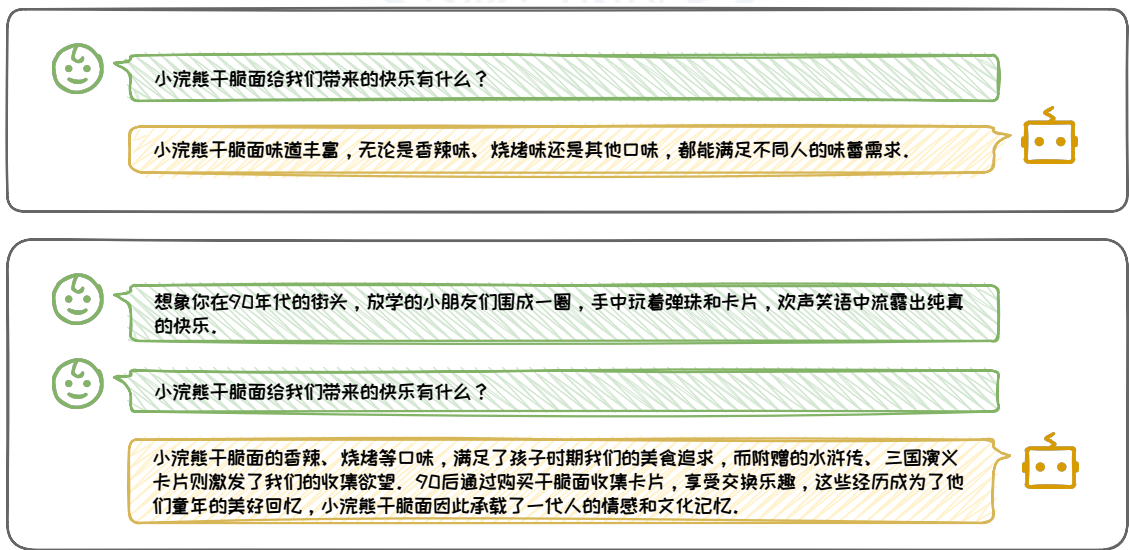


图 3.29: 情景代入示例。

通过情景代入，模型能够深入理解并反映特定情境下的文化社会背景与现实环境，从而生成更加丰富和有深度的回答。在图 3.29 例子中，当模型被置于 90 年代的街头情景中，它不仅能够描述小浣熊干脆面的口味，还能够捕捉到那个时代特有的文化现象——通过收集卡片来交换乐趣。这种回答不仅提供了具体的信息，还唤起了用户的情感共鸣，增强了交互的情感连接。

在本节中，我们深入探讨了提升 Prompt 技巧的多种策略，以增强大语言模型的交互效率和输出质量。这些技巧主要包括规范 Prompt 编写、合理归纳提问、适时使用思维链、以及善用心理暗示。这些技巧和策略的应用，不仅可以提升了提示的有效性，使得模型能够更准确地理解和回应用户的需求，还显著提高了大语言模型在复杂任务中的表现。

3.5 相关应用

Prompt 工程的应用极为广泛，几乎涵盖了所有需要与大语言模型进行高效交互的场景。这项技术不仅能够帮助我们处理一些基础任务，还能显著提升大语言模型在应对复杂任务时的表现。Prompt 工程在构建 Agent 完成复杂任务、进行数据合成、Text-to-SQL 转换，以及设计个性化的 GPTs 等方面，发挥着不可或缺的作用。下面我们将依次介绍 Prompt 工程在这些应用场景中的具体作用。

3.5.1 基于大语言模型的 Agent

智能体（Agent）是一种能够自主感知环境并采取行动以实现特定目标的实体 [35]。作为实现通用人工智能（AGI）的有力手段，Agent 被期望能够完成各种复杂任务，并在多样化环境中表现出类人智能。然而，以往的 Agent 通常依赖简单的启发式策略函数，在孤立且受限的环境中进行学习和操作，这种方法难以复制人类水平的决策过程，限制了 Agent 的能力和应用范围。近年来，大语言模型的不

发展，涌现出各种能力，为 Agent 研究带来了新的机遇。基于大语言模型的 Agent（以下统称 Agent）展现出了强大的决策能力。其具备全面的通用知识，可以在缺乏、训练数据的情况下，也能进行规划、决策、工具调用等复杂的行动。

Prompt 工程技术在 Agent 中起到了重要的作用。在 Agent 系统中，大语言模型作为核心控制器，能够完成规划、决策、行动等操作，这些操作很多都依赖 Prompt 完成。图 3.30 展示了一个经典的 Agent 框架，该框架主要由四大部分组成：配置模块（Profile）、记忆模块（Memory）、计划模块（Planning）和行动模块（Action）[35]。Prompt 工程技术贯穿整个 Agent 流程，为每个模块提供支持。

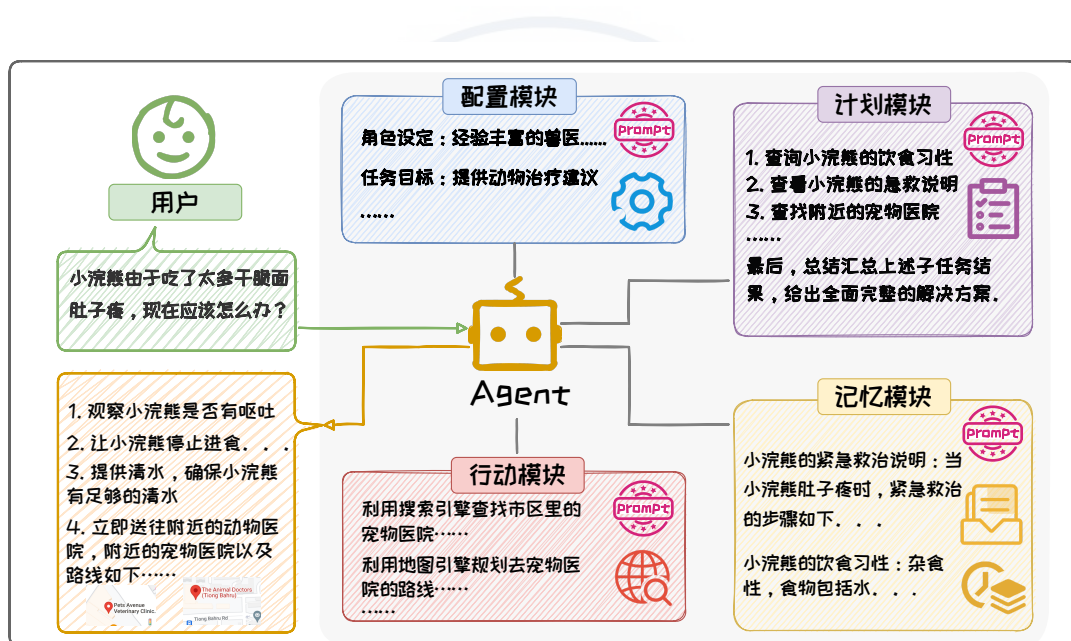


图 3.30: 基于大语言模型的 Agent 框架流程示意图。

在 Agent 中，上述四个组件各司其职，分工协作，共同完成复杂任务：（1）**配置模块**利用 Prompt 工程中的角色扮演技术，来定义 Agent 的角色。设定 Agent 的背景、技能、职责等信息，这些角色设定信息以上下文的形式嵌入到 Agent 每一次交互的 Prompt 中。（2）**记忆模块**是 Agent 的知识与交互记忆的存储中心。记忆模块通过检索增强等技术获取记忆，这一过程涉及到使用 Prompt 工程中的上下文学

习技术来构造和优化查询，从而帮助更加精准检索到相关记忆。在获取记忆之后，将这些记忆将被添加到交互的 Prompt 中，帮助 Agent 利用这些记忆知识，实现更为准确高效的决策与行动。(3) **计划模块**则扮演着任务分解者的角色，它将复杂的任务细化为一系列更为简单、易于管理的子任务。在这一过程中，通过 Prompt 工程中的思维链技术，让大语言模型分解任务并进行规划，按照链式顺序输出子任务；同时还利用了上下文学习技术，构造少样本示例来调控分解出的子任务的粒度，确保整个任务流程的顺畅与高效。(4) **行动模块**负责将计划模块生成的计划转化为具体的行动步骤，并借助外部工具执行这些步骤以实现 Agent 的目标。通常会为 Agent 提供工具 API 的接口，把调用 API 接口的示例作为上下文，让大语言模型生成调用 API 的代码，之后执行这些代码，从而得到执行步骤的结果。

如图 3.30 展示了一个小浣熊健康助理 Agent 处理用户请求“小浣熊由于吃了太多干脆面肚子疼，现在应该怎么办？”的流程。在这个例子中，首先，**配置模块**为 Agent 设定了明确的角色定位——经验丰富的兽医，并明确了任务目标为提供专业的动物治疗建议。为确保角色设定与任务目标的一致性，相关信息始终以上下文形式嵌入至输入给大语言模型的 Prompt 中，从而为后续处理奠定坚实基础。其次，**计划模块**根据用户的具体请求，精心规划了救助小浣熊的行动方案。该模块将整体任务细化为多个子任务，包括搜寻小浣熊的相关资料及救助信息、查找附近的宠物医院以及总结各个子任务的执行结果以给出全面回答等。接着，**记忆模块**从知识库中搜寻小浣熊的相关信息，包括小浣熊饮食习性、如何急救小浣熊等，提供了必要的背景知识支持。然后，**行动模块**通过调用搜索工具，迅速搜索到附近最近的宠物医院，并调用地图工具规划出最佳的送医路线。最后，**计划模块**将多个子任务的执行结果进行汇总，并综合考虑各种因素，执行最佳行动方案。该方案不仅包括观察小浣熊的行为、提供专业的照顾建议，还详细阐述了如何为小浣熊紧急送医的步骤，以生动形象的方式向用户展示整个救助过程。

基于大语言模型的 Agent，在不同行业和应用场景都展现出了巨大的潜力。斯坦福大学利用 GPT-4 模拟了一个虚拟的西部小镇 [25]。他们创建了一个虚拟环境，让多个基于 GPT-4 的 Agent 在其中生活和互动。这些 Agent 通过 Prompt 工程里面的角色扮演技术设定了不同的角色，如医生、教师和市长等，并根据自己的角色和目标自主行动，进行交流，解决问题，并推动小镇的发展。HuggingGPT [30] 则以 ChatGPT 为核心控制器，用户给定一个任务后，它首先将任务拆分为多个子任务，并从 Huggingface 上调用不同的模型来解决这些子任务。在得到子任务的结果之后，HuggingGPT 将这些结果汇总起来，返回最终结果，展现了其在复杂任务编排规划和模型调度协同方面的强大能力。这些 Agent 的研究不仅推动了大语言模型的发展与应用落地，也为 Agent 在现实世界中的应用提供了新的视角和方法。

3.5.2 数据合成

数据质量是制约大语言模型性能上限的关键要素之一，正所谓“Garbage in, Garbage Out” [27]，无论模型架构多么优秀，训练算法多么优秀，计算资源多么强大，最终模型的表现都高度依赖于训练数据的质量。然而，获取高质量数据资源面临挑战。研究显示，**公共领域**的高质量语言数据，如书籍、新闻、科学论文和维基百科，预计将在 2026 年左右耗尽 [33]。**特定领域**的垂直数据因隐私保护和标注难度高等问题，难以大量提供高质量数据，限制了模型的进一步发展。

面对这些挑战，数据合成作为一种补充或替代真实数据的有效手段，因其可控性、安全性和低成本等优势而受到广泛关注。特别是利用大语言模型生成训练数据，已成为当前研究的热点议题，其通过 Prompt 工程技术，利用大语言模型强大的思维能力、指令跟随能力，来合成高质量数据，其中一个代表性方法是 Self-Instruct [37]。Self-Instruct 通过 Prompt 工程技术构建 Prompt，通过多步骤调用大语言模型，并依据已有的少量指令数据，合成大量丰富且多样化的指令数据。以金融

场景为例，我们可以先人工标注少量金融指令数据（如数百条），例如“请根据提供的几只基金情况，为我挑选出最佳基金”。随后，我们运用 Self-Instruct 方法调用大语言模型，我们能够将这些数据扩展至数万条，且保持高质量和多样性，如生成“请指导我如何选择股票和基金进行投资”等指令。

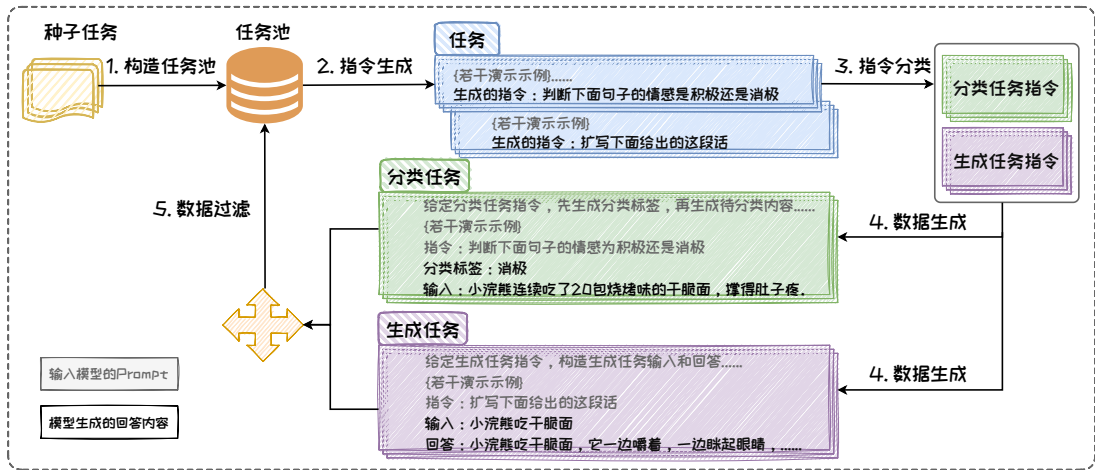


图 3.31: Self-Instruct 流程示例图。

如图 3.31 所示，Self-Instruct 包含构建任务池、指令生成、指令分类、数据生成、数据过滤五个步骤。任务池存储了初始的指令数据以及存储后续生成的指令数据；指令生成负责参考任务池中的样例，生成指令数据中的**指令部分**；指令分类将生成的指令分类为**分类任务**或**生成任务**，在这两种任务模式下，生成数据的方式不同；数据生成则根据已有的指令，生成指令数据中的**输入部分**和**回答部分**；数据过滤则复杂去除低质量的数据，保证生成的指令数据质量。它从一个有限的手动编写任务种子集开始，通过与大语言模型交互，不断生成指令数据，扩充原始的数据集。

- 1. **构建任务池**。人工手工设计了 175 个指令数据集，作为初始任务池，后续模型会不断参考任务池的示例，来生成指令数据，并将生成的指令数据加入任务池中。

- **2. 指令生成。**从任务池中随机抽取 8 个现有指令作为演示示例组成 Prompt 中的上下文，以少样本学习的方式构造 Prompt 让模型生成指令。图中生成了两个指令：“扩写下面给出的这段话”和“判断下面句子的情感为积极还是消极”。
- **3. 指令分类。**编写若干条“指令-分类任务/生成任务”样例对，作为上下文构造 Prompt，让模型判断该指令对应的任务是分类任务还是生成任务。图中“扩写下面给出的这段话”和“判断下面句子的情感为积极还是消极”分别被分类为生成任务和分类任务。
- **4. 数据生成。**对分类任务和生成任务使用 Prompt 工程中的上下文学习技术，构造不同的 Prompt 来生成指令数据中的输入部分和回答部分。对于“扩写下面给出的这段话”这条指令，它是分类任务，在 Prompt 中指示模型先生成类别标签，再生成相应的输入内容，从而使得生成的输入内容更加偏向于该类别标签；对于“判断下面句子的情感为积极还是消极”这条指令，它是生成任务，在 Prompt 中指示模型先生成输入内容，再生成对应的回答。
- **5. 数据过滤。**通过设置各种启发式方法过滤低质量或者高重复度的指令数据，然后将剩余的有效任务添加到任务池中。

上述过程中，第二步到第五步不断循环迭代，直到任务池中收集到足够的数据时停止。

数据合成的意义在于，它不仅能够缓解高质量数据资源的枯竭问题，还能通过生成多样化的数据集，提高模型的泛化能力和鲁棒性。此外，数据合成还能在保护隐私的前提下，为特定领域的垂直数据提供有效的补充。并且，通过利用大型语言模型进行数据合成，生成的数据可以用于微调小型模型，显著提升其效果，实现模型蒸馏。

3.5.3 Text-to-SQL

互联网技术进步带动数据量指数增长，目前金融、电商等各行业的海量高价值数据主要存储在关系型数据库中。从关系型数据库中查询数据需要使用结构化查询语言（Structured Query Language，SQL）进行编程。然而，SQL 逻辑复杂复杂，编程难度较高，只有专业人员才能够熟练掌握，这为非专业人士从关系型数据库中查询数据设置了障碍。为了降低数据查询门槛，零代码或低代码来的数据查询接口亟待研究。Text-to-SQL 技术可以将自然语言查询翻译成可以在数据库中执行的 SQL 语句，是实现零代码或低代码数据查询的有效途径。通过 Text-to-SQL 的方式，我们只需要动动嘴，用大白话就能对数据库进行查询，而不必自己编写 SQL 语句。这让广大普通人都可以自由操作数据库，挖掘数据库中隐含的数据价值。

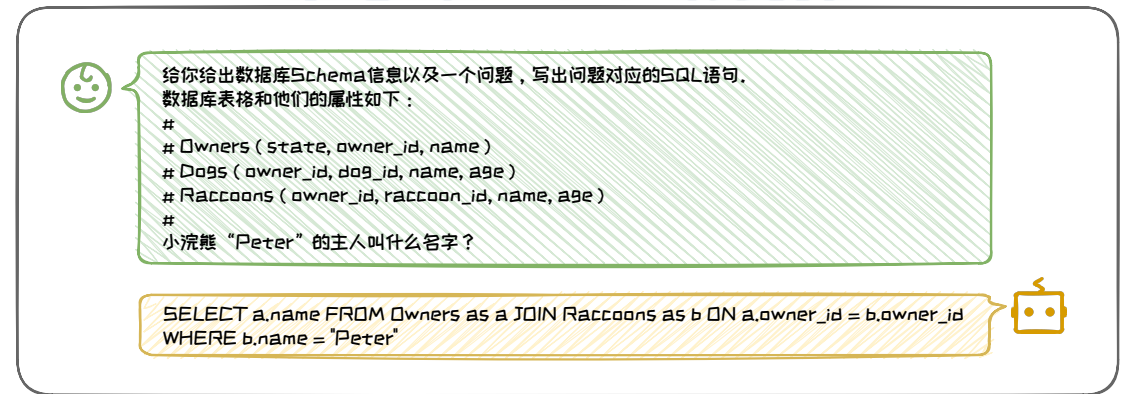


图 3.32: Text-to-SQL 示例。

传统的 Text-to-SQL 方法通常使用预训练-微调的范式训练 Text-to-SQL 模型。这些方法需要大量训练数据，费时费力且难以泛化到新的场景。近年，大语言模型涌现出的代码生成能力，为零样本 Text-to-SQL 带来可能。图 3.32 展示了一个应用大语言模型进行零样本 Text-to-SQL 的例子。用户把问题输入给大语言模型，询问“小浣熊“Peter”的主人叫什么名字？”。大语言模型会根据用户的问题，生成对应的 SQL 语句，即“SELECT a.name FROM Owners as a JOIN Raccoons as b ON

a.owner_id = b.owner_id WHERE b.name = 'Peter';”。随后，SQL 语句可以在对应的数据库中执行，得到用户提问的答案，并返回给用户。

最早使用大语言模型来做零样本 Text-to-SQL 的方法是 C3 [7]。C3 的核心在于 Prompt 工程的设计，给出了如何针对 Text-to-SQL 任务设计 Prompt 来优化生成效果。如图 3.33 所示，C3 由三个关键部分组成：**清晰提示**（Clear Prompting）、**提示校准**（Calibration with Hints）和**一致输出**（Consistent Output），分别对应模型输入、模型偏差和模型输出。

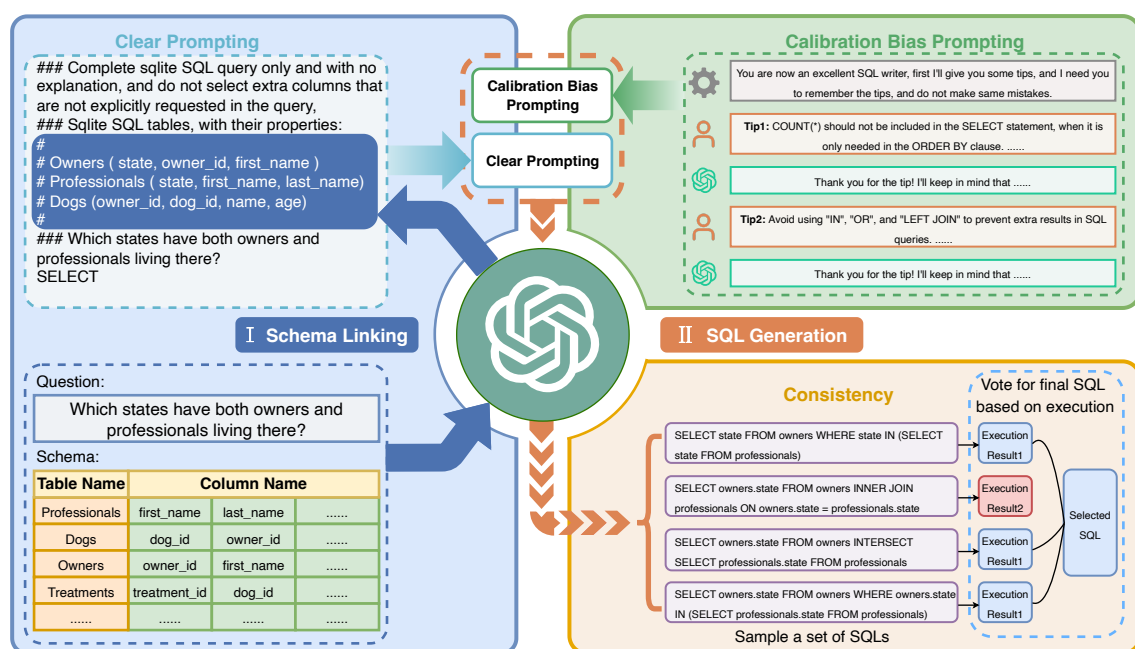


图 3.33: C3 方法整体框架图。

在模型输入端，C3 提出要采用 **清晰提示**（Clear Prompting）。其包含两部分：（1）**清晰布局**（Clear Layout），通过明确符号划分指令、上下文和问题，确保指令模板清晰，显著提升 ChatGPT 对问题的理解能力。这一策略体现了 Prompt 技巧中的“排版清晰”原则，确保信息有效传递。（2）**清晰上下文**（Clear Context），设计零样本 Prompt，指示 ChatGPT 从数据库中召回与问题相关的表和列。此举旨在检索关键信息，去除无关内容，减少上下文长度和冗余信息，从而提高生成 SQL

的准确性。这体现了 Prompt 技巧中“上下文丰富且清晰”的原则，确保了模型在处理任务时能够聚焦于关键信息。

为应对 ChatGPT 本身的固有偏差, C3 中采用**提示校准(Calibration with Hints)**。通过插件式校准策略, 利用包含历史对话的上下文提示, 将先验知识纳入 ChatGPT。在历史对话中, 设定 ChatGPT 为优秀 SQL 专家角色, 通过对话引导其遵循预设提示, 通过这种角色扮演的方式, 有效校准偏差。

在模型的输出端, C3 采用**输出校准 (Output Calibration)** 来应对大语言模型固有的随机性。C3 将 Self-Consistency 方法应用到 Text-to-SQL 任务上, 对多种推理路径进行采样, 选择最一致的答案, 增强输出稳定性, 保持 SQL 查询的一致性。

3.5.4 GPTS

GPTs 是 OpenAI 推出的支持用户自定义的 GPT 应用, 允许用户通过编写 Prompt, 添加工具等方式创建定制版的 GPT 应用, 也可以使用别人分享的 GPTs 模型。

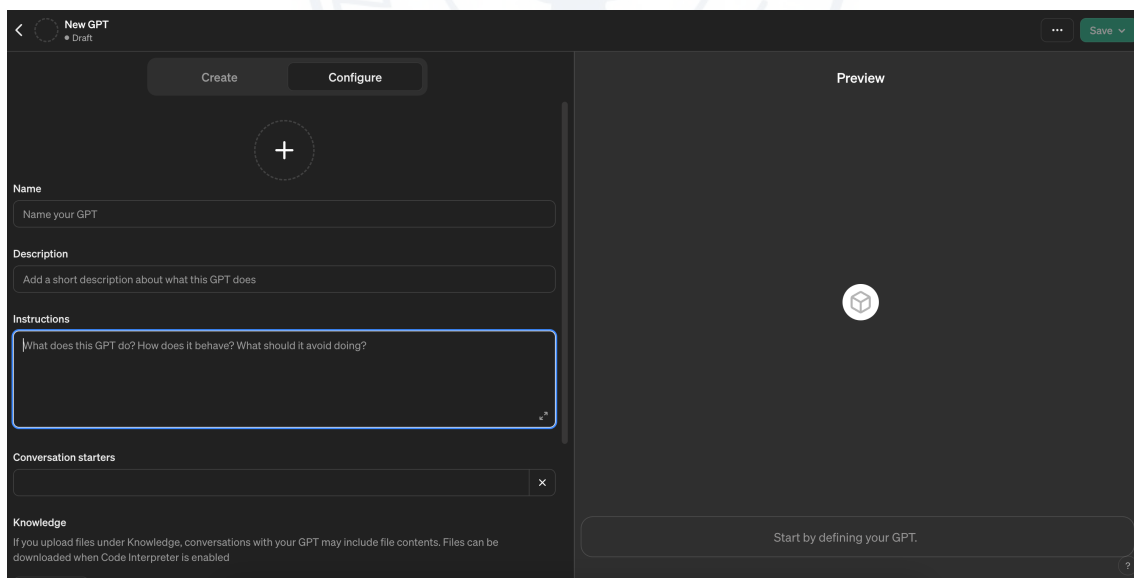


图 3.34: 制作 GPTS 的页面图⁶。

图 3.34展示了如何制作 GPTs。在这个专属页面中，用户拥有充分的自由度来自定义 GPTs 的能力和函数。通过“Description”指明该 GPTs 的功能，方便其他用户快速理解其功能和作用；而“Instructions”则是为 GPTs 预设 Prompt，使得 GPTs 能够实现预期的功能。运用本章所介绍的 Prompt 工程技术编写这些关键内容，可以显著提高 GPTs 的性能。此外，用户还可以定制专属的知识库，并根据需求选择所需的能力，例如网络搜索、图像生成、代码解释等。完成个性化 GPTs 的制作后，用户可以选择将其分享，供其他用户使用。

在本节中，我们详细探讨了 Prompt 工程在多个领域中的具体应用场景应用场。Prompt 工程在提升大语言模型的交互和执行能力方面具有重要作用。无论是在任务规划、数据合成、个性化模型定制，还是 Text-to-SQL 中，Prompt 工程都展现了其独特的优势和广阔的应用前景。

参考文献

- [1] Maciej Besta et al. “Graph of Thoughts: Solving Elaborate Problems with Large Language Models”. In: *AAAI*. 2024.
- [2] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *arXiv preprint arXiv:2005.14165* (2020).
- [3] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *NeurIPS*. 2020.
- [4] Stephanie C. Y. Chan et al. “Data Distributional Properties Drive Emergent In-Context Learning in Transformers”. In: *NeurIPS*. 2022.
- [5] Aakanksha Chowdhery et al. “PaLM: Scaling Language Modeling with Pathways”. In: *Journal of Machine Learning Research* 24 (2023), 240:1–240:113.
- [6] DeepSeek-AI et al. “DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model”. In: *arXiv preprint arXiv:2405.04434* (2024).
- [7] Xuemei Dong et al. “C3: Zero-shot Text-to-SQL with ChatGPT”. In: *arXiv preprint arXiv:2307.07306* (2023).
- [8] Philip Gage. “A new algorithm for data compression”. In: *The C User’s Journal* 12.2 (1994), pp. 23–38.

- [9] Dan Hendrycks et al. “Measuring Massive Multitask Language Understanding”. In: *ICLR*. 2021.
- [10] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.
- [11] Huiqiang Jiang et al. “LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models”. In: *EMNLP*. 2023.
- [12] Takeshi Kojima et al. “Large Language Models are Zero-Shot Reasoners”. In: *NeurIPS*. 2022.
- [13] Aobo Kong et al. “Better Zero-Shot Reasoning with Role-Play Prompting”. In: *arXiv preprint arXiv:2308.07702* (2023).
- [14] Jannik Kossen, Yarin Gal, and Tom Rainforth. “In-Context Learning Learns Label Relationships but Is Not Conventional Learning”. In: *arXiv preprint arXiv:2307.12375* (2024).
- [15] Junlong Li et al. “Self-Prompting Large Language Models for Zero-Shot Open-Domain QA”. In: *arXiv preprint arXiv:2212.08635* (2024).
- [16] Xiaonan Li et al. “Unified Demonstration Retriever for In-Context Learning”. In: *ACL*. 2023.
- [17] Jiachang Liu et al. “What Makes Good In-Context Examples for GPT-3?” In: *ACL*. 2022, pp. 100–114.
- [18] Nelson F Liu et al. “Lost in the middle: How language models use long contexts”. In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 157–173.
- [19] Yao Lu et al. “Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity”. In: *ACL*. 2022.
- [20] Man Luo et al. “In-context Learning with Retrieved Demonstrations for Language Models: A Survey”. In: *arXiv preprint arXiv:2401.11624* (2024).
- [21] Ziyang Luo et al. “Wizardcoder: Empowering code large language models with evol-instruct”. In: *arXiv preprint arXiv:2306.08568* (2023).
- [22] Yuren Mao et al. “FIT-RAG: Black-Box RAG with Factual Information and Token Reduction”. In: *arXiv preprint arXiv:2403.14374* (2024).
- [23] Sewon Min et al. “Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?” In: *EMNLP*. 2022.

- [24] Jane Pan et al. “What In-Context Learning ”Learns” In-Context: Disentangling Task Recognition and Task Learning”. In: *ACL*. 2023.
- [25] Joon Sung Park et al. “Generative Agents: Interactive Simulacra of Human Behavior”. In: *UIST*. 2023.
- [26] Allan Raventós et al. “Pretraining task diversity and the emergence of non-Bayesian in-context learning for regression”. In: *NeurIPS*. 2023.
- [27] L. Todd Rose and Kurt W. Fischer. “Garbage in, garbage out: Having useful data is everything”. In: *Measurement: Interdisciplinary Research and Perspectives* 9.4 (2011), pp. 224–226.
- [28] Alexander Scarlatos and Andrew Lan. “RetICL: Sequential Retrieval of In-Context Examples with Reinforcement Learning”. In: *arXiv preprint arXiv:2305.14502* (2024).
- [29] Mike Schuster and Kaisuke Nakajima. “Japanese and korean voice search”. In: *ICASSP*. 2012.
- [30] Yongliang Shen et al. “HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face”. In: *NeurIPS*. 2023.
- [31] Seongjin Shin et al. “On the Effect of Pretraining Corpora on In-context Learning by a Large-scale Language Model”. In: *NAACL*. 2022.
- [32] Rohan Taori et al. “Alpaca: A strong, replicable instruction-following model”. In: *Stanford Center for Research on Foundation Models*. (2023).
- [33] Pablo Villalobos et al. “Will we run out of data? An analysis of the limits of scaling datasets in Machine Learning”. In: *arXiv preprint arXiv:2211.04325* (2022).
- [34] Changan Wang, Kyunghyun Cho, and Jiatao Gu. “Neural machine translation with byte-level subwords”. In: *AAAI*. 2020.
- [35] Lei Wang et al. “A survey on large language model based autonomous agents”. In: *Frontiers of Computer Science* 18.6 (2024), p. 186345.
- [36] Xuezhi Wang et al. “Self-Consistency Improves Chain of Thought Reasoning in Language Models”. In: *ICLR*. 2023.
- [37] Yizhong Wang et al. “Self-Instruct: Aligning Language Models with Self-Generated Instructions”. In: *ACL*. 2023.
- [38] Jason Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *NeurIPS*. 2022.

- [39] Jason Wei et al. “Emergent Abilities of Large Language Models”. In: *Transaction of Machine Learning Research* 2022 (2022).
- [40] Sang Michael Xie et al. “An Explanation of In-context Learning as Implicit Bayesian Inference”. In: *ICLR*. 2022.
- [41] An Yang et al. “Qwen2 Technical Report”. In: *arXiv preprint arXiv:2407.10671* (2024).
- [42] Shunyu Yao et al. “Tree of Thoughts: Deliberate Problem Solving with Large Language Models”. In: *NeurIPS*. 2023.
- [43] Kang Min Yoo et al. “Ground-Truth Labels Matter: A Deeper Look into Input-Label Demonstrations”. In: *EMNLP*. 2022.
- [44] Tao Yu et al. “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task”. In: *EMNLP*. 2018.
- [45] Chao Zhang et al. “FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis”. In: *SIGMOD*. 2024.
- [46] Zhuosheng Zhang et al. “Automatic Chain of Thought Prompting in Large Language Models”. In: *ICLR*. 2023.
- [47] Wayne Xin Zhao et al. “A Survey of Large Language Models”. In: *arXiv preprint arXiv:2303.18223* (2023).
- [48] Yuxiang Zhou et al. “The Mystery of In-Context Learning: A Comprehensive Survey on Interpretation and Analysis”. In: *arXiv preprint arXiv:2311.00237* (2024).