第 1 章

绪　　论

1

随着时代的演进，AI迅猛发展，DeepSeek应用、豆包、Manus.ai等AI Agent开始打破人们生活的原有平衡。可以说，这是属于AI Agent的时代。作为本书的第1章，将带领读者从全局初步认识Agent，同时介绍本书编写的初衷与学习价值，内容包括AI Agent演进之路、Agent市场前景及本书内容安排。希望本书能够成为读者在AI时代乘风踏浪的起点，与你一起扬帆起航。

1.1　AI Agent 演进之路

仿佛一夜之间，AI Agent（人工智能体，简称智能体）的浪潮席卷而来，悄然重塑我们的生活——它已不再是遥不可及的技术概念，而成为每个人日常中不可或缺的伙伴，显著提升各行业的效率，也让相关的从业者能够更大程度地发挥他们的创造力。AI Agent不仅作用于重复性的工作，更悄然渗透至复杂、创新的核心领域。它远非一次普通的产品迭代，其影响之深远，足以与蒸汽时代、电力时代、信息时代相提并论。大胆推测，未来的史书或将称这一阶段为“第五次工业革命”。

作为本书的第1节，愿它成为读者与AI Agent之间因果之链的起点。本节将带领读者回溯AI Agent的完整演进脉络：从大语言模型（LLM）的奠基，到当前蓬勃发展的AI Agent，再到未来迈向物理世界的Physical Agent。通过本节的学习，读者将串联起AI发展的全链路，或许会对个人乃至时代的新使命有不同的见解。

1.1.1　开幕：LLM 与 ChatGPT

从学术概念上描述，AI Agent是一种具备感知环境并进行理解、决策和执行行为的智能体。提到AI Agent，就不得不引出LLM（Large Language Model，大语言模型），因为所有形态的Agent都是基于LLM开发的，Agent中最重要的理解、角色及执行能力也是LLM赋予的，可以说没有LLM

就没有AI Agent。

对于非算法领域工作者，LLM最早进入视野应是在2022年末，OpenAI公司的GPT-3.5模型发布，并于一个月后发布了世界上第一个AI应用ChatGPT，它支持以类人的方式理解用户的问题并进行有效对话。AI热潮的浪花也从那时开始真正波动起来。ChatGPT示例如图1-1所示。

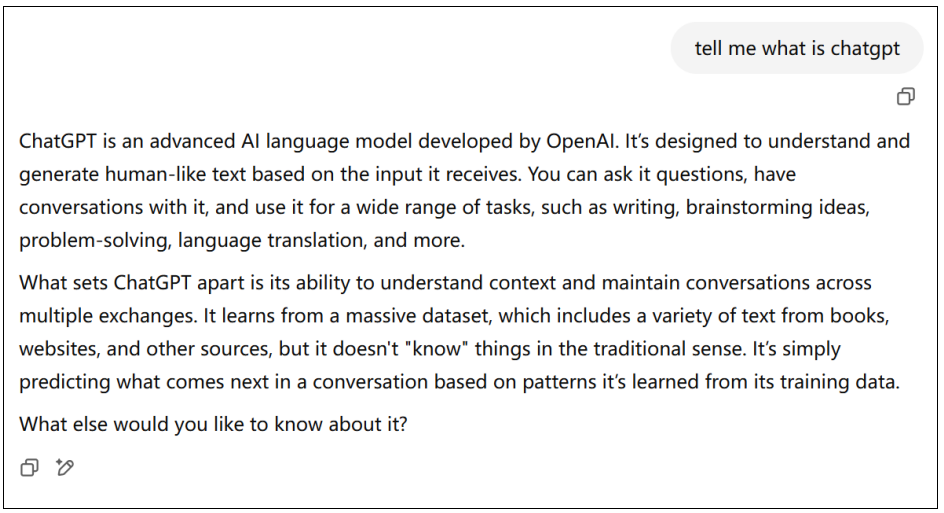

图 1-1　ChatGPT 示例

由于ChatGPT的使用受到一定的限制，很多读者不能第一时间体验，因此不能立即感知到AI对时代变化的深远影响。虽然ChatGPT直到2022年末才掀起第一波AI浪潮，但LLM的起源实际上可以追溯到2017年底。

1.1.2　前传：Transformer 架构

2017年以前，主流的序列模型架构主要依赖循环神经网络（Recurrent Neural Network，RNN）及其变种。这种模型架构的输出会使用一个隐藏状态层加权得出，可以理解为信息传递需要通过隐藏状态完成，其中每个时刻的隐藏状态层都依赖当前的输出加上前一个时刻的隐藏状态层，数学公式表示如下：

$$\boldsymbol{h}_t=\sigma(\boldsymbol{W}_h\boldsymbol{h}_{t-1}+\boldsymbol{W}_x\boldsymbol{x}_t+\boldsymbol{b})$$

其中：

- $\boldsymbol{W}_h$为隐藏状态层的权重矩阵。
- $\boldsymbol{W}_x$是输入内容的权重矩阵。
- $\boldsymbol{b}$是偏置项。
- σ是激活函数。

也就是说，当前的隐藏状态层 $\boldsymbol{h}_t$ 需要前一个隐藏状态层 $\boldsymbol{h}_{t-1}$ 与输入内容 x_t 分别加权并用偏置项 $\boldsymbol{b}$ 处理后，使用激活函数计算得到。最终 RNN 的输出将消费当前的隐藏状态层 $\boldsymbol{h}_t$ 加权得到的结果，采用以下公式完成：

$$\boldsymbol{y}_h = \boldsymbol{W}_y\boldsymbol{h}_t + \boldsymbol{b}_y$$

同理，$\boldsymbol{W}_y$和$\boldsymbol{b}_y$分别为输出内容的权重矩阵和偏置项。可以看出，RNN的信息传递依赖一种循环结构来递进式地计算时间步，就像爬一个特殊的楼梯一样，需要一阶一阶地爬上去，每一步都依赖之前的动作，记住上一阶楼梯的位置后，下一阶楼梯才会出现，就像图1-2所示的示例，时间步4在到达时间步3后才可以得出。

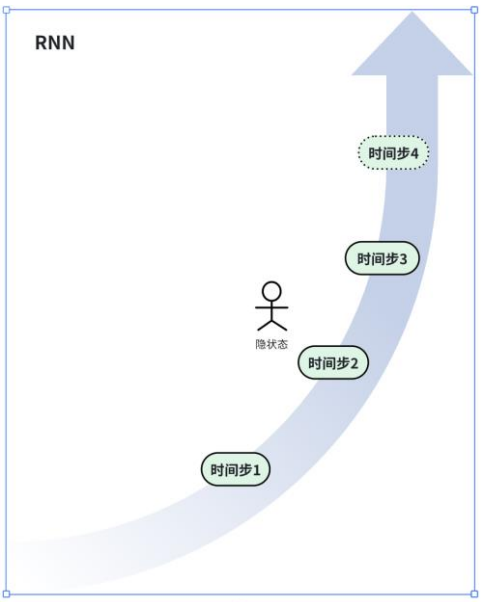

图 1-2　RNN 的信息传递

从这个过程中不难发现 RNN 的一些局限性：

- 梯度消失：假设已经爬到很高的楼层，回想第一阶的情况，就会遗忘细节。这会导致RNN在长程依赖的复杂场景中存在信息精度丢失的问题。
- 并行化差：RNN的隐藏状态需要一步一步传递过去，不能跳过某个楼层直达，这就导致计算效率比较低，并且因为需要顺序计算的缘故，GPU的并行计算能力不能得到充分发挥，计算资源也就浪费了。

LLM应对的复杂场景需要有更多的信息专注度和高效的训练学习方式，而基于循环结构的RNN由于并行化差和计算成本高的原因，很难衍生出百亿甚至千亿参数的LLM。虽然学术界也提出了一些基于RNN的变种架构，例如长短时记忆网络（LSTM）和门控循环单元（GRU），对训练

架构进行了一些优化，但是本质上仍未脱离RNN的循环结构，并行化差和计算成本高的问题没有得到根本解决。

直到2017年，Google Brain团队于*Attention is All You Need*中提出了Transformer架构。Transformer架构做出的重要改进之一就是完全摒弃了RNN的循环架构，转而提出了自注意力机制，彻底解决了传统循环结构计算速度慢、无法并行化以及梯度消失等问题。

自注意力机制采用计算序列中各元素之间的关系来捕捉长程依赖的信息。在这个过程中，每个元素（字或者词，也就是token）都会被映射为查询（Query）、键（Key）和值（Value）向量，它们分别代表信息中的查询、标识符和实际信息。映射过程则是使用训练得到的特定权重矩阵加权完成。以查询为例，加权公式如下：

$$\boldsymbol{Q}_i = \boldsymbol{W}_Q \boldsymbol{X}_i$$

其中，$\boldsymbol{Q}_i$是对应的查询向量，$\boldsymbol{W}_Q$是训练得到的查询权重矩阵，$\boldsymbol{X}_i$是输入值，其他向量同理。

在映射词得到3个向量后，自注意力机制会使用向量自身的查询（$\boldsymbol{Q}_i$）和其他向量的键（$\boldsymbol{K}_j$）通过点积的方式来计算相似度，用于反映当前输入元素对其他元素的关注程度，也称注意力得分（Attention Score）：

$$\text{AttenionScore}(\boldsymbol{Q}_i, \boldsymbol{K}_j) = \boldsymbol{Q}_i \boldsymbol{K}_j^{\mathrm{T}}$$

值得一提的是，这个过程会对所有 token 执行，意味着每个查询向量 $\boldsymbol{Q}_i$ 都将与其他所有键向量 $\boldsymbol{K}_j$ 计算相似度，从而获得各位置之间的相似性信息。在获得所有的相似度信息后，需要先基于键向量的维度 d_k 进行缩放，避免点积过大导致梯度爆炸或消失，再使用 softmax 函数完成归一化处理，将相似度转换为概率分布，也称注意力权重（Attenion Weight）：

$$\text{AttenionWeight}_i = \text{soft}\max\left(\frac{\text{AttenionScore}(\boldsymbol{Q}_i, \boldsymbol{K}_j)}{\sqrt{d_k}}\right)$$

注意力权重与注意力得分类似，都用来反映当前输入元素对其他元素的“关注程度”，不同的是，它是一个更为具化的权重，能够用来给值向量 $\boldsymbol{V}_j$ 消费。在获取所有 token 的注意力权重后，就可以对它们进行加权求和，得到最终的输出结果：

$$\text{Output}_i = \sum_j \text{AttentionWight}_{ij} \boldsymbol{V}_j$$

上述就是整个自注意力机制的过程，不难发现，自注意力机制就像一个承接对外需求的团队，团队中有大量的成员（token），管理者（Query、Key、Value 的权重矩阵）会分别计算每个成员的 3 个核心向量：

- Key是个人在团队中的身份。
- Query是个人在团队中的需求，需要团队支持的资源。

- Value是个人在团队中的价值。

当团队承接到外部需求后，每个成员都会用自己的身份卡（Query）和他人的令牌（Key）进行匹配，以判断各团队成员之间的在这次需求中的配合密切程度（Attention Score）。再经过统一处理后，每个团队成员都会得到一个与其他团队成员交互配合的关系表（Attention Weight）。最终成员按照既定分配，用自己的价值（Value）来完成与关系表（Attention Weight）中各个成员的配合（加权），最终输出外部需求的结果。整个过程如图1-3所示。

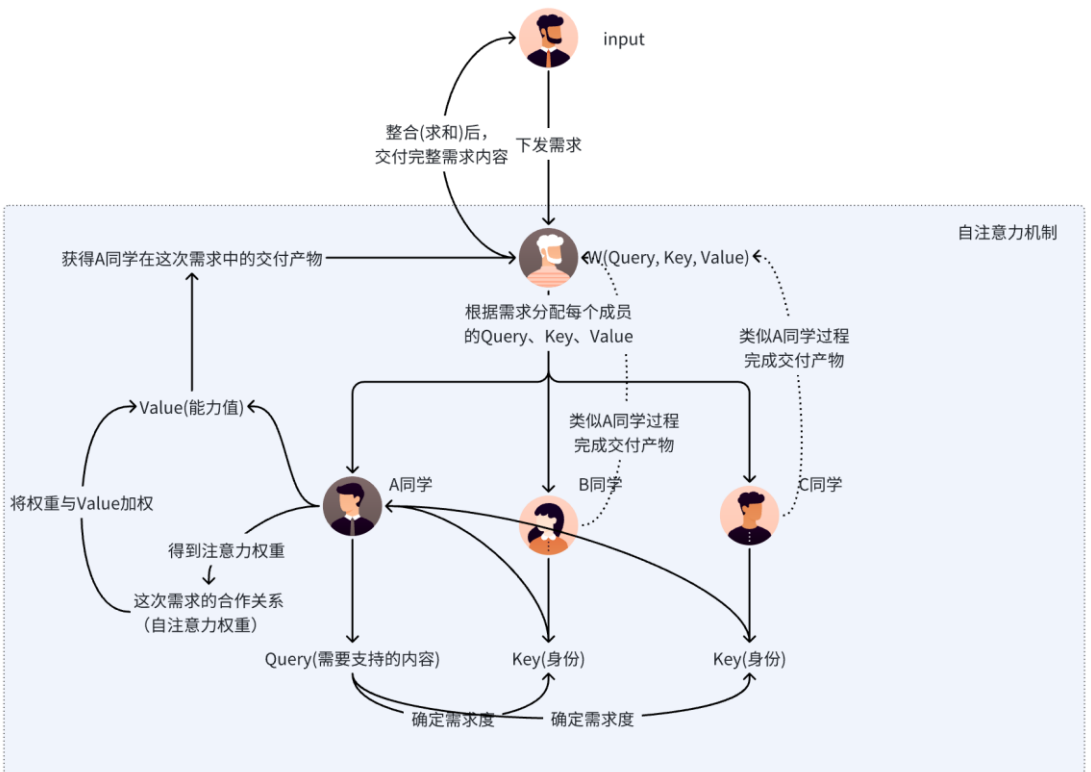

图 1-3　Transformer 架构的信息传递

不难发现，以自注意力机制为核心的Transformer架构相比RNN等基于循环结构的模型架构，在并行化能力和计算效率上有着显著的优势。

Transformer架构中的自注意力机制允许在处理每个词时，同时关注序列中所有其他位置的信息。这种方式使得每个词的表示可以并行计算，大大提高了计算效率，最大程度地发挥了现代多核处理器的能力，这也为LLM的出现奠定了基础。

更有意思的是，回顾自注意力机制的过程可以发现，如果基于Transformer架构的模型需要承接更大的需求或者对需求提供更精准的答复，主要提高以下两点：

（1）每个词的3个核心权重矩阵（$\boldsymbol{W}_Q$、$\boldsymbol{W}_K$、$\boldsymbol{W}_V$），这些矩阵决定了词在需求中的定位以及词与词之间应该以何种权重配合联系。

（2）模型中的词数量，它决定了能承接多大的需求或者说更擅长哪些方向的需求。

类比现实社会中的概念，那就是提高管理者的判断力，对每个成员针对需求进行更合适的能力评估和分工配合，以及扩大团队的成员规模。就像要提高团队的战斗力，既要提高管理者对团队成员的判断和分工，也需要补充各种垂类领域的人才。这也是LLM训练过程中最为关注的两个指标。

虽然Transformer架构在理论上非常强大，但要实现高效的大规模预训练和生成出真正能够广泛应用的LLM，仍面临许多技术挑战，例如实际的模型架构、GPU等硬件的发展、训练策略等，因此距真正可大规模应用的LLM问世仍有一段不短的时间。

1.1.3　初见端倪：单向自回归解码

标准的Transformer架构中包含两个相辅相成的组件：Encoder（编码器）和Decoder（解码器）。Encoder用于将输入信息理解后，转换为隐藏层表示；Decoder则用于将隐层表示转换为可理解的输出。同时，它们都采用自注意力机制来保证过程中对每个元素的关注和联系，这也被称为Encoder-Decoder架构，如图1-4所示。

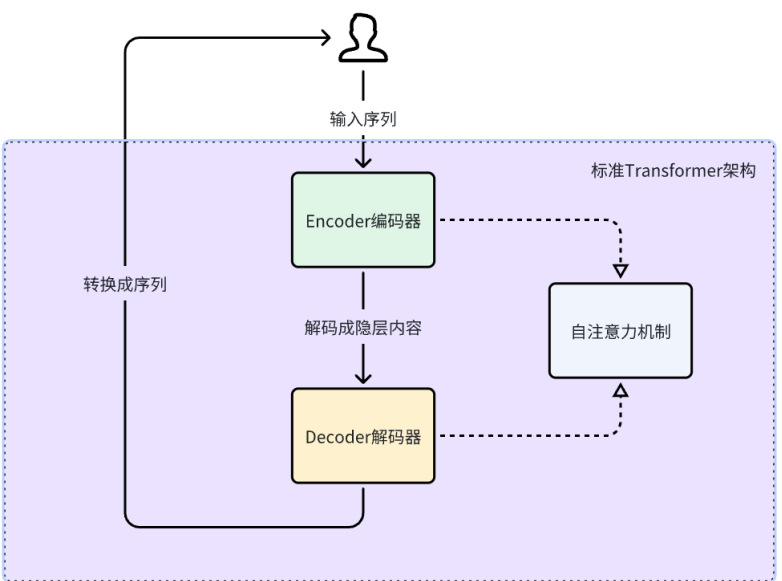

图 1-4　标准 Transformer Encoder-Decoder 架构

基于Transformer架构，2018年自然语言处理（Natural Language Processing，NLP）领域出现了两个有影响力的模型系列，BERT（Bidirectional Encoder Representations from Transformers，

Transformer双向编码器）和GPT（Generative Pre-trained Transformer，生成式预训练Transformer）。有意思的是，它们没有直接使用标准的Transformer架构，而是将Encoder-Decoder架构拆开，分别使用了其中的一个组件，以更好地适应不同领域的任务。

BERT专注于文本理解任务，例如文本分类、情感分析等，是理解型的LLM。它使用编码器部分，并且采用双向自注意力机制。双向自注意力指的是在训练过程中，模型同时关注上下文的前后信息，其输出预测并不是下一个词，而是句子中的随机掩码标记，整个过程如图1-5所示。

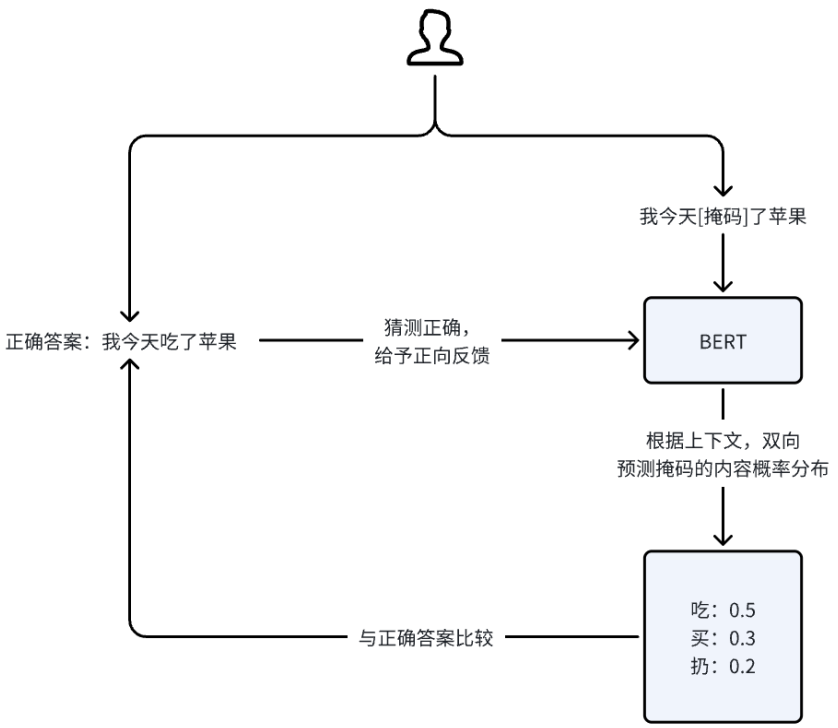

图 1-5　BERT 双向自注意力的训练过程

GPT则专注于生成任务，例如文本生成、对话生成和续写文本等，是生成型的LLM。对于GPT而言，基于上下文前后双向的预测不再适用，因为在生成任务中，需要逐字生成，无法用未来的词来预测现在的内容。这与自然语言的生成过程是一致的，人类在表达思想时，总是从左到右（或从过去到未来）构建句子，无法在生成的同时利用后续的内容。

基于这个特性，GPT使用了Transformer架构中的解码器部分，并且采用单向自回归解码的策略完成文本生成任务。自回归指的是每个词的生成都基于已生成的词和上下文完成，而单向则指的是GPT只会基于之前生成的词进行预测，即从左到右的顺序，不会考虑未来的词，整个过程如图1-6所示。

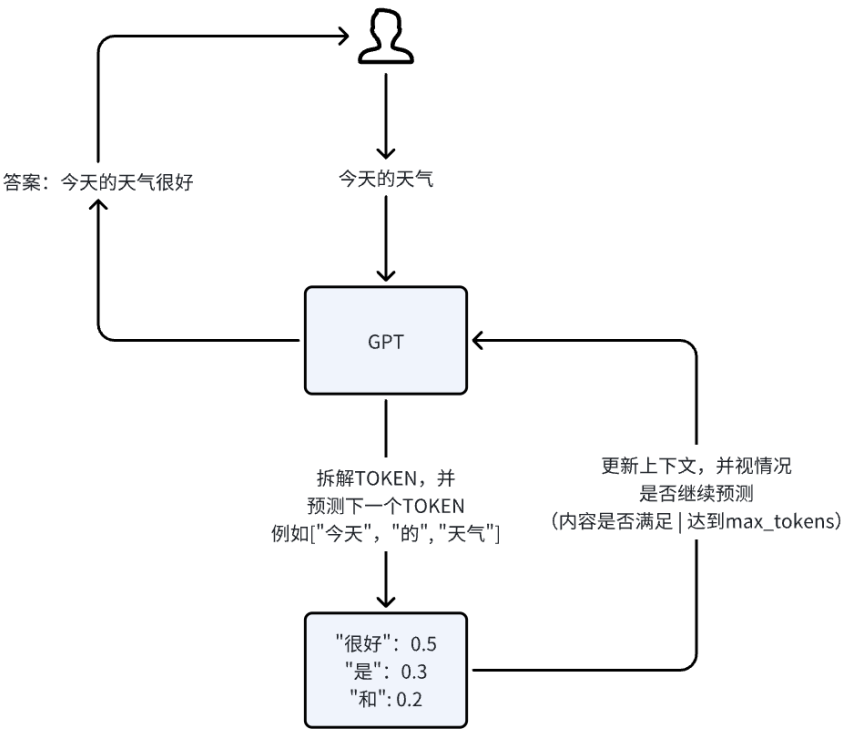

图 1-6　GPT 单向自回归解码的过程

这种单向解码自回归的机制在文本生成场景中表现非常出色。2018年6月，GPT的首个版本发布，其参数规模达1.17亿，并基于4.6GB高质量文本数据训练，是生成类LLM的重大突破。这个参数量对于真正泛化可落地的大模型来说还相差甚远，但已经是一个坚实基础，也为未来具备强大生成能力的LLM问世做好了铺垫。

1.1.4　问世：ChatGPT

在GPT第一版本取得突破后，OpenAI进一步训练扩充了模型参数，于2019年2月发布了GPT-2，其参数规模达15亿，并基于40GB的数据进行训练，如图1-7所示。与第一版本不同，GPT-2已经能够在不针对垂直领域进行特定微调的情况下，完成回答问题、翻译、撰写文章等通用性任务。这进一步证明了扩大模型规模和训练数据的重要性。

随后，OpenAI趁热打铁，于2020年6月发布了GPT-3，其参数规模达1750亿。相比GPT-2勉强达到基本可沟通的程度，GPT-3真正意义上表现出了强大的通用能力，它能够在没有干预的情况下，顺畅地完成有逻辑的上下文沟通和信息输出，并且能够在一定程度上完成创新性工作。例如，OpenAI官网中提到一个应用GPT-3的示例，业务方分析用户对结账体验的满意度，并给出解决方案，如图1-8所示。

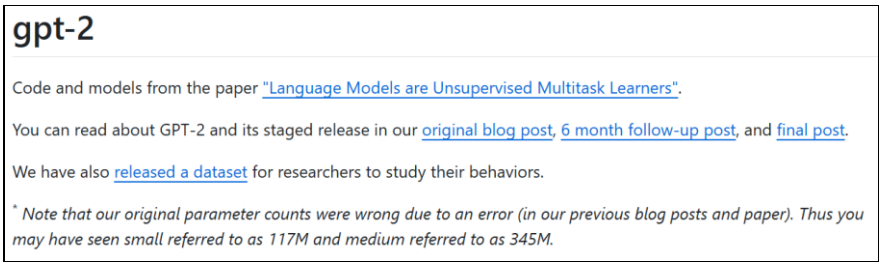

图 1-7　GPT-2 开源仓库 Readme.md

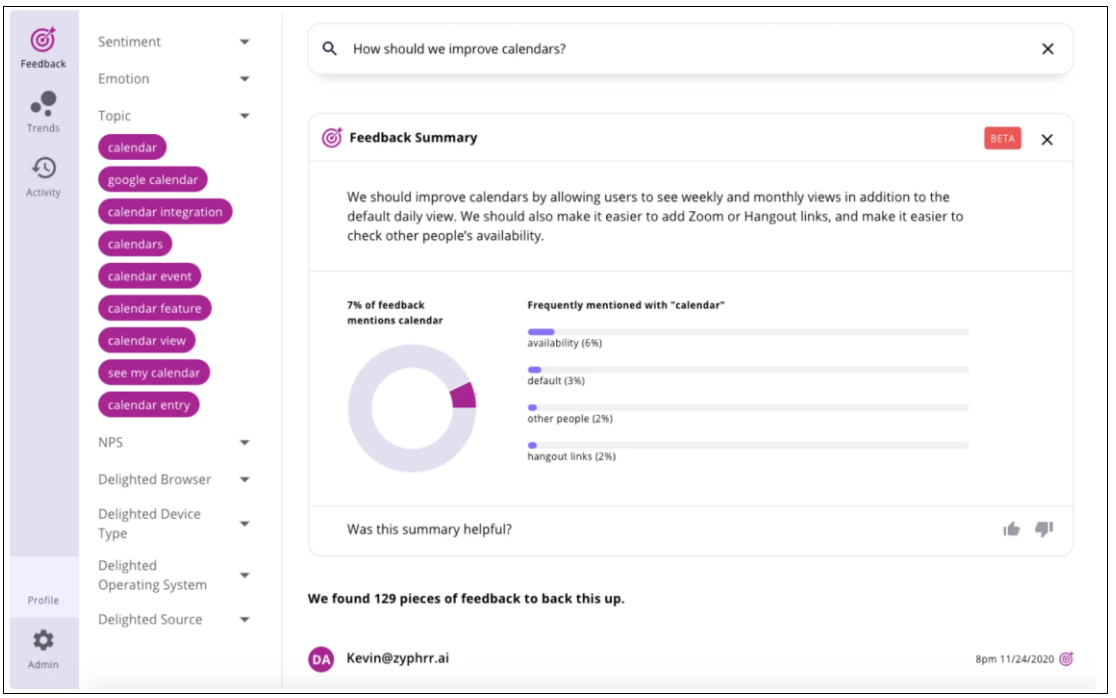

图 1-8　GPT-3 应用于分析用户结账体验的示例

在从亿级参数膨胀到千亿级参数的过程中，GPT系列模型的能力获得了巨大的提升，但也暴露了新的问题——“幻觉”，即LLM生成的内容经常与事实不符或者无意义，像一本正经地胡说八道。如果用户对提问问题的答案没有一个大致的认知，就无法区分模型给出的答案中哪些是正确的。

为了解决这个问题，研究人员提出了SFT（Supervised Fine-Tuning，监督微调）的调优方案。它通过示例调优，即高质量的输入/输出示例集合来训练模型，指导模型完成指定场景的任务。例如，如果希望GPT能更好地应对法律场景，就需要搜集一定体量的高质量法律场景的输入/输出，类似以下格式：

```
    [
        {
```

```
            "input": "What is the legal age for drinking alcohol in the United States?",
            "output": "The legal age for drinking alcohol in the United States is 21."
        },
        {
            "input": "What is the difference between first-degree murder and
second-degree murder?",
            "output": "First-degree murder is premeditated, while second-degree murder
is unplanned but intentional."
        },
        {
            "input": "What is a contract?",
            "output": "A contract is a legally binding agreement between two or more
parties."
        }
    ]
```

当然，要实际用于训练，数据量肯定远不止3条，这只是一个格式上的示例，每一次针对某个场景的高质量一问一答即是一条数据。准备好数据集，再经过训练、评估等步骤，模型后续在面对这些场景时，将会模仿数据集中的人类行为来进行答复，整个过程如图1-9所示。

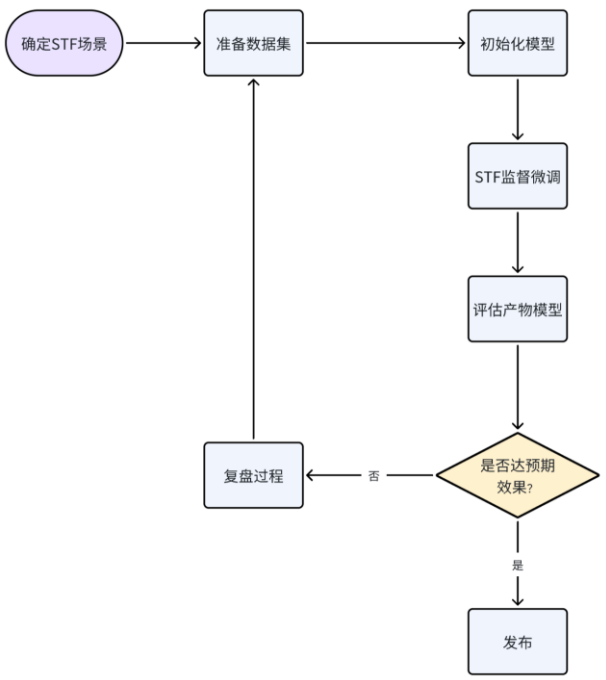

图 1-9　STF 训练过程

STF的数据集如果是精心搜集的，并且能够反映预期行为和结果，将对模型的效果有显著提升。

它的缺点也很明显，STF强依赖数据集的体量和质量。数据集就像模型的课本，在数据量不足或者质量不达标的情况下，模型不仅不能调优，可能还会有劣化和过拟合的风险。

同时，数据集本质上是对人类行为的采集。若数据已通过用户埋点等方式实现平台化收集，则仅需人工进行标注与筛选，成本尚可接受；但若需从零开始依赖人工全量准备，这一过程将极为密集且耗时。

不难看出，针对未经过数据沉淀，需要从零搜集数据集的场景，如果团队资源有限，那么STF的训练方式就存在局限性和困难。基于这个背景，一种更具扩展性和效果的训练方式被提出了——RLHF（Reinforcement Learning from Human Feedback，基于人类反馈的强化学习）。

与STF不同，RLHF不再通过提供一问一答式的数据集让模型模仿人类行为，而是通过人工评估模型输出结果，并对好的结果进行奖励的方式训练。同样针对法律场景的调优，RLHF提供给模型的输入将是一个问题，并且让模型针对这个问题做出初步答复，在这之后，人类专家会针对这次答复进行评分和更具化的指导意见。上述整个过程将得到一个类似如下的产物：

```
    What is the legal age for drinking alcohol in the United States? // 问题
    The legal age for drinking alcohol in the United States is 18. // 模型答复
    Score 2.The legal age for drinking alcohol in the United States is 21, not 18.
Please revise the answer accordingly.   // 人工评分和指导意见
```

当然，这只是一个例子，实际上可能会有更多的法律问题遵循上述步骤得到类似产物，然后基于这个产物去训练一个奖励模型，用于将模型生成的产物与人类反馈进行对比，从而得出评分。通过奖励模型进一步对原始模型进行强化训练，让它的答案逐步达到奖励模型的高分标准，就是RLHF的整个过程，如图1-10和图1-11所示。

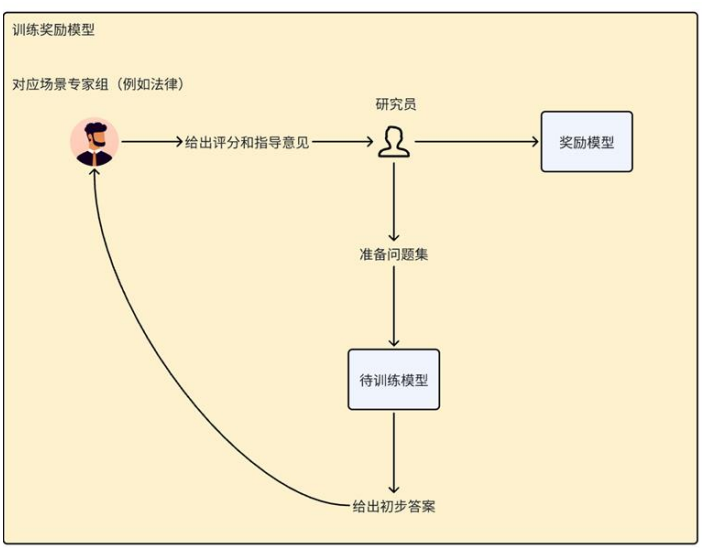

图 1-10　RLHF 训练奖励模型的过程

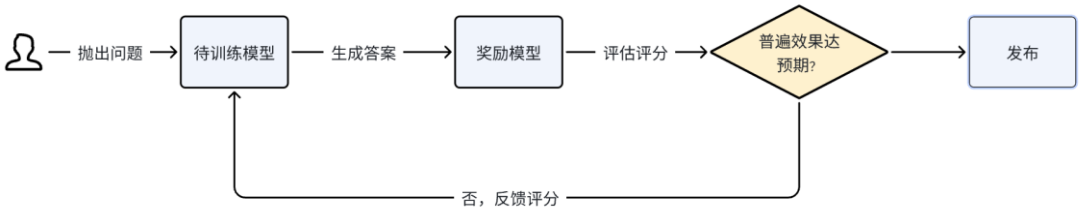

图 1-11　RLHF 强化学习优化的过程

不难看出，RLHF通过将人类反馈融入训练过程，不仅降低了数据集收集的成本，而且显著提升了模型生成可靠且被人类认可的答案的能力。

研究员同时基于上述两种调优方案，有效缓解了GPT-3的幻觉问题，并在2022年3月推出了GPT-3的升级版——GPT-3.5。GPT-3.5是生成式LLM历史上第一款落地效果佳、能基本满足应用层开发需要的模型。2022年11月，OpenAI又基于GPT-3.5推出了ChatGPT，如图1-12所示。至此，LLM开始与大部分读者接轨，AI开始影响我们生活的各个方面。

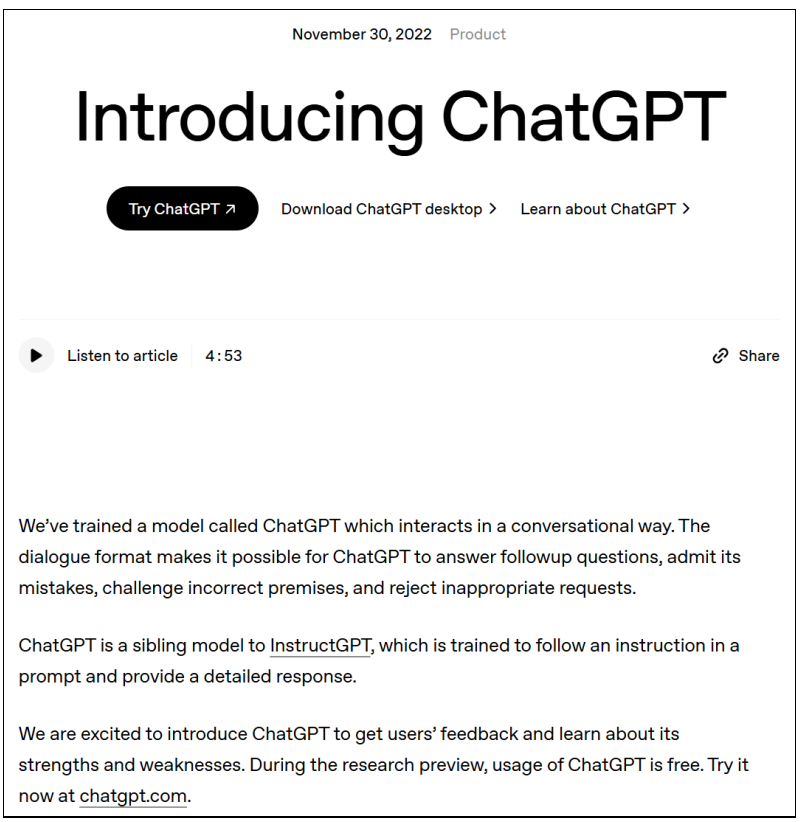

图 1-12　ChatGPT 官网

那段时间，OpenAI作为LLM的领头羊，如日中天，在ChatGPT推出不久，又抛出几个重磅炸弹——于2023年至2024年间，推出了GPT-4、GPT-4V和GPT-4o等模型。它们相比GPT-3.5，除了文本生成能力得到提升外，还是MLLMs（Multimodal Large Language Models，多模态大语言模型）。

MLLMs的推出意味着生成式LLM的输入交互不再只是文本，而是开拓至图像、音频、视频等多种媒体，这让生成式LLM在一些需要多媒体材料的行业（如医疗、教育等）产生了更深远的影响，并让人们与AI正常沟通、对话并做出一定的交互也开始成为可能。

1.1.5　风波再起：DeepSeek

OpenAI的成功让各行各业看到了AI在这个时代的巨大潜力。从2023年起，很多公司纷纷开始训练自己的模型。例如，Anthropic推出的Claude系列模型在编码领域表现优异，而Google推出的Gemini系列模型在综合性能上也表现突出，如图1-13和图1-14所示。

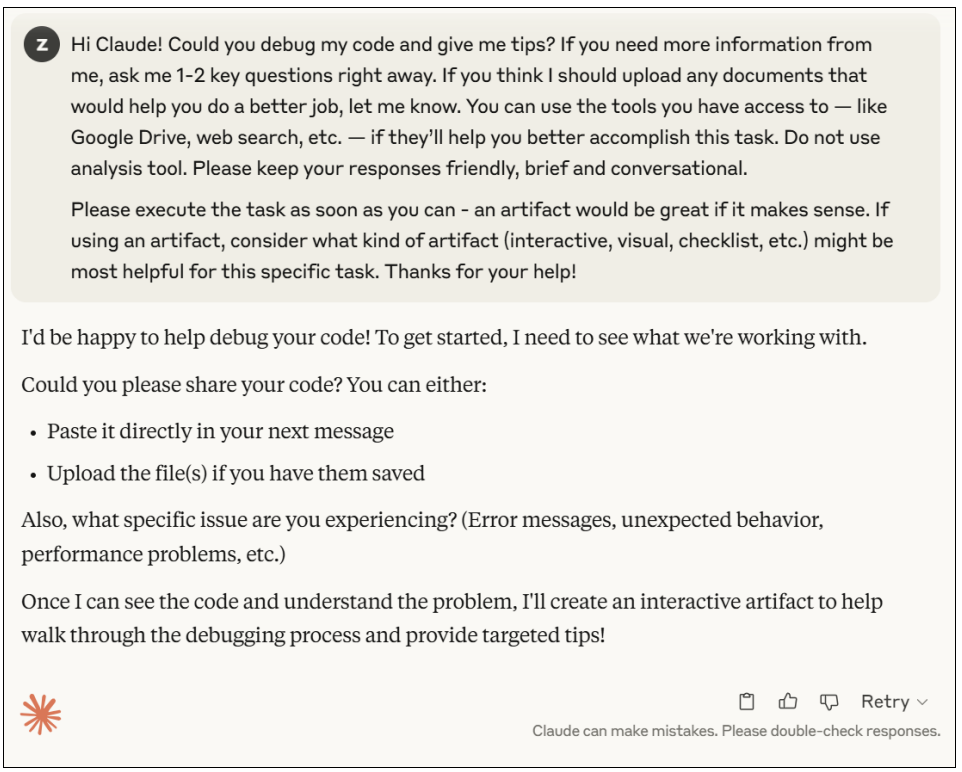

图 1-13　Claude 模型示例

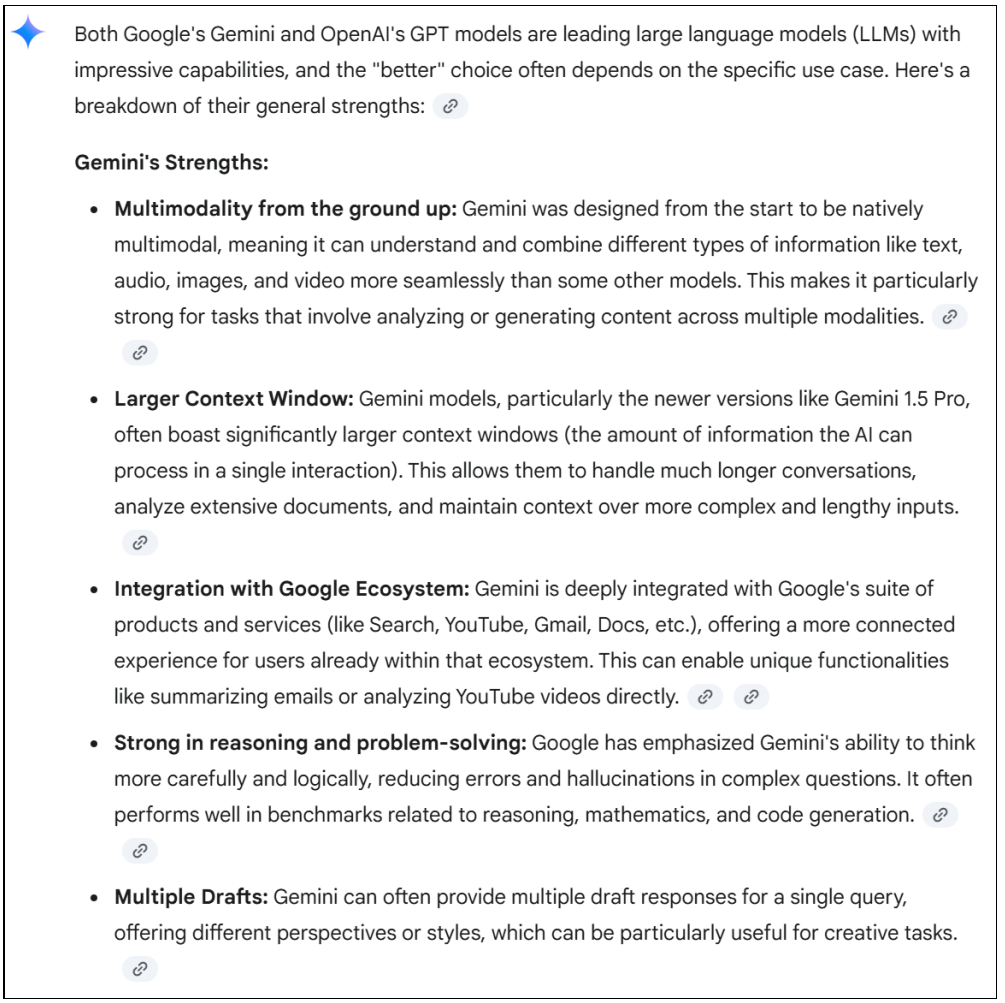

图 1-14　Gemini 模型示例

与此同时，国内大厂与独角兽企业也推出了多个模型应用。例如，阿里巴巴的通义千问、字节跳动的豆包、腾讯的混元、月之暗面的Kimi，均是基于自研模型构建的AI应用。然而，不可否认的是，在AI赛道百舸争流的早期阶段，即2023年至2024年初，国内模型整体与国际先进水平仍存在明显差距，国内表现较优的模型仅能与GPT-3.5或GPT-4勉强对标或持平，尚不足以引起国外主要竞争者的重视。在这一时期，国际模型在技术领先性与生态影响力上依然占据主导地位。

这一差距的主要原因是生成式LLM的训练成本和时间积累庞大。除非在模型架构层面出现有效创新，否则想从零开始迎头赶上难度极大。直到2024年12月，国内的深度求索（DeepSeek）公司首次发布的DeepSeek系列模型——DeepSeek-V3，才真正引起全球的广泛关注与震动。

DeepSeek-V3的参数规模达6710亿，性能和世界顶尖的闭源模型GPT-4o及Claude-3.5-Sonnet不相上下。DeepSeek官网公布的模型评估结果如图1-15所示。

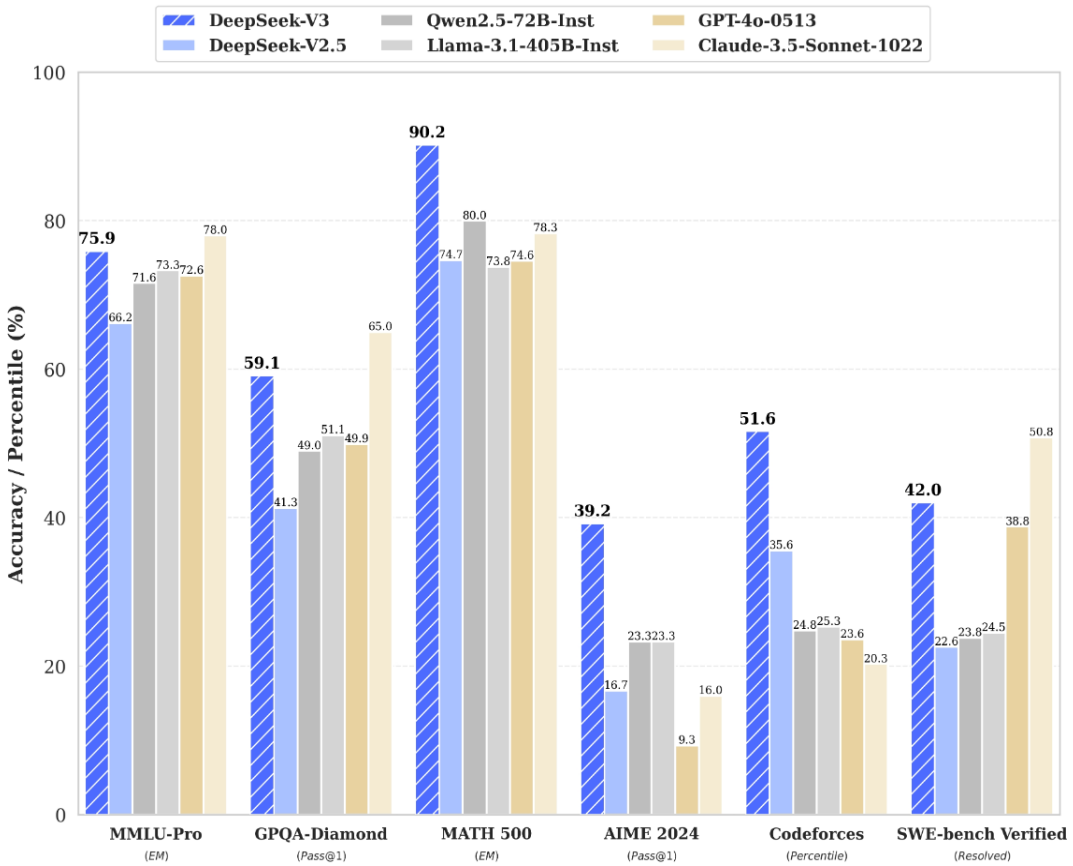

图 1-15　DeepSeek 官网公布的模型评估结果

值得一提的是，得益于DeepSeek-V3优化后的模型架构和训练手段，其训练成本极低，仅约560万美元，而OpenAI训练GPT-4的成本已超过1亿美元。由于训练成本低，DeepSeek-V3的API收费也显著低于其他竞品模型。以百万tokens为单位，在不考虑优惠时段的情况下，DeepSeek API的收费约为GPT-4o的几十分之一，如图1-16和图1-17所示。

这种训练成本与性能表现之间的强烈对比，打破了“依赖堆砌算力资源实现AI进步”的传统认知，证明了在具备先进强化学习技术的前提下，低训练成本即可实现弯道超车。这一突破重新让人们的关注焦点回归到算法创新与模型架构本身。

Pricing

Latest models

New: Save on synchronous requests with flex processing.

Text tokens

Price per 1M tokens · Batch API price

Model	Input	Cached input	Output
gpt-4.1 ↳ gpt-4.1-2025-04-14	$2.00	$0.50	$8.00
gpt-4.1-mini ↳ gpt-4.1-mini-2025-04-14	$0.40	$0.10	$1.60
gpt-4.1-nano ↳ gpt-4.1-nano-2025-04-14	$0.10	$0.025	$0.40
gpt-4.5-preview ↳ gpt-4.5-preview-2025-02-27	$75.00	$37.50	$150.00
gpt-4o ↳ gpt-4o-2024-08-06	$2.50	$1.25	$10.00
gpt-4o-audio-preview ↳ gpt-4o-audio-preview-2024-12-17	$2.50	-	$10.00
gpt-4o-realtime-preview ↳ gpt-4o-realtime-preview-2024-12-17	$5.00	$2.50	$20.00
gpt-4o-mini ↳ gpt-4o-mini-2024-07-18	$0.15	$0.075	$0.60
gpt-4o-mini-audio-preview ↳ gpt-4o-mini-audio-preview-2024-12-17	$0.15	-	$0.60
gpt-4o-mini-realtime-preview ↳ gpt-4o-mini-realtime-preview-2024-12-17	$0.60	$0.30	$2.40
o1 ↳ o1-2024-12-17	$15.00	$7.50	$60.00

图 1-16　OpenAI API 每百万 tokens 收费

模型[1]		deepseek-chat	deepseek-reasoner
上下文长度		64K	64K
最大思维链长度[2]		-	32K
最大输出长度[3]		8K	8K
标准时段价格 (北京时间 08:30-00:30)	百万tokens输入（缓存命中）[4]	0.5元	1元
	百万tokens输入（缓存未命中）	2元	4元
	百万tokens输出[5]	8元	16元
优惠时段价格[6] (北京时间 00:30-08:30)	百万tokens输入（缓存命中）	0.25元（5折）	0.25元（2.5折）
	百万tokens输入（缓存未命中）	1元（5折）	1元（2.5折）
	百万tokens输出	4元（5折）	4元（2.5折）

图 1-17　DeepSeek API 每百万 tokens 收费

DeepSeek-V3不仅性能卓越，训练成本与服务收费也更低，且其完整的模型架构、参数、训练方法及部署评估方案已对外开源，构建了可落地的社区生态体系。这极大地降低了私有化部署DeepSeek模型与应用落地的门槛，也对全球AI生态产生了深远影响。相较之下，目前全球顶尖模型通常是闭源的，如OpenAI发布的GPT-3.5及以上系列模型，用户只能通过OpenAPI的方式来消费，无法通过私有化部署、微调等方式进行私有化商用落地。

这些重磅举措已经把全球AI领域搅得天翻地覆，但风波还未平息。仅一个月后，2025年1月，深度求索公司发布了DeepSeek-R1模型。该模型延续了DeepSeek-V3的核心优势：卓越的性能、低训练成本和收费，并继续开源核心架构与参数权重等信息。更进一步，DeepSeek-R1还提供了先前模型所未支持的深度思考能力。

DeepSeek-R1模型能够展示其思考过程，具备更好的情绪理解和上下文分析能力。例如，如果我们用烦躁焦虑的语气询问：“好烦，能不能帮我写个演讲的开场白啊？”DeepSeek-R1能在思考过程中体现出它已经分析出用户的情绪状态，并做出有针对性的回复调整，如图1-18所示。

DeepSeek-R1优质的模型输出体验和深度思考能力令人眼前一亮，给人的第一感觉是：作为一款生成式LLM，它更像人类了。这场风波彻底改变了中国乃至世界的AI布局，OpenAI的垄断地位逐渐被打破。基于DeepSeek的开源模型权重和架构，其他模型有了更进一步的创新空间，国产优质模型也开始在全球舞台上崭露头角，逐步与国际竞争者同台竞技。

图 1-18　DeepSeek-R1 使用示例

1.1.6　百花齐放：Agent

回顾生成式LLM的发展历程，不禁让人感慨万千——从Transformer架构的奠基，到如今模型基建层的繁荣，生成式LLM的演进之路确实非同寻常，它的时间线如图1-19所示。

模型基建层的百花齐放，不仅推动了AI应用层的发展，也激发了更深层次的创新。早期的对话式聊天已让人惊艳，而随着模型的不断进化，开发者和用户渐渐不再满足于单纯的对话交互，开始探索具备自主反应、思考和行动能力的AI智能代理——AI Agent，现在习惯称之为智能体。

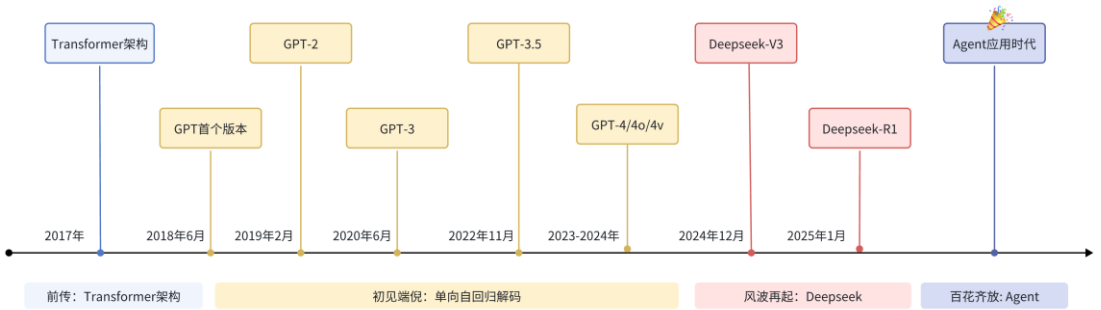

图 1-19　生成式 LLM 时间线

1. Single Agent

早期，由于模型能力的限制、幻觉问题以及Agent工程化经验尚不成熟等原因，Agent的设计相对简单。通常，Agent会在某个垂类场景中接入模型API，绑定一些工具类或交互组件，在对话式基础上进行特定的交互。这种在垂类领域专注于完成某一任务的Agent被称为Single Agent。例如，笔者在2023年初，ChatGPT刚刚引起广泛关注时，便开发过一个自动生成单元测试（简称单测）的Agent，架构如图1-20所示。

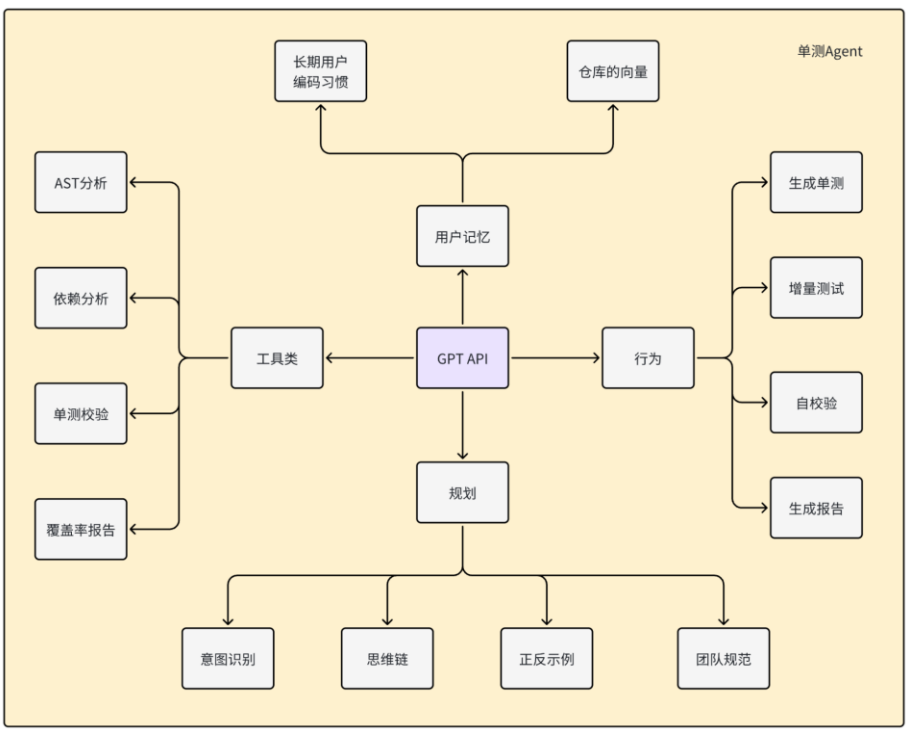

图 1-20　单测 Agent 的架构设计

上述单测Agent的架构设计包含几个工程化模块：

（1）用户记忆：通常缓存用户的短期和长期记忆行为，以使Agent能根据用户的喜好和习惯做出响应，达到千人千面的效果。

（2）规划：预置一些规划策略，如意图识别和思维链等，协助模型在垂类场景中做出更符合预期的行为，随着模型自身能力的提升，规划部分的作用逐步弱化。

（3）工具/行为：工程化内置的一些能力函数，Agent做出判断后，由工程系统调起指定能力以实现行为的效果。工具和行为可以设计为同一领域，也可以根据需求异构为两个模块。

这种设计就是Agent模式中最常用的ReAct（Reason+Action）架构：通过输入确定规划，再由规划制定行为。通过这种架构，即使单测场景的工具链复杂，仍能进行有效整合和联动，生成高质量且可验证的单测。不过，该产品只用于单测这个明确的垂类场景，缺少任务和步骤间的流转，也没有多个Agent的协调合作，因此仍属于Single Agent模式。

当场景变得复杂，涉及更多节点和流转时，单一的工程化处理就不再足够。LLM的幻觉问题在复杂场景中会随着节点流转而逐步加剧。

2. Multi Agent

尽管使用LLM已能较好地完成某些垂类任务，但这并没有满足人们对Agent的预期。更激进的想法是，真正的Agent能够像人类一样，主动拆解任务、完成节点流转和校验，最终输出高质量的结果。这样的Agent被称为Multi Agent。虽然开发者们的预期很高，但由于模型幻觉、工程化范式尚未成熟等原因，真正能被称为Multi Agent且效果不错的产品并不多。

在这些产品中，较为典型的案例是Manus.ai，它由中国的Monica.im团队研发，并于2025年3月发布。作为全球首款通用型Agent产品，Manus.ai可以支持旅游规划、股票分析、教育内容等40多种领域的复杂产品生成和交互，用户可以通过Web来使用，如图1-21所示。

图 1-21　Manus.ai 首页示例

虽然从外部交互看，其整体形态与ChatGPT等类似，但Manus.ai支持对复杂任务的拆解，逐步执行并呈现给用户。以北医三院附近常住人口及疾病流行率预测为例，整个过程如图1-22所示。

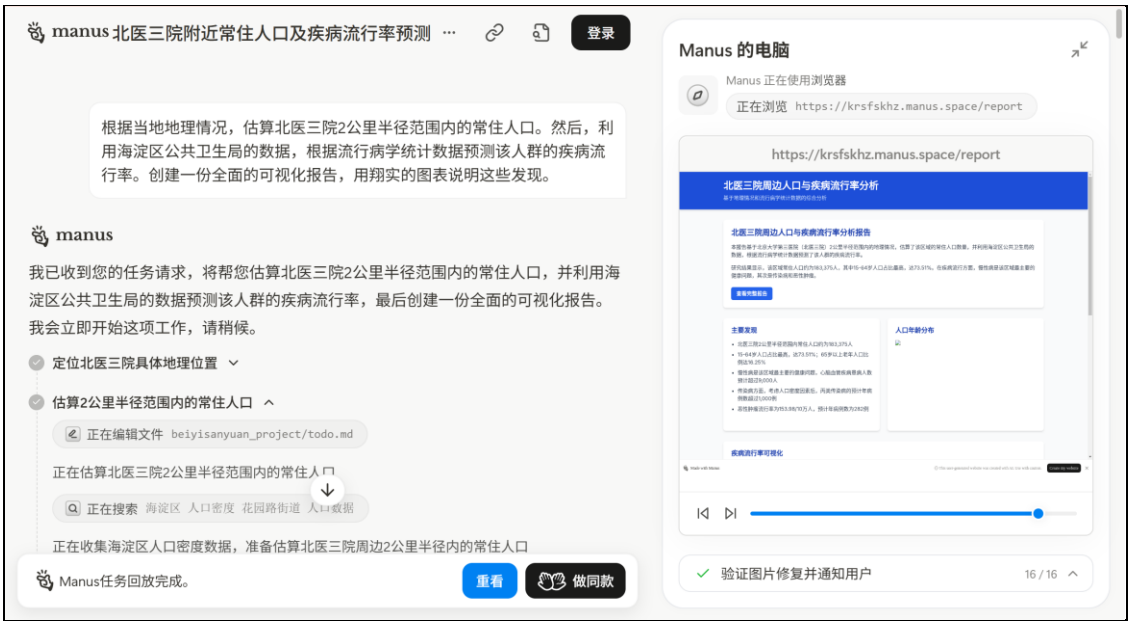

图 1-22　Manus.ai 输出流行病预测的过程

在流行病预测任务中，Manus.ai将任务拆解为8个步骤：

01 定位北医三院的具体位置。
02 估算 2 公里半径范围内的常住人口。
03 获取海淀区公共卫生局相关数据。
04 收集并分析流行病学统计数据。
05 分析目标人群疾病流行率。
06 创建数据可视化图表。
07 编制综合分析报告。
08 提交最终报告和相关文件。

在完成所有的步骤后，Manus.ai最终会输出如图1-23所示的交付产物。

与ChatGPT等对话式的AI应用相比，Manus.ai不仅提供了文本咨询功能，还能够拆解复杂任务并逐步执行，整体效果令人惊艳。同时，Manus.ai的架构设计也很有趣，它的整个执行过程通过虚拟机执行不同脚本完成。例如，通过Python脚本完成一些统计任务，如图1-24所示。

图 1-23　Manus.ai 输出的流行病预测产物

图 1-24　Manus.ai 通过 Python 脚本完成统计任务

即使是对整个任务的拆解，Manus.ai也依然采用markdown的方式来呈现，如图1-25所示。

图 1-25　Manus.ai 使用 markdown 拆解任务

Manus.ai之所以采用虚拟机结合多种类型文件拆解的方式来执行任务，主要是因为它是Web形态的Agent，受限于浏览器的安全沙箱机制，无法通过网页或JavaScript直接访问用户的文件系统、终端、网络接口等本地系统资源。Manus.ai的整个流程如图1-26所示。

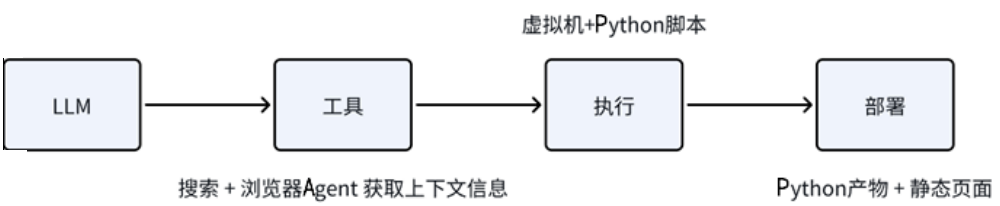

图 1-26　Manus.ai 流程简图

当然，实际架构比这个简图复杂得多，这只是Manus.ai单次任务的宏观流程。例如，LLM层需要异构出独立元素来完成任务的制定、执行和自校验等，以保证整个复合型任务执行稳定。本节不展开介绍，将在后续章节中通过实例说明。

基于浏览器这一载体的特性，Manus.ai的输入/输出存在一定局限性。例如，在生成PPT或文档等产物时，需要依赖Python强大的自动化生态以脚本形式完成，而不能直接调起对应的软件。同时，在部署产物时，由于无法调起用户本地计算机的端口，除了Python产物外，Manus.ai提供的产物多以静态页面为主。

基于上述讨论，Multi Agent的能力取决于以下几个关键点：

- 模型能力：影响Agent执行过程中的幻觉程度；同时，更多的token支持也可以弱化工程化在Multi Agent中的占比，避免过度设计。
- 承载形态：不同的承载形态决定了Multi Agent的控制面，例如浏览器的沙箱形态限制了Multi Agent对用户计算机的控制，在能力上也有局限。
- 工具链：任何Agent本质上只能进行决策和规划，可选的工具链才是Agent执行产物的关键，丰富的工具链能够扩展Agent的上限，为此社区里推出了Function Calling、MCP、A2A等方式扩展Agent工具链，这个在后文中也会详细介绍。
- 步骤调度能力：由于模型的能力仍存在限制，因此还需要一定的工程化架构设计来缓解模型的幻觉，并将多个过程节点有秩序地调度起来。针对不同的场景，也需要定制。

不难发现，Manus.ai在ReAct架构的基础上额外增加了调度能力，用于流转步骤节点之间的能力，这个调度模块被称为Supervisor，并在Multi Agent系统中有着广泛应用。然而，由于Manus.ai的承载形态是浏览器应用，存在与用户计算机的沙箱安全隔离，在一定程度上限制了其产物的上限。

提到承载形态，就不得不提及另一个更优的Multi Agent形态——IDE（Integrated Development Environment，集成开发环境）。尽管IDE通常应用于编程环境，但作为Agent载体，它其实具有得天独厚的优势。相比浏览器等沙箱环境，面向开发者的IDE在授权的情况下，可以自由调起个人计算机中的终端，并在端口中绑定进程服务，同时还可以根据用户计算机的实际情况进行兼容性编程。由此，得到的产物形态和质量上限会比其他承载形态更高。

在AI IDE领域，最为知名的Multi Agent产品就是Cursor。它由Anyspere公司开发，集成了GPT-4、Claude-4等编码效果突出的模型，除了提供代码补全、问答等基于AI的编码提效能力外，还提供了Agent能力。它支持从零搭建指定项目，涵盖图形界面、服务部署、进程绑定等在内的软件开发全流程，生成的质量也是编码类Agent中相对较高的，如图1-27所示。

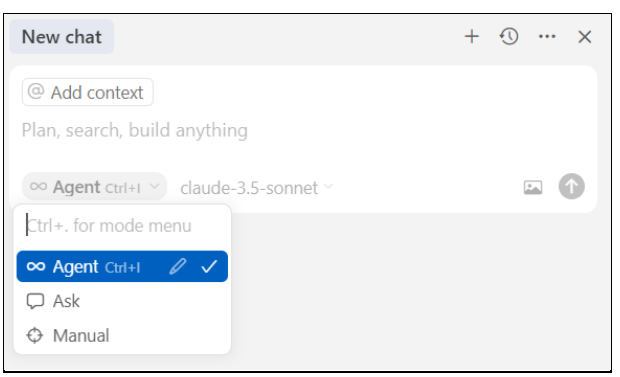

图 1-27　Cursor Agent 入口

使用Cursor Agent可以将想法快速从零实现，例如让Cursor Agent从零实现一个贪吃蛇游戏，1分钟不到就可以生成一个可运行的贪吃蛇游戏，如图1-28所示。

图 1-28　Cursor Agent 使用示例和产物效果

整个过程几乎不需要开发者介入，即使遇到不满意或需要调整的地方，也只需在Agent窗口中继续追问并迭代需求即可。与其说Cursor Agent的诞生提升了开发者的效率，不如说它真正降低了每个人将想法落地为产品的门槛。以往要将想法落地，需要具备开发技术并投入大量时间进行研发；如今，依托Cursor Agent的能力，在原型图不严格的场景下，可以快速完成一个MVP（Minimum Viable Product，最小可行性产品）并投入使用。

相比早期的对话式AI应用或单一功能的Agent，如今的AI应用场景日益复杂，功能也往往相互联动。可以预见，未来Agent的发展趋势将更加复杂，并朝着复合型Agent的方向演进。这里所说的“复合型Agent”并非单指Single Agent或Multi Agent等工程角度的架构设计，而是从宏观产品层面出发，能够适配未来多样化、复杂业务场景的软件Agent交互形态，能为用户在真实世界中完成真正有价值的任务和创造更大的收益。

3. Physical Agent

前文提到，复合型Agent很可能是未来3~5年的热门方向。然而，无论复合型Agent采用何种工程化架构，其本质仍停留在软件层面，尚未与现实生活直接结合，所产生的结果主要是数字产物，难以对人们的生活造成物理层面的直接影响。

除了精神和知识的效率提升，推动物理世界的变革也同样重要。例如，宇树科技研发的AI机器人，便是将Agent能力延伸到现实物理层面的实践案例，并因此获得了数亿元人民币的融资。尽管当前模型能力和硬件协同还不足以支撑大规模的物理落地，但从更长时间尺度来看，Physical Agent（物理智能体）极有可能成为未来的重要方向。

芝加哥大学的Fouad Bousetouane于2025年1月发布的论文*Physical AI Agents: Integrating Cognitive Intelligence with Real-World Action*中提到：Physical AI Agent具备颠覆性的潜力，并提出了基于Perception（感知）、Cognition（认知）和Actuation（驱动）的Physical Agent架构范式，为未来Agent从虚拟走向现实提供了可参考的方向，如图1-29所示。

图 1-29　Physical Agent 架构范式

在这一架构范式中，感知和驱动主要处于硬件层面，用于提供和触发现实世界的交互媒介，例如视觉、听觉、触觉及机械动作等；而认知层则由复合型Agent承担，负责调度和控制感知与驱动之间的高效联动。硬件层的多项关键技术在当前已经相对成熟，但真正决定Physical Agent能力上限的依然是位于认知层的复合型Agent。

因此，复合型Agent不仅在当前AI应用中扮演着举足轻重的角色，更是未来Physical Agent架构中不可或缺的重要一环，将在推动虚拟智能体向真实物理世界演化的进程中发挥变革性作用。

1.2　复合型 AI Agent 的市场前景

前文介绍了AI Agent的演进之路，从LLM的诞生到具备自主决策和行动能力的AI Agent，这一过程经历了漫长的发展与积淀。如今，复合型AI Agent正如雨后春笋般涌现，正在深刻改变着世界的运作方式。无论是社区讨论还是各大开发者大会，都在为Agent的开发布局提供支持，其重要性不言而喻。

在此前的小节中提到，在不考虑模型能力变更的前提下，复合型Agent的开发效率与能力上限的高低主要取决于架构设计是否合理，以及是否拥有完善、丰富的工具链支持。基于这一共识，开发者社区从2024年起便持续推动降低Agent开发的工程化成本，同时试图在Agent与工具之间建立稳定的连接与共享关系。例如，推出用于低代码构建Agent工作流的Coze（即扣子），简化Agent服务搭建的框架LangChain、LangGraph，以及用于工具链连接与Agent通信生态的MCP（Model-Context Protocol，模型上下文协议）和A2A（Agent-to-Agent，智能体到智能体）等，整个生态正逐步充实完善，为Agent的规模化落地奠定坚实基础。

另一方面，在2025年5月举办的多场国际大型开发者大会中，Agent均成为核心主题并被大力推广。例如，在Microsoft Build开发者大会（见图1-30）与Google I/O大会上，均发布了数十款全新

Agent产品，全面押注Agent赛道，持续丰富智能体生态。微软全球首席传播官Frank X. Shaw更是公开表示：“我们已迈入AI智能体时代。”

图 1-30　Microsoft Build 2025 议题

不仅国外科技公司在大力投入Agent领域，国内公司也在积极布局、竞相角逐。以类Cursor的AI IDE产品为例，BAT（百度公司、阿里巴巴集团、腾讯公司三大公司首字母缩写）等国内大厂已推出超过4款同类产品，包括阿里的通义灵码IDE、Qoder、Opensumi IDE框架，字节跳动的Trae IDE，以及腾讯的CodeBuddy等。此外，在大厂业务线内部，围绕Agent能力孵化的各类产品更是数不胜数，几乎每条业务线都在探索如何将Agent融入具体场景并形成产品竞争力。

与此同时，各家公司对Agent方向人才的招聘需求和投入也在持续增长，市场对具备Agent工程化和应用落地能力的开发者需求旺盛，如图1-31所示。这一趋势正在快速推动Agent技术从探索走向规模化落地。

图 1-31　Agent 开发的招聘需求

由此可见，复合型AI Agent及其研发人才在未来相当长的一段时间内都将处于高度紧缺状态。学习和掌握AI Agent的研发能力，将为相关从业者带来显著的个人成长与丰厚的职业回报。

1.3　本书的内容安排

随着LLM的快速发展，人类对AI的应用也在不断深入演进：从早期的对话式交互，到后来的链式调用的简单AI应用，再到如今能适配多场景、身兼多职的复合型AI Agent。当前的LLM，就像一个信息广博、思维紧密的大脑，但它仍缺乏直接处理输入和输出的能力，也需要人类介入来监督和保障完整流程的正确执行。

要想将LLM真正整合为可投产落地、持续创造价值的Agent，仍需大量的工程化手段加持。工程化的作用在于，一方面为Agent提供丰富且稳定的工具链，支持多种形态的输入与输出能力；另一方面，通过工程辅助手段联动Agent的各个流程节点，缓解幻觉和偏差等问题，让Agent能够稳定高效地服务真实业务场景。

如今，Agent开发与工程化已形成庞大而复杂的知识图谱，但在社区与现有书籍中，仍缺乏体系化且示范完整的实践资料，使得许多想进入AI新时代的读者无从下手。本书正是希望填补这一空白，作为读者学习AI Agent开发的起点，帮助大家在AI新时代中扬帆起航。

在章节设计上，遵循由浅入深的原则，除本章绪论外，设置了7个相互关联、循序渐进的工程化章节，每章均配备可独立运行的完整示例，帮助读者从零掌握复合型AI Agent开发的理论与实践，从而快速构建系统化知识体系并具备真实落地能力。具体章节结构如图1-32所示。

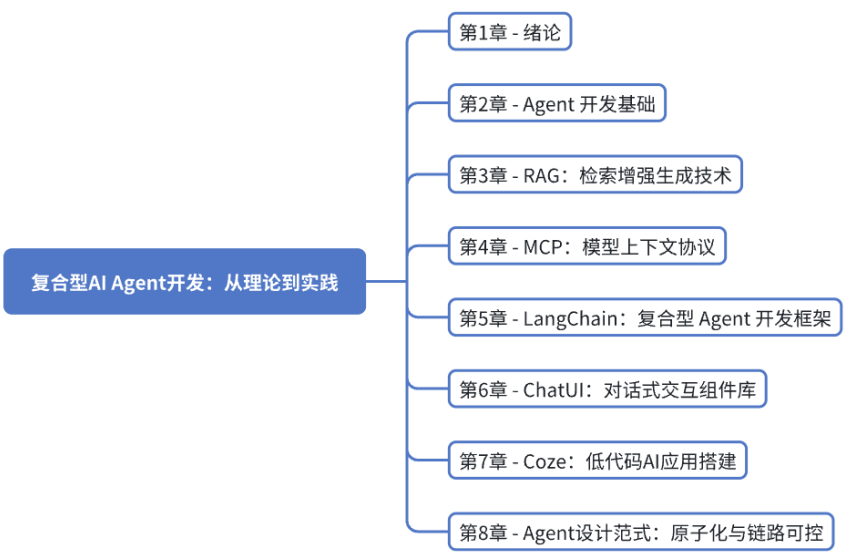

图 1-32　本书章节设计的思维导图

各章内容均围绕Agent开发与工程化实践展开，具体安排如下：

第1章为绪论，详细讲解AI Agent的演进历程，从Transformer架构到GPT系列诞生，再到DeepSeek掀起浪潮，直至Agent逐步渗透新时代的每个角落。时代的年轮推动了AI Agent的市场前景，也奠定了本书的写作初衷。

第2章将系统介绍Agent开发基础，从Agent核心三要素（LLM、Prompt、Action）出发，讲解最简单的Agent是如何运作的，并基于示例深入解析Model API、Prompt Engineering和Function Calling，最后从全局视角介绍现代Agent常见架构及工程化设计范式，为后续框架和模块的学习打下坚实基础。学完本章后，读者将具备基本的Agent开发能力，并对核心模块与Agent全局架构设计有完整认知。

第3章将聚焦RAG（检索增强生成技术）。RAG是成熟的数据相似匹配技术，通过向量化和向量存储，利用向量距离衡量数据相似度，可高效构建垂类私域场景下的知识库并服务于模型的信息检索增强。本章将从零实现RAG基础设施，使用RAGAS完成召回评估，并介绍自我矫正的CRAG，帮助读者全面掌握RAG增强流程并具备独立搭建私域知识库的能力。

第4章将聚焦于模型上下文协议（Model Context Protocol，MCP）。借助MCP，不论是第三方开发者还是个人，都可以将自研能力无缝接入Agent，使传统的Agent开发方式延伸到全球能力网络。本章将从MCP的定位与价值切入，系统解析其核心作用、架构原理及工作流程，并结合实用案例，提供MCP能力封装方法与开源MCP Servers的调用示例，帮助读者完成从理论理解到实战落地的跨越，快速构建功能多样、可持续扩展的复合型现代Agent。

第5章将介绍LangChain生态。作为现代复合型Agent服务开发框架，LangChain能加速并简化Agent服务层开发流程，提供稳定、可复用能力以构建系统级Agent服务，已在诸多Agent开发岗位中占据核心地位。本章将详细介绍LangChain、LangGraph等生态模块及其在链式AI应用和复合型Agent搭建中的实践，帮助读者具备系统级Agent开发能力。

第6章将介绍专为对话式交互场景打造的前端组件库ChatUI。借助ChatUI，开发者可以在极短时间内搭建出美观、功能完善的Agent图形界面，并与LangChain实现无缝联动，大幅降低对话式Agent的前端开发门槛。本章将结合实际示例，展示如何利用ChatUI快速构建高体验度的交互界面，为Agent赋予更友好、更高效的用户交互能力。

第7章将介绍低代码Agent应用开发平台Coze。前文各章节主要聚焦于代码驱动的Agent开发，而对于模块能力和流程编排较为明确的场景，低代码方式往往能够更高效地完成开发与构建。Coze提供了完善的低代码工具链与Workflow编排能力，使开发者无须从零编码即可快速实现Agent逻辑。学习完本章后，读者将能够将低代码Agent与代码Agent灵活联动，根据不同业务场景与需求选择最优方案，从而独立完成功能丰富、结构灵活的复合型Agent开发。

第8章作为全书的收官之作，将以全局视角系统梳理Agent的核心设计范式，内容涵盖框架设计、通信设计、渲染设计、工作流优化范式、成本优化策略等关键领域。本章不仅总结前文的知识

脉络，还将从整体架构与长远演进的角度，探讨如何在功能、性能与成本之间取得最佳平衡，为读者构建可扩展、可维护、可持续优化的现代Agent体系提供完整的参考蓝图。

Agent的时代已经开始，时代的风浪愈演愈烈，希望本书能成为大家坚实的船基，帮助各位掌舵人在时代的风浪下，踏浪而行，扬帆起航！

第 2 章

Agent开发基础

2

本章将介绍Agent开发的核心基础。首先讲解构建Agent所需的三大要素：LLM（Large Language Model，大语言模型）、Prompt（提示词）与Action（动作）。这三者构成了智能体推理与执行的基本框架。接着，介绍Model API的使用方法，包括基础调用、参数配置、流式响应、system message及深度思考标签等内容。随后，探讨Prompt Engineering（提示词工程）的常用技巧，如思维链（CoT）、结构化提示、正反示例引导和分治策略。最后，介绍Function Calling（函数调用）能力，实现模型与外部工具的联动，这是构建工具增强型Agent的关键技术。通过本章的学习，读者将全面掌握Agent的基本组成、开发流程与实用技巧，具备独立构建和调优智能体的能力。

2.1 Agent 三要素：LLM、Prompt、Action

复合型Agent是近年来大语言模型应用中最具代表性的一类架构，广泛用于搜索问答、任务助手、机器人流程自动化、代码生成、数据分析等复杂应用场景。尽管不同Agent的实现路径各异，但它们本质上都由三个关键组成部分驱动：LLM（大语言模型）、Prompt（提示词）、Action（动作/工具调用）。这三者共同构建了Agent感知、推理、行动的闭环系统，构成了智能体能力的核心骨架。

1. LLM：复合型 Agent 的大脑

LLM是Agent的核心推理引擎，基于海量语料训练而成，具备强大的自然语言理解和生成能力。LLM决定了Agent的“智力上限”——它能否听懂复杂语句、做出合理判断、具备上下文记忆与推理能力，完全依赖于所用模型的能力边界。

目前主流的可用于Agent构建的LLM包括：

- OpenAI的GPT-4/GPT-3.5系列（如ChatGPT）。
- Anthropic的Claude系列。
- Google的Gemini系列。
- 国内自研的Qwen（通义千问）、DeepSeek、Seed/Doubao（豆包）系列。

在具体实现中，LLM通常以API的形式提供。Agent在接收到用户输入后，将其转换为Prompt并发送给LLM，再由模型生成响应，参与后续决策流程。后续章节会结合案例详细介绍LLM API的能力和使用方式。

2. Prompt：驱动模型行为的语言接口

Prompt是Agent与LLM沟通的桥梁，决定了模型“如何理解”和“如何回应”输入。它不仅是一段文本，更是一种对模型行为的控制语言。提示词工程已成为构建强大Agent的核心技能之一。在日常使用中，Prompt可以是直接的任务描述或自然语言提问，例如：

```
“请帮我生成一份周报的开头段落。”
“列出三种提高工作效率的方法。”
```

然而，但在更复杂的场景中，仅靠简单提问往往难以获得稳定、精准的输出。此时，可以通过一些特定技巧来优化Prompt，从而显著提升模型的推理深度和任务完成效果。例如Few-shot Prompting（少样本提示）的一个示例：

```
下面是用户的提问和对应的意图分类，请根据示例判断最后一句的意图：
用户：我想预订明天晚上去北京的机票。
意图：机票预订

用户：帮我查一下附近的餐厅。
意图：餐厅推荐

用户：请提醒我下午三点开会。
意图：日程提醒

用户：我想买一台笔记本电脑。
意图：
```

与常规提问不同，Few-shot Prompting通过在提示词中提供多个示范问答对，引导模型模仿示例的格式和风格，从而提升输出的准确性和一致性。这种方法特别适合分类、意图识别、格式转换等任务。它与常规提问的差异如图2-1所示。

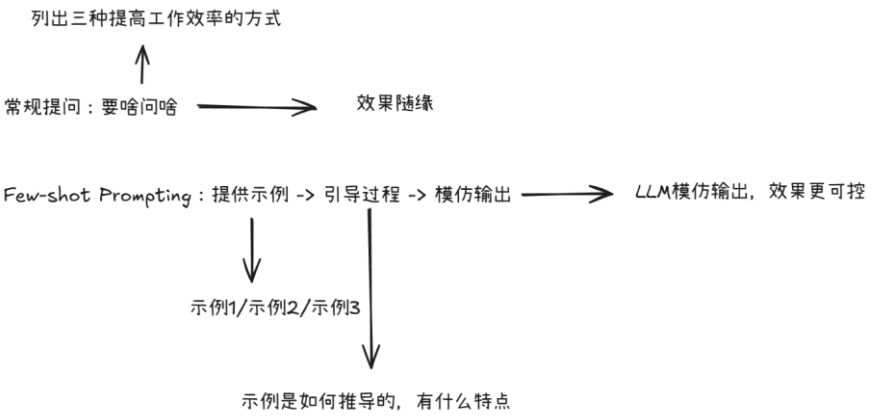

图 2-1　Few-shot Prompting 与常规提问的差异

类似Few-shot的提示设计只是提示词工程中的一种优化策略，除此之外，还有结构化输出模板（让模型输出更规整易解析）、正反示例引导（通过对比强化行为偏好）、角色设定与语境构建（通过设定身份、语气来控制模型风格）等多种手段，它们都是提升Agent表现、增强模型可控性的关键方法。后续章节将逐一展开讲解。

不难发现，通过精心设计提示词（Prompt），可以将模型从一个“泛化的大脑”变成一个“定制化的智能助手”。无论是设定角色语气、分解任务步骤，还是引导模型执行结构化任务，Prompt都是控制智能体行为表现的关键工具。在实际开发中，Prompt的优化往往是提升Agent表现最直接、最有效的手段之一。后续章节也将结合具体案例，深入探讨不同场景下的提示词工程（Prompt Engineering）策略。

3. Action：连接模型与真实世界的执行接口

Action是让模型从“说”走向“做”的关键机制。LLM本身无法访问互联网、数据库或本地环境，其生成结果仅基于上下文预测。而通过Action，Agent可调用外部工具（tool）或函数（function），执行真正的任务操作，其工作流如图2-2所示。

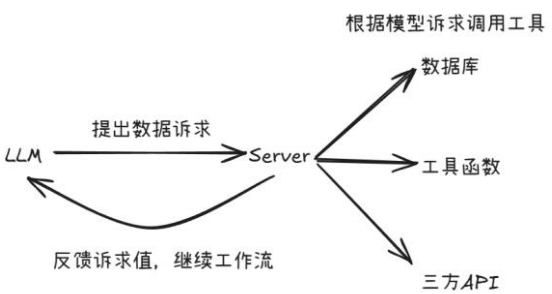

图 2-2　Action 工作流

在具体实现中，Agent会先将用户输入转换为合适的Prompt，通过调用LLM API获取模型响应，随后根据模型的输出判断是否触发对应的Action来完成任务，这就是Function Calling（函数调用）技术。后续章节将结合实例，深入讲解如何利用Function Calling技术实现模型与工具的灵活联动。

因此，一个真正可用的Agent需要LLM、Prompt和Action三者紧密协同：LLM提供语言理解与决策能力，是智能体的核心引擎；Prompt负责对输入进行结构化和语义引导，确保模型按预期完成任务；Action则将模型的决策转换为具体操作，实现任务的落地执行。三者相辅相成，共同构建了智能体从理解到推理再到执行的完整闭环，是打造高效可靠Agent的关键基础。

2.2　LLM：以 DeepSeek API 为例

目前，大多数大语言模型以云端托管服务的形式提供，通过开放API向开发者提供推理能力，这类服务称为LLM API。

OpenAI率先推出此类服务，其GPT-3.5和GPT-4通过chat/completions端点实现交互对话，成为行业标准。受其影响，很多模型厂商都采用了相似接口，以简化接入和迁移流程。本节将以DeepSeek API为例，系统讲解模型API的关键概念与实用技巧，包括：

- 基础API调用方式：如何构造请求、设置参数并处理响应。
- 流式（Streaming）请求机制：如何实现响应内容的实时处理与渲染。
- System Message（系统消息）：如何通过系统指令设定模型角色、语气与行为边界。
- DeepThinking（深度思考）标签：DeepSeek首次推出的深度思考控制机制，用于体现推理过程。

通过这一部分的学习，读者将掌握调用云端大模型的核心方法，为后续构建LLM应用打下坚实基础，本章示例将使用Python完成。

2.2.1　基础调用

在学习调用DeepSeek API之前，首先需要了解什么是API keys。API keys是由服务提供商生成的身份凭证，调用者在程序中通过请求头发送给服务器，用于身份鉴权和权限控制。除了验证身份外，API keys还可用于监控调用次数和计费管理。开发者可以登录DeepSeek官方开发平台，在“创建API keys”页面生成专属于自己的密钥，如图2-3所示。

根据官网指引，创建API keys时，系统会为账户生成唯一的密钥。调用DeepSeek API时，需要将该API keys放入请求头中，用于身份验证。

除了准备API keys，调用API前还需为账户充值，服务器会根据每次调用所消耗的token数量扣费。这里的token通常相当于一个英文单词、一个标点符号或1~2个中文字符，一次对话的总token消耗为输入（Prompt）和输出（Completion）token的总和（token也被翻译为“词元”）。DeepSeek

的收费标准在各个模型厂商中相对偏低，其2025年5月的收费标准如图2-4所示。

图 2-3　DeepSeek 官网 API keys 页面

模型(1)		deepseek-chat	deepseek-reasoner
上下文长度		64K	64K
最大思维链长度(2)		-	32K
最大输出长度(3)		8K	8K
标准时段价格（北京时间 08:30-00:30）	百万tokens输入（缓存命中）(4)	0.5元	1元
	百万tokens输入（缓存未命中）	2元	4元
	百万tokens输出 (5)	8元	16元
优惠时段价格(6)（北京时间 00:30-08:30）	百万tokens输入（缓存命中）	0.25元（5折）	0.25元（2.5折）
	百万tokens输入（缓存未命中）	1元（5折）	1元（2.5折）
	百万tokens输出	4元（5折）	4元（2.5折）

图 2-4　DeepSeek API 收费标准

创建好API keys并充值后，就可以开始调用DeepSeek API了，示例代码如下：

```
import requests     # 导入requests库用于发送HTTP请求
import json         # 导入json库用于处理JSON数据

# 设置API密钥
api_key = ''

# 设置DeepSeek API的端点URL
url = "https://api.deepseek.com/chat/completions"

# 构建消息数组，包含系统角色提示词和用户输入
```

```
    msg = [
        {"content": "You are a helpful assistant","role": "system"},# 系统角色提示词
        {"content": "你好", "role": "user"}           # 用户输入的消息
    ]

    # 构建请求体并转换为JSON字符串
    payload = json.dumps({
        "messages": msg,        # 必填字段，消息数组，包含content和role属性，至少一条消息
        "model": "deepseek-chat" # 必填字段，模型名称，可选"deepseek-chat"或
"deepseek-reasoner"
    })

    # 设置HTTP请求头，包含身份验证和内容类型
    headers = {
        'Content-Type': 'application/json',          # 请求内容类型为JSON
        'Accept': 'application/json',                # 接收的响应内容类型为JSON
        'Authorization': f'Bearer {api_key}'         # API keys，用于身份验证
    }

    # 发送POST请求到DeepSeek API，获取响应
    response = requests.request("POST", url, headers=headers, data=payload)

    # 打印响应类型和内容
    print(response)                                  # 打印API返回的响应
```

在上述示例中，完整展示了如何从零开始调用 DeepSeek API：

（1）首先通过设置请求地址和API keys来完成身份认证；接着构造标准的请求体并转换为JSON字符串，其中messages字段包含一组对话消息，每条消息都包含角色（如user或assistant）和消息内容（content），model字段用于指定调用的模型类型（如"deepseek-chat"）。

（2）设置请求头，确保包括Authorization（携带API keys）和Content-Type（指定为application/json）。

（3）使用Python的requests库发起POST请求并接收模型返回的响应文本。

运行代码，效果如图2-5所示。

```
{"id":"2a6fc8d1-51fb-4e9b-a2b6-d3b23f798664","object":"chat.comple
tion","created":1750468681,"model":"deepseek-chat","choices":[{"in
dex":0,"message":{"role":"assistant","content":"你好！😊 很高兴见
到你～有什么我可以帮你的吗？"},"logprobs":null,"finish_reason":"st
op"}],"usage":{"prompt_tokens":9,"completion_tokens":15,"total_tok
ens":24,"prompt_tokens_details":{"cached_tokens":0},"prompt_cache_
hit_tokens":0,"prompt_cache_miss_tokens":9},"system_fingerprint":"
fp_8802369eaa_prod0425fp8"}
```

图 2-5　DeepSeek API 调用示例

可以看到，DeepSeek API的返回结果是一个JSON格式的字符串。为了便于查看和调试，下面对它进行格式化处理，使其结构清晰、层级分明，如图2-6所示。

图 2-6　DeepSeek API 结构化输出

图2-6中展示的外层JSON字段，包含了本次模型调用的基本信息，如调用时间、请求标识符、所使用的模型类型、模型返回的回复内容，以及本次请求中输入与输出token的使用统计。这些字段共同构成了完整的响应结构，详细说明如表2-1所示。

表 2-1　DeepSeek API 外层基础字段

字　　段	类　　型	说　　明
Id	string	请求标识符，唯一标识本次请求，可用于请求追踪或调试
Object	string	本次返回的对象类型
Created	number	创建聊天完成时的时间戳
Model	string	使用的 DeepSeek 模型名称
choices	object	模型响应回复的相关信息结构体
usage	object	标识本次模型调用的 token 消耗情况
system_fingerprint	string	标识当前模型的配置版本信息

值得一提的是，上述的choices数组字段与调用的响应直接相关，记录了模型补全的结果。数组中的每个元素代表一次独立的回复，适用于多轮回复或多样性生成的场景。每个回复中包含多个字段，用于描述具体的消息内容、消息角色以及生成状态等信息。该字段的详细说明如表2-2所示。

表 2-2　DeepSeek API 响应体中的 choices 字段

字　段	类　型	说　明
index	number	choices 数组的元素索引，0 代表第一个回复
message	object	模型生成的 completion 消息
finish_reason	string	模型停止生成 token 的原因
logprobs	object	该 choice 的对数概率信息，可为 null

choices数组中的message字段用于记录模型生成的补全消息（completion），是对模型输出内容的详细描述。它通常包含两个核心字段：role表示消息的角色（如"assistant"），content表示模型生成的具体文本内容。message字段的详细说明如表2-3所示。

表 2-3　DeepSeek API 响应体中的 message 字段

字　段	类　型	说　明
role	string	标识对话的角色，取值包含 system、user、assistant、tool、function
content	string	对话的内容，可以是任意文本，单位为 token

2.2.2　请求参数

在调用大模型接口的过程中，请求参数的设置对模型响应的质量、随机性以及调用成本都有显著影响。OpenAI接口提供了丰富的参数控制能力，而DeepSeek API在此基础上保留并扩展了多个关键参数，供开发者根据实际需求进行灵活配置。

这些参数不仅可以用来控制生成内容的风格与稳定性，比如设定回答是否更加严谨或更具创意，还可用于控制token消耗、是否采用流式响应等性能与成本维度的优化。详细参数说明如表2-4所示。

表 2-4　DeepSeek API 请求参数说明

字　段	类　型	是否必填	说　明
model	string	是	指定使用的模型
messages	array	是	聊天上下文消息数组，必须包含至少一条用户信息
max_tokens	number	否	限制一次请求中模型输出 completion 的最大 token 数，默认值为 4096
temperature	number	否	用于控制回答的随机性。值越大生成的内容越发散，值越小则生成的内容越严谨稳定
top_p	number	否	作为调节采样温度的替代方案，控制候选词的范围，top_p 越小，对候选词的剪枝比例越多
stream	boolean	否	指定响应消息是否为流式输出，默认值为 false
stop	string	否	用于指定模型生成内容的终止符，匹配后提前终止输出

（续表）

字　　段	类　　型	是否必填	说　　明
presence_penalty	number	否	用于惩罚重复话题，提高输出新颖度
frequency_penalty	number	否	值越高，token 在已有文本中因重复次数多而受到的惩罚就越高，从而减少重复内容，默认值为 0
response_format	object	否	指定模型必须输出的格式，支持 text 和 json_object，默认值为{"type":"text"}
tools	array	否	定义可调用的外部函数列表（用于 Function Calling），默认值为 null，后续小节会详细说明

值得一提的是，temperature 与 top_p 是两个常用于调控模型输出随机性与创造性的采样策略参数，它们在实际应用中往往被混淆，实则各有侧重：

- temperature（温度）：作用于模型输出的概率分布本身。较高的值（如1.0）使输出更加多样且不确定；较低的值（如0.2）会趋向于输出概率最高的token，生成内容更确定但较保守。
- top_p（核采样）：作用于模型输出的候选集合。它按概率累计对token进行裁剪，仅从累计概率不超过top_p的token中进行选择，能在一定程度上剔除“长尾”词，平衡多样性与合理性。

这两个参数在调优策略上通常只需要设置其一，以避免模型行为不稳定。在实际场景中：

- 若追求输出内容的稳健与可控(如代码生成、问答场景),可适当降低top_p(如设置为0.2);
- 若应用场景偏向于开放式创作（如写作、角色扮演、脑暴等），则可设为默认值或适当提高值，以提升生成内容的多样性和创造力。

除此之外，其他参数也在模型的输出方式和交互能力上发挥着关键作用，例如可实现实时响应的流式输出（stream）、支持结构化结果解析的格式设置（response_format），以及具备调用外部函数能力的Function Calling（tools与tool_choice）等，这些将在后续小节中逐一展开详解。

2.2.3　流式请求

前一小节中对DeepSeek模型的请求参数进行了整体梳理，包括必选参数与可选参数的基本含义与用途。本节将重点介绍可选参数中的stream参数——用于控制流式输出的机制。

流式是一种数据处理模式，它允许模型将生成内容以分块（chunk）的形式逐步输出，而非一次性返回完整响应。这种机制在处理大文本、提高响应速度、优化用户体验等方面具有显著优势。特别是在WebSocket或SSE（Server-Sent Events）等网络传输协议中，流式传输可以实现“边生成边显示”的效果，使用户无须等待整个响应完成，即可开始阅读模型生成的内容。

在DeepSeek API中，stream参数的类型为boolean，默认值为false，表示关闭流式输出。若希望启用该特性，只需在请求体中显式设置：

```
    payload = json.dumps({
        "messages": msg,
        "model": "deepseek-chat",
        "stream": True, # 启用流式输出
        "max_tokens": 60,
    })
```

流式请求发送后，服务器通过 SSE 协议以 text/event-stream 格式分批推送生成内容。客户端通过处理可读流，逐块接收并打印数据，实现实时渲染。以下示例将展示客户端如何获取和处理流式数据。

```
    import requests     # 导入requests库，用于发送HTTP请求
    import json         # 导入json库，用于处理JSON数据

    api_key = ''        # 设置DeepSeek API的访问密钥
    url = "https://api.deepseek.com/chat/completions"  # 设置DeepSeek API的端点URL

    msg = [
        {"content": "You are a helpful assistant", "role": "system"},  # 系统角色提
示词
        {"content": "宋朝延续多少年,输出不超过50字", "role": "user"}   # 用户问题
    ]  # 定义对话消息列表，包含系统提示和用户问题

    payload = json.dumps({                # 构建API请求的负载数据并转换为JSON字符串
        "messages": msg,                  # 设置对话消息
        "model": "deepseek-chat",         # 指定使用的AI模型
        "stream": True,                   # 启用流式输出
        "max_tokens": 60                  # 设置最大生成token数
    })

    headers = {  # 设置HTTP请求头
        'Content-Type': 'application/json',      # 指定请求内容类型为JSON
        'Accept': 'application/json',            # 指定接收的响应类型为JSON
        "Authorization": f"Bearer {api_key}"     # 设置API认证信息
    }

    # 逐块处理响应
    try:
        response = requests.request("POST", url, headers=headers, data=payload,
stream=True)                                  # 发送POST请求并获取响应
        response.raise_for_status()           # 检查HTTP响应状态，如果不是200则抛出异常

        for chunk in response.iter_lines():  # 逐行读取流式响应
            if chunk:  # 如果数据块不为空
```

```
                decoded_chunk = chunk.decode('utf-8').strip()  # 将字节流解码为字符串并
去除首尾空白
                print(decoded_chunk)  # 打印解码后的数据块

    except requests.exceptions.RequestException as e:
        print(f"\n请求失败: {e}")      # 打印错误信息
```

在上述示例中，通过在请求体中设置"stream": true，服务端被指示采用流式输出模式，分块返回生成内容；在调用requests.request()时设置 stream=True，客户端以流式方式接收响应数据，二者协同配合，才能实现端到端的流式交互。随后使用response.iter_lines()逐行解析SSE格式的响应（通常以"data:"开头并以换行分隔），并通过UTF-8解码字节流为字符串，最终打印出每个数据块的内容。效果如图2-7所示。


```
data: {"id":"25709ee2-d8f8-4f5d-b5cd-bf5e0e19c62d","object":"chat.completion.chunk"
,"created":1751075613,"model":"deepseek-chat","system_fingerprint":"fp_8802369eaa_p
rod0623_fp8_kvcache","choices":[{"index":0,"delta":{"role":"assistant","content":""
},"logprobs":null,"finish_reason":null}]}
data: {"id":"25709ee2-d8f8-4f5d-b5cd-bf5e0e19c62d","object":"chat.completion.chunk"
,"created":1751075613,"model":"deepseek-chat","system_fingerprint":"fp_8802369eaa_p
rod0623_fp8_kvcache","choices":[{"index":0,"delta":{"content":"宋朝"},"logprobs":nu
ll,"finish_reason":null}]}
data: {"id":"25709ee2-d8f8-4f5d-b5cd-bf5e0e19c62d","object":"chat.completion.chunk"
,"created":1751075613,"model":"deepseek-chat","system_fingerprint":"fp_8802369eaa_p
rod0623_fp8_kvcache","choices":[{"index":0,"delta":{"content":"延续"},"logprobs":nu
ll,"finish_reason":null}]}
data: {"id":"25709ee2-d8f8-4f5d-b5cd-bf5e0e19c62d","object":"chat.completion.chunk"
,"created":1751075613,"model":"deepseek-chat","system_fingerprint":"fp_8802369eaa_p
rod0623_fp8_kvcache","choices":[{"index":0,"delta":{"content":"约"},"logprobs":null
,"finish_reason":null}]}
data: {"id":"25709ee2-d8f8-4f5d-b5cd-bf5e0e19c62d","object":"chat.completion.chunk"
,"created":1751075613,"model":"deepseek-chat","system_fingerprint":"fp_8802369eaa_p
rod0623_fp8_kvcache","choices":[{"index":0,"delta":{"content":"320"},"logprobs":nul
l,"finish_reason":null}]}
```


图 2-7　DeepSeek API 流式请求的返回内容

图2-7展示了连续的流式输出数据块，每个数据块均以“data:”开头，后跟JSON格式内容，其结构化数据如图2-8所示。


```
{
    "id": "25709ee2-d8f8-4f5d-b5cd-bf5e0e19c62d",
    "object": "chat.completion.chunk",
    "created": 1751075613,
    "model": "deepseek-chat",
    "system_fingerprint": "fp_8802369eaa_prod0623_fp8_kvcache",
    "choices": [
        {
            "index": 0,
            "delta": {
                "role": "assistant",
                "content": ""
            },
            "logprobs": null,
            "finish_reason": null
        }
    ]
}
```


图 2-8　流式响应的结构化输出

在上述结构化输出中，每个数据块都包含了模型生成的部分内容及其相关元信息，如生成的文本片段、消息角色、调用的模型名称和当前请求的唯一标识符等。这些信息有助于客户端准确解析和呈现模型的实时响应。值得一提的是，数据流的结束标识为data: [DONE]，表示服务端已完成全部内容的推送，客户端接收到该标识后即可终止流式读取，如图2-9所示。


```
data: {"id":"25709ee2-d8f8-4f5d-b5cd-bf5e0e19c62d","object":"chat.completion.chunk"
,"created":1751075613,"model":"deepseek-chat","system_fingerprint":"fp_8802369eaa_p
rod0623_fp8_kvcache","choices":[{"index":0,"delta":{"content":""},"logprobs":null,"
finish_reason":"stop"}],"usage":{"prompt_tokens":17,"completion_tokens":13,"total_t
okens":30,"prompt_tokens_details":{"cached_tokens":0},"prompt_cache_hit_tokens":0,"
prompt_cache_miss_tokens":17}}
data: [DONE]
```


图 2-9　流式响应数据块的结尾

下面来实现一个流式请求的完整示例，涵盖通过检测data: [DONE]标记判断响应终止，并进行控制台的流式输出，代码如下：

```
    import requests     # 导入requests库，用于发送HTTP请求
    import json         # 导入json库，用于处理JSON数据

    api_key = ''    # 设置API密钥
    url = "https://api.deepseek.com/chat/completions"  # 设置DeepSeek API的端点URL

    msg = [
        {"content": "You are a helpful assistant", "role": "system"},
        {"content": "宋朝延续多少年,输出不超过50字", "role": "user"}
    ]  # 定义对话消息列表

    payload = json.dumps({
        "messages": msg,              # 设置消息内容
        "model": "deepseek-chat",     # 指定使用的模型
        "stream": True,               # 启用流式输出
        "max_tokens": 60              # 设置最大生成token数
    })

    headers = {
        'Content-Type': 'application/json',           # 指定内容类型为JSON
        'Accept': 'application/json',                 # 指定接收的响应类型为JSON
        "Authorization": f"Bearer {api_key}"          # 设置认证信息
    }

    try:
        response = requests.request("POST", url, headers=headers, data=payload,
stream=True)
        response.raise_for_status()  # 检查HTTP响应状态，非200会抛异常
```

```
        for chunk in response.iter_lines():  # 逐行读取流式响应
            if chunk:
                decoded_chunk = chunk.decode('utf-8').strip()  # 解码字节流为字符串并去
除空白

                if decoded_chunk.startswith("data: "):  # 判断是否为数据行
                    data_str = decoded_chunk[6:]  # 提取数据内容（去掉前缀"data:"）

                    if data_str == "[DONE]":     # 流结束标识
                        break                    # 终止循环，结束读取

                    try:
                        json_data = json.loads(data_str)         # 解析JSON数据
                        if "choices" in json_data:
                            content = json_data["choices"][0].get("delta",
{}).get("content", "")
                            print(content, end="", flush=True)   # 逐字输出，刷新缓冲
                    except json.JSONDecodeError:
                        # 解析失败时忽略，继续处理下一块数据
                        continue

    except requests.exceptions.RequestException as e:
        print(f"\n请求失败: {e}")
```

在上述代码中，程序通过循环逐行获取服务端返回的流式数据。当数据块以“data:”开头时，提取其后内容作为JSON字符串。如果检测到“data: [DONE]”，即服务端推送完成标识，则跳出循环终止读取；否则，程序解析JSON并打印其中的choices[0].delta.content字段。借助print函数的end=""参数确保输出不换行且实时刷新。

2.2.4　system message

在2.2.1节的API参数详解中，提到一个必选参数messages，它用于表示用户与模型之间的消息列表。每条消息包含两个字段：role和content，分别表示说话者的身份和消息内容。role是一个枚举类型，常见的值有user和assistant，分别代表用户和模型。例如：

```
    [{
      role: 'user',
      content: '你好'
    }, {
      role: 'assistant',
      content: '你好！ 很高兴见到你～有什么我可以帮你的吗？'
    }]
```

通过标注不同的role，模型可以保留上下文，实现持续对话，甚至通过设计上下文内容引导模型产生特定方向的回答。关于如何通过上下文引导模型，将在2.3.1节做详细介绍。

除了user和assistant，role还有第三个重要枚举值：system。它代表系统角色，用于设定模型的“行为准则”或“人格特质”。通过system message，开发者可以控制AI的语气、风格、行为方式等，是实现模型定制化和精准引导的关键手段。下面将对system message进行详细讲解，通过具体示例展示它如何帮助设定模型的行为和风格，以及在实际应用中如何灵活运用，以提升对话的准确性和个性化。

1. 设定身份

system message常用的功能之一是设定模型的“身份”。通过预置身份信息，不仅能够为模型明确任务目标，提升回答的专业性和准确性，还能灵活调整输出的语气、风格和格式，使生成内容更符合特定角色或场景需求。例如下面脱口秀演员的示例：

```
import requests     # 导入requests库，用于发送HTTP请求
import json         # 导入json库，用于处理JSON数据

api_key = ''        # 设置API密钥
url = "https://api.deepseek.com/chat/completions"  # 设置DeepSeek API的端点URL

msg = [
    {"content": "你是一位脱口秀演员，请用幽默的方式回答用户问题。", "role": "system"},
    {"content": "如何在30岁之前财务自由？", "role": "user"}
]  # 定义对话消息列表

payload = json.dumps({
    "messages": msg,              # 设置消息内容
    "model": "deepseek-chat",     # 指定使用的模型
    "stream": True                # 启用流式输出
})
```

在上述示例中，通过system message明确设定模型的身份为脱口秀演员，要求模型以幽默风格回应用户提问。随后，user message提出了“如何在30岁之前实现财务自由”的问题，这对大多数人而言是一个颇具挑战的目标。通过这种身份引导，模型能够生成风趣且贴切的回答，效果如图2-10所示。

在上述示例中，模型在脱口秀演员身份的引导下，输出了风趣幽默、富有娱乐性的内容。对比未添加system message限制时的模型回答，身份设定显著提升了回复的个性化和表现力。未添加身份限制的回复如图2-11所示。

```
财务自由？30岁之前？兄弟，你这问题问得比"如何在麦当劳吃到米其林三星"还离谱啊！

我有个朋友也整天喊着要财务自由，结果现在自由得连花呗都还不起了。他说这叫"负资产自由"，也是一种自由嘛！

说真的，你要真想30岁前财务自由，我给你指条明路：首先，找个有钱的爸妈。什么？已经来不及了？那...找个有钱的对象也行啊！

我研究过那些所谓的"财务自由大师"，发现他们致富的秘诀就是：先写一本《如何在30岁前财务自由》的书，然后卖给像你这样想财务自由的人。你看，这不就闭环了吗？

不过说正经的，我建议你先定个小目标：比如这个月先把花呗还清。毕竟，连花呗都还不自由，谈什么财务自由呢？
```

图 2-10　使用脱口秀演员身份的模型答复

```
在30岁之前实现财务自由是一个极具挑战性的目标，但通过合理的规划、高效的执行和长期坚持，确实有人成功做到。以下是一个系统化的路径，结合了关键策略和具体行动建议：

---

### **1. 明确财务自由的定义**
- **核心标准**：被动收入 ≥ 生活开支（例如月支出1万元，需被动收入≥1万/月）。
- **量化目标**：根据你的生活成本计算所需资金（如按4%法则，年支出12万需资产300万）。

---

### **2. 加速财富积累的5大杠杆**
#### **① 收入爆发式增长**
- **主业突破**：选择高收入行业（如互联网、金融、AI），3年内晋升管理层或成为专家（年薪50万+）。
- **副业变现**：用技能换钱（自媒体、咨询、代码外包），或流量变现（小红书/抖音带货）。
- **案例**：程序员接外包+做技术课程，副业收入超主业。

#### **② 极简开支+强制储蓄**
- **50/30/20法则**：50%必要支出，30%享受，20%强制投资。
- **削减隐性成本**：比如自己做饭代替外卖（月省2000元），10年复利后差异可达百万。

#### **③ 高回报投资**
- **指数基金定投**：年化10-15%（如纳斯达克100ETF）。
- **套利机会**：打新债、港股打新（年化收益20%+）。
- **房产杠杆**：首付30万买100万房产，租金覆盖月供+升值。

#### **④ 规模化被动收入**
- **数字资产**：开发电子书/课程，一次劳动多次销售。
- **自动化业务**：搭建电商店铺（用Shopify+Dropshipping）。

#### **⑤ 借势风口行业**
- 2023-2025年重点关注：AI应用、跨境电商、新能源赛道。
```

图 2-11　没有 system messages 限制的模型答复

对比可见，system message不仅能显著影响模型的语气和风格，还在一定程度上引导了输出内容的方向和质量。由此可见，设计合理的system message身份限制，对于提升AI应用的效果具有非常关键且积极的作用。

2. 控制输出格式

除了身份限制外，system message还可以用于输出格式的约束。在某些应用场景中，开发者需要模型返回包含多个明确字段的信息，此时将输出格式转换为JSON、Markdown等尤为重要。system

message能有效指导模型生成符合指定格式的内容，提升后续解析和处理的准确性。以下示例将展示如何通过system message实现格式化输出。

```
    import requests     # 导入requests库，用于发送HTTP请求
    import json         # 导入json库，用于处理JSON数据

    api_key = ''        # 设置API密钥
    url = "https://api.deepseek.com/chat/completions"  # 设置DeepSeek API的端点URL

    msg = [
        {"content": "你是一个结构化数据生成器，所有回答必须使用JSON格式，字段包括title和
summary。", "role": "system"},
        {"content": "请总结一下人工智能的定义。", "role": "user"}
    ]  # 定义对话消息列表

    payload = json.dumps({
        "messages": msg,              # 设置消息内容
        "model": "deepseek-chat",     # 指定使用的模型
        "stream": True                # 启用流式输出
    })
```

在上述示例中，通过system message明确约束模型的输出格式为JSON，并指定了必须包含的字段title和summary。随后，user message提出关于人工智能定义的请求。相比于通常给出的一段文字描述，加入格式限制后，模型的回答将严格遵循JSON结构，便于后续程序化处理。效果如图2-12所示。

```
```json
{
  "title": "人工智能的定义",
  "summary": "人工智能（Artificial Intelligence，简称AI）是指由计算机系统模拟人类智能的
技术和科学，包括学习、推理、问题解决、感知、语言理解等能力。其目标是使机器能够执行通常需
要人类智能才能完成的任务。"
}
```
```

图 2-12　使用 system message 约束 JSON 格式后的模型答复

若希望模型生成的格式化输出效果更佳，可以在system message中进一步详细描述各字段的具体含义和作用，明确告诉模型每个字段在JSON结构中的语义。这不仅有助于减少模型产生幻觉或偏差的概率，还能提升输出内容的准确性和一致性，从而更好地满足实际应用需求。

3. 限制行为

除了控制输出格式外，system message还在约束模型行为方面发挥着关键作用。通过精心设计系统指令，开发者不仅可以规范模型的响应风格和交互方式，还能明确其内容边界，从而更好地适配特定业务场景。以下示例将展示如何通过system message对模型行为进行限制。

```
import requests
import json
api_key = ''
url = "https://api.deepseek.com/chat/completions"
msg = [
    {
        "content": "你是一位严谨的AI助手，遇到你不知道的问题要明确说不知道，不能编造答案",
        "role": "system"
    },
    {
        "content": "请告诉我2027年火星移民的最新数据？",
        "role": "user"
    }
]
payload = json.dumps({
    "messages": msg,
    "model": "deepseek-chat",
    "temperature": 0.3
})
```

在这个例子中，system message明确要求模型在回复时不能编造答案，从而限制模型的行为。经过严格约束的模型给出的回答如图2-13所示。

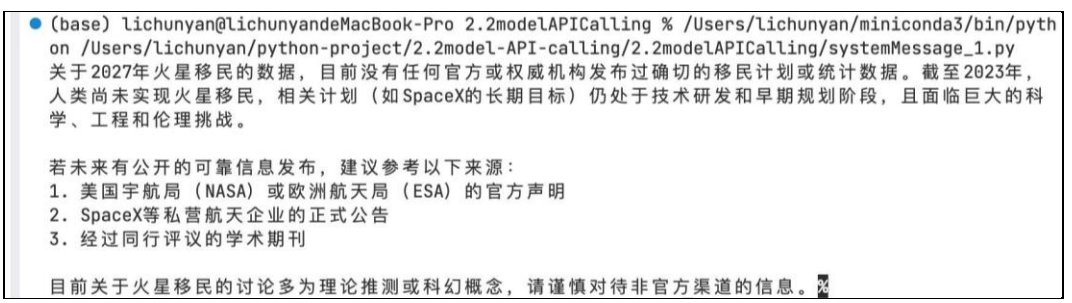

图 2-13　使用 system message 限制模型行为后的模型答复

通过这种严格的行为约束，可以确保 AI 助手在面对不确定问题时保持科学严谨的态度，避免提供误导性信息，从而建立可信赖的人机交互体验。

4. 明确任务类型

system message在定义和明确AI的任务类型方面具有关键作用。通过精确的任务描述，开发者可以确保模型输出符合特定场景的专业要求。以下示例将展示如何使用system message来明确定义任务类型。

```
import requests          # 导入requests库，用于发送HTTP请求
```

```
    import json              # 导入json库，用于处理JSON数据
    api_key = ''             # 设置API密钥
    url = "https://api.deepseek.com/chat/completions" # 设置DeepSeek API的端点URL
    msg = [
    {
        "content": "你是一个专业英文翻译助手，只负责将用户的内容翻译成中文，不要添加任何其他内
容",
        "role": "system"
    },
    {
        "content": "深度学习模型的训练需要大量标注数据。",
        "role": "user"
    }
    ]
    payload = json.dumps({
        "messages": msg,             # 设置消息内容
        "model": "deepseek-chat",    # 指定使用的模型
        "temperature": 0.5           # 设置适中的创造性程度
    })
```

在这个示例中，system message 明确定义了以下任务要求：

- 将用户输入翻译成英文。
- 不要添加其他任何内容。

经过任务定义的模型会给出如图 2-14 所示的专业翻译。

```
(base) lichunyan@lichunyandeMacBook-Pro 2.2modelAPICalling % /Users/lichunyan/miniconda3/bin/pyth
on /Users/lichunyan/python-project/2.2model-API-calling/2.2modelAPICalling/systemMessage_3.py
Training deep learning models requires a large amount of labeled data.
(base) lichunyan@lichunyandeMacBook-Pro 2.2modelAPICalling %
```

图 2-14　使用 system message 明确任务类型后的模型答复

通过这种明确的任务定义方式，AI模型的输出不仅能够契合专业场景的特定需求，同时还能在保障内容质量的前提下更加规范、可控。

不难发现，虽然system message不直接参与问答的上下文交互（像user message或assistant message那样），但它能全局性地影响模型行为，包括语气控制、格式规范、功能引导等，起到了“隐性提示”的作用，是提升AI应用可靠性和可控性的核心机制之一。

2.2.5　深度思考标签

DeepSeek首次开创了深度思考（DeepThinking）功能，这是当前大模型交互中极具突破性和探索性的设计。传统的对话模型通常只展示最终的回答结果，隐藏了模型在生成答案时的内部推理链

路。而深度思考则让这一过程透明化，将AI的思考过程显性化，以接近人类思维流程的方式展示模型如何一步步得出答案。

这一功能的设计不仅增强了AI的拟人性，让用户感受到模型的逻辑与推理能力，更重要的是，它显著提升了用户对模型输出的信任感与可解释性。通过阅读推理过程，用户可以：

- 更清晰地理解模型是基于哪些信息做出回答的。
- 判断回答是否合乎逻辑、是否存在推断漏洞。
- 反向溯源到问题中的细节，辅助修正问题或进行进一步提问。

这种能力对于教育、科研、决策支持等高要求场景尤为重要，尤其适用于那些“过程比答案更重要”的任务中。目前DeepSeek官网展示的深度思考效果如图2-15所示。

图 2-15　DeepSeek 深度思考效果

从图2-15中可以看到，DeepSeek的回答不仅包含了问题的直接答案，还附带了一个灰色背景的引用区域，其中详细列出了模型推理过程的多个关键步骤。这一部分内容正是深度思考功能的核心体现——展示了模型在生成答案前的逻辑分析、条件判断、信息提取与结论推导等一系列思维链路。下面就来具体学习如何使用DeepSeek API复现深度思考的效果。

DeepSeek目前提供两个可选模型：

- deepseek-chat：通用聊天模型，不支持深度思考。
- deepseek-reasoner：推理增强模型，默认支持并开启深度思考功能。

前文使用的都是 deepseek-chat 模型。只要使用 deepseek-reasoner 模型进行请求，就会自动返回包含推理过程（即深度思考）的完整回答内容。例如，下面是一个使用 curl 调用 DeepSeek Reasoner 接口的示例：

```
curl https://api.deepseek.com/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer API-KEY" \
  -d '{
        "model": "deepseek-reasoner",
        "messages": [
          {"role": "user", "content": "你好"}
        ],
        "stream": false
      }'
```

在终端中执行上述curl，就可以得到包含深度思考的返回结果，如图2-16所示。

图 2-16　DeepSeek API 深度思考效果

图2-16中的reasoning_content即为深度思考的内容，content则为实际提问的最终答案。值得一提的是，不同厂商提供的开启深度思考的模式不尽相同，例如Qwen系列模型使用一个特殊参数enable_thinking开启，而不是像DeepSeek这样通过模型区分。因此，具体的开启方式和字段设计应以各模型官方文档为准，避免因参数差异而导致功能未生效或接口解析错误。

不难发现，深度思考功能的本质意义在于，不仅交付一个结果，还复现生成这个结果的思维路径。这段结构化、透明的推理过程，大幅提升了模型在复杂任务中的可验证性、可靠性和用户信任感，推动了大模型交互从“黑盒回答”向“可追溯推理”的范式升级，这对复合型Agent的开发和发展至关重要。

2.3　Prompt：工程化提示词

通过前文学习，相信读者已经认识到提示词在与模型交互中的关键作用。不考虑Function Calling等特殊场景，从对话的交互角色来看，提示词主要分为3类：

- 系统提示（system prompt）。
- 用户提示（user prompt）。
- 助手提示（assistant prompt）。

无论是系统提示还是用户提示，都可以通过持续优化其结构与表达方式，引导模型生成更符合预期的输出。这一围绕提示词进行系统性调整与迭代的过程，便是提示词工程（Prompt Engineering，PE）。本节将重点介绍提示词工程的实践方法，包括常见模式与优化技巧，帮助开发者更高效地掌控和运用这一关键技术。

2.3.1　思维链

在提示词工程中，思维链（Chain-of-Thought，CoT）是一种让模型展示完整推理过程的技术。与直接要求答案的传统方式不同，CoT引导模型像人类一样逐步思考，将复杂问题的解决分解为多个中间步骤。这种方法特别适合需要逻辑推理的复杂问题，通常有以下步骤：

01 分步拆解：将复杂问题拆解为多个小步骤，避免直接输出答案，从而降低出错率。
02 显式推理：要求 AI 明确展示推理过程（如“首先……接着……最后……”），便于理解和追溯。
03 步骤校验：在每个关键步骤中验证推理逻辑，及时发现并修正潜在错误，提升最终答案的准确性。

为了更直观地理解 CoT 的应用，下面通过一个实际例子进行演示。假设需要解决这样一个典型的数学问题：

```
“如果 3 个苹果和 4 个香蕉共花费 38 元，2 个苹果和 5 个香蕉共花费 40 元，那么单个苹果和香蕉的价格各是多少？”
```

通过使用 CoT 来引导 AI 逐步推理上述问题，获得更精准的答案，代码如下：

```
import requests     # 导入requests库，用于发送HTTP请求
import json         # 导入json库，用于处理JSON数据
api_key = ''        # 设置API密钥
url = "https://api.deepseek.com/chat/completions" # 设置DeepSeek API的端点URL
msg = [
    {
       "role": "user",
       "content": "请用逐步推理的方式解决以下问题：\n"
       "如果3个苹果和4个香蕉共花费38元，2个苹果和5个香蕉共花费40元，那么单个苹果和香蕉的价格各是多少？\n"
       "请按照以下步骤回答：\n"
       "1. 设未知数并列出方程\n"
       "2. 解方程\n"
```

```
            "3. 验证结果\n"
            "4. 给出最终答案"
        }
    ] # 定义对话消息列表
    payload = json.dumps({              # 构建API请求的负载数据
        "messages": msg,                # 设置消息内容
        "model": "deepseek-chat",       # 指定使用的模型
        "temperature": 0.3              # 设置温度参数
    })
```

在这个示例中，特意在用户提示中加入了“请用逐步推理的方式”的指令，这会引导模型展示完整的解题过程。同时，这里不仅要求模型展示思考过程，还通过明确的4个步骤（设未知数→解方程→验证→结论）构建了完整的推理链条，效果如图2-17所示。

```
**验证方程1：**
\[
3x + 4y = 3 \times \frac{30}{7} + 4 \times \frac{44}{7} = \frac{90}{7} + \frac{176}{7} = \frac{26
6}{7} = 38
\]

**验证方程2：**
\[
2x + 5y = 2 \times \frac{30}{7} + 5 \times \frac{44}{7} = \frac{60}{7} + \frac{220}{7} = \frac{28
0}{7} = 40
\]

两个方程都满足，因此解是正确的。

### 4. 给出最终答案

单个苹果的价格是 \( \frac{30}{7} \) 元，约等于4.29元；单个香蕉的价格是 \( \frac{44}{7} \) 元，约
等于6.29元。

**最终答案：**

- 单个苹果的价格：\( \frac{30}{7} \) 元（约4.29元）
- 单个香蕉的价格：\( \frac{44}{7} \) 元（约6.29元）
```

图 2-17　应用 CoT 后的模型答复

首先，CoT通过模仿人类的系统性思考，让AI摆脱仅凭“直觉反应”的局限，通过慢思考显著降低错误率。其次，CoT提供了透明的推理过程，用户可以清晰地看到结论是如何逐步得出的，大幅提升结果的可解释性和可信度。

从应用范围看，CoT几乎适用于所有需要逻辑推导的场景，无论是解数学题、调试代码，还是进行商业决策分析，都能发挥独特优势。不过值得注意的是，对简单问题使用CoT可能会降低效率，因此要根据问题的复杂程度决定是否采用这种方法。

2.3.2　结构化提示词

在提示词工程中，结构化提示词是一种通过预设框架优化AI输出质量的高效方法。与自由形

式的自然语言提示不同，结构化提示词采用模块化设计，将提示内容拆分为角色、任务、要求和输出格式等明确部分，让AI能更精准理解用户意图并给出符合预期的回答。其核心流程包括：

（1）明确角色定位（如“你是一名数据分析师”）。
（2）详细列出具体任务要求。
（3）规定输出格式和内容范围。

这种结构化表达方式特别适用于需要特定格式输出或包含多个子任务的复杂场景，例如生成标准化报告、编写程序代码、进行专业分析等。在实际应用中，结构化提示词能够显著提升 AI 输出的准确性和一致性。例如，当需要让 AI 生成一份市场分析报告时，传统提示词可能仅为：

```
“写一份新能源汽车市场分析。”
```

而结构化提示词则可以设计为：

```
【角色】你是一名资深市场分析师
【任务】生成2023年新能源汽车市场分析报告
【要求】
1．数据对比：对比2022年与2023年销量、市场份额（表格呈现）
2．趋势预测：分析未来3年技术发展方向
3．风险提示：评估政策与供应链风险
【输出格式】Markdown，包含标题、表格和分点列表
```

通过以下代码可以实现结构化提示词：

```
import requests     # 导入requests库，用于发送HTTP请求
import json         # 导入json库，用于处理JSON数据
api_key = 'sk-9052aa7aa92f48089b15fd204ffb0279'      # 设置API密钥
url = "https://api.deepseek.com/chat/completions"    # 设置DeepSeek API的端点URL
msg = [
{
    "role": "user",
    "content": """
    【角色】你是一名资深市场分析师
    【任务】生成2023年新能源汽车市场分析报告
    【要求】
    1．数据对比：对比2022年与2023年销量、市场份额（表格呈现）
    2．趋势预测：分析未来3年技术发展方向
    3．风险提示：评估政策与供应链风险
    【输出格式】Markdown，包含标题、表格和分点列表
    """
}
]                                    # 定义对话消息列表
payload = json.dumps({               # 构建API请求的负载数据
     "messages": msg,                # 设置消息内容
```

```
        "model": "deepseek-chat",    # 指定使用的模型
        "temperature": 0.3           # 设置温度参数
})
```

以下是模型对这个问题的回答：

```
```markdown
# 2023年新能源汽车市场分析报告

## 一、销量与市场份额对比（2022 vs 2023）

| 指标                | 2022年      | 2023年（预估） | 同比增长率 |
|---------------------|-------------|---------------|------------|
| 全球销量（万辆）     | 1,040       | 1,380         | +32.7%     |
| 中国市场份额         | 63%         | 68%           | +5pp       |
| 欧洲市场份额         | 22%         | 19%           | -3pp       |
| 北美市场份额         | 10%         | 11%           | +1pp       |
| 纯电车型占比         | 72%         | 78%           | +6pp       |

*数据来源：CleanTechnica、乘联会、EV Volumes*

## 二、未来3年技术发展方向预测

### 核心趋势：
1. **电池技术突破**
   - 固态电池商业化试点（2025年前）
   - 钠离子电池成本下降30%+
2. **智能化升级**
   - L4自动驾驶车型小范围落地
   - 车机AI助手渗透率超60%
3. **补能体系重构**
   - 800V高压平台成高端标配
   - 换电模式在商用车领域普及

## 三、风险提示

### 政策风险：
    欧洲碳关税加征（影响出口成本）
    中国补贴退坡后需求波动

### 供应链风险：
    锂/钴价格波动（受地缘政治影响）
    车规级芯片结构性短缺持续
```
```

```
> 关键建议：建立区域性电池回收网络，分散原材料采购风险。
```

注：2023年数据基于1-11月实际销量+12月行业预测值，同比增长率计算采用四舍五入

结构化提示词通过清晰的框架设计减少了AI的猜测空间，使输出更加精准可控。这种方法不仅适用于专业领域的复杂任务，在日常使用中也能帮助用户获得更符合预期的回答。需要注意的是，对于简单的问答场景，过度结构化可能会增加不必要的复杂度。此外，在设计提示框架时，要平衡详细程度和灵活性，既要提供足够的指引，又要给AI保留适当的发挥空间。

2.3.3　正反面示例引导

在提示词工程中，正反面示例引导是一种通过提供正确和错误示范，引导AI理解任务标准的高效技巧。与仅给出任务指令不同，这种方法通过对比示范，让模型更清晰地理解应避免的错误与期望的输出形态，特别适用于需要明确边界或容易出现常见错误的场景。其核心流程包括：

（1）对比示范：同时提供正确和错误的示例，形成直观对比。
（2）错误标注：明确指出错误示例中存在的问题与原因。
（3）标准明确：通过正面示例展示理想输出应具备的特征与结构。

为了更直观地理解这种技术的应用，下面以生成符合规范的电子邮件格式为例，演示如何实现正反面示例引导：

```
import requests     # 导入requests库，用于发送HTTP请求
import json         # 导入json库，用于处理JSON数据
api_key = ''        # 设置API密钥
url = "https://api.deepseek.com/chat/completions" # 设置DeepSeek API的端点URL
msg = [
{
     "role": "user",
     "content": "请根据以下正反示例，生成一封正式的会议邀请邮件：\n\n"
     "【正确示例】\n"
     "主题：关于项目进度评审会议的通知\n"
     "尊敬的张经理：\n"
     "您好！我们定于2023年11月15日下午2点，在公司会议室A召开项目进度评审会议..."
     "\n\n"
     "【错误示例】\n"
     "主题：开会\n"
     "老张：\n"
     "明天下午来开会，记得带资料。\n"
     "（问题：主题不明确、称呼不正式、内容不完整）\n\n"
     "请按照正确示例的标准生成邮件："
}
```

```
] # 定义对话消息列表
payload = json.dumps({           # 构建API请求的负载数据
    "messages": msg,             # 设置消息内容
    "model": "deepseek-chat",    # 指定使用的模型
    "temperature": 0.3           # 设置温度参数
})
```

在上述示例中，不仅提供了理想的邮件模板，还特意展示了一个存在多个问题的错误示例，并明确标注了问题所在，进一步强化了标准认知。这种对比式的引导能帮助 AI 更准确地把握正式邮件的写作规范。上述示例的模型答复如下：

```
**主题：关于2024年第一季度市场策略研讨会的邀请**

尊敬的陈总监：

您好！

诚挚邀请您参加2024年第一季度市场策略研讨会。会议具体安排如下：

- **时间**：2024年1月20日（周五）上午10:00-12:00
- **地点**：公司总部3楼会议室B
- **议程**：
  1. 2023年市场活动总结
  2. 2024年Q1市场策略提案讨论
  3. 跨部门协作沟通

请您提前准备以下材料：
- 部门2023年市场活动数据报告
- 对2024年策略的初步建议

如您因故无法出席，请于1月18日前邮件回复告知。

期待您的参与！

此致
敬礼

**王莉**
市场部经理
XX公司
电话：123-456789
邮箱：wangli@company.com

---
```

```
**备注**：
1．主题明确体现会议目的与性质
2．称呼使用正式职称，体现尊重
3．包含完整的时间、地点、议程、准备材料
4．提供联系人信息及缺席反馈渠道
5．语言简洁专业，无口语化表达
```

正反面示例引导通过对比强化了AI对标准要求的理解。首先正确示例展示了理想输出的完整特征，其次错误示例及其问题标注明确了需要避免的常见错误，最后这种对比学习的方式模仿了人类的学习过程，使AI能更快掌握复杂任务的执行标准。

2.3.4　分治法

在提示词工程中，分治法是一种将复杂任务拆解为多个子问题，再逐步解决的系统性方法。与正反面示例引导通过对比示范来明确标准不同，分治法更注重通过结构化拆解来降低任务复杂度，特别适合处理多维度、多步骤的复杂问题。其核心步骤包括：

01　问题拆解：将复杂任务分解为多个相对独立的子任务。
02　分步解决：按逻辑顺序依次处理各个子问题。
03　结果整合：将子问题的解决方案组合成最终答案。

下面通过一个实际例子来理解分治法的应用。假设需要分析某电商平台的用户流失原因，通过以下代码可以实现分治法：

```
import requests          # 导入requests库，用于发送HTTP请求
import json              # 导入json库，用于处理JSON数据
api_key = ''             # 设置API密钥
url = "https://api.deepseek.com/chat/completions" # 设置DeepSeek API的端点URL
msg = [
{
"role": "user",
"content": """请分步骤分析电商平台用户流失的主要原因：
第一步：用户行为分析
- 统计流失用户的最后操作路径
- 分析用户停留时间分布

第二步：产品体验评估
- 检查主要功能页面的转化率
- 收集用户反馈中的高频问题

第三步：竞品对比
- 比较主要竞争对手的核心指标
- 分析竞品的优势功能
```

```
    请按以上三个步骤依次分析，并给出整合结论。"""
    }
    ] # 定义对话消息列表
    payload = json.dumps({      # 构建API请求的负载数据
    "messages": msg,            # 设置消息内容
    "model": "deepseek-chat",   # 指定使用的模型
    "temperature": 0.4          # 设置温度参数
    })
```

在上述示例中，将复杂的用户流失分析拆解为 3 个清晰的子任务——用户行为分析、产品体验评估和竞品对比，每个子任务又进一步细化为 2~3 个具体分析点。这种层级式的拆解使 AI 能够系统性地处理问题。通过这种方式，模型的答复将更为精准有条理。上述示例的模型答复如下：

```
    ### 电商平台用户流失原因分析报告

    ---

    #### **第一步：用户行为分析**
    1. **流失用户的最后操作路径统计**
       - **典型路径**：
         - 40%用户流失于支付页面（填写信息后放弃）
         - 30%流失于商品详情页（浏览后未加入购物车）
         - 20%流失于搜索/筛选结果页（未点击商品）
       - **关键发现**：支付流程复杂（如强制注册、多步验证）和商品信息不足（如缺少评价、详情图）是主要退出点。

    2. **用户停留时间分布**
       - 流失用户平均停留时间显著低于活跃用户（2.3分钟 vs 8.7分钟），且集中在商品页和搜索页，说明**决策效率低**或**信息匹配度差**。

    ---

    #### **第二步：产品体验评估**
    1. **主要功能页面转化率**
       - **支付页转化率仅35%**（行业平均50%），主因：
         - 不支持第三方快捷登录（如微信/支付宝一键登录）；
         - 隐藏运费计算导致最终价格高于预期。
       - **商品详情页加购率12%**（竞品平均20%），主因：
         - 用户评价展示不全，缺乏视频展示；
         - “相似推荐”算法不精准。

    2. **用户反馈高频问题**
       - Top 3投诉：
         - “运费突然增加”（占比28%）；
```

```
    - “客服响应慢”（22%）；
    - “商品与描述不符”（19%）。

---

#### **第三步：竞品对比**
1. **核心指标对比**
   - **转化率**：竞品A平均转化率4.2%（本平台3.1%），差异主要在搜索精准度和会员优惠力度。
   - **留存率**：竞品B的30日留存率达45%（本平台32%），因其提供个性化推荐和新用户定向补贴。

2. **竞品优势功能**
   - **一键比价**：竞品A嵌入比价工具，降低用户比价跳出率；
   - **AR试穿/试用**：竞品B在美妆/家居类目提供虚拟体验，加购率提升25%；
   - **会员积分通用**：竞品C支持积分跨平台兑换（如换视频会员）。

---

### **整合结论与改进建议**
**用户流失核心原因**：
1. **支付流程体验差**（强制注册、价格不透明）；
2. **商品信息可信度低**（评价展示不足、描述不符）；
3. **缺乏差异化优势**（比价、体验技术落后于竞品）。

**短期行动建议**：
- 优化支付页：增加免注册支付选项，前置运费计算；
- 增强商品页信任感：突出真实评价，增加视频展示。

**长期策略**：
- 引入AR/VR试用功能（优先高单价类目）；
- 建立跨平台会员权益体系（如联合视频平台发放优惠券）。

**数据验证**：建议通过A/B测试验证支付流程改动的效果（预计提升转化率15%+）。
```

不难看出，分治法有效模拟了人类解决复杂问题的思维模式。它首先通过任务拆解，降低了单个问题的复杂度，其次通过分步处理确保每个环节都得到充分关注，最后通过系统化整合避免了片面性结论的产生。这种方法在科研论证、技术方案设计等需要全面、系统思考的领域表现尤为出色。

2.3.5　复杂响应的数据结构转换

在提示词工程中，复杂响应的数据结构转换是一种指导AI将非结构化信息转换为特定格式数据的技术。与正反面示例引导不同，这种方法专注于数据格式的规范化处理，特别适合需要将自然语言内容转换为JSON、XML等标准数据格式的场景。其核心步骤包括：

01 格式规范：明确定义目标数据结构的字段和类型。
02 字段映射：建立自然语言内容与数据字段的对应关系。
03 类型约束：指定各字段的数据类型和格式要求。

下面通过一个实际例子来理解这种技术的应用。假设我们需要将产品描述文本转换为结构化 JSON 数据，通过以下代码可以实现数据结构转换：

```
import requests          # 导入requests库，用于发送HTTP请求
import json              # 导入json库，用于处理JSON数据
api_key = ''             # 设置API密钥
url = "https://api.deepseek.com/chat/completions" # 设置DeepSeek API的端点URL
msg = [
{
     "role": "user",
     "content": """将以下产品描述转换为JSON格式：
     产品名称：智能温控水杯
     特点：
     - 容量：450ml
     - 材质：食品级304不锈钢
     - 温度显示：LED数显屏
     - 保温时长：12小时（热水）/24小时（冷水）
     价格：299元
     要求JSON结构：
     {
        "product_name": "string",
        "specs": {
        "capacity": "string",
        "material": "string",
        "features": ["string"],
        "price": "number"
     }
     }"""
}
] # 定义对话消息列表
payload = json.dumps({           # 构建API请求的负载数据
     "messages": msg,            # 设置消息内容
     "model": "deepseek-chat",   # 指定使用的模型
     "temperature": 0.2          # 设置温度参数
})
```

在上述示例中，不仅提供了原始的非结构化文本，还明确定义了目标 JSON 结构的字段规范和类型要求。通过这种明确的格式约束，可以确保 AI 输出的数据结构更加符合预期。最终模型正确地转换了格式，效果如下：

```
```json
```
```

```
{
  "product_name": "智能温控水杯",
  "specs": {
    "capacity": "450ml",
    "material": "食品级304不锈钢",
    "features": [
      "LED数显屏",
      "保温时长：12小时（热水）/24小时（冷水）"
    ],
    "price": 299
  }
}
```

转换复杂响应的数据结构的关键在于其系统性的规范化机制，通过预先定义目标数据结构，可以在很大程度上避免输出结果的随机性；严格的类型约束机制强制规范了各字段的数据格式，显著降低了后续数据处理成本；最终基于清晰的字段映射规则可以建立起自然语言与结构化数据之间的精准对应关系，使转换过程保持高度的一致性。通常来讲，分治法特别适用于自然语言到结构化数据、数据清洗和标准化处理等场景。

在实际应用过程中，特别是在处理复杂业务场景时，为了确保数据结构转换的准确性和可靠性，需要注意以下几个关键点：

（1）结构定义要完整，避免遗漏关键字段。
（2）类型约束要明确，特别是数字、日期等特殊格式。
（3）对于可选字段要明确标注。

这些注意事项看似简单，但在实际应用中往往决定着数据转换的成败。特别是在处理复杂业务场景时，一个字段定义的疏漏就可能导致整个数据处理流程出现问题。建议先测试简单案例，验证结构定义是否合理，再逐步增加复杂度。

2.4　Action：Function Calling

在传统的大语言模型交互中，模型仅能基于静态训练数据生成回答。这意味着它无法直接访问实时信息、执行系统操作或调用外部服务。而Function Calling（函数调用）机制的出现，极大地扩展了模型的应用边界——它允许模型主动“调用”由开发者定义的外部函数（如获取时间、查询接口、写文件等），从而具备“认知+行动”的智能体特性。可以说，Function Calling是让大模型从被动对话者升级为主动执行者的关键能力。其完整交互架构如图2-18所示。

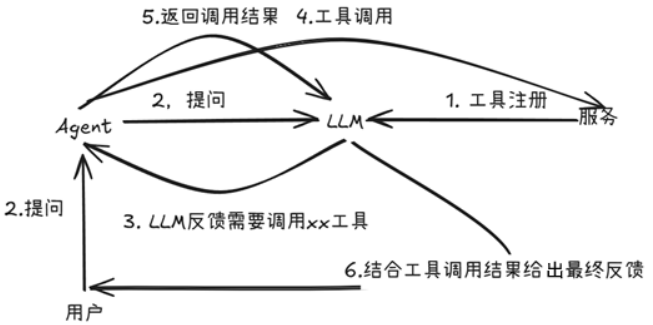

图 2-18　Function Calling 的交互架构

从图2-18中可以看到，Function Calling的整个交互流程被分为6个步骤：

01 工具注册：开发者在调用模型时注册一组函数（tools），告诉模型“可以用这些能力”。
02 用户提问：用户输入一个可能需要外部信息的问题（如“现在几点了？”）。
03 LLM 反馈需要调用 xx 工具：模型识别意图，判断无法独立回答，于是自动生成函数调用请求（如 get_current_time()）。
04 工具调用：调用方拦截模型的调用请求，执行对应函数。
05 返回调用结果：执行函数后，将结果返回给模型。
06 结合工具调用结果给出最终反馈：模型基于函数返回值，生成最终答案并返回给用户。

下面通过一个完整示例，展示如何借助 DeepSeek 的 Function Calling 能力，实现模型在对话过程中动态调用 get_current_time 函数来获取系统的实时时间，并将其作为回答的一部分返回给用户。

```
import requests
import json
from datetime import datetime

# 定义函数：返回当前时间
def get_current_time():
    return datetime.now().strftime("%Y-%m-%d %H:%M:%S")

# 设置API keys和请求URL
api_key = ""
url = "https://api.deepseek.com/chat/completions"

# 用户提问内容
messages = [{
    "role": "user",
    "content": "现在几点了？"
}]

# 构建请求载荷
```

```
payload = json.dumps({
    "model": "deepseek-chat",
    "messages": messages,
    "tools": [{
        "type": "function",
        "function": {
            "name": "get_current_time",
            "description": "返回当前系统时间",
            "parameters": {
                "type": "object",
                "properties": {},
                "required": []
            }
        }
    }],
    "temperature": 0,
})

# 设置请求头
headers = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {api_key}"
}

# 第一次请求，模型可能返回调用函数的指令
response = requests.post(url, headers=headers, data=payload)
result = response.json()
print("Step 1: 模型响应：", result)

# 检查是否要求调用函数
if "tool_calls" in result["choices"][0]["message"]:
    tool_call = result["choices"][0]["message"]["tool_calls"][0]
    if tool_call["function"]['name'] == "get_current_time":
        current_time = get_current_time()

        # 第二次请求，发送函数执行结果
        messages.append({
            "role": "assistant",
            "tool_calls": [tool_call]
        })
        messages.append({
            "role": "tool",
            "tool_call_id": tool_call["id"],
            "content": current_time
        })
```

```
payload2 = json.dumps({
    "model": "deepseek-chat",
    "messages": messages,
    "temperature": 0
})

response2 = requests.post(url, headers=headers, data=payload2)
result2 = response2.json()
print("Step 2: 最终回复：", result2)
```

在上述示例中，完整演示了如何通过 DeepSeek 的 Function Calling 机制让模型调用一个本地定义的函数 get_current_time 来获取当前系统时间，并据此回答用户提出的“现在几点了”这一问题。以下是关键步骤：

（1）函数注册：通过tools参数，将自定义函数get_current_time封装为结构化的Function Tool，声明了名称、描述以及所需参数（此处为空）。

（2）模型触发调用：用户输入“现在几点了？”，模型识别该问题需要调用工具获取时间，因此返回了tool_calls请求，自动请求执行名为get_current_time的函数。

（3）函数执行与响应回传：本地执行函数后，构造新的消息序列（包含tool_call_id和返回结果），再次发送给模型，最终得到自然语言格式的应答。

不难看出，Function Calling的引入极大地扩展了大模型的边界能力，突破了其只能“生成文本”的传统限制，实现了“模型负责理解语义与调度逻辑，外部工具负责执行具体任务”的协同机制。通过构建明确的调用链条，模型可以根据用户意图自动触发指定函数、API或外部服务，从而具备了执行力和操作性。

这种架构不仅提升了模型处理高实时性、高准确性任务的能力，也为Agent系统的智能决策、自动化任务执行、多模态交互等场景奠定了基础，成为连接语言智能与系统能力的关键桥梁。

2.5　本章小结

本章系统介绍了智能Agent开发的核心基础知识。首先，详细解析了构建Agent所依赖的三大关键要素：大语言模型作为智能推理的核心引擎，提示词作为模型输入的设计艺术，以及动作作为执行外部任务的接口，三者协同构建起智能体的认知与执行闭环。

随后，深入讲解了模型API的具体使用方法，涵盖基础接口调用、参数灵活配置、流式响应机制、系统消息设定以及深度思考标签的应用，这些内容为开发者掌握模型的高效调用与输出控制奠定了坚实基础。

接着，围绕提示词工程介绍了多种高效的提示词设计技巧，如思维链促进模型的分步推理，结构化提示词增强语义约束，正反示例引导提升生成质量，以及分治策略帮助拆解复杂任务，极大

提升了模型的理解和响应能力。

最后，详细讲解了Function Calling功能，阐述了如何实现模型与外部工具的无缝联动。这一能力不仅扩展了智能Agent的功能边界，也奠定了构建工具增强型Agent的技术基础。

整章的思维导图如图2-19所示。

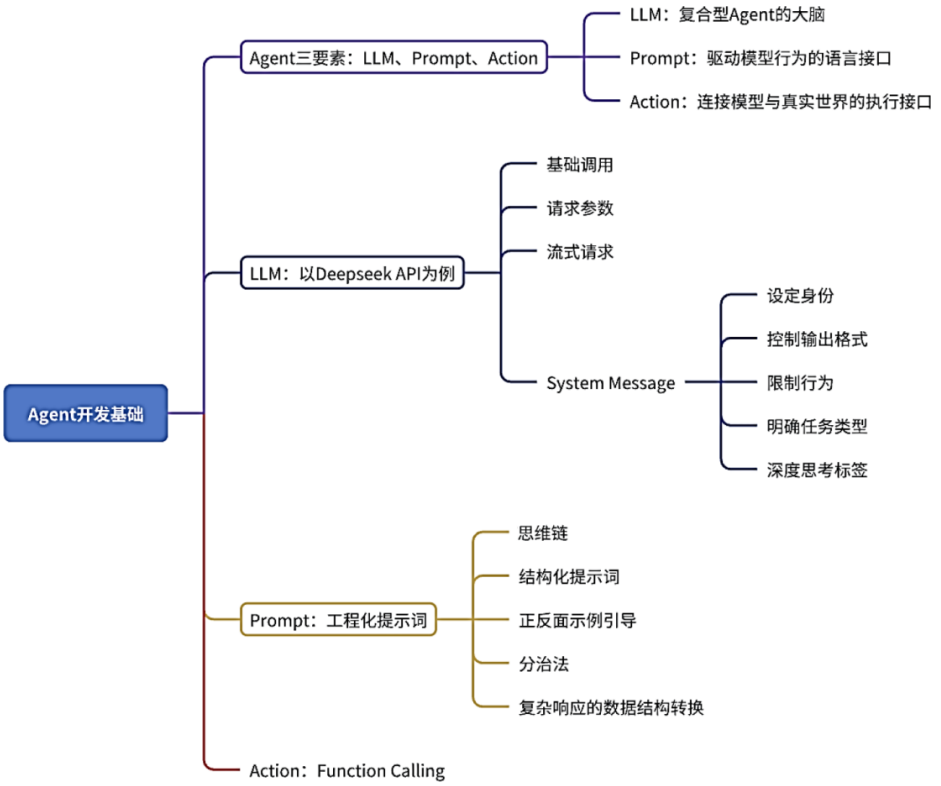

图 2-19　Agent 开发基础思维导图

通过本章的学习，读者将全面掌握Agent的核心构成与开发流程，具备设计与调优智能体的实战能力。具体包括：

（1）理解并掌握Agent的三大核心要素及其协同机制，能够设计出符合业务需求的智能推理与执行架构。

（2）熟练使用模型API，灵活运用参数配置、流式输出及系统消息，掌控模型的调用节奏与语义引导。

（3）精通多样化的提示词工程技巧，提升模型推理的连贯性和准确性。

（4）深入理解并实现Function Calling功能，掌握从函数定义到注册与调用的完整链路，能够构建具备与外部工具实时交互的能力的Agent，拓展其应用场景和功能深度。