

Fix the error in the given paragraph.
 Remove any repeating sentences, meaningless characters, not English sentences, and so on.
 Remove unnecessary repetition. Rewrite any incomplete sentences.
 Return directly the results without explanation.
 Return directly the input paragraph if it is already correct without explanation.

在经过 ChatGPT 的评估与改进后，5000 个图像-文本对中最终保留下 3500 对符合要求的高质量数据，用于下一阶段的模型微调。具体的数据格式如图 7.15 所示，包含基本的图像信息和更加全面的文本描述。

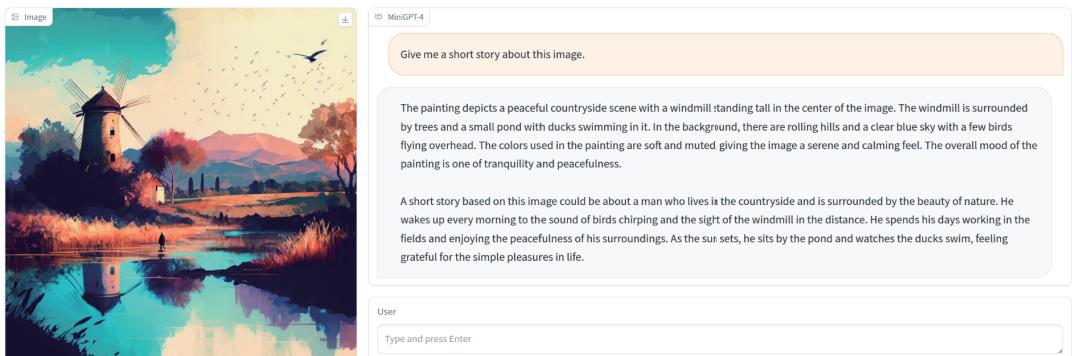


图 7.15 高质量图像-文本对的数据格式

在预训练的基础上，研究人员使用精心构建的高质量图像-文本对对预训练的 MiniGPT-4 模型进行微调。在训练过程中，MiniGPT-4 同样要完成类似的文本描述生成任务，不过具体的任务指令不再固定，而是来自一个更广泛的预定义指令集。例如，“详细描述此图像”、“你可以为我描述此图像的内容吗”，或者“解释这张图像为什么有趣”。微调训练只在训练数据集和文本提示上与预训练过程略微不同，在此不再介绍相关的代码实现。

微调结果表明，MiniGPT-4 能够产生更加自然、更加流畅的视觉问答反馈。同时，这一训练过程也是非常高效的，只需要 400 个训练步骤，批量大小为 12，使用单块 NVIDIA A100 80GB GPU 训练 7 分钟即可完成。

在微调完成后，研究人员发现 MiniGPT-4 具备其他各种有趣的能力，这是在 GPT-4 的演示中没有体现的，例如：通过观察诱人的食物照片，直接生成详细的食谱；识别图像中存在的问题并提供相应的解决方案；直接从图像中检索出有关人物、电影或绘画作品的事实信息。如图 7.16 所示，用户希望 MiniGPT-4 指出输入的海报出自哪部电影，这本质上是一个根据图像进行事实检索的问题。MiniGPT-4 能够轻松识别出海报出自美国电影《教父》。

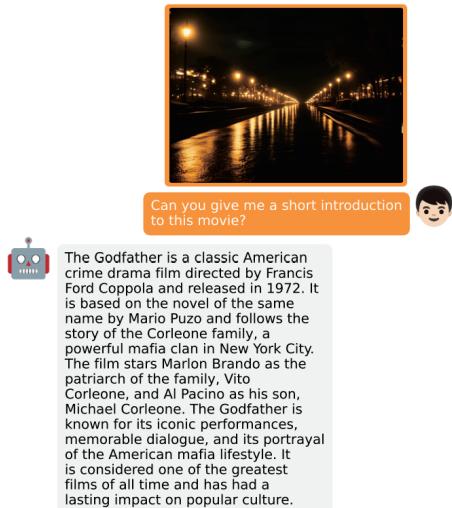


图 7.16 MiniGPT-4 根据图像进行事实检索

8. 大模型智能体

一直以来，实现通用类人智能都是人类不懈追求的目标，智能体，也是在该背景下提出的。早期的智能体主要是基于强化学习实现的，不仅计算成本高，需要用大量的数据训练，而且难以实现知识迁移。随着大模型的发展，其在诸多领域展现出惊人的语义处理能力，能够快速生成文本、回答问题，甚至完成一些复杂的知识推理任务。研究人员开始思考如何将大模型与智能体结合，从而突破大模型本身不具备与外部世界联系，无法感知外部环境以及调用外部工具的问题。同时，智能体借助大模型强大的多模态理解与生成优势，可以快速处理信息、规划行动。智能体与大模型结合展现出了强大的能力，因此近年来大模型智能体受到了越来越多的关注并在很多应用领域取得了很好的实践结果。

本章将重点介绍智能体的发展、大语言模型智能体架构，最后以 LangChain 为例介绍大语言模型智能体实践。

8.1 智能体基础

“智能体”（Agent）也称为智能代理，这一概念源远流长，其历史渊源可上溯至亚里士多德、休谟等先哲的相关论述。从哲学维度剖析，“智能体”意指具备行动潜能的实体，而“代理”一词，则侧重于对这种行动潜能的施行与展现^[337]。智能体的范畴颇为广泛，既涵盖人类个体，亦囊括物理世界以及虚拟空间中的其他各类实体。尤为关键的是，智能体概念的核心聚焦于个体的自主性，即赋予其运用意志、抉择判断以及付诸行动的能力，使之摆脱了单纯被动回应外部刺激的模式。

本节将从智能体的发展历史和大模型智能体应用范式角度介绍智能体发展的大体历程以及大模型智能体在实际应用中的具体范式。

8.1.1 智能体发展历史

自 20 世纪 80 年代中后期起，人工智能研究人员开展了智能体相关研究^[338-341]。与此同时，智能体的内涵也历经演变，与哲学意义上的智能体逐渐有所区别。就人工智能范畴而言，智能体本质上是一种计算实体^[342, 343]。由于哲学范畴内容关于智能体的定义涉及意识、欲望等概念，这些

对于计算实体来说很难定义和度量^[344]，我们所能直接观测到的仅仅是计算实体的外在行为表现。因而，包括艾伦·图灵在内的诸多人工智能研究者提议，暂且搁置有关智能体是否“真正”在思考，又或者是否真正持有“思想”这类问题的探讨^[345]。研究人员转而采用诸如自主性、响应性、主动性以及社交性等其他特性，用以辅助阐释智能体^[342]。从根本上来说，人工智能领域的智能体与哲学意义层面的智能体并非同一概念，人工智能领域的智能体是智能体哲学概念于人工智能语境下的具象化呈现。

自 20 世纪 90 年代开始，人工智能领域智能体研究开始更快速发展，从整体上看智能体技术的发展与人工智能发展紧密相关，可以粗略地划分为以下三个阶段：符号智能体、基于强化学习的智能体、以及基于大模型的智能体。

在人工智能发展的早期阶段，符号智能体扮演着关键角色，主要关注转导、表征和推理问题^[346]。具体而言，转导问题侧重于将来自环境的低层次感知数据，诸如传感器读取的数据，转换为高层次的符号表示；表征和推理问题，则聚焦选择和设计适当的符号表示来有效地描述和处理智能体所涉及的知识信息，并确保基于符号逻辑的推理过程能够高效进行^[346]。符号智能体具备明确和可解释的推理能力，以及出众的表达效能^[347-349]，基于知识构建的专家系统便是其典型范例。不过，当应对不确定性情境以及大规模现实世界难题时，符号智能体暴露出诸多短板^[350, 351]。并且由于符号推理算法的复杂性，找到一种能在有限时段内产出有价值结果的高效算法，更是颇具挑战性^[352]。

伴随计算性能的跃升以及数据获取便利性的提升，加之学界与业界对智能体同环境交互问题研究的不断关注，研究人员开始使用强化学习手段，训练智能体以应对更为繁复、极具挑战性的任务^[353, 354]。核心关注点聚焦于如何引导智能体借由与环境的互动开展学习，进而确保其在特定任务执行进程中斩获最大化的累积奖励^[355]。最初，基于强化学习构建的智能体，主要凭借策略搜索、值函数优化等基础性技术落地实践，诸如 Q-learning^[356] 与 SARSA^[357] 等典型范例。而随着深度学习的兴起，深度强化学习应运而生，它融合了强化学习与深度神经网络技术^[358, 359]，促使智能体具备从高维输入数据中学习复杂策略的能力，得以在未知环境里自如探索、自主学习，进而从电子游戏竞技到机器人操控等诸多领域广泛渗透，产生了 AlphaGo^[360]、DQN^[361] 等一系列重要成果。但是，强化学习智能体依旧面临着训练周期冗长、采样效率欠佳以及稳定性不足等棘手难题，尤其在错综复杂的现实世界应用场景中，这些短板更加凸显^[355]。

2023 以来，大模型异军突起，其所展现出的惊人的能力引发广泛瞩目，基于大模型构建的智能体也日益备受瞩目^[362-365]。大模型智能体具有感知、决策、行动和记忆的能力，通过感知模块捕获周围环境的信息，利用大模型进行推理和决策，通过执行器实施具体行动，同时还能存储和管理记忆，以支持持续学习和适应动态环境^[366]。大模型智能体将大模型作为智能体的核心中枢，即大脑或控制器的关键构成要素，同时借助多模态感知、工具运用等策略，全方位拓展智能体的感知范畴与行动边界。凭借思维链、问题分解等技术手段，大模型智能体得以彰显出可与符号智能体媲美的推理规划潜能。不仅如此，它们还能从外界反馈中持续学习，执行全新行动，获得与强

化学习智能体那样与环境互动的能力。当前，大模型智能体已在软件开发^[367]、科学探索^[368]、网络购物^[25]、医疗健康^[369] 等诸多现实世界场景取得很好的实践效果。尤为突出的是，鉴于其天然的自然语言理解与生成能力，它们能够通过自然语言达成无缝对接式交互，为多智能体之间的协作和竞争奠定了坚实基础，引发广泛关注与深入讨论。

8.1.2 大模型智能体范式

从大模型智能体的应用范式来看，其可进一步细分为单智能体、多智能体交互以及人-智能体交互三种主要模式，如图8.1所示。单智能体具备核心决策能力与任务执行能力，已在众多应用领域中展现出卓越的性能。多智能体之间则能够通过协作或对抗性交互不断推动能力的提升与优化。而在人-智能体交互模式下，智能体不仅能够通过人类反馈提高任务执行的效率与安全性，同时也能够为人类提供更加优质的服务。本文将围绕上述三种范式展开详细论述，深入探讨其特性与应用场景。

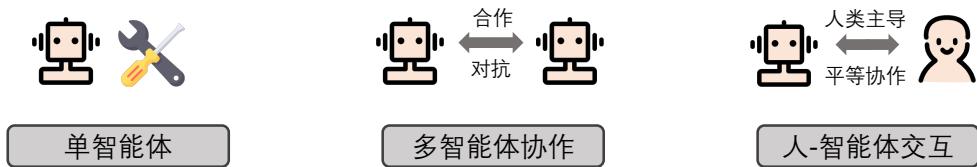


图 8.1 大模型智能体范式^[370]

1. 单智能体

单智能体范式是指基于大模型构建的具备自主决策与任务执行能力的独立智能体。不同于传统的大模型应用，这类智能体能够在复杂环境中实现自我调节与持续优化，从而高效地完成任务并解决问题。单智能体在多个领域展现出了巨大的应用前景，其具体应用可以划分为面向任务、面向研究创新以及面向生命模拟的智能体场景。

面向任务的智能体主要聚焦于解决明确的任务或问题，如自然语言问答、图像识别和数据分析等传统领域。这类智能体能够将复杂任务分解为多个子任务并逐步完成。例如，ChatGPT 等对话式智能体不仅可以实现自然语言交流，还能够通过调用 API 与外部工具交互，以应对更加复杂的任务需求。DeepMind 开发的多模态智能体 GATO^[371] 展示了其在多任务处理上的卓越能力，从图像分类和文本生成到机器人控制，均能出色完成。而 Codex^[100] 则能够将自然语言描述转化为代码，并具备代码调试、修改与优化的能力。这些基于大模型的单智能体在对话交互、控制系统以及编程开发等领域展现了广泛的适用性，极大地拓展了智能体的实际应用范围。

面向研究创新的智能体则专注于科学探索、技术研发和创新性问题的解决。这类智能体需要具备强大的推理能力与创新思维，因此推理与决策模块在其设计中尤为关键^[372]。例如，在化学、数学等领域，ChemCrow^[373] 和 FunSearch^[374] 等基于大模型的智能体已经展现了在自动化任务执

行方面的巨大潜能。通过智能体的辅助，研究人员能够更高效地完成复杂推理、公式验证和实验设计，从而推动科学的研究进步。

面向生命模拟的智能体应用则聚焦于模拟人类或其他生物的行为与社会互动。这类智能体不仅需要具备自然语言理解与生成能力，还需拥有常识推理与社会认知能力。例如，在斯坦福小镇^[362]的实验中，智能体能够基于对环境和自身状态的理解，通过基本观察总结出高级别的认知，模拟人类或生物体的日常行为与决策过程。而 RoleLLM^[375] 通过非参数提示学习直接为智能体注入角色数据，使其能够模拟不同角色的行为特征。Humanoid Agent^[376] 则通过模拟人类的基本需求与情感，增强智能体的真实感与适用性，使其在社交互动和仿真环境中表现得更加自然。这些智能体不仅能够模仿特定角色的语言风格与知识体系，还能体现角色的个性与思维过程，在社会行为模拟、游戏角色扮演和个性化助理等领域具有重要的应用价值。

2. 多智能体协作

在大模型智能体的应用中，社群协作范式主要包括两种核心交互模式：合作互动与对抗互动。通过这两种模式，智能体在协作中实现能力互补，在对抗中推动性能提升。以下将详细阐述这两种交互模式的具体实现方式及其在智能体发展中的重要意义。

合作互动模式强调通过多个智能体之间的协作与资源共享，实现任务的高效解决与能力的优势互补。合作互动的显著优势在于能够充分发挥每个智能体的特长，优化资源配置，从而提升整个系统的效率与可靠性。

在合作互动模式下，不同智能体通过明确的角色分工和高效的交流机制，共享资源与信息，达成复杂任务的高效解决。例如，Voyager^[372] 构建了一个共享的技能库，允许不同智能体在探索和执行复杂任务时相互协作与补充；AgentSims^[377] 提出的“Mayor”模式中，一个智能体作为“领导者”分配任务，其他智能体则负责完成诸如招聘员工、组建公司等具体工作，最终通过协作完成整体目标。MetaGPT^[378] 则通过让智能体分别扮演不同的角色（如产品经理、架构师、项目经理和工程师），在软件开发的过程中进行交流与监督，从而提升代码生成的质量。此外，MedAgents 通过构建多个专注于不同医疗领域的智能体专家团队，共同进行会诊，大幅提高了诊断的成功率^[379]。类似的合作框架也广泛应用于软件开发^[380]、推荐系统^[381] 等领域。

对抗互动模式则通过在智能体之间设计具有竞争性的任务和环境，促进整体性能的提升。这一模式的核心思想在于引入“辩论”机制，通过智能体之间的相互挑战与反馈推动系统进步。

在对抗互动中，每个智能体承担不同的角色，提出各自的观点或解决方案，并根据预设的规则和标准展开辩论。这种机制能够帮助智能体发现自身的不足之处，进而进行优化和完善。例如，DebateGPT 让多个智能体围绕同一问题展开辩论，各自提出观点并根据既定的评价标准进行评估，从而促进智能体的改进。在电影推荐场景中，不同智能体通过对推荐结果展开讨论与反馈，逐步优化最终的推荐质量^[381]。在医疗诊断领域，智能体可以分别扮演不同医学专家的角色，通过辩论的方式共同讨论诊断方案，从而提高诊断的准确性与可靠性^[379]。

3. 人-智能体交互

无论智能体的具体形式为何，其核心目标始终是服务于人类。人-智能体交互范式通过引入人类的参与，实现人机之间的智能互动。从功能与角色的分工来看，人-智能体交互范式可以进一步细分为“人类主导范式”和“人机平等协作范式”两种模式。

在人类主导范式中，人类通过提供指导与反馈，对智能体的行为和决策施加直接影响。这种模式强调人类的主导地位，智能体在任务执行过程中高度依赖人类的指令与修正。例如，HuggingGPT^[382]通过人类提供的任务描述调用不同的模型来完成具体任务。在这一模式下，人类负责任务的规划与管理，而智能体则执行具体的操作以辅助人类完成目标。

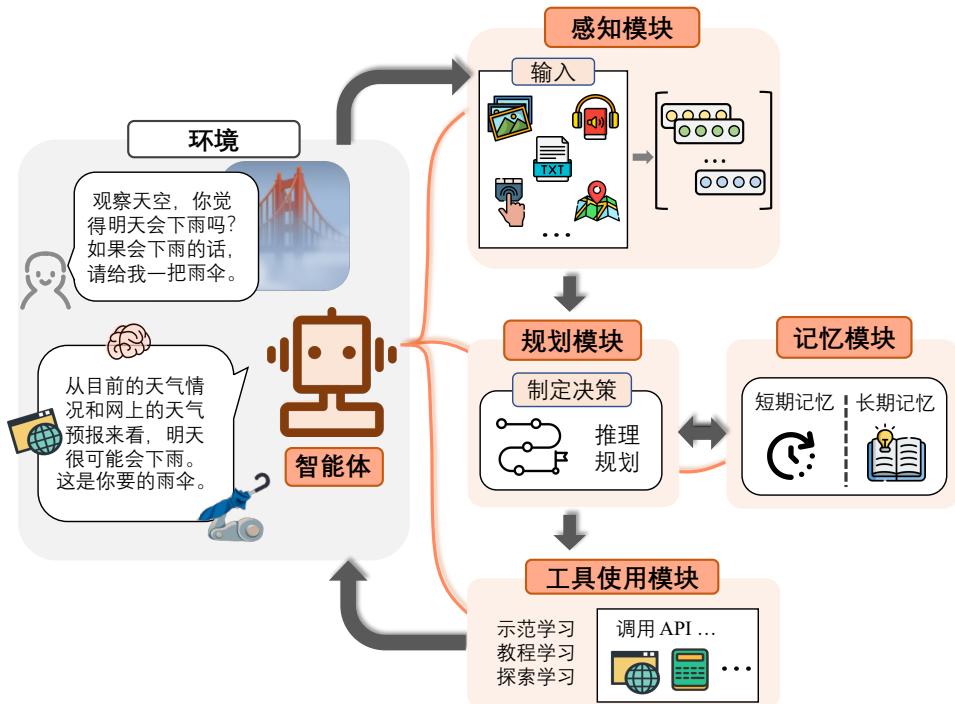
这一范式的显著优势在于能够确保智能体的行为始终符合人类的需求与期望。由于每一步操作均受到人类的监督与控制，智能体在任务完成的准确性和可靠性方面得以显著提升 [132]。此外，在面对意外情况或复杂任务时，人类能够及时调整策略，从而有效应对多变的环境和要求^[383]。通过这一协作模式，人类不仅能够熟悉智能体所提供的辅助功能，还能在有效监督的条件下进一步提升整体的工作效率和用户体验。

在人机平等协作范式中，强调智能体与人类作为平等的合作伙伴，共同参与任务的规划与执行。这种模式注重智能体的适应性与自主性，通过协同合作实现任务的高效完成。例如，在任务执行过程中，智能体能够主动寻求人类的反馈，并根据反馈动态调整其行为策略^[384]。与人类主导范式不同，这种模式不仅要求智能体具备执行能力，还需要其通过自主学习与优化不断提升自身能力。

随着智能体在环境感知、推理与决策能力方面的进步，人机交互的效率与深度也将不断提高。通过持续的优化与协同，人类与智能体之间可以实现真正的无缝合作，使智能体成为人类创新和效率提升的重要伙伴。

8.2 大语言模型智能体架构

智能体可以被视为独立的个体，能够接收并处理外部信息，进而给出响应。大模型智能体基本组成如图8.2所示，主要包含以下几个核心模块：感知模块、规划模块、记忆模块、工具使用模块。对于外界输入，智能体借助多模态能力将文字、音频、图像等多种形式的信息转换为机器能够理解的表现形式；进而由规划模块对这些信息进行处理，结合记忆模块完成推理、规划等复杂任务；智能体可能会利用工具使用模块执行相应的动作，对外部输入做出响应。本节将分别介绍智能体各个模块的基本功能。

图 8.2 智能体框架结构^[370]

8.2.1 感知模块

感知模块负责从环境中获取文本、视觉、听觉等多种形式的信息，并将其传递给其他模块进行处理。多模态感知能力对于大模型智能体的发展至关重要。通过整合这些多样化的输入，智能体能够深入理解其所处的环境，做出更明智的决策，在复杂多变的任务中发挥出色表现。赋予大模型智能体多模态感知能力已成为一个重要的研究方向，除了常见的输入形式之外，触觉反馈、手势以及 3D 雷达等其他潜在输入也可以丰富智能体的感知范围，使其在复杂环境中保持灵活、全面的感知能力。

文本作为人类与世界交互的核心载体，在大模型智能体的发展中扮演着重要角色。同时，文本作为承载数据、信息和知识的主要媒介，也是人机交互的核心。现有主流大模型智能体如 AutoGPT^[385]等已具备通过文本进行交互的基础能力。然而，准确理解文本背后的隐含意义，如用户的隐式意图，仍然是一大挑战。一些研究尝试通过强化学习技术，捕捉这些隐含意义，并利用模型反馈机制推导出用户偏好，使智能体能够做出更个性化和精准的响应。随着任务越来越复杂，尤其是在

陌生场景下，提升智能体的文本感知能力显得尤为重要。

在视觉感知领域，尽管大语言模型在理解和处理多轮对话方面展现了卓越的性能^[65]，但仍然无法处理视觉模态信息。视觉输入通常包含丰富的环境信息，例如物体的属性、空间关系以及场景布局。将视觉信息与其他模态数据相结合，能够使智能体对外部环境的理解更加全面且精准^[255]。为了赋予智能体理解视觉信息的能力，一种直接的方法是将视觉输入通过图像描述生成技术转换为对应的文本描述^[386]。这种方法的优点在于其高度的可解释性，并且无需为生成描述进行额外训练，从而显著节约计算资源。然而，此方法在转换过程中可能会丢失大量潜在信息，导致视觉信息的表达不完整。为解决上述问题，研究人员尝试将大语言模型与视觉编码器相结合，并通过增加一个可学习的接口层来对齐视觉编码与大模型的语言理解能力，从而增强大模型对视觉信息的感知能力^[387]。这一方法有效降低了大模型在学习视觉语言对齐任务中的负担，并显著提升了其在视觉感知方面的性能。

在音频感知方面，声音信息是外界环境中不可或缺的重要组成部分，为大模型智能体赋予听觉感知能力，能够显著增强其对交互内容、环境状况乃至潜在危险的感知能力。目前，已有多种针对音频处理的模型和方法被开发，但这些模型通常仅在特定任务中表现优异^[288, 387, 388]。鉴于大模型智能体在工具使用方面的强大能力，研究人员提出了一种直观的方案，即通过将大模型作为控制中心，级联调用现有的工具集或模型库以感知音频信息，从而实现多模态感知的高效融合。然而，与视觉感知类似，这种通过外部模型进行听觉感知的方法仍存在信息丢失的隐患。因此，如何将听觉感知能力直接融入大模型体系，成为当前亟待解决的重要研究课题。

此外，感知模块的发展还应涵盖其他潜在的输入形式，如触觉、嗅觉等，以进一步拓展大模型智能体的感知能力。未来的智能体可能具备更加丰富的感官系统，能够像人类一样感知并理解多样化的现实世界信息。例如，通过配备特定的触觉和嗅觉器官，智能体可以在与物体交互时获取更为详尽的信息；同时，其还能够对环境中的温度、湿度、光照强度等要素进行精准感知，从而实现适应性行动。总体而言，感知模块的多模态扩展不仅能够帮助智能体更全面地理解并适应外部环境，还将显著提升其在复杂任务中的执行能力。未来研究的核心将聚焦于赋予大模型更强的多模态理解能力，以进一步增强其感知与决策水平。这一领域的突破将为大模型智能体的全面发展奠定重要基础。

8.2.2 规划模块

规划模块是大模型智能体的核心，其主要职责是通过对环境与任务的深刻理解，生成并优化任务执行计划，制定合理的行动步骤以实现既定目标。研究表明，大模型的推理与规划能力随着模型参数规模和训练数据量的增加呈现出显著的阶跃式提升^[62]。尤其是在模型参数量达到数百亿级别时，即使缺乏直接与任务相关的数据，大模型也能够通过在输入提示中加入包含任务中间推理步骤的示例，或通过引导模型逐步输出推理过程，逐步构建任务的解决方案。将大模型作为规划模块的核心，充分发挥其强大的推理能力和丰富的知识库，可以在复杂且动态的环境中实现

快速决策，并灵活应对各种变化。目前，这一领域的研究主要集中于无反馈规划与带反馈规划两大方向，为探索大模型在规划能力上的潜力提供了重要的研究路径。

1. 无反馈规划

无反馈规划（Planning without feedback）指在规划阶段一次性生成完整的任务和子任务拆分计划，并严格按照该计划逐步执行，而不根据外界变化进行实时调整。在这种模式下，大模型智能体会在任务开始前，根据当前环境和任务要求生成一个完整的执行方案，并在执行过程中始终遵循初始计划。无反馈规划的主要优势在于其执行效率较高，适用于环境相对稳定、变化较少的任务场景。例如，在文档生成任务中，智能体可以根据预先设定的主题、段落结构和内容要求，生成包含所有预定义内容的完整文章，并在生成过程中不因外部反馈而修改文章内容。目前，无反馈规划的典型方法是将思维链推理技术扩展至智能体领域^[389]。在这种方法中，大模型智能体能够利用思维链推理技术预生成完成任务所需的所有子任务拆分计划，并为每个子任务设计相应的执行动作，以便在真实环境中逐步完成。然而，这种方法的挑战在于，预生成的计划可能在实际环境中面临执行困难或效果不佳的问题，特别是在忽略外部数据变化的情况下，智能体可能无法有效应对突发事件或异常情况。

2. 带反馈规划

带反馈规划（Planning with feedback）是一种更为复杂且灵活的规划方式，智能体在执行任务的过程中能够持续获取环境反馈或监控环境变化，并基于反馈信息动态调整行动计划。在这一模式下，智能体不仅会在任务开始前制定初步的执行计划，还能够在任务执行过程中实时监测环境变化和任务进展，依据实际情况不断优化和修正计划。带反馈规划强调智能体与环境的交互，通过不断更新计划以确保任务的顺利完成。其显著优势在于高度的适应性与灵活性，尤其适用于环境复杂且变化频繁的任务场景。ReAct^[390]方法是大模型智能体带反馈规划方法的经典方法，其核心在于将任务执行过程与推理规划过程相结合。在任务执行的每一步中，大模型智能体依据已完成的子任务和获得的环境反馈，动态生成当前步骤的子任务及相应的执行动作，并将其在真实环境中执行。完成后，环境反馈会被传递回智能体，用于下一步的任务规划。通过这一反复循环的过程，ReAct方法使大模型智能体能够根据环境反馈实现动态任务规划。

在实际应用中，通常将无反馈规划与带反馈规划相结合，以兼顾效率与灵活性。例如，在自主配送系统中，可以首先利用无反馈规划生成初步的配送路线，并在实际执行过程中通过带反馈规划进行实时调整，以应对突发情况和动态变化。通过融合无反馈规划的高效性与带反馈规划的适应性，规划模块赋予了大模型自主智能体灵活且高效的决策能力，使其能够在多样化的任务环境中表现出色，从而完成复杂的任务目标。

8.2.3 记忆模块

记忆模块是在大模型智能体中承担着管理与操作智能体记忆的核心功能，包括对长短期记忆的存储、读取、处理以及反思等任务。该模块不仅负责存储历史数据与经验，还能够高效提取和更新信息，从而实现长期记忆与短期记忆之间的有机交互。通过记忆模块的支持，智能体在处理连续性任务时能够保持上下文的连贯性，并基于以往经验做出更加准确的判断与决策。

1. 记忆模型

大模型智能体所采用的记忆模型包括长期记忆和短期记忆两部分。它们各自有不同的功能和实现方式，但都依赖于大模型的强大计算和理解能力。

短期记忆通常通过将记忆内容以提示语句的形式嵌入大模型输入的上下文中，借助大模型的上下文理解能力来实现。包括存储和使用两个部分：(1) 存储：在任务执行过程中，关键的上下文信息与事件会被实时记录，形成短期记忆内容；(2) 使用：在后续任务中，这些记忆内容会作为提示语句输入至大模型的上下文中，帮助模型基于提示进行推理与决策。例如，将前几步的操作结果及重要的环境信息作为输入内容，支持模型在接下来的步骤中做出更加合理的判断与选择。

长期记忆通过构建记忆库来实现管理和检索，支持知识的持久化存储与高效调用。包括构建和检索两个部分：(1) 构建：在长期任务的执行过程中，智能体会将累积的经验、知识以及数据系统化地存储至记忆库中。记忆库的形式可以包括向量数据库、知识图谱等。(2) 检索：智能体需要获取以往的经验或相关知识时，可以通过查询记忆库进行检索，并将检索到的记忆内容作为大模型的输入，与当前任务需求结合后进行处理。例如，在面对类似问题时，智能体能够检索到此前解决相似问题的经验，从而显著提升问题解决的效率与准确性。

2. 记忆操作

智能体的记忆操作则包括写入、读取和反思等多个环节，这些操作旨在确保智能体能够高效地管理和利用其记忆资源，从而提升任务执行能力与智能化水平。

记忆写入指将新的信息或经验存储到记忆模块中。在短期记忆中，写入的方式通常是将新的文本信息直接插入到上下文中，而在长期记忆中，则需要将信息存储到记忆库中，并对其进行索引与标记，以便后续检索使用。例如，当智能体完成某项任务后，可以将任务的执行过程及其结果记录为参考数据，供未来使用。通过不断积累经验，智能体能够逐步优化其能力，实现更高水平的智能表现。

记忆读取是指从记忆模块中提取与当前任务相关的信息，以支持任务的完成。在短期记忆中，读取操作通常是直接从上下文中提取提示信息并加以使用；而在长期记忆中，则通过检索记忆库来获得相关内容，通常通过匹配任务需求与记忆信息的方式完成。例如，当处理一个复杂问题时，智能体可以从长期记忆库中提取具有参考价值的解决方案，从而提供精准的建议或策略，提升任务处理的效率与质量。

记忆反思是智能体对已存储记忆进行回顾与分析的一种机制，旨在进一步优化其行为策略与

决策能力。在基于大模型的智能体系统中，Reflexion^[391] 方法为记忆模块引入了反思功能。通过对过往任务的回顾与结果分析，智能体能够总结经验教训，并生成改进建议。例如，在完成多项任务后，智能体可以反思并评估哪些方法行之有效，哪些需要调整，并将这些反思结果存储到记忆模块中，以指导未来任务的执行。这种机制不仅提升了智能体对任务的适应能力，还为其持续优化提供了重要支持。

8.2.4 工具使用模块

工具使用模块是大模型智能体连接外部环境的关键环节之一，通过调用外部工具和资源来执行特定任务，从而扩展了智能体的功能边界并提升其问题解决能力与效率。此模块的设计与实现，显著增强了大模型智能体在实际应用中的灵活性与实用性，使其能够完成复杂计算、获取外界数据并与其他系统进行交互。对于大模型智能体而言，扩展其工具使用能力的核心在于如何充分激发大语言模型的潜力，使其具备高效的工具操作能力。

工具使用模块的核心是如何让大语言模型获得工具使用能力，它的实现离不开有效的工具学习策略，这些策略主要分为以下三类：(1) 示范学习：通过观察具体的工具使用案例进行学习；(2) 教程学习：通过工具手册或操作指南获取知识；(3) 探索学习：通过尝试和反馈不断优化工具使用能力，通常涉及强化学习的应用。

1. 示范学习

示范学习是智能体通过模仿人类专家操作工具的行为模式，逐步掌握工具使用方法的一种过程。这种学习方式类似于人类通过观看教学视频或观察他人操作来掌握新技能。通常，基于示范学习的工具掌握过程可以分为以下两个阶段：(1) 示范数据收集：首先，需要构建一个包含大量工具使用示范数据的训练集。这些数据形式可以包括详细的操作步骤记录、工具使用视频等内容，以确保覆盖工具使用的关键场景和步骤。(2) 模型训练：随后，将收集到的示范数据输入到大语言模型中，通过监督学习的方式训练模型，使其能够理解并模仿示范中的工具操作流程，从而具备执行类似任务的能力。

示范学习的优势在于能够快速帮助大模型掌握具体工具的使用方法，特别适用于操作步骤明确且流程固定的工具。然而，其局限性也较为明显：一方面，示范学习高度依赖高质量的示范数据；另一方面，其在工具操作的灵活性和创新性方面存在一定不足，较难应对需要动态调整的复杂任务。

2. 教程学习

教程学习通常通过将工具手册作为提示输入到大模型中，使其直接从手册内容中理解工具的功能与使用方法。这一方法的核心理念来源于人类通过阅读手册或观察演示学习新技能的行为方式。同样，大模型可以借助其强大的上下文理解能力，通过提示语句从工具手册中获取相关知识并掌握工具的操作。然而，尽管 OpenAI 系列的大模型凭借卓越的上下文理解能力能够较好地完

成教程学习任务，现有的开源大模型却因其上下文理解能力的不足，难以通过教程学习有效掌握工具使用技能。

针对这一问题，ToolLLM^[392] 提出了通过构建 ToolBench 数据集，为 3000 余种工具（涵盖 16000 多个 API）自动生成任务指令，并利用深度优先搜索算法自动化构建解决方案路径，从而对开源大模型进行微调，显著提升其基于教程学习的工具使用能力。此外，该方法还通过 API 检索器推荐最适合的 API，以进一步优化工具选择与操作过程，成功解决了开源大模型在依赖工具手册提示语句进行学习时效果受限的问题。

教程学习的显著优势在于其系统性与全面性。大模型能够通过详细的文档深入学习工具的功能与操作方法，从而赋予智能体更为全面且强大的工具使用能力。这种学习方式不仅能够帮助智能体高效掌握工具，还为其在复杂任务场景中的灵活应用奠定了坚实基础。

3. 探索学习

探索学习（Exploratory Learning）是一种通过自主尝试与实验来掌握工具使用的方法。在这一过程中，智能体通过自主探索和反复试验，逐步学习工具的操作技巧及其最佳使用方式。智能体能够根据环境反馈和人类反馈动态调整操作策略，从而不断优化工具的使用方法。

在实际操作中，环境反馈通常通过智能体与外部环境交互后所获得的结果进行优化；具体而言，结果反馈用于评估智能体一系列动作的整体效果，而中间反馈则着重考察每一步操作的即时表现。例如，在 WebShop^[393] 场景中，智能体通过对比其购买行为与人类购买行为之间的相似性来获得结果反馈，从而评估其表现的有效性。在此基础上，人类反馈强化学习通过模拟人类奖励机制，结合强化学习算法优化智能体的策略，以提升其决策能力和执行效果。同时，智能体会将每次尝试的结果系统化地记录下来，构建经验库。这一过程不仅使智能体能够积累丰富的操作经验，还能逐步提升其对工具的使用熟练度和操作效率。

探索学习的关键在于通过持续的试探与调整，使智能体在动态环境中不断完善其工具使用能力。这种方法不仅赋予智能体更强的适应性与自主性，还为其在多变任务场景中的高效表现提供了坚实的技术支持。

当前研究的重点在于如何通过整合多种学习策略来优化模型性能，从而全面提升大模型智能体的表现能力。例如，将示范学习的精确性与探索学习的灵活性相结合，可以显著增强模型在未知环境中的适应能力；而教程学习与示范学习的结合，则能够为模型理解复杂工具操作提供双重支持。这种多策略融合不仅提升了模型的学习效率，还为处理更复杂的多工具任务开辟了新路径。

8.3 大模型智能体训练

大模型智能体的核心能力涵盖了感知、规划、记忆以及工具使用，这些能力使其能够弥补传统大模型无法与外部世界交互的局限性。然而，大语言模型在最初的设计中并不具备这些核心能力。大语言模型主要依赖于大规模的文本数据训练，擅长语言生成和理解，但无法直接使用外部

工具，也不能很好对任务进行多步骤的规划。同时大语言模型构建之初也没有考虑记忆和使用用户全部对话历史。为了弥补这些不足，研究者们开始系统地研究如何提升大语言模型解决上述问题的能力。本节将重点介绍大语言模型工具使用能力提升、推理规划能力提升以及长期记忆构建与应用的策略方法。

8.3.1 工具学习

大模型工具学习（Tool Learning）是指通过让大语言模型学会使用各种工具的调用方式，进而利用合适的工具去实现特定的功能需求。例如，用户输入“请告诉我上海今天的天气。”具备工具使用能力的大语言模型会给出如下响应：

```

1. 识别任务类型： 天气查询任务。
2. 调用天气 API： 调用天气 API：模型请求外部天气服务 API（如 WeatherMap），发送查询参数。
response = requests.get("https://api.weathermap/data/2.5/weather",
    params={
        "q": "Shanghai",
        "date": "2025-1-6",
        "appid": "your_api_key",
        "units": "metric"
    })
weather_data = response.json()
3. 返回结果： API 返回数据：上海当前气温为 3°C，天气晴朗。
4. 模型最终输出：“上海今天的天气是晴朗，当前气温为 3°C。”

```

当前训练大语言模型使用工具的方法主要依赖于通过工具交互轨迹生成的大规模数据集，对预训练模型应用有监督微调方法进行训练。文献 [392] 描述了工具学习数据集构造的方法，主要包括三个阶段：API 收集、指令生成和解决路径标注。以下是每个阶段的详细总结：

(1) API 收集：ToolLLaMA 的 API 数据集来源于 RapidAPI 平台，这是一个提供大量真实世界 RESTful API 的市场。通过爬取 RapidAPI 的工具和 API 文档，包括 API 的功能描述、必选参数、可选参数、请求体、调用代码片段及示例响应，初始收集了 10,853 个工具 (53,190 个 API)。为了确保数据质量，过滤掉了不可用或质量较低的 API (如返回 404 错误的 API)，最终保留 3,451 个高质量工具 (16,464 个 API)，涵盖 49 个类别和 500 多个细分类别集合。

(2) 指令生成：通过 ChatGPT 自动生成与 API 功能相关的多样化指令，特别注重单工具和多工具场景的结合。指令生成过程从 API 文档出发，随机抽取单个或多个 API，并结合提供的人工撰写的种子指令示例，指导 ChatGPT 创造符合实际应用场景的指令。这些指令分为三类：单工具指令、同类别多工具指令和同集合多工具指令，最终生成了近 20 万条指令-API 数据对，确保覆盖广泛的工具使用场景。

(3) 解决轨迹标注：为每条生成的指令，通过 ChatGPT 的函数调用功能标注有效的解决路径（即多步 API 调用序列）。使用深度优先搜索决策树（DFSDT）扩展搜索空间，允许模型探索多个推理路径，并在必要时放弃当前路径以扩展新节点。相比传统方法，DFSDT 有效解决了推理错误传播和探索不足的问题，最终生成了 126,486 条高质量的指令-解决路径对，为模型训练提供了丰富的数据支持。

ToolLLaMA-2-7B^[392] 是通过使用上述方法构建的包含 12.6 万条数据的大规模数据集，在 LLaMA-2 模型上进行的有监督微调。然而，这种基于大规模数据集的微调方法往往忽略了工具使用中的任务特定特征，从而导致模型性能的瓶颈。即使经过如此大量的数据训练，ToolLLaMA-2-7B 的工具调用效果也仅能达到 GPT-4 的 80% 左右。

文献 [394] 指出，当前用于工具学习的数据集大多通过 GPT-4 等模型自动构建，数据中存在不小比例的错误。例如，RoTLLaMA 的训练集包含 12,247 条由 GPT-4 生成并经过筛选的多轮工具调用轨迹，但其中约 17% 的轨迹存在工具使用错误。这些错误的轨迹会对利用其进行训练的模型带来了显著的负面影响。此外，通过对 ToolLLaMA-2-7B-v2 和 NexusRaven-13B-v2 的实验结果进行分析发现，当模型选择了错误的工具时，通常会选择一个与正确工具具有相同前缀的工具。进一步研究表明，通过手动纠正模型第一个错误预测的词元，模型往往能够生成正确的后续词元。这一现象说明，某些关键词元（Key Tokens）对于任务的成功至关重要。研究还表明，模型在工具调用中的错误可以根据工具类型、参数以及内容分为有限的几种类别。这为后续针对性地优化工具学习数据集和提升模型性能提供了重要参考。

根据上述分析，文献 [394] 提出的 TL-Training 方法通过错误数据影响缓解、关键词元优先级排序以及强化学习策略有效缓解了上述问题。在有监督微调阶段，其核心目标是使大型语言模型（LLM）与训练数据的分布保持一致。然而，训练数据中的错误交互路径可能对模型的决策产生负面影响，进而增加工具调用错误的概率。为了解决这一问题，TL-Training 设计了一种自动化流程，用于识别错误的交互路径并阻止这些路径的反向传播，从而减少它们对模型性能的有害影响。给定一个数据序列 $(q, t_{0..s}, o_{0..s})$ ，旨在识别错误的工具调用轨迹 $\mathbb{T}_e \subseteq \{t_0, t_1, \dots, t_s\}$ 。由于直接判断某个特定的工具调用 t_i 是否正确是颇具挑战性的。TL-Training 利用工具调用后生成的反馈 o_i 。这些反馈通常包含了结构化的错误报告信息，由于工具调用错误种类较为固定，可以通过依次分析 o_i 来提取错误调用轨迹 \mathbb{T}_e ，从而实现对错误调用的自动识别。在识别出错误调用轨迹 \mathbb{T}_e 后，通过在训练过程中阻止这些错误交互路径的反向传播，减轻它们对模型的负面影响。这一机制通过修改损失函数实现，其具体形式如下：

$$\mathcal{L}_{MAE} = - \sum_{\mathbb{D}} \sum_{t_s \notin \mathbb{T}_e} \log p_M(t_s | q, \mathbb{T}, t_{0..s-1}, o_{0..s-1}) \quad (8.1)$$

其中 \mathbb{D} 表示整个训练数据集。

其次，在文献 [394] 中研究发现，工具名称的首个词元，连同其后那些与其他工具名称有共同

前缀的词元，在成功识别工具方面起着更为关键的作用。标准的有监督微调训练会不加区分地最大化每个词元的条件概率，将所有词元视为同等重要。为解决这一局限，TL-Training 提出了一种根据词元相对重要性自适应调整其训练权重的方案。

给定一个数据序列 $(q, t_{0..s}, o_{0..s})$ ，其中每个工具 $t_i = (t_i^0, t_i^1, \dots, t_i^{l_i})$ 由 l_i 个词元构成，将这些词元划分为两个集合：

$$K_i = \{t_i^m \in t_i \mid t_i^m \text{ 是关键词元}\} \quad (8.2)$$

$$NK_i = \{t_i^m \in t_i \mid t_i^m \text{ 不是关键词元}\} \quad (8.3)$$

然后，依据它们的相对重要性来调整 K_i 和 NK_i 的权重，使模型能够更侧重于关键词元。

$$w_i^m = \begin{cases} \text{CLIP}\left(\frac{|NK_i|}{|K_i|}, 1, w_{\max}\right) & \text{如果 } t_i^m \in K_i \\ 1 & \text{否则} \end{cases} \quad (8.4)$$

其中， w_{\max} 是最大调整乘数，而 $\text{CLIP}(x, \min, \max)$ 函数用于将调整因子限制在 $[\min, \max]$ 这个区间范围内。符号 $|\cdot|$ 表示集合的大小。（注：由于 K_i 始终至少包含工具名称的首个词元，所以避免了除数为零的风险。）

利用这些权重，在训练过程中按照以下目标优先考虑关键词元：

$$\mathcal{L}_{PKT} = - \sum_{\mathbb{D}} \sum_{t_s} \sum_{t_s^m} w_s^m \cdot \log p_M(t_s^m \mid q, \mathbb{T}, t_{0..s-1}, o_{0..s-1}, t_s^0 \dots t_s^{m-1}) \quad (8.5)$$

最后，由于工具调用过程出现的错误类型有限，这使得可以基于这些特定错误引入一种奖励机制，并应用强化学习算法，提升其工具使用能力。为实现这一目标，TL-Training 针对工具使用任务定义了一组奖励函数，并采用近端策略优化（PPO）算法来优化模型性能。

对于大语言模型生成的工具调用预测 t_i 及其相应的标准答案，基于模型在各种场景下工具使用的质量定义了以下奖励函数：

$$R(t_i) = \begin{cases} -2 & \text{如果 } t_i \text{ 无法解析} \\ -2 & \text{如果 } t_i \text{ 包含工具幻觉} \\ -1.5 & \text{如果 } t_i \text{ 调用了错误的工具} \\ R_p(t_i) & \text{如果 } t_i \text{ 存在参数识别问题} \\ -0.25 & \text{如果 } t_i \text{ 存在内容填充问题} \\ 1 & \text{如果 } t_i \text{ 正确} \end{cases} \quad (8.6)$$

其中， $R_p(t_i)$ 定义为：

$$\begin{aligned} R_p(t_i) = & -0.8 \cdot \mathbb{I}(t_i \text{ 存在参数幻觉}) \\ & - 0.5 \cdot \mathbb{I}(t_i \text{ 包含冗余参数}) \\ & - 0.5 \cdot \mathbb{I}(t_i \text{ 存在缺失参数}) \end{aligned} \quad (8.7)$$

这里 $\mathbb{I}(\cdot)$ 表示指示函数。

奖励函数 $R(\cdot)$ 针对大型语言模型工具使用中不同的潜在错误，提供了一个结构化的评分系统来评估性能。基于该奖励函数，应用 PPO 算法，通过迭代优化模型参数来最大化这些奖励，具体如下：

$$\mathcal{M}^* = \arg \max_{\mathcal{M}} \mathbb{E}_{\mathbb{D}} \left[\sum_{t_s} (R(t_s) - \beta \text{KL}(\mathcal{M}(\cdot) || \mathcal{M}_{\text{sft}}(\cdot))) \right] \quad (8.8)$$

其中， β 用于调节与初始监督微调（SFT）模型 \mathcal{M}_{sft} 的偏差。

TL-Training 方法使大语言模型能够逐步完善其对工具使用的理解，并随着时间推移提高工具使用的准确性。文献 [394] 给出的实验结果表明，该方法仅使用 1217 个训练数据，就可以使得 CodeLLaMA-2-7B 模型在工具使用性能方面达到 GPT-4o 的能力。

8.3.2 推理规划

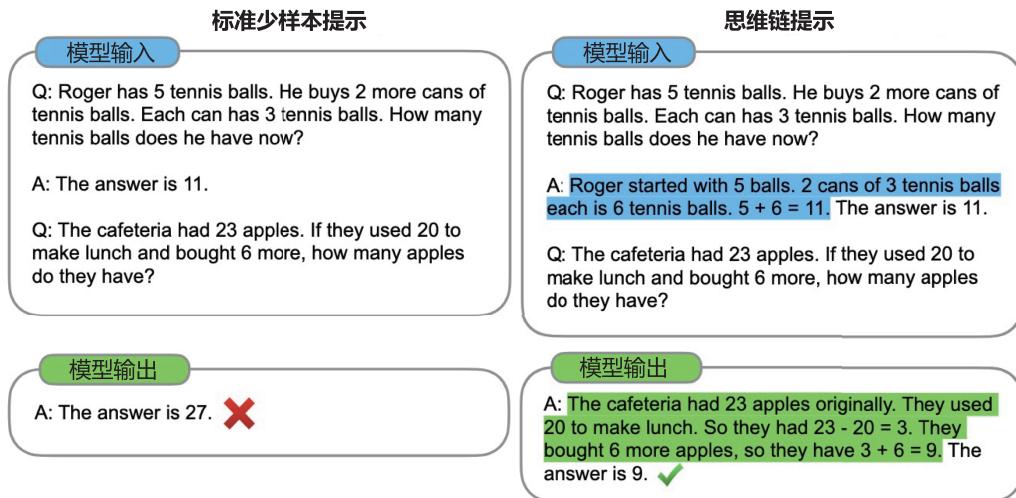
推理规划能力是大模型智能体的核心能力。只有提升大模型的推理和规划能力，才能使模型对环境和任务有深刻理解，从而生成并优化任务执行计划，制定合理的行动步骤以实现既定目标。然而，仅仅通过扩大语言模型的规模，并不能显著提升推理（Reasoning）能力，如常识推理、逻辑推理、数学推理等。通过示例（Demonstration）或者明确指导模型在面对问题时如何逐步思考，促使模型在得出最终答案之前生成中间的推理步骤，可以显著提升其在推理任务上的表现。这种方法被称为思维链提示（Chain-of-Thought Prompting）^[395]。同样地，面对复杂任务或问题时，大语言模型可以展现出良好的规划（Planning）能力。通过引导模型首先将复杂的问题分解为多个较为简单的子问题，然后逐一解决这些子问题，可使模型得出最终答案，这种策略被称为由少至多提示^[396]。本节将重点介绍如何利用思维链提示和由少至多提示这两种方式，提升大语言模型的推理规划能力。

1. 思维链提示

语言模型在推理能力方面的表现一直未能令人满意，一些研究人员认为这可能是因为此前的模式是直接让模型输出结果，而忽略了其中的思考过程。人类在解决包括数学应用题在内的、涉及多步推理的问题时，通常会逐步书写整个解题过程的中间步骤，最终得出答案。如果明确告知模型先输出中间的推理步骤，再根据生成的步骤得出答案，是否能够提升其推理表现呢？针对这

个问题，Google Brain 的研究人员提出了思维链（Chain-of-Thought, CoT）提示方式^[395]，除了将问题输入模型，还将类似题目的解题思路和步骤输入模型，使得模型不仅输出最终结果，还输出中间步骤，从而提升模型的推理能力。研究人员甚至提出了零样本思维链（Zero-shot Chain-of-Thought, Zero-shot CoT）提示方式，只需要简单地告知模型“让我们一步一步思考（Let's think step by step）”^[397]，模型就能够自动输出中间步骤。

思维链提示方式如图8.3所示，标准少样本提示（Standard Few-shot Prompting）技术在给模型的输入里面提供了 k 个[问题，答案]对，以及当前问题，由模型输出答案。而思维链提示在给模型的输入里面提供了 k 个[问题，思维链，提示]元组及当前问题，引导模型在回答问题之前先输出推理过程。可以看到在标准少样本提示下，模型通常直接给出答案，但是由于缺少推理步骤，直接给出的答案准确率不高，也缺乏解释。而在思维链提示下，模型输出推理步骤，在一定程度上降低了推理难度，最终结果的准确率有所提升，同时具备了一定的可解释性。

图 8.3 思维链提示方式^[395]

文献 [395] 使用了人工构造的思维链。然而，通过实验发现，使用由不同人员编写的符号推理示例在准确率上存在高达 28.2% 的差异，而改变范例的顺序在大多数任务中则只产生了不到 2% 的变化。因此，如果能够自动构建具有良好问题和推理链的范例，则可以大幅度提升推理效果。文献 [398] 发现，仅通过搜索相似问题并将其对应的推理过程作为范例对于效果提升而言作用十分有限，但是问题和推理链示例的多样性对于自动构建范例至关重要。因此，上海交通大学和 Amazon Web Services 的研究人员提出了 Auto-CoT^[398] 方法，通过采集具有多样性的任务和生成推理链来构建范例。Auto-CoT 算法的整体过程如图8.4 所示。Auto-CoT 包括以下两个主要阶段。

(1) 问题聚类：将给定数据集中的问题划分为几个簇（Cluster）。

(2) 范例采样：从每个簇中选择一个代表性问题，并基于简单的启发式方法使用 Zero-shot CoT 生成问题的推理链。

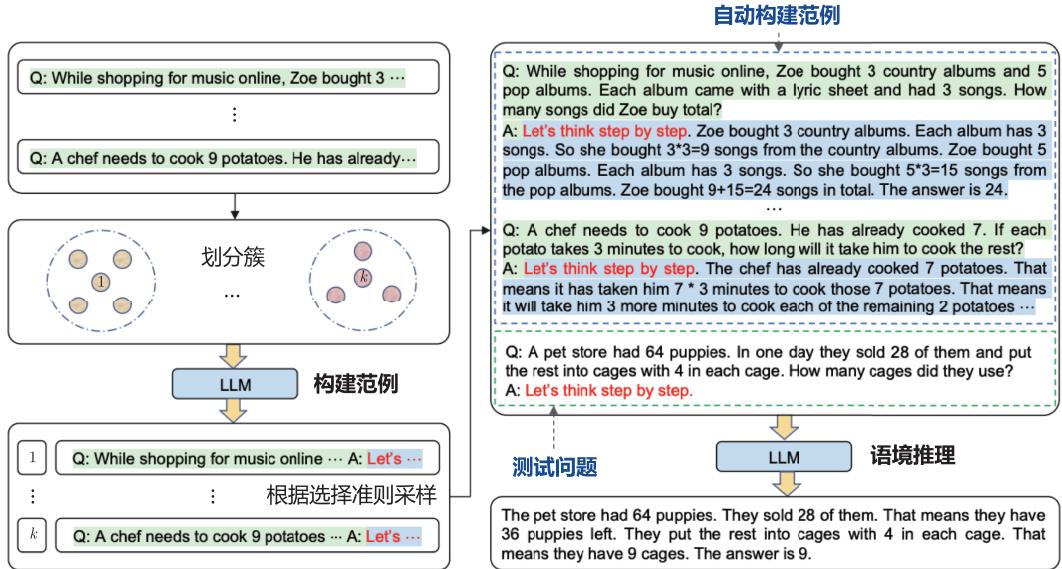


图 8.4 Auto-CoT 算法的整体过程^[398]

由于基于多样性的聚类可以降低相似性带来的错误，因此 Auto-CoT 算法对于给定的问题集合 Q 首先进行聚类。使用 Sentence-BERT^[399] 为 Q 中的每个问题计算一个向量表示。然后，使用 K-means 聚类算法根据问题向量表示生成 K 个问题簇。对于簇 i 中的问题，按照到簇中心的距离升序排列，并将排序后的列表表示为 $\mathbf{q}^{(i)} = [\mathbf{q}_1^{(i)}, \mathbf{q}_2^{(i)}, \dots]$ 。

在聚类的基础上，需要为问题生成推理链，采样生成符合选择标准的范例。对每个簇 i 构建一个范例 $\mathbf{d}^{(i)}$ ，包括问题、解释和答案。对于簇 i ，根据排序列表 $\mathbf{q}^{(i)} = [\mathbf{q}_1^{(i)}, \mathbf{q}_2^{(i)}, \dots]$ 迭代选择问题，直到满足条件为止。从距离簇 i 中心最近的问题开始考虑。如果当前选择了第 j 个问题 $\mathbf{q}_j^{(i)}$ ，则构建提示输入 $[\mathbf{Q} : \mathbf{q}_j^{(i)}, A : [P]]$ ，其中 $[P]$ 是一个单一提示“让我们一步一步思考”。将这个提示输入使用 Zero-Shot CoT^[397] 的大语言模型中，得到由解释 $\mathbf{r}_j^{(i)}$ 和提取的答案 $\mathbf{a}_j^{(i)}$ 组成的推理链。最终得到范例 $\mathbf{d}_j^{(i)} = [\mathbf{Q} : \mathbf{q}_j^{(i)}, A : \mathbf{r}_j^{(i)} \circ \mathbf{a}_j^{(i)}]$ 。如果 $\mathbf{r}_j^{(i)}$ 中的推理步骤小于 5 步，并且 $\mathbf{q}_j^{(i)}$ 中的词元小于 60 个，则将 $\mathbf{d}_j^{(i)}$ 纳入 $\mathbf{d}^{(i)}$ 。

此外，一些研究人员提出了对思维链提示的改进方法，例如从训练样本中选取推理最复杂的样本来形成示例样本，被称为 Complex-CoT^[400]。也有研究人员指出可以从问题角度考虑优化思维链提示，通过将复杂的、模糊的、低质量的问题优化为模型更易理解的、高质量的问题，进一步提升思维链提示的性能，这一方法被称为 Self-Polish^[401]。

2. 由少至多提示

当面对复杂任务或问题时，人类通常倾向于将其转化为多个更容易解决的子任务/子问题，并逐一解决它们，得到最终想要的答案或者结果。这种能力就是通常所说的任务分解（Task Decomposition）能力。基于这种问题解决思路，研究人员提出了由少至多提示（Least-to-Most Prompting）方法^[396]。这种方法试图利用大语言模型的规划能力，将复杂问题分解为一系列的子问题并依次解决它们。

由少至多提示流程如图8.5所示，主要包含问题分解阶段和逐步解决子问题阶段。在问题分解阶段中，模型的输入包括 $k \times [\text{原始问题}, \text{子问题列表}]$ 的组合，以及要测试的原始问题；在逐步解决子问题阶段中，模型的输入包括 $k \times [\text{原始问题}, m \times (\text{子问题}, \text{子答案})]$ 元组，以及要测试的原始问题和当前要解决的子问题。

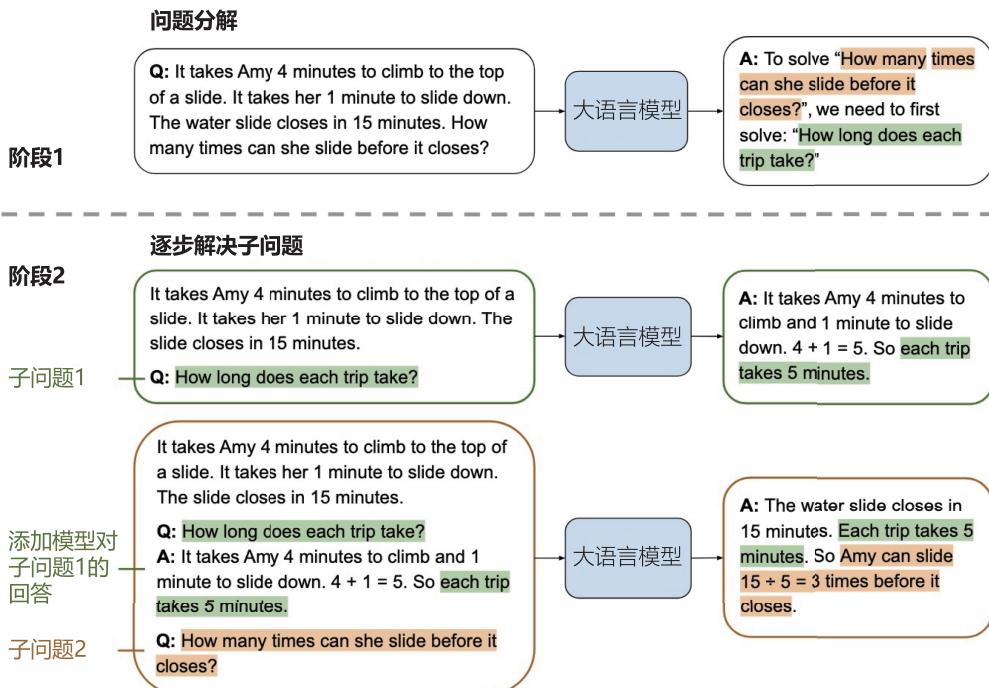


图 8.5 由少至多提示流程^[396]

上述过程的示例代码如下：

```

def CoT_Prompting(question, problem_reducing_prompt_path, problem_solving_prompt_path):
    # 读取prompt
    with open(file=problem_reducing_prompt_path, mode="r", encoding="utf-8") as f:
        problem_reducing_prompt = f.read().strip()
    with open(file=problem_solving_prompt_path, mode="r", encoding="utf-8") as f:
        problem_solving_prompt = f.read().strip()

    # 问题分解
    # 构造模型输入
    problem_reducing_prompt_input = problem_reducing_prompt + "\n\nQ: {}\nA:".format(question)
    # 调用模型得到回复
    problem_reducing_response = create_response(problem_reducing_prompt_input)
    # 得到分解后的子问题列表
    reduced_problem_list = get_reduced_problem_list_from_response(problem_reducing_response)

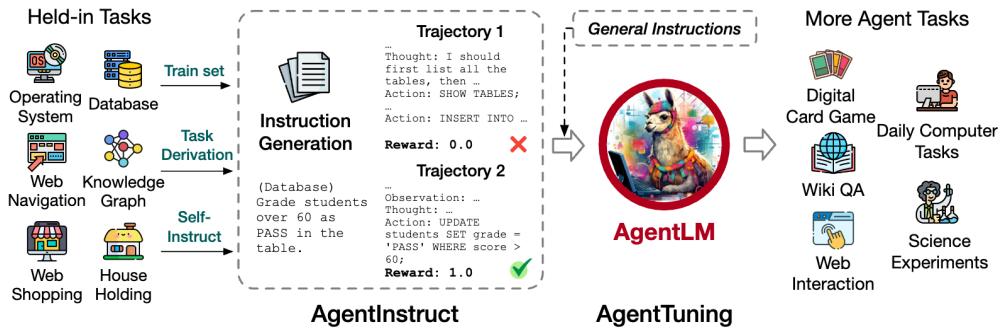
    # 串行解决问题
    problem_solving_prompt_input = problem_solving_prompt + "\n\nQ: {}".format(question)
    for sub_problem in reduced_problem_list:
        # 构造解决子问题的prompt
        problem_solving_prompt_input = problem_solving_prompt_input
            + "\n\nQ: {}\nA:".format(sub_problem)
        # 调用模型得到回复
        sub_problem_response = create_response(problem_solving_prompt_input)
        sub_answer = get_sub_answer_from_response(sub_problem_response)
        # 把当前子问题的答案拼接到之前的prompt上面
        problem_solving_prompt_input = problem_solving_prompt_input + sub_answer

    # 得到最终答案
    final_answer = answer_clean(sub_answer)
    # 返回答案
    return final_answer

```

3. AgentTuning

为提升大模型的通用推理能力，文献 [402] 提出了一种名为 AgentTuning 的方法，如图8.6所示。AgentTuning 主要由两个核心组件构成：一个轻量级的指令调优数据集 AgentInstruct，以及一种混合指令调优策略。该方法旨在增强模型的智能体能力的同时，尽可能保留其泛化能力。

图 8.6 AgentTuning 方法框架^[402]

AgentInstruct 数据集包含 1,866 条经过严格验证的交互轨迹。这些轨迹包含高质量的逐步推理过程（即 Chain-of-Thought），并涉及六种不同的智能体任务，包括 AlfWorld^[403]、WebShop^[393]、Mind2Web^[404]、知识图谱、操作系统和数据库。对于每个智能体任务，AgentInstruct 的构建包括三个主要阶段：指令构造、轨迹交互以及轨迹过滤。

对于 AlfWorld、WebShop、Mind2Web 以及知识图谱等已有训练集的任务，AgentInstruct 直接利用其训练数据，依次完成轨迹交互和轨迹过滤两个阶段。对于缺乏训练集的任务（如操作系统和数据库），则采用任务推导（Task Derivation）和自指令生成（Self-Instruct）^[405]的方法构建相应的指令，以确保数据的完整性与多样性。

在数据库任务的指令构建过程中，使用了 BIRD^[406] 数据集作为基础，该数据集是一个仅包含 SELECT 语句的数据库基准数据集。任务推导的过程分为两种方法：1) 基于 BIRD 数据集的子任务，通过问题和参考 SQL 语句生成轨迹。具体而言，执行参考 SQL 语句以查询数据库并获取结果，将其作为智能体的提交答案，并利用 GPT-4 根据上述信息生成智能体的推理过程。通过这一方式，可以直接从 BIRD 数据集中生成正确的交互轨迹。然而，该方法的交互轮次固定为 2，限制了轨迹的多样性。2) 直接构建指令而非轨迹。其具体步骤为：首先，将 BIRD 中的问题输入 GPT-4，与数据库进行交互以生成轨迹；随后，执行 BIRD 中的参考 SQL 语句，并将其结果与 GPT-4 生成的答案进行比对；最后，筛选出生成正确答案的轨迹。通过过滤错误答案，该方法仅保留正确的交互轨迹，从而构建出高质量且多样化的轨迹数据集。

在操作系统任务中，由于涉及终端操作的指令较难获取，采用了自指令生成方法构建该任务。具体而言，首先通过 GPT-4 生成与操作系统相关的任务，包括任务说明、参考解决方案以及评估脚本。随后，使用 GPT-4 作为求解器，依据生成的任务完成操作并记录其交互轨迹。在任务完成后，运行参考解决方案，并利用评估脚本将其结果与求解器的解答结果进行比对，仅保留参考解决方案与求解器解答一致的轨迹作为有效数据。

在初步构建指令后，AgentInstruct 数据集构造选用 GPT-4 (gpt-4-0613) 作为智能体模型进行

轨迹交互任务。在评估方法上，采用了 1-shot 评估策略，主要是为了满足智能体任务中对输出格式精确性的严格要求。对于每个任务，均提供来自训练集的完整交互过程作为示例。轨迹交互过程主要包括两个阶段。首先，向模型提供任务描述及一个成功的 1-shot 示例，以帮助其理解任务要求。随后进入正式交互阶段，向模型输入当前指令和必要的上下文信息。模型基于这些信息及此前的反馈内容，生成“思考”（Thought）并采取相应的行动。环境则根据模型的操作提供反馈，反馈内容可能包括状态变化或新的信息。上述过程循环进行，直至模型完成任务目标或达到 Token 限制。若模型连续三次生成相同的输出，则被视为重复性失败。若模型输出的格式不符合要求，则通过 BLEU 指标将其与所有可能的操作选项进行比较，并选择最接近的选项作为该步骤的操作。

在涉及真实场景的智能体任务中，由于任务复杂性较高，即便是 GPT-4 在此类任务上的表现也未能达到预期。为了确保数据质量，AgentInstruct 数据集构造过程中还对其交互轨迹进行了严格的过滤。每条交互轨迹都会获得一个奖励值，基于此奖励值，可以自动筛选出高质量的轨迹数据。最终构建了 1,866 条轨迹。

采用 AgentTuning 方法对 Llama 2 模型进行微调，并构建了开源的 AgentLM 模型。AgentLM 在未知智能任务中展现了很好的性能，同时在 MMLU、GSM8K、HumanEval 和 MT-Bench 等通用任务上依然保持了优异的表现。开源的 AgentLM-70B 在智能体任务表现上可与 GPT-3.5-turbo 相媲美。

8.3.3 长期记忆

大模型智能体的记忆模型由长期记忆和短期记忆构成。短期记忆可以通过将记忆内容以提示语句嵌入大模型输入上下文，依靠大模型的上下文理解能力实现存储和使用。长期记忆则通过构建记忆库来管理和检索，以实现知识的持久化存储与高效调用。在长期任务中，智能体将经验、知识等存储到记忆库，需要时检索记忆内容，与当前任务需求结合，提升问题解决效率。

大模型智能体实现长期记忆的常见方法之一是引入外部记忆库。长期记忆可存储为灵活的形式，例如文本文件或结构化数据库，并通过检索机制与反思机制进行访问与更新。外部记忆库可以采用向量数据库或可读写的神经网络记忆库等模式，模型能够动态地获取或更新所需知识。其中，检索增强生成（Retrieval-Augmented Generation, RAG）是一种典型方法，将检索与生成有机结合，适用于知识动态变化的场景。然而，该方法在应用中仍面临检索效率和记忆库质量的挑战，这对系统性能具有重要影响。

文献 [407] 提出了 MemoryBank 方法，允许模型调用相关记忆，通过持续的记忆更新不断进化，通过综合之前交互的信息，随着时间的推移理解和适应用户个性。MemoryBank 框架如图 8.7 所示，它由记忆存储、记忆检索以及记忆更新模块组成，每次用户输入的提示词会与记忆模块检索结果一起构成记忆增强的提示词。记忆存储作为主要的数据存储库，保存了对话的详细记录、事件总结和用户个性评估。记忆检索允许根据上下文进行记忆回忆。记忆更新受到艾宾浩斯遗忘曲线理论（Ebbinghaus Forgetting Curve Theory）的启发，改理论认为遗忘在学习之后立即开始，而且遗

忘的进程并不是均匀的，最初遗忘速度很快，以后逐渐缓慢。根据时间的流逝，帮助 AI 记住、有选择地忘记和加强记忆。MemoryBank 具有较好灵活性，可以适应开源和闭源的大语言模型，支持中英双语，并且可以与遗忘机制一起使用。

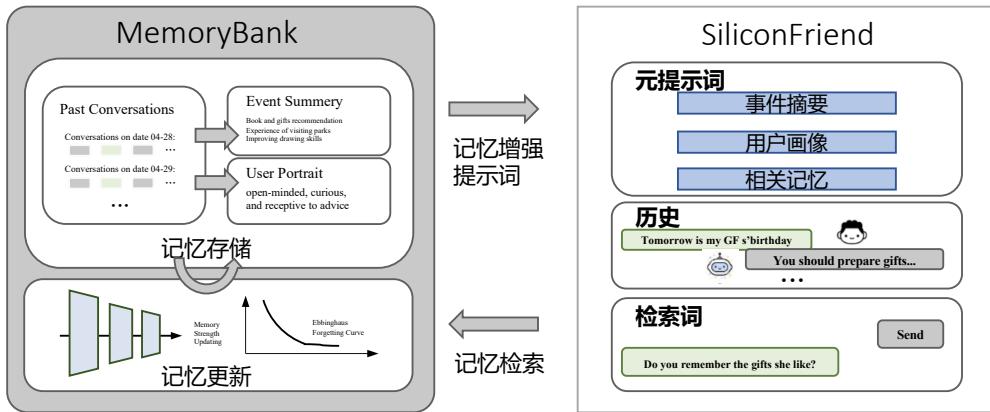


图 8.7 MemoryBank 方法框架^[407]

记忆存储（Memory Storage）是 MemoryBank 的核心组件之一，存储了丰富的信息，包含日常对话记录、过去的事件总结和用户个性评估的演变，从而构建了一个动态的多层次记忆全景图。通过按时间顺序记录多轮对话并添加时间戳，构建了有序的交互历史。这种细致的记录不仅支持精确的记忆检索，还为后续记忆更新提供了详细索引。

MemoryBank 借鉴了人类记忆的复杂性，不仅简单存储，还对对话进行提炼，生成每日事件总结，并进一步凝练为全局总结，形成层次化的记忆结构，为用户交互和重要事件提供鸟瞰式视角。具体来说，以之前的每日对话或每日事件为输入，要求大语言模型使用提示“总结内容 / 对话 / 事件 / 中的事件和关键信息”来总结每日事件或全局事件。此外，还专注于用户个性理解，通过长期交互不断评估和更新个性洞察，最终形成对用户个性的全局理解。这种多层次方法使 AI 伴侣能够学习、适应并根据用户独特特质定制响应，从而显著提升用户体验。以每日对话或个性分析为输入，要求大语言模型使用提示：“根据以下对话，请总结用户的个性特征和情绪。”或“以下是一段时间内用户表现出的个性特征和情绪。请提供一个高度简洁和概括的用户个性总结。”来分析用户。

MemoryBank 所采用的记忆检索机制类似于知识检索任务，具体实现上采用了一种类似稠密篇章检索（Dense Passage Retrieval, DPR）的双塔稠密检索模型^[408]。每次对话及其对应的总结均被视为一个记忆片段 m ，并通过编码器模型 $E(\cdot)$ 进行预编码，生成该片段的上下文向量化表示 h_m 。整个记忆存储 M 被预编码为 $M = \{h_m^0, h_m^1, \dots, h_m^{|M|}\}$ 。随后，这些向量表示通过 FAISS 方法^[409]进行索引，以实现高效的检索操作。在实际检索过程中，当前对话的上下文 c 同样通过编码器 $E(\cdot)$