Agent入门

Agent被誉为AI的终极形态，甚至是最有可能通往AGI的道路，那么什么是Agent?

OpenAI华人科学家翁丽莲给出的答案是：

Agent=大模型+记忆+主动规划+工具使用

但也有人认为这个定义缺失了对环境交互能力的强调。

举个例子：当用户输入给大模型：我想要开一家服装店，请给我一些建议。直接把这个query输入给LLM得到的回答往往平平无奇并且没有任何指导意义，但如果是一个人类助手来解决这个问题，通常会选择去查阅相关的网页、询问有经验的人、查询当地的天气和当前流行的趋势，并依据已知的信息修改后面的计划，在实践中反思、修正行为，这就是Agent可以做的。

它完全模仿了人类在面对问题、解决问题过程中的思维方式和工具使用行为，可能只需要一些创意和调优手段，普通人也能通过一些agent平台（比如字节的coze、百度的Agentbuilder等）做出让人眼前一亮的agent，从而获得商业上的价值，但是Agent发挥价值基于这样一个假设：LLM拥有媲美人脑的优秀的推理能力，但这一点目前还没有完全实现。

因此目前Agent的实现还需要结合大量的传统技术，比如搜索规则、知识图谱等，更重要的是，Agent需要知道自己的能力边界。

AI Agent系统组成

LLM就相当于Agent的大脑，Lilian Weng认为一个AI Agent系统由以下几个部分组成：

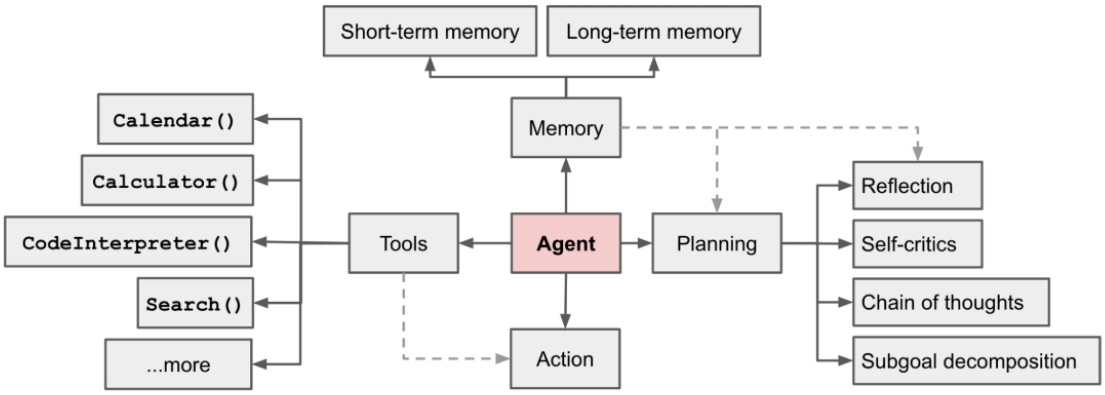


Fig. 1. Overview of a LLM-powered autonomous agent system.

第一部分：Planning

复杂任务的处理常常涉及多个步骤，因此Agents需要对这些步骤做一个提前的规划，侧重点在以下两个方面：

- 子目标分解：将大型任务分解为更小、更易于管理的子目标，相关的方法有CoT、ToT、LLM+P等
- 自我反思：对过去的行为进行自我批评和反思，从错误中吸取教训，从而修正未来的执行步骤，相关的方法有ReAct、Reflexion、CoH、正则化项、算法蒸馏等

CoT

CoT目前已经是提升模型解决复杂问题的标准解法，通过让模型“Let’s think step by step”，将大型任务转化为多个可管理的子任务，有很多相关的工作都通过简单的方法取得了不错的效果。

ToT

在CoT的基础上，ToT通过在每一步探索多种推理可能性来扩展模型的性能。首先将问题分解为多个思维步骤，每个步骤生成多个思维，从而创建一个树状结构。搜索过程可以是BFS和DFS，每个状态由分类器或多数票进行评估。

LLM + P

LLM用于将自然语言指令转换为机器可以理解的形式，并生成PDDL描述，接下来PDDL描述可以被规划器P使用，生成合理的计划并决定分配的任务。

> **PDDL（Planning Domain Definition Language）**
>
> PDDL是一种标准化的通用领域规划语言，它是能够用于描述可行动作、初始状态和目标状态的语言，用于帮助规划器生成计划。PDDL通常被用于AI的自动规划问题，例如机器人路径规划、调度问题、资源分配等。

自我反思：

自我反思可以让Agents能够通过改进过去的行动检测、纠正过往的错误以不断提高自身表现。

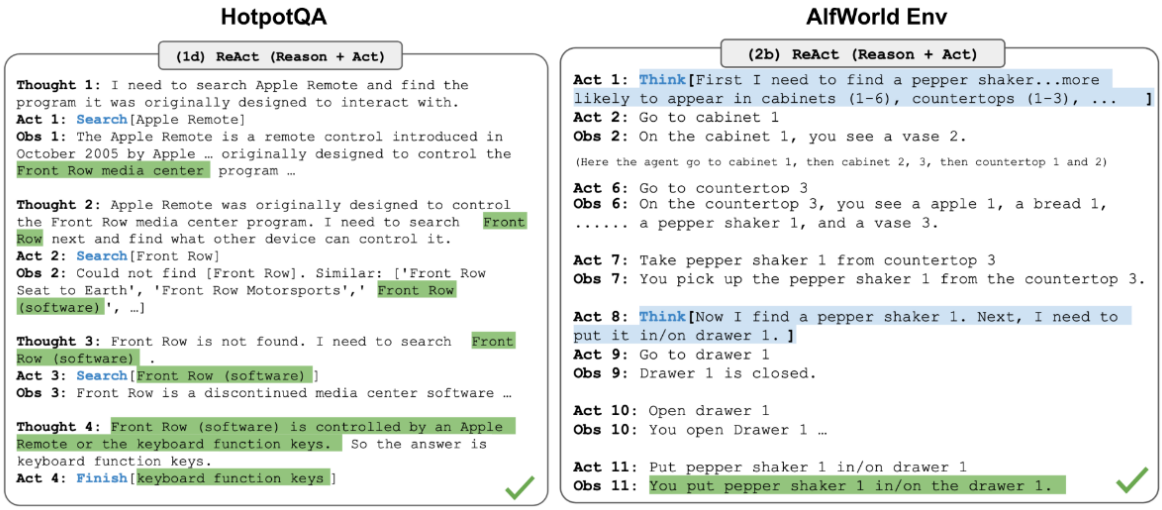
ReAct

ReAct的核心思想就是让语言模型不仅仅停留在“做”行动，还要通过“思考”来指导行动。推理与行动的结合，使得LLM在知识密集型任务（如百科查询）和需要决策的任务（如选择最佳方案）中，能够做得更好。就是让LLM把内心碎碎念“说”出来，然后再根据内心思考做相应的动作，模仿人类的推理过程，以提高LLM答案的准确性。

此外，进一步提高ReAct准确率的方法，就是用推理运动轨迹作为数据集对模型进行微调，相当于人类“内化”知识的过程。

ReAct的prompt template如下：

```
Thought: ...
Action: ...
Observation: ...
... (Repeated many times)
```

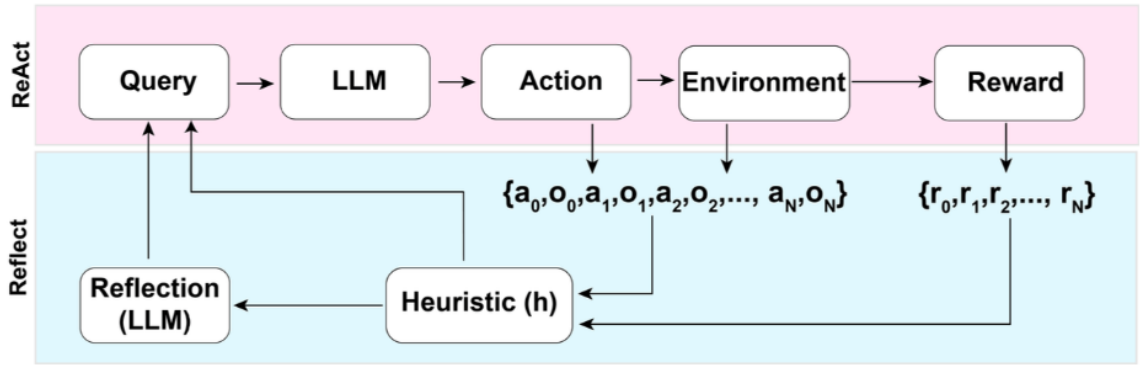


知识密集型任务的推理轨迹示例

Reflexion

Reflexion 是一种强化学习框架，旨在通过动态记忆和自我反思增强智能体的推理能力。它结合了 **强化学习**（RL）和 **自我反思** 的概念，使得智能体不仅能从成功的行动中学习，还能从失败中获得反馈，改进未来的决策。

该框架中的奖励模型使用的是简单的二元奖励（如，成功/失败），动作空间使用类似ReAct中的设置。在每个动作a_t之后，Agents会计算一个启发式值h_t，决定是否重置环境或继续执行任务。



Reflexion框架的图解

1. **启发式函数**：
   - **启发式函数（Heuristic Function）**：它帮助Agent判断某个轨迹（即一系列的行动）是否低效或者存在 **幻觉**，如果是，则停止该轨迹并重新开始。
     - **低效规划（Inefficient Planning）**：指Agent采取的行动过于缓慢或没有带来实际的进展，尽管进行了很多步骤。
     - **幻觉（Hallucination）**：指Agent重复做相同的动作，且这些动作并未产生新的观察或变化，导致Agent进入了死循环，无法取得实际进展。
2. **自我反思**：
   - **两步示例（Two-shot Examples）**：通过给Agent提供 **两步示例**，其中每个示例包括：
     1. **失败轨迹**：Agent尝试过的、失败的行动序列。
     2. **理想反思**：对于失败轨迹的理想反思，指导Agent如何在未来改变计划，避免类似的失败。
   - 这些反思会被加入到Agent的 **短期记忆** 中，最多存储三个反思。短期记忆允许Agent在执行任务时引用这些反思，以帮助其做出更好的决策。

Chain of Hindsight

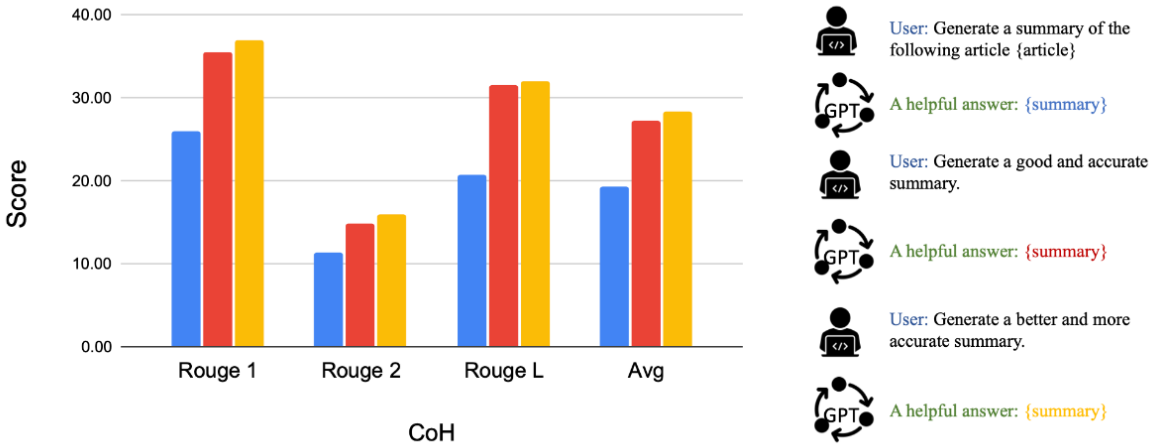
CoH是一种通过利用 **自我反思** 来改进模型输出的训练方法，目标是使得模型能够基于过去的输出和反馈生成更高质量的结果。这个方法通过 **人类反馈数据** 和 **历史输出的序列** 来进行模型的 **微调**，从而帮助模型提高其生成结果的质量。

人类反馈数据是一个元组，$D_h = (x, y_i, r_i, z_i)_{i=1}^n$， 其中，x是prompt， y_i是模型的补全， r_i是人类对y_i的评分， z_i 是人类对模型输出提供的事后反馈（hindsight feedback），这些反馈元组按 奖励排序（reward-ranked），即人类评分较高的输出排在前面，给模型提供了反馈的优先顺序。

在训练过程中，CoH通过 **监督微调** 来调整模型的行为。具体来说，训练数据是一个**序列**，模型基于历史的输出和反馈进行自我反思，生成更好的答案。训练的目标是让模型学会从 **过去的输出序列** 中 **自我反思**，并基于这些反思生成更好的输出。例如，给定一段历史输出和反馈，模型学会如何改进其当前的输出。

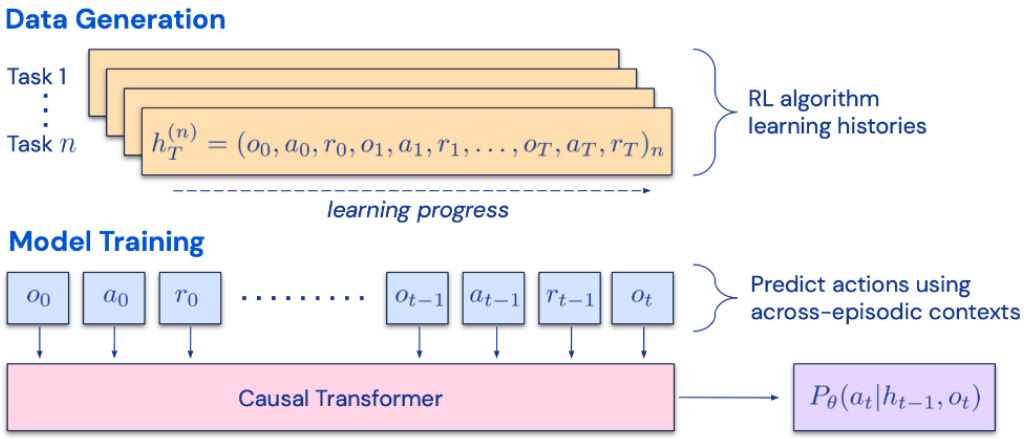
CoH方法的核心在于让模型能够反思自己的输出，并从过去的经验中学习改进策略。通过在模型训练时给出多个 **过去的输出序列** 和对应的 **反馈**，模型不仅要预测当前最优的输出，还要理解过去输出的不足之处，从而在新任务中避免相同的错误。

为了避免模型在训练中过度依赖历史数据中的某些常见词汇或模式（从而产生过拟合），CoH在训练过程中加入了一个正则化项（Regularization Term），确保模型能够泛化。具体做法是，在训练时 **随机遮蔽0%-5%的过去token**，迫使模型不依赖于某些常见的词汇或模式，而是能够进行更广泛的学习。



使用CoH进行微调后，模型可以按照指令生成顺序增量改进输出

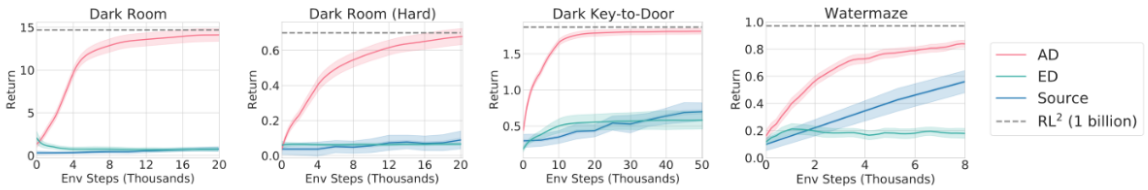
CoH的概念与 **算法蒸馏（AD）** 相似，后者应用于强化学习任务中。具体来说，**Algorithm Distillation (AD)** 是一种将强化学习中的“学习历史”提取并蒸馏成一个策略网络的技术。它的目标是通过 **行为克隆（Behavioral Cloning）** 的方式，将Agent从过去的学习历史中获得的知识传递给模型，而不是直接训练一个任务特定的策略。AD主要的创新在于通过将多个学习回合（episode）的历史信息作为输入，来训练一个任务无关的强化学习策略，进而提升模型在多任务环境中的表现。



AD的工作原理

1. **训练过程中使用学习历史**：
    - 在训练阶段，AD通过从多个强化学习回合中收集 **学习历史**，并将这些历史串联起来，作为模型的输入。这样，模型可以利用整个学习过程中的经验来改进当前的决策。
    - **多回合历史**：每个回合中的数据并不是孤立的，AD通过将 **2-4个回合的历史数据** 作为输入，帮助模型从过去的学习经验中抽取规律，预测当前最优的动作。
    - **目标**：通过学习这些历史信息，模型不再需要依赖于每个特定任务的策略，而是学会一种 **任务无关的通用决策策略。**
2. **强化学习的蒸馏过程**：
    - 传统的 **强化学习（RL）** 需要通过与环境的交互来更新策略，但在AD中，重点不在于每个单独的回合，而是 **将多个回合的历史进行蒸馏。**这种蒸馏方法的关键在于通过行为克隆将 **跨回合的学习历史** 转化为一个神经网络模型，这样该模型就能直接根据过去的学习历史做出更好的决策，而不需要每次都进行完全的强化学习训练。
    - 这样，AD通过 **行为克隆** 的方式，将强化学习过程中的学习历史和环境交互抽象成策略，从而获得一个可以在新的任务中快速适应的通用策略。

论文使用了三个基线进行对比：ED（专家蒸馏，使用专家轨迹而非学习历史的行为克隆）、源策略（用于生成UCB蒸馏的轨迹）、RL^2（2017年提出的一种在线强化学习方法，作为上限进行比较）。ED依赖于专家的行为示范，而AD通过多个回合的历史数据进行蒸馏，因此AD不依赖于专家轨迹，而是依赖于**跨回合的学习历史**。在与RL^2的对比中，AD通过 **离线强化学习** 达到了接近RL^2的效果，且 **训练速度比其他基线方法快。**AD通过 **历史信息蒸馏** 的方式，显著提高了 **样本效率**，使得模型在较少的交互数据下就能得到较好的结果。



第二部分：Memory

可以把记忆存储在向量数据库中，需要的时候再使用ANN等MIPS方法进行**最大内积搜索**，从而以损失少量的精度来换取巨大的速度提升。

第三部分：Tool Use

给大模型配备工具可以显著扩展大模型的功能以及处理复杂任务的能力。

MRKL (Modular Reasoning, Knowledge, and Language)

MRKL是一种 **神经符号架构**（neuro-symbolic architecture），旨在将 **语言模型（LLM）与专家模块** 相结合。这些模块可以是神经网络（例如深度学习模型）或符号系统（例如数学计算器、货币转换器、天气API等）。在MRKL系统中，LLM充当 **路由器**（router），根据任务的需求将查询请求发送给最适合的专家模块。换句话说，LLM决定哪个外部工具（如计算器、API等）适合处理当前的问题。MRKL框架的一个实验表明，LLM在解决 **口头算术问题** 时比在 **明确陈述的算术问题** 上更难得到正确答案，尤其是在无法正确提取算式的参数时。这揭示了LLM的 **能力局限性**，即知道何时以及如何使用外部工具是非常关键的。

TALM (Tool Augmented Language Models)

TALM通过**微调（fine-tuning）**语言模型，使其学会如何 **调用外部工具API**（例如搜索引擎、计算器等）。这种方法的核心在于：当模型学会了如何调用API并利用外部工具时，能更好地处理和解决特定任务。TALM通过增加**API调用的注解**来扩展训练数据集，并评估这些新增的API调用注解是否能够提高模型输出的质量。换句话说，模型通过学习如何使用外部工具，提升其对复杂任务的处理能力。

Toolformer

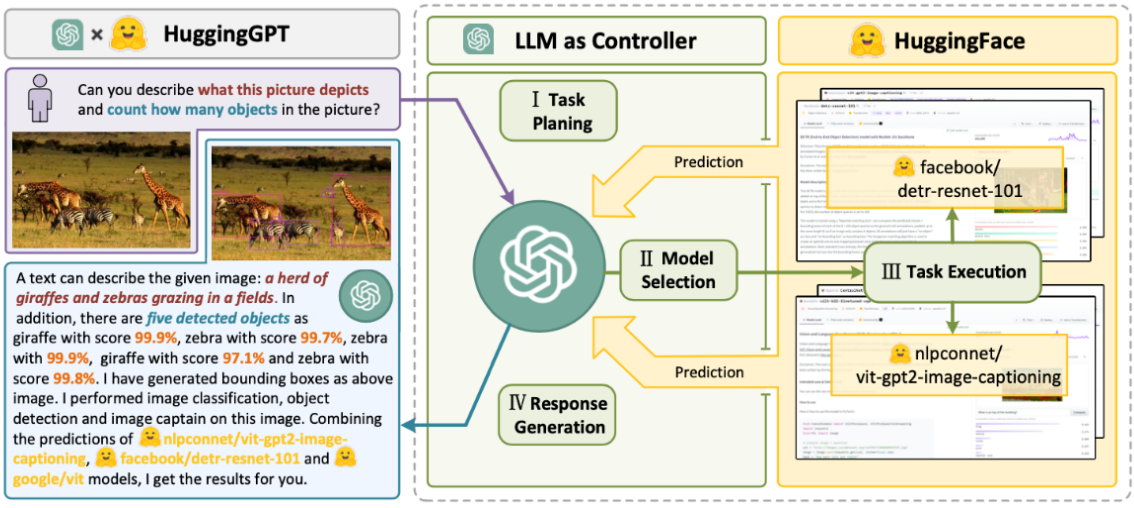
Toolformer是另一种微调语言模型的技术，目的是让语言模型学会 **使用外部工具API**，类似于TALM。不同之处在于，Toolformer框架更侧重于 **基于模型输出质量来选择合适的API调用。**通过微调，使得模型能够基于实际情况（例如任务的具体要求）自动选择和调用外部工具，从而提高任务完成的效率和准确性。

ChatGPT Plugins 和 OpenAI API Function Calling

- **ChatGPT Plugins**：ChatGPT的插件机制使得用户可以通过安装插件扩展ChatGPT的功能。这些插件通常由 **第三方开发者提供**，允许ChatGPT访问外部的 **服务或工具**（例如查询天气、计算机算式、执行交易等）。
- **OpenAI API Function Calling**：类似于插件，但功能更为 **自定义**，通过API调用的方式，允许开发者定义函数，使得LLM可以与这些函数进行交互，完成特定任务。

HuggingGPT

HuggingGPT是一个框架，它通过将 **ChatGPT** 作为任务规划器来 **选择合适的模型**，这些模型可以来自HuggingFace平台。HuggingGPT会根据 **模型描述** 来选择合适的任务执行模型，然后基于执行结果来 **总结回应。**



HuggingGPT的工作原理

该系统包括四个阶段：

- 任务规划：LLM作为大脑，负责将用户请求解析为多个任务，每个任务有4个关联的属性：任务类型、ID、依赖项、参数，通过few-shot的方式，指导LLM进行任务解析和规划。指令如下（中文版）：

```
AI助手可以将用户输入解析为多个任务：[{"task": 任务, "id": 任务_id, "dep": 依赖任务_id, "args": {"text": 文本, "image": 图片_URL, "audio": 音频_URL, "video": 视频_URL}}]。其中，“dep”字段表示当前任务依赖于先前任务生成的新资源。特殊标签“-task_id”指代依赖任务生成的文本、图片、音频和视频资源。任务必须从以下选项中选择：{{ 可用任务列表 }}。任务之间存在逻辑关系，请注意它们的顺序。如果用户输入无法解析，您需要回复空的JSON格式。以下是几个参考案例：{{ 示例演示 }}。聊天历史记录作为{{ 聊天历史 }}。从聊天历史中，您可以找到用户提到的资源路径来规划任务。
```

- 模型选择：LLM可以根据任务类型将请求分配给相应的专家模型，这些模型一般限制为多项选择题的类型。然后，LLM提供一个可供选择和使用的模型列表。由于上下文长度的限制，LLM需要基于任务类型进行过滤。

```
根据用户请求和调用命令，AI助手帮助用户从模型列表中选择一个合适的模型来处理用户请求。AI助手仅输出最合适模型的模型ID。输出必须严格遵循JSON格式：“id”: “id”, “reason”: “选择该模型的详细原因”。我们为您提供了可供选择的模型列表{{ 候选模型 }}，请选择一个模型。
```

- 任务执行：专家模型执行并记录特定任务的结果。

```
根据用户输入和推理结果，AI助手需要描述处理过程和结果。前面的阶段可以形成如下内容：

- 用户输入: {{ 用户输入 }}
- 任务规划: {{ 任务 }}
- 模型选择: {{ 模型分配 }}
- 任务执行: {{ 预测结果 }}

你必须首先直接回答用户的请求。然后描述任务过程，并以第一人称的方式向用户展示分析和模型推理结果。如果推理结果包含文件路径，必须告知用户完整的文件路径。
```

- 响应生成：LLM接收执行结果并向用户提供汇总结果。

挑战

1、有限的上下文长度：历史信息、指令的细节、API call的内容、模型的回复，其实都需要放到上下文中，尤其是像self-reflection这种方式依赖于从过去的错误中学习，缺乏长期记忆会导致Agent分心，从而无法专注于目标，导致不可预测的行为，所以需要比较长的上下文窗口，虽然现在有一些方法比如外挂向量知识库的方式解决试图这个问题，但也不如完全将这些信息都包括在上下文中效果好。

2、长期计划和任务分解：对于比较长期和复杂的任务而言，LLM还做不到像人类那样，从错误中学习和修正自身行为。

3、过于依赖自然语言：当前LLM仍然存在幻觉、输出内容不标准等问题，所以对模型输出进行解析和后处理也是一个关注点。

参考：Weng, Lilian. (Jun 2023). “LLM-powered Autonomous Agents”. Lil’Log.