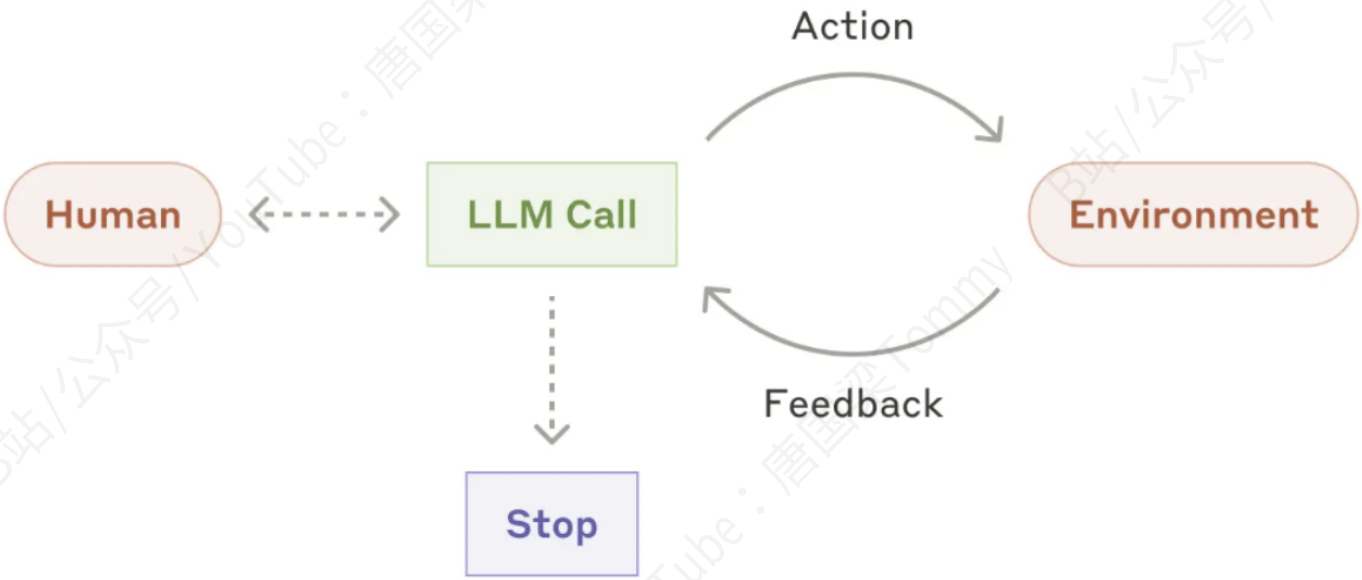
一. AI Agent 概述

1. 什么是 AI Agent?

AI Agent 是一种由大语言模型（LLM）驱动的自主智能体，通过整合**记忆、规划、工具使用和自我反思**等多种能力，完成复杂任务。这些智能体能够结合人类提供的初始目标，利用 LLM 的自然语言理解能力，处理复杂指令并与外部工具协作。



2. 核心组件

2.1 速览

- **规划**
  - **子目标和任务分解**：智能体将大型任务分解为更小的、可管理的子目标，从而高效处理复杂任务。
  - **反思与改进**：智能体能够对过去的行为进行自我批评和反思，从错误中学习，并对未来步骤进行改进，从而提升最终结果的质量。
- **记忆**
  - **短期记忆**：所有上下文学习可以视为利用模型的短期记忆。
  - **长期记忆**：提供智能体保留和回忆（无限）信息的能力，通常通过外部向量存储和快速检索来实现。
- **工具使用**
  - 智能体学习调用外部API，以补充模型权重中缺失的信息（通常难以在训练后更改），包括当前信息、代码执行能力、专有信息资源访问等。

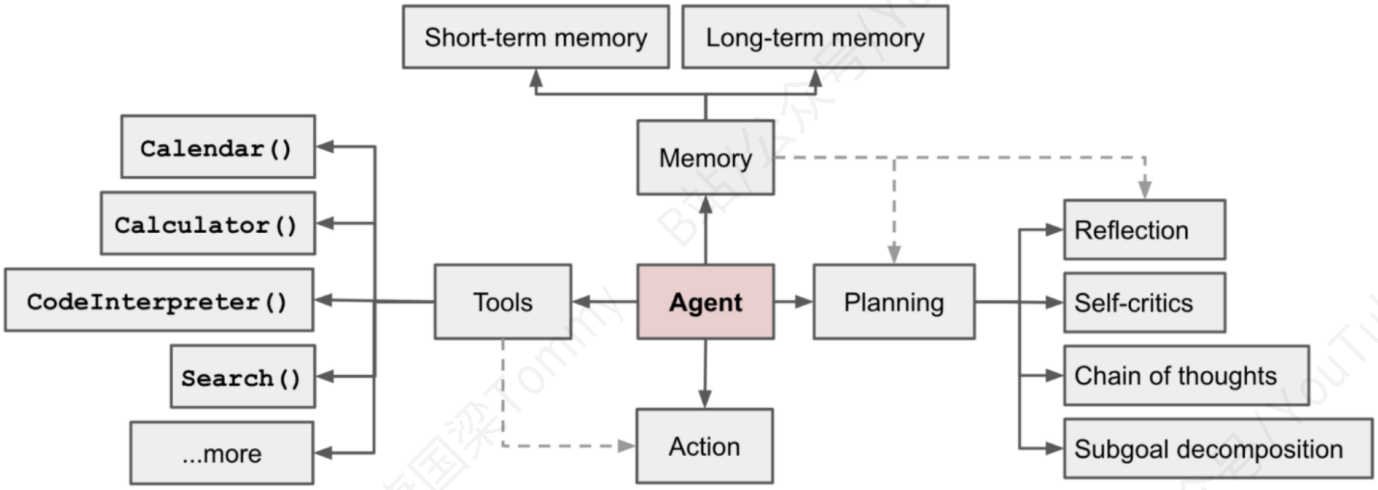
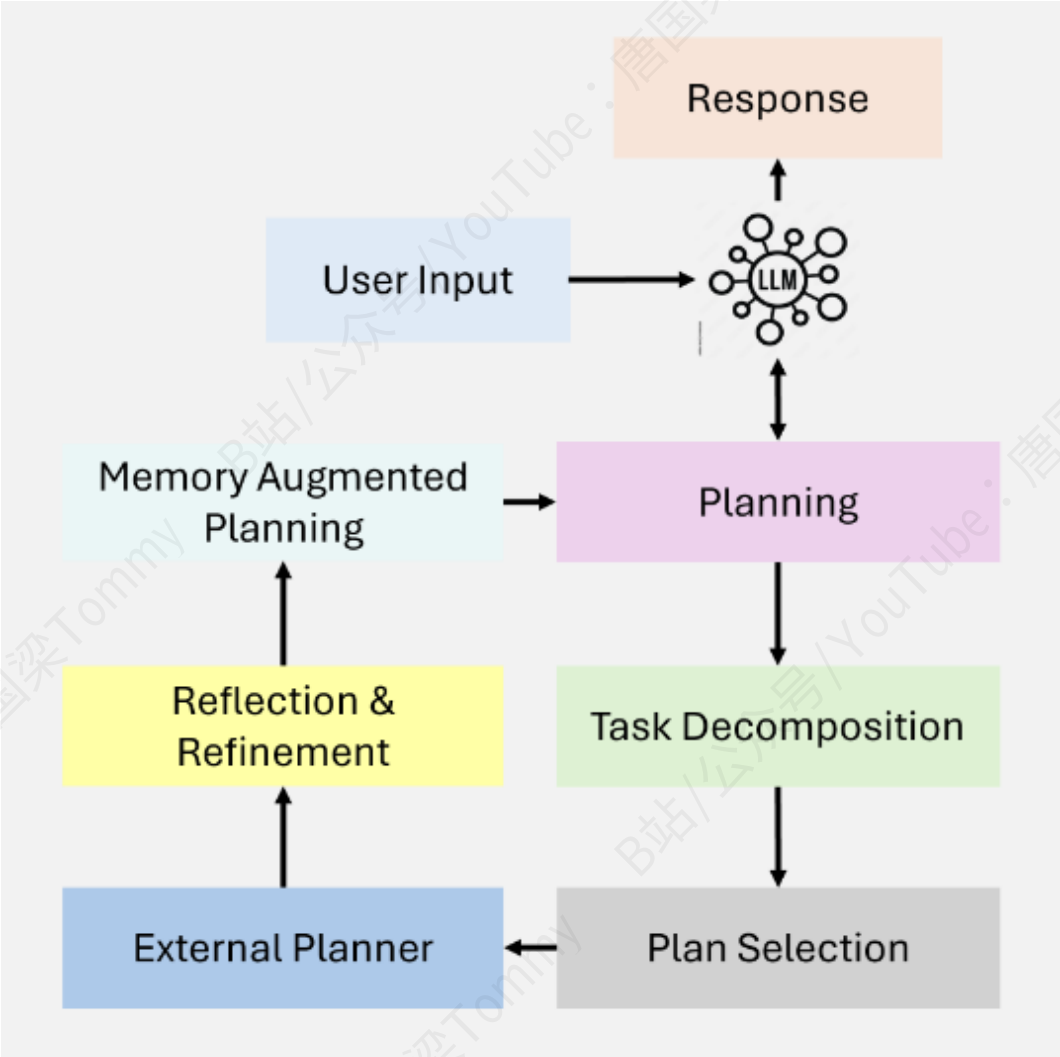


Fig. 1. Overview of a LLM-powered autonomous agent system.

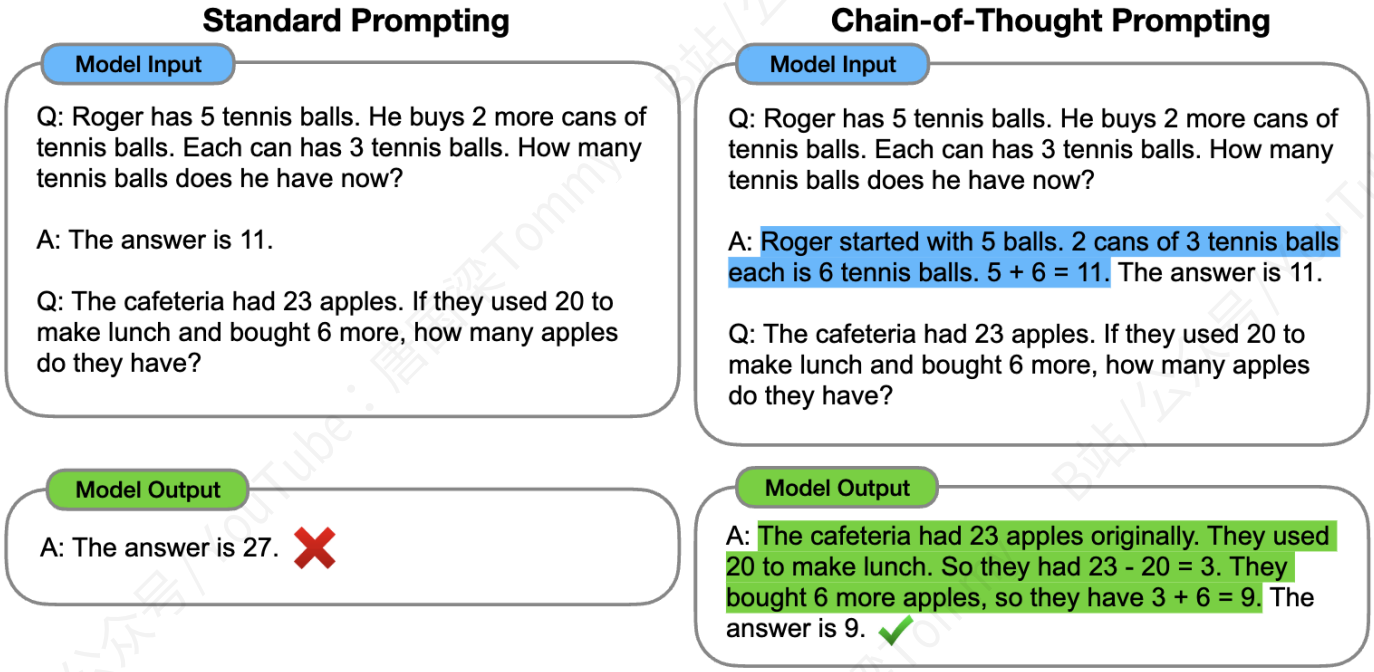
2.2 组件一：规划

（1）任务分解

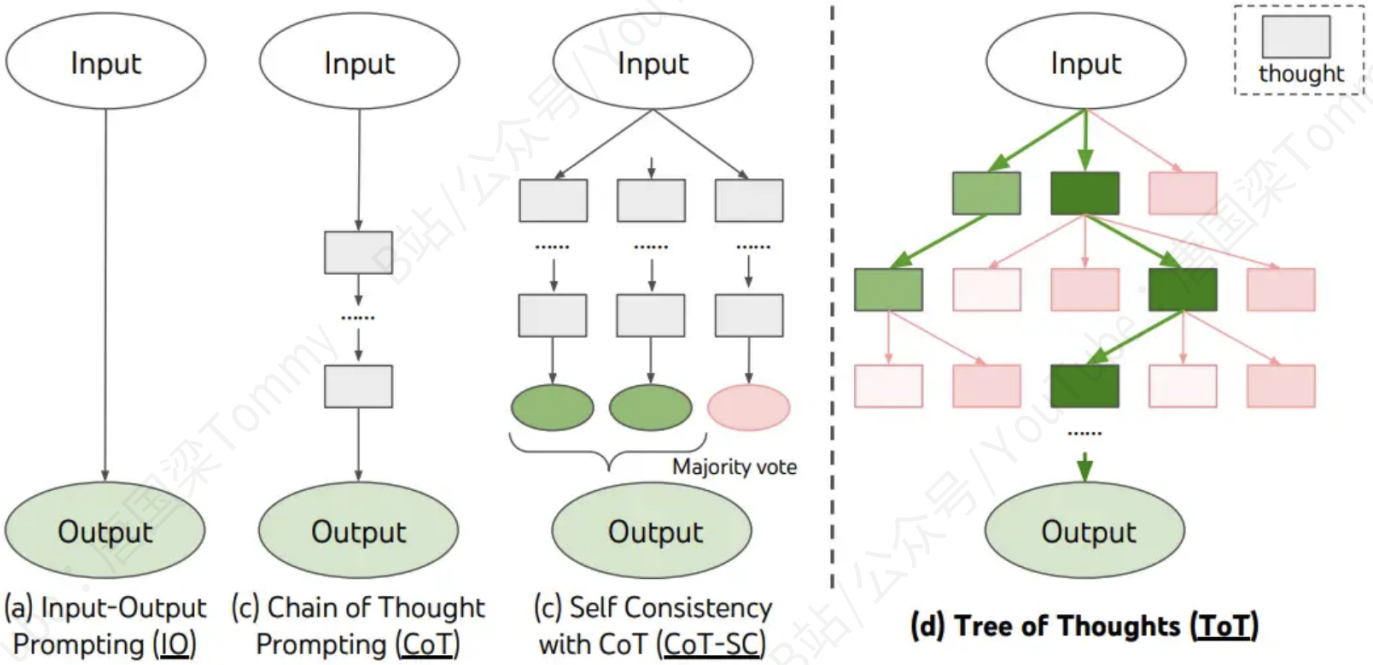


- **思维链（Chain of Thought，CoT；Wei等，2022年）**

已成为提升复杂任务模型性能的标准提示技术。该模型被指导“逐步思考”，以利用更多推理时间计算能力，将复杂任务分解为更小、更简单的步骤。CoT将大型任务转化为多个可管理的子任务，并为理解模型的思维过程提供了指引。



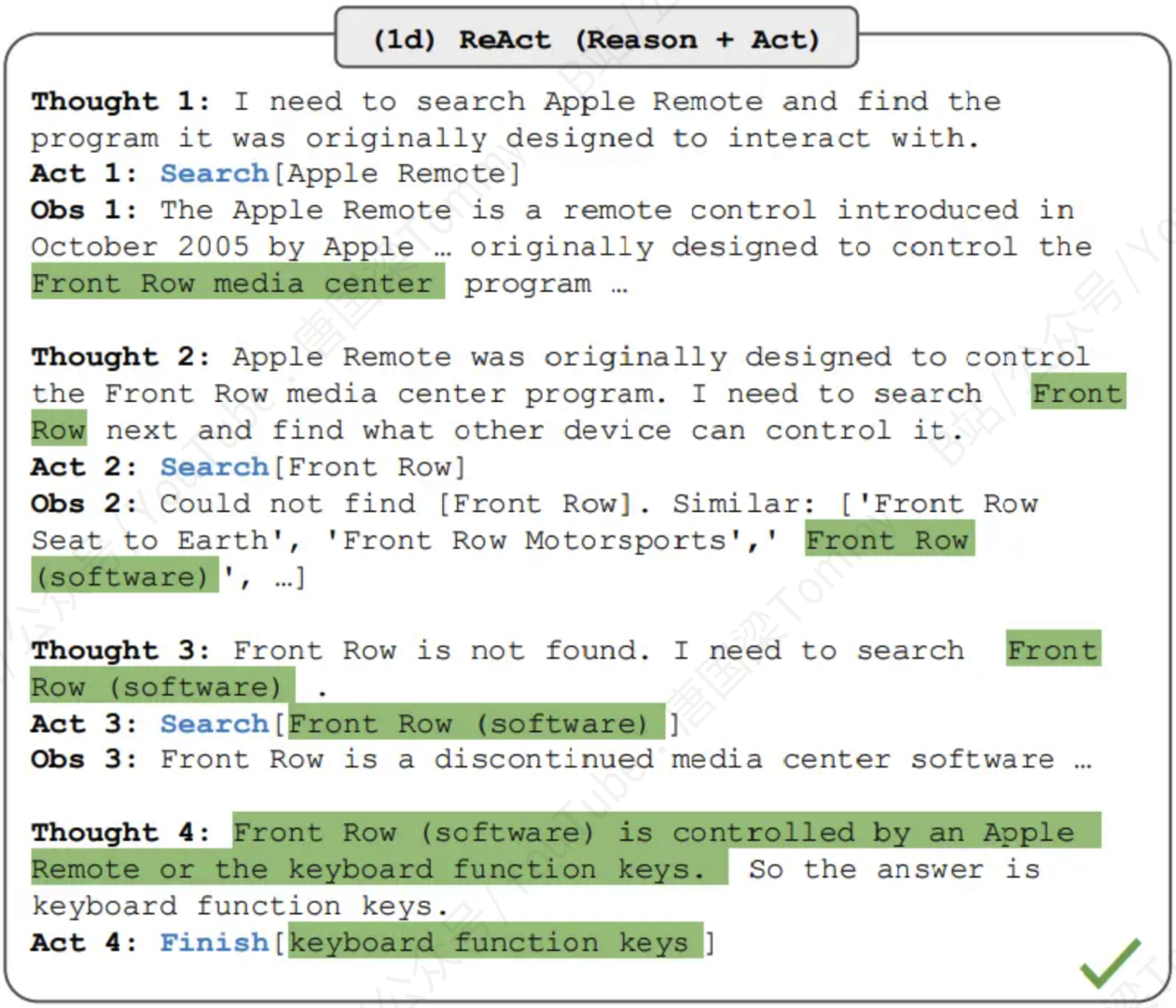
- **思维树（Yao等，2023年）**
  通过在每个步骤中探索多种推理可能性扩展了CoT。它首先将问题分解为多个思维步骤，并为每个步骤生成多个思维，形成树状结构。搜索过程可以是BFS（广度优先搜索）或DFS（深度优先搜索），每个状态通过分类器（通过提示）或多数投票进行评估。



（2）自我反思

自我反思是一个重要的方面，它允许自主智能体通过改进过去的行动决策和修正先前的错误来进行迭代改进。在实际任务中，试错是不可避免的。

- **ReAct（Yao等，2023年）**
  通过扩展操作空间，将任务特定的离散动作和语言空间相结合，将推理和操作集成到LLM中。前者使LLM能够与环境交互（例如：使用维基百科搜索API），而后者则提示LLM在自然语言中生成推理轨迹。



2.3 组件二：记忆

记忆可以定义为获取、存储、保持和随后检索信息的过程。在人类大脑中有多种类型的记忆：

1. **感觉记忆（Sensory Memory）**：
   这是记忆的最早阶段，提供保留感官信息印象的能力（视觉、听觉等），即使原始刺激已经结束。感觉记忆通常只能持续几秒钟。其子类别包括图像记忆（视觉）、回声记忆（听觉）和触觉记忆（触觉）。
2. **短期记忆（Short-Term Memory, STM）或工作记忆（Working Memory）**：
   这种记忆存储我们当前意识到并需要用来完成复杂认知任务（如学习和推理）的信息。短期记忆的容量通常被认为约为7个项目（Miller，1956），并且持续时间为20-30秒。
3. **长期记忆（Long-Term Memory, LTM）**：
   长期记忆可以将信息存储非常长的时间，从几天到几十年，并且拥有几乎无限的存储容量。长期记忆有两种子类型：

- **显性/陈述性记忆（Explicit/Declarative Memory）**：这种记忆包括事实和事件，指那些可以被有意识地回忆起来的记忆，包括情景记忆（事件和经历）和语义记忆（事实和概念）。
- **隐性/程序性记忆（Implicit/Procedural Memory）**：这种记忆是无意识的，涉及自动执行的技能和例行事务，例如骑自行车或打字。

2.4 组件三：工具使用

工具使用是人类显著且独特的特征。我们创造、修改并利用外部工具来完成超越我们身体和认知极限的任务。为LLM配备外部工具可以显著扩展模型能力。

工具使用模块包含4个阶段：

(1) 任务规划：

LLM 作为大脑，将用户请求解析为多个任务。他们使用少样本示例指导LLM进行任务解析和规划。

(2) 工具选择:

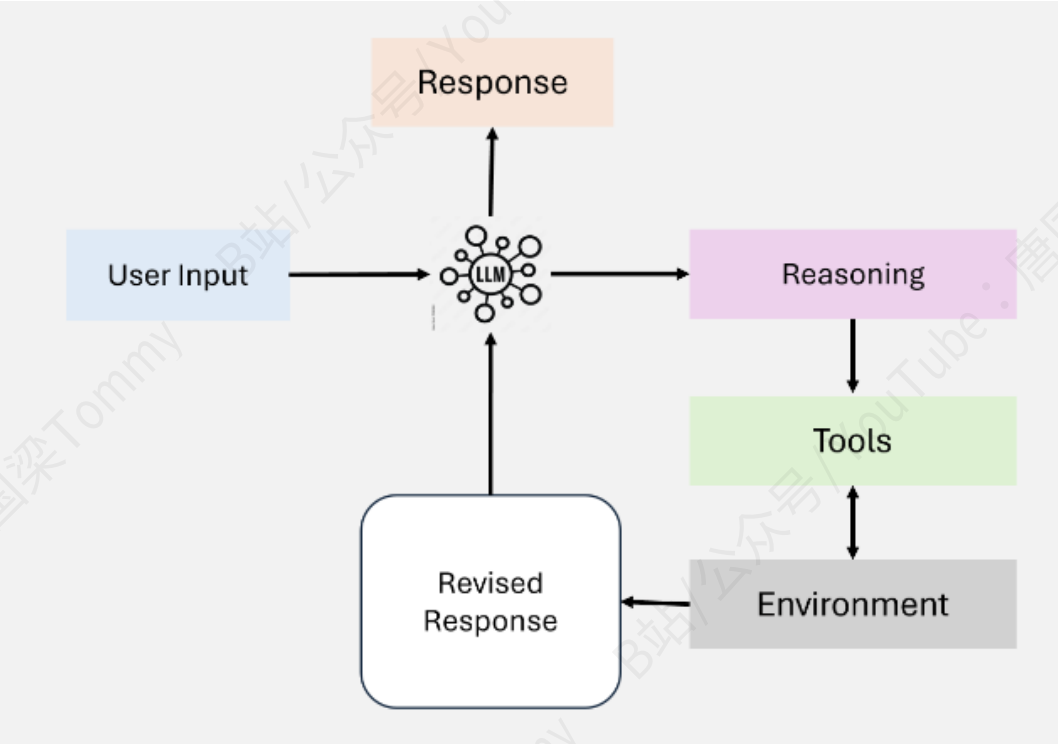
从可用的工具集中选择合适的工具来解决子问题。

(3) 工具调用:

根据工具描述，从用户查询中提取所需的工具调用参数，并向工具服务器发送请求。

(4) 响应生成:

将工具返回的结果与 LLM 自身的知识相结合，生成最终的响应。



二. Lagent 项目简介

Lagent是一个受PyTorch设计理念启发的轻量级LLM智能体开发框架。通过类似神经网络层的抽象方式，让Agent的开发流程更加清晰直观。开发者只需要专注于创建各个功能层并定义它们之间的消息传递方式，就能以Pythonic的方式快速构建复杂的Agent应用。

1. 特性

1.1 模型即智能体

- 采用AgentMessage作为统一的通信数据结构
- 支持同步和异步两种接口，方便调试和大规模推理
- 内置多种LLM模型支持(如InternLM、Qwen、Llama等)

1.2 记忆即状态

- 智能体自动维护对话历史记忆
- 支持多会话隔离，便于并发场景
- 提供灵活的状态导入导出机制

1.3 灵活的消息处理

- 可自定义消息聚合器，支持few-shot示例
- 提供多种响应格式化选项
- 支持工具调用结果的解析和处理

1.4 多智能体协作

- 支持构建复杂的多智能体工作流
- 提供异步并行处理能力
- 内置多种预设智能体模板

2. 核心模块

2.1 agents 模块

- Agents 模块是 Lagent 的核心，负责协调 LLM 与各种工具之间的交互，实现智能决策和任务执行。
- 包含基础代理类Agent和异步代理类AsyncAgent
- 支持记忆管理、输出格式化、消息聚合等功能

2.2 actions 模块

- Actions（动作）模块是 Lagent 中用于提供 LLM 与真实世界交互的工具集合。动作，也被称为工具，提供了一套LLM驱动的智能体用来与真实世界交互并执行复杂任务的函数。
- 包含简单工具（只提供一个API接口供调用）和工具包（实现多个API接口，承担不同的子任务）。
- 提供代码执行、网络搜索等具体功能

2.3 distributed 模块

- 这个模块主要用于将 Lagent 部署到生产环境，支持分布式推理和服务化调用。用户可以根据需求选择 HTTP 服务或 Ray 分布式方案。

2.4 hooks 模块

- Hooks 模块提供了一个钩子系统，用于在 Agent 和 Action 执行过程中进行**消息处理和干预**。Hooks 模块是 Lagent 的重要扩展机制，通过它可以实现：(1) 消息格式转换 (2) 交互过程记录 (3) 错误处理和恢复 (4) 自定义处理逻辑 (5) 监控和调试

2.5 llms 模块

- LLMs 模块是 Lagent 的语言模型接口层，负责统一管理和封装不同的语言模型，提供标准化的调用接口，处理模型响应的标准化。

（1）提供统一的LLM接口抽象，定义了基础的LLM接口类BaseLLM和API接口类BaseAPILLM；

（2）支持多种LLM实现，包括：OpenAI的GPT系列，Anthropic的Claude (ClaudeAPI)，HuggingFace模型，LMDeploy部署模型，vLLM优化的模型；

（3）同步/异步双接口设计，为每个LLM实现提供同步和异步两种接口；

（4）支持多种部署方式：本地Pipeline模式 (如LMDeployPipeline)，客户端/服务器模式 (如LMDeployClient/LMDeployServer)，分布式部署支持。

2.6 memory 模块

- Memory 模块负责管理 Agent 的对话历史和状态信息，提供会话隔离和状态管理功能。

（1）基础记忆管理

- 提供消息历史的存储和检索功能
- 支持按时间顺序记录代理与用户的对话内容
- 允许设置记忆容量限制

（2）多会话管理

- 支持多个会话的独立记忆管理
- 通过会话ID区分不同对话上下文
- 动态创建和管理会话实例

（3）记忆过滤和检索

- 支持获取最近N条记忆
- 提供自定义过滤函数功能
- 灵活的记忆检索机制
- 支持按不同条件筛选历史记录

（4）状态管理

- 作为Agent的状态管理系统
- 在每次对话过程中自动记录输入输出
- 支持状态的序列化和反序列化

2.7 prompts 模块

Prompts 模块负责管理和解析提示词模板，为 Agent 和 LLM 之间的交互提供标准化的消息格式。

（1）提示模板管理

- 提供统一的提示模板定义和管理机制
- 支持多种模板格式（字符串、字典、列表等）
- 实现模板的动态组装和渲染

（2）模板解析系统

- LMTemplateParser：用于语言模型的提示模板解析
- APITemplateParser：用于API调用的模板解析
- 支持元模板（meta-template）机制

- 处理角色转换和提示合并

（3）输出格式化

- 提供多种输出解析器
  - ToolParser：工具调用输出解析
  - JSONParser：JSON格式解析
  - MixedToolParser：混合工具解析
- 支持自定义输出格式定义

三. Lagent 项目实战

1. 环境配置

```
# 1. 创建虚拟环境
conda create -n lagent python=3.11
conda init bash && source /root/.bashrc
conda activate lagent
conda install ipykernel
ipython kernel install --user --name=lagent   # 设置kernel，--user表示当前用户，lagent为虚拟环境

# 2. 安装项目依赖包
cd lagent
pip install -e .

# 3. 安装其他依赖包
pip install vllm accelerate lmdeploy
```

2. 原创案例实践 + 源码解析

case-1：基础对话模型调用

主要功能：

- 实现了最基础的大语言模型调用
- 使用VllmModel加载Qwen2.5模型
- 创建简单的Agent进行单轮对话

Case 2 - Few-shot示例增强

主要功能：

- 实现了支持few-shot示例的消息聚合器FewshotAggregator
- 通过继承DefaultAggregator扩展功能
- 在对话中加入示例对来提升模型回答质量
- 支持系统提示、历史消息和few-shot示例的组合

Case 3 - Agent驱动的Python代码执行器

主要功能：

- 实现了一个智能体(Agent)控制Python代码执行的框架，通过标准化的消息格式实现智能体与执行工具的通信
- 智能体负责决策和指令生成，可以思考问题并生成相应的Python代码作为解决方案
- 执行工具(IPythonInteractive)负责安全地执行Python代码并返回执行结果
- 整个过程通过消息处理器(CodeProcessor)实现消息格式转换，确保智能体和工具之间的无缝通信

Case 4 - 高德天气API集成

主要功能：

- 实现了WeatherAPIAction自定义动作类
- 集成高德地图天气API
- 使用ActionExecutor执行API调用
- 处理API返回结果和错误状态

Case 5 - 高德位置搜索和导航

主要功能：

- 实现了LocationSearchAction和NavigationAction两个动作类
- 集成高德地图API进行位置搜索和路线规划
- 使用异步LLM模型处理导航描述
- 提供人性化的路线导航指南

Case 6 - 代码生成与执行代理

主要功能：

- 实现了专门的Coder代理类
- 集成代码生成和执行功能
- 支持多轮对话优化代码
- 提供详细的代码生成和执行过程

Case 7 - 异步博客生成系统

主要功能：

- 实现了异步的博客生成器AsyncBlogger
- 包含写作助手和评论家两个角色
- 支持并行处理多个写作任务
- 通过多轮迭代优化文章质量

3. 项目中的Demo实践

四. 参考资料


```
# 1. Building effective agents
https://www.anthropic.com/research/building-effective-agents

# 2. LLM Powered Autonomous Agents
https://lilianweng.github.io/posts/2023-06-23-agent/

# 3. Multi-Agent Collaboration Mechanisms: A Survey of LLMs
https://arxiv.org/abs/2501.06322v1

# 4. A Survey on LLM-based Multi-Agent System: Recent Advances and New Frontiers in Application
https://arxiv.org/abs/2412.17481v2

# 5. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models
https://arxiv.org/abs/2201.11903v6

# 6. Tree of Thoughts: Deliberate Problem Solving with Large Language Models
https://arxiv.org/abs/2305.10601v2
```