Agent Planning1 基础方法

Planning

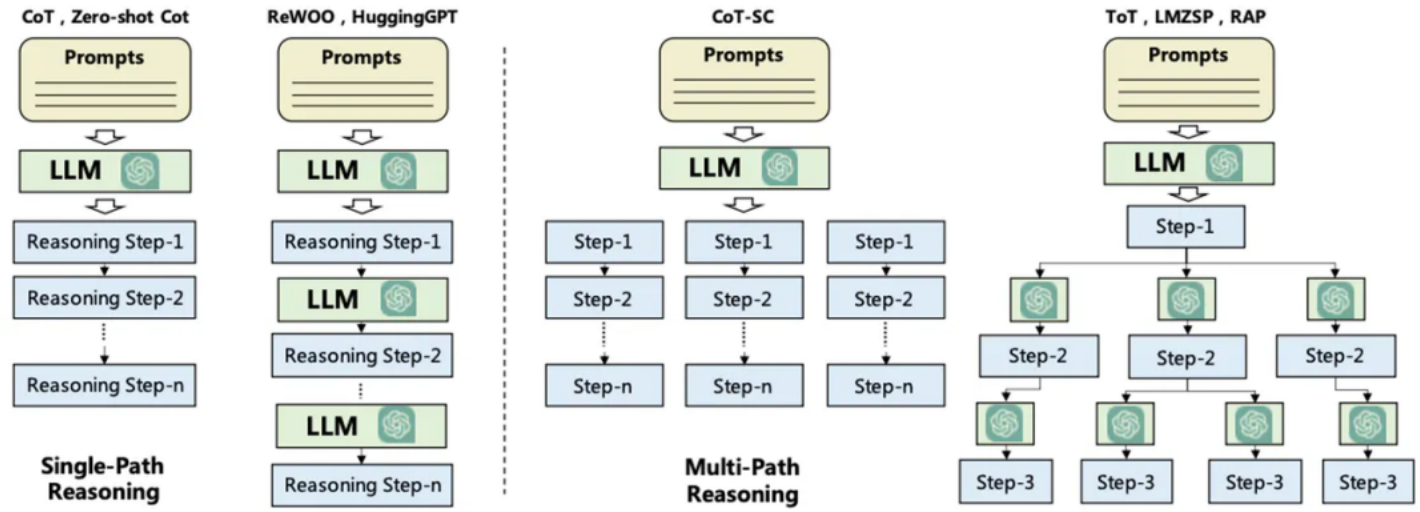
CoT

思维链将复杂的问题分解为更简单的任务，逐步解决问题，使用CoT能在算数、常识和推理任务都提高了性能。但这会增加推理的时间。CoT可以分为Few-Shot 和Zero-Shot（只需要在prompt中加入“让我们一步步的思考”）两种。使用Langchain可以轻松的实现CoT：

```
# 创建聊天模型
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI(temperature=0)
# 设定 AI 的角色和目标
role_template = "你是一个xx工作的AI助手,目标是xx"

# CoT 的关键部分，AI 解释推理过程，并加入一些先前的对话示例（Few-Shot Learning）
cot_template = """
请你按部就班的思考，先理解用户需求，再进行信息检索，再做出决策
一些示例:xx
"""
from langchain.prompts import ChatPromptTemplate, HumanMessagePromptTemplate, SystemMessagePromptTemplate
system_prompt_role = SystemMessagePromptTemplate.from_template(role_template)
system_prompt_cot = SystemMessagePromptTemplate.from_template(cot_template)

# 用户的询问
human_template = "{human_input}"
human_prompt = HumanMessagePromptTemplate.from_template(human_template)

# 将以上所有信息结合为一个聊天提示
chat_prompt = ChatPromptTemplate.from_messages([system_prompt_role, system_prompt_cot, human_prompt])
prompt = chat_prompt.format_prompt(human_input="xx").to_messages()

# 接收用户的询问，返回回答结果
response = llm(prompt)
print(response)
```

ToT

在需要多步骤推理的任务中，引导语言模型搜索一棵由连贯的语言序列（解决问题的中间步骤）组成的思维树，而不是简单地生成一个答案。ToT框架的核心思想是：让模型生成和评估其思维的能力，并将其与搜索算法（如广度优先搜索和深度优先搜索）结合起来，进行系统性地探索和验证。对于每个任务，将其分解为多个步骤，为每个步骤提出多个方案，在多条思维路径中搜寻最优的方案。



LLM+P

大型语言模型不擅长解决长期规划问题。相反，一旦以一种规范的方式给出问题，传统的规划方法就能够运用有效的搜索算法快速找到正确的，甚至是最优的解决方案。LLM+P把这两者的优势结合起来，接收一个用自然语言描述的规划问题，将语言描述转化为一个用规划领域定义语言（PDDL）编写的文件，然后利用传统规划方法快速找到解决方案，最后将找到的解决方案翻译回自然语言。

PDDL包含领域定义和问题定义两部分：

- 领域定义：描述可能的动作、动作前提条件、和导致结果。
- 问题定义：描述一个具体的规划问题，包含初始状态和目标状态

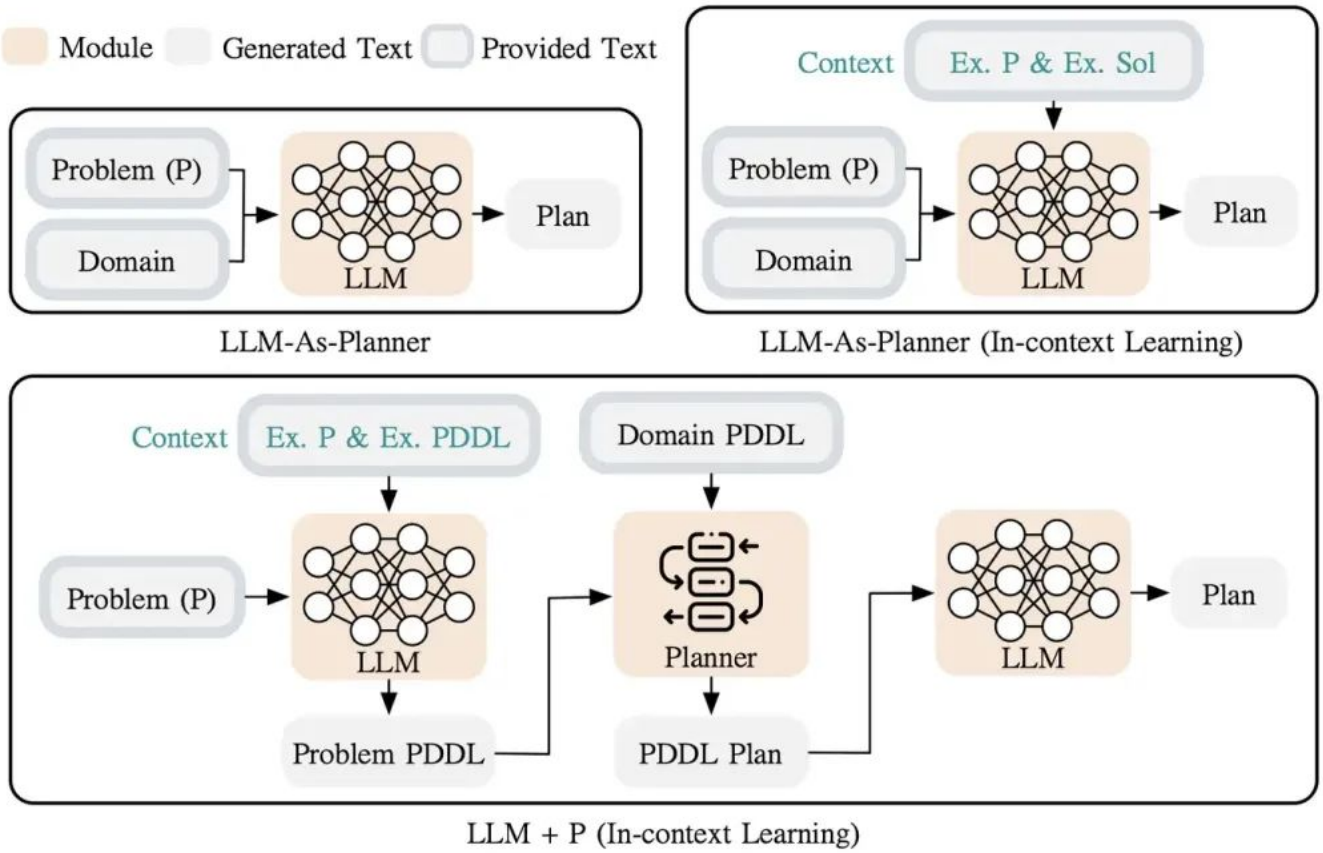


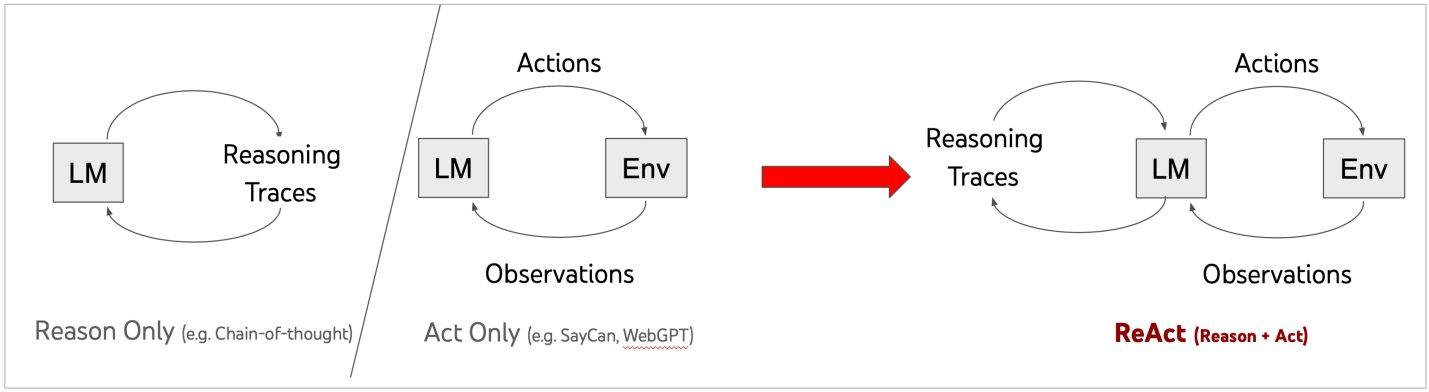
Figure 1: LLM+P makes use of a large language model (LLM) to produce the PDDL description of the given problem, then leverages a classical planner for finding an *optimal* plan, then translates the raw plan back to natural language using the LLM again.

ReAct

ReAct: Synergizing Reasoning and Acting in Language Models（不是前端哪个react框架）。其实现了“行动”和“推理”之间的协同作用，使得大模型能够作为智能代理，**生成推理痕迹和任务特定行动来实现更大的协同作用**。

ReAct的任务解决轨迹是Thought-Action-Observation，可以简化为模型按照Reasoning-Acting框架。Reasoning包括了对当前环境和状态的观察，并生成推理轨迹。这使模型能够诱导、跟踪和更新操作计划，甚至处理异常情况。ReAct的每一个推理过程都会被详细记录在案，这也改善大模型解决问题时的可解释性和可信度；Acting在于指导大模型采取下一步的行动，比如与外部源（如知识库或环境）进行交互并且收集信息，或者给出最终答案。

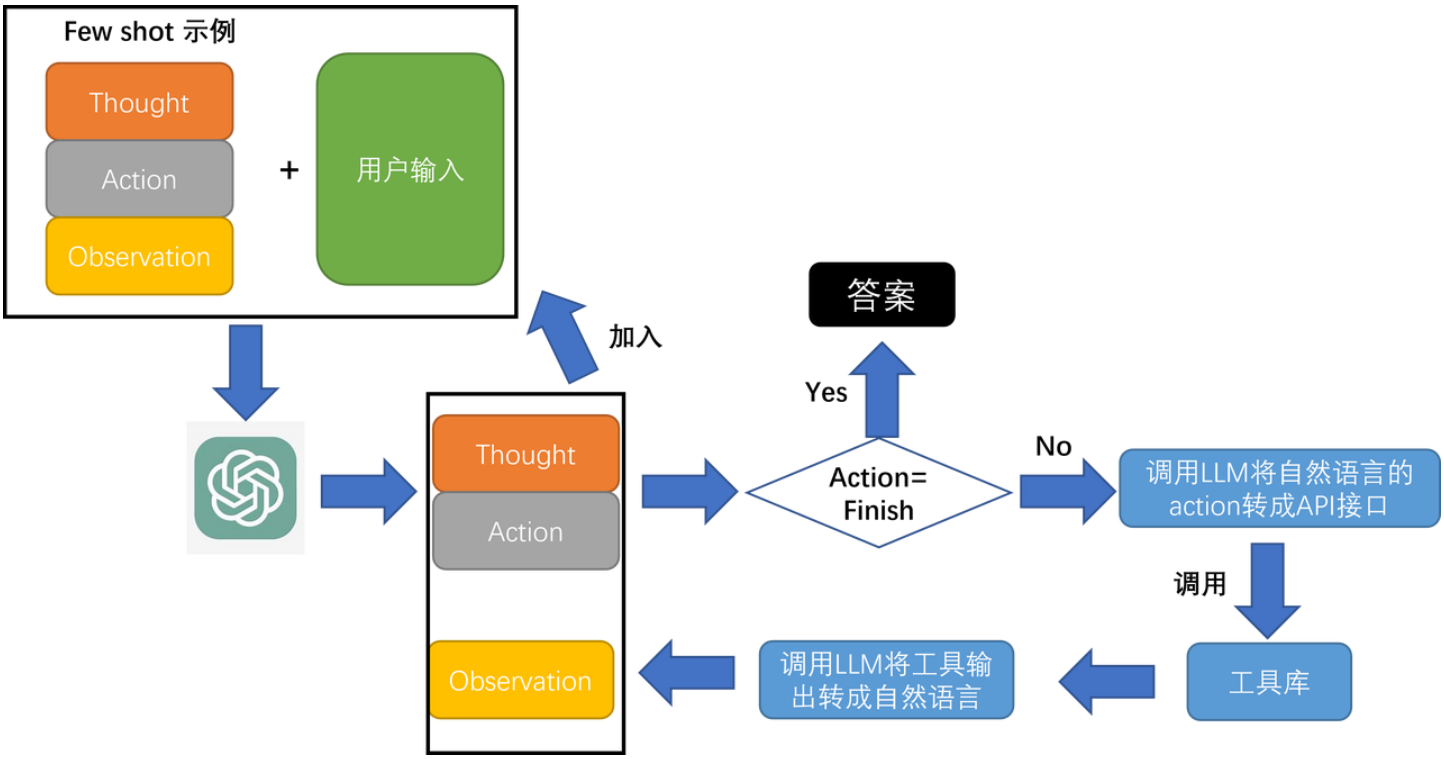
与仅仅使用CoT不同的是，这会导致模型存在幻觉，没有与外部工具交互的功能。而将ReAct框架与CoT结合，就能够让大模型在推理过程同时使用内部知识和获取到的外部信息，提升模型的可解释性和可信度。



langchain实现了ReAct框架：

```
llm = ChatOpenAI(model=os.environ["LLM_MODELEND"], temperature=0)
tools = load_tools(["serpapi", "llm-math"], llm=llm)
agent = initialize_agent(
    tools, llm, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True)
agent.run(prompt)
```

React+CoT的训练流程如下，注意之前每轮的输出会加入prompt作为后续轮次模型的输入。



引用：

1. 一文带你了解基于大模型的Agent
2. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency
3. REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS