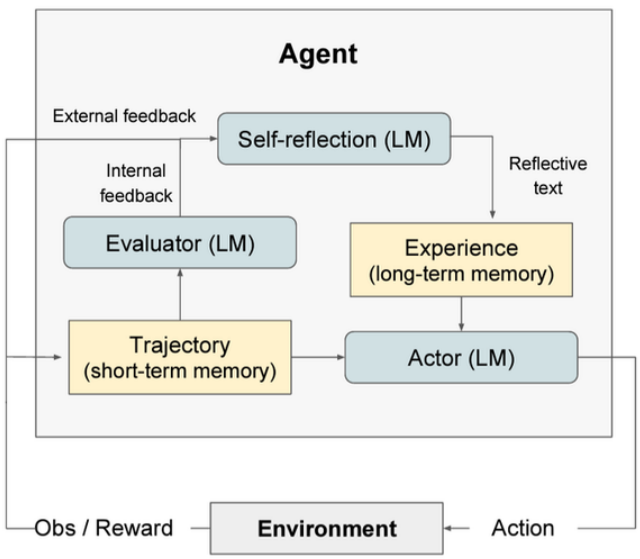
Agent Planning3 反思

Reflexion

Reflexion Agent在生成每一个trajectory后，进行启发式评估，生成反思文本并保留在记忆缓冲区中，以诱导在随后的尝试中做出更好的决策。启发式函数用于确定trajectory效率低下或包含幻觉时应当停止。效率低下的规划指的是长时间未成功完成的trajectory。幻觉定义为一系列连续相同的行动，这些行动导致在环境中观察到相同的结果。

Reflexion包含三个不同的模型：一个**执行者（Actor）**，用 M_a 表示，它生成文本和动作；一个**评估者模型（Evaluator）**，由 M_e 表示，它对 M_a 产生的输出进行打分；以及一个**自我反思模型（Self-Reflection model）**，用 M_{sr} 表示，它协助执行者自我提升。（如下图

- Actor利用llm根据状态观察生成文本和动作，采用类似强化学习的设置，从策略采样行动，并从环境接受观察，生成trajectory，可以采用React框架。
- Evaluator评估行动的价值，将trajectory作为输入，计算奖励分数。
- Self-reflection 通过生成自我反思来为未来的尝试提供有价值的反馈，存储到记忆中。
- Memory 存储短期记忆和长期记忆的概念。在推理时，Actor根据短期和长期记忆做出决策，轨迹历史作为短期记忆，而Self-Reflection模型的输出则存储在长期记忆中。



Algorithm 1 Reinforcement via self-reflection

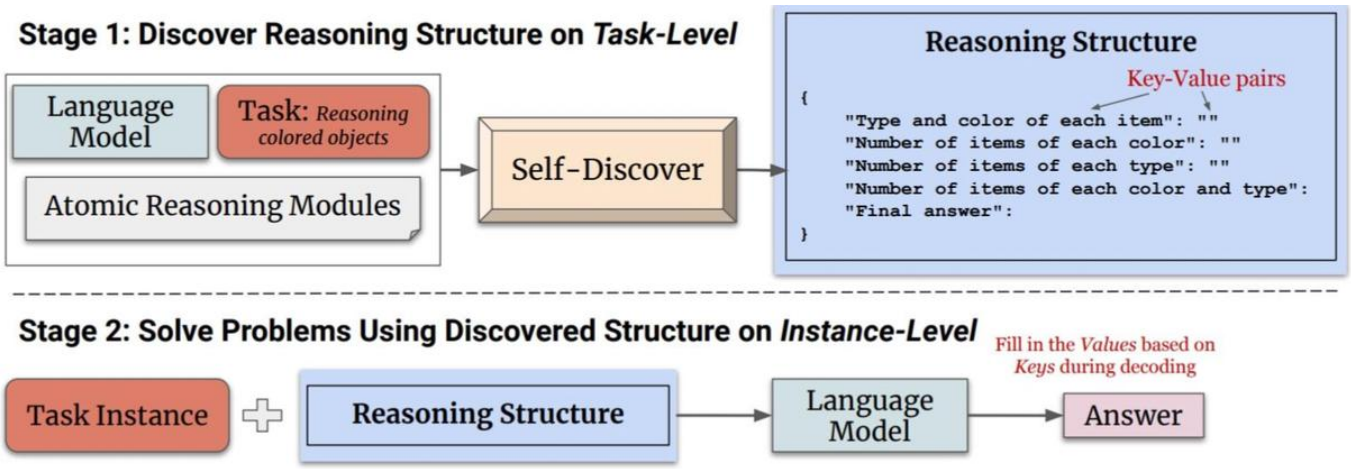
Initialize Actor, Evaluator, Self-Reflection: M_a, M_e, M_{sr}
Initialize policy $\pi_\theta(a_i|s_i), \theta = \{M_a, mem\}$
Generate initial trajectory using π_θ
Evaluate τ_0 using M_e
Generate initial self-reflection sr_0 using M_{sr}
Set $mem \leftarrow [sr_0]$
Set $t = 0$
while M_e not pass or $t <$ max trials **do**
  Generate $\tau_t = [a_0, o_0, \dots a_i, o_i]$ using π_θ
  Evaluate τ_t using M_e
  Generate self-reflection sr_t using M_{sr}
  Append sr_t to mem
  Increment t
end while
return

本质上是强化学习的思路，但传统的强化学习需要大量的训练数据和昂贵的模型微调，自我反思提供了一种轻量级替代方案，不需要微调底层语言模型，从而使其在数据和计算资源方面更加高效。和Basic reflection 相比，引入了外部数据来评估回答是否准确，并强制生成响应中多余和缺失的方面，这使得反思的内容更具建设性。

prompt方面：会让大模型针对问题在回答前进行反思和批判性思考，反思包括有没有漏掉(missing)或者重复(Superfluous)，然后回答问题，回答之后再有针对性的修改(Revise)

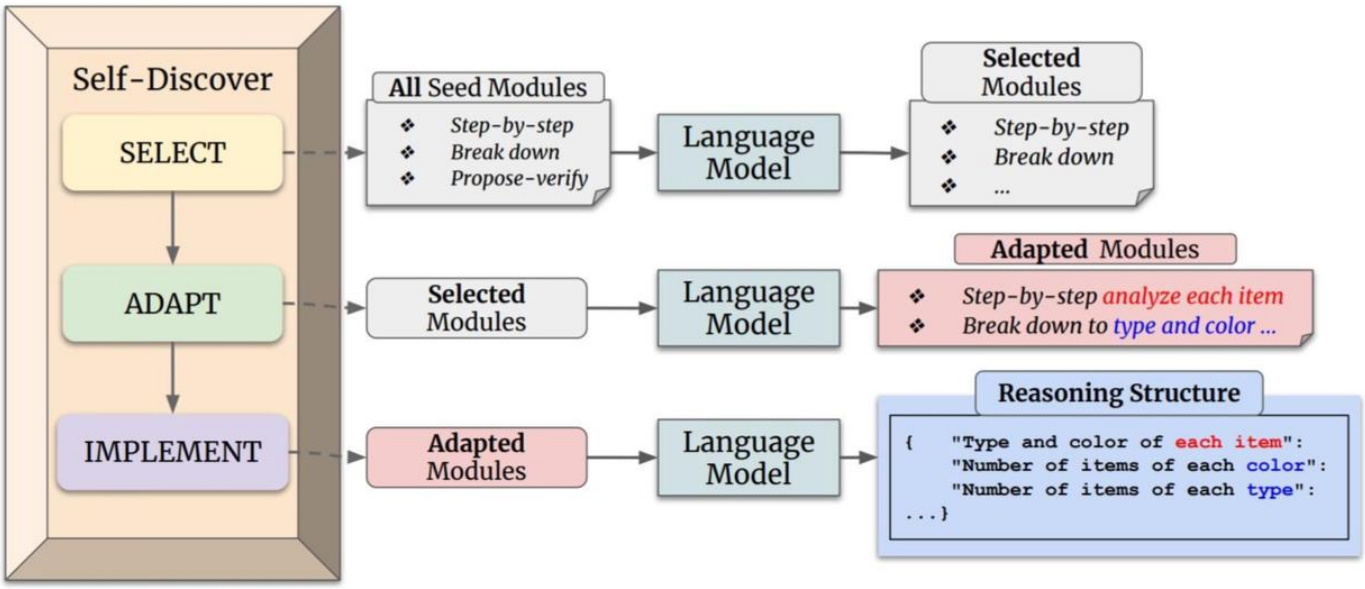
Self-DisCover

Self-discover 的核心是让大模型在更小粒度上 task 本身进行反思，比如前面Agent planning2中的提到的 Plan&Slove 是反思 task 是不是需要补充，而 Self-discover 是对 task 本身进行反思。



本方法主要分为两个阶段：利用SELF-DISCOVER 构建了任务特定的推理结构、应用推理结构解决问题。其中第一步又可以分为以下三个操作：

- **选择**：模型从一组原子推理模块（例如“批判性思维”和“逐步思考”）中选择对于解决特定任务有用的模块。模型通过一个元提示来引导选择过程，这个元提示结合了任务示例和原子模块描述。选择过程的目标是确定哪些推理模块对于解决任务是有助的。
- **适应**：一旦选定了相关的推理模块，下一步是调整这些模块的描述使其更适合当前任务。这个过程将一般性的推理模块描述，转化为更具体的任务相关描述。例如对于算术问题，“分解问题”的模块可能被调整为“按顺序计算每个算术操作”。同样，这个过程使用元提示和模型来生成适应任务的推理模块描述。
- **实施**：在适应了推理模块之后，Self-Discover框架将这些适应后的推理模块描述转化为一个结构化的可执行计划。这个计划以键值对的形式呈现，类似于JSON，以便于模型理解和执行。这个过程不仅包括元提示，还包括一个人类编写的推理结构示例，帮助模型更好地将自然语言转化为结构化的推理计划。



LATS

下面介绍一篇论文，该算法融合了ToT、React、Plan&solve、Reflection和强化学习等思想：Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models

- 预备知识：

给定自然语言x和y，模型 $p_\theta(x)$ 的任务是推理出最接近y的答案，通常prompt和x一起作为输入，生成过程可以表示为 $y\ p_\theta(prompt_{IO}(x))$ 。

React框架引入了外部环境的交互，定义行动空间 $a \in A$ 和 CoT的推理路径 $z \in Z$ ，将两者合并为最后的行动空间 $\hat{A} = A \cup Z$ ，外部环境的观察定义为o。给定观察o，下一个行动的生成表示为

$$a_i \sim p_\theta^{\text{ReAct}}(x, o_{1\cdots i-1}, a_{1\cdots i-1}),$$

CoT、ToT和React框架面临着以下问题：1）CoT和React的自回归训练会忽略特定状态的潜在连续名词 2）CoT和ToTal只依赖LLM自有的能力，可能造成幻觉。3）ToT无法利用外部环境的反馈。4）以上方法无法利用过去的经验。

蒙特卡洛树搜索是一种决策树算法，树的结点表示状态，边表示行动。从初始状态根节点出发，每轮训练包含2个步骤：1）从当前状态p中探索多个子状态s，并采样n个动作。2）采取上致信度（UCT）最高的动作，定义为：

$$UCT(s) = V(s) + w\sqrt{\frac{\ln N(p)}{N(s)}}, \qquad (1)$$

V(s)表示节点s的期望，N(s)表示访问节点s的次数，w是权重参数。当一个episode结束时，进行反向传播，用奖励r更新路径上的每个节点的value值：

$$V(s) = \frac{V_{\text{old}}(s)(N(s)-1)+r}{N(s)},$$

- LATS方法

本文提出的LATS遵循React框架的Thought-Action-Observation流程，参照蒙特卡洛树，每轮采样n个行动产生多个trajectory，以克服LLM的随机性并扩大探索域，从而找到最优trajectory。

LATS包含以下图中的6个步骤，并循环迭代，直到采样了k个trajectory后任务完成或者计算资源限制。其中 p_θ 同时作为agent，value function和反馈生成器，充分利用LLM的表征能力。

- selection：根据蒙特卡洛树选择UCT值最大的下一个节点。
- expansion：从当前状态p采样n个行动，与环境交互得到n个子节点。
- evaluation：为每个子节点计算value值，参考ToT，通过提示工程将 p_θ 作为一个评估值函数，并且这里还引入了环境反馈。还引入了基于self-consistency的启发，认为选择次数更多的action更精确：

$$V(s) = \lambda * \mathrm{LM}(s) + (1 - \lambda) * \mathrm{SC}(s), \qquad (2)$$

- simulation重复之前的过程直到到达终点状态，如果达到最优解就直接结束，反之进行Backpropagation和reflection。
- backpropagation：更新蒙特卡洛树中trajectory上的每一个节点，$N(s_i) = N(s_{i-1}) + 1, V(s_i) = \frac{V(s_{i-1}N(s_{i-1})) + r}{N(s_i)}$，其中r是奖励。
- reflection：通过prompt工程，让 p_θ 根据trajectory和奖励进行self-relection，总结推理过程中的错误，并选择更好的选项。将错误的trajectory和relection存储在记忆中，在随后的迭代中，这些被加入到agent和value函数的上下文。

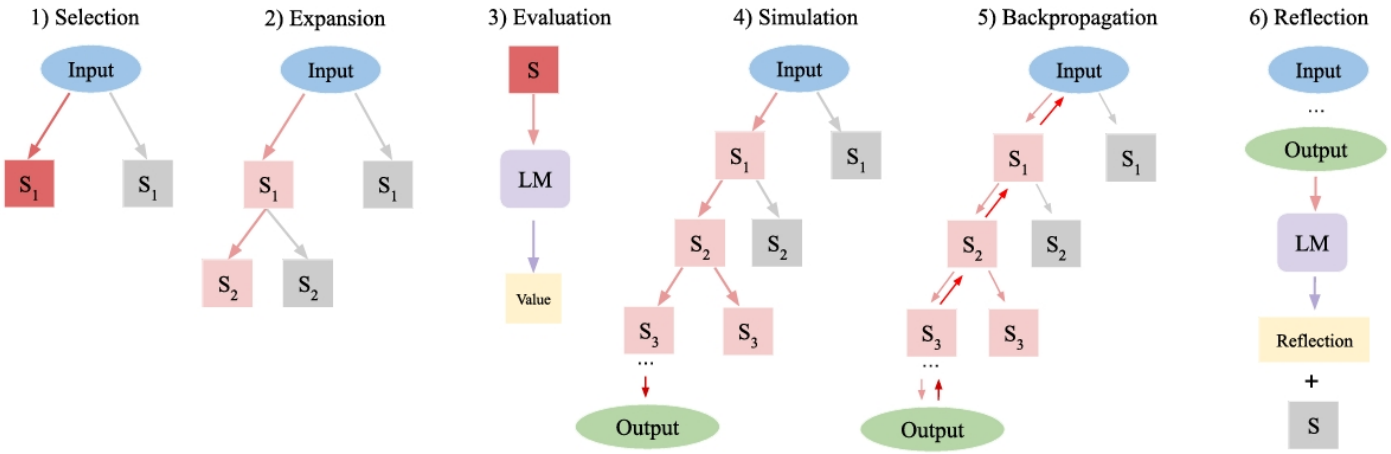


Figure 2. Overview of the six operations in LATS. A node is *selected*, *expanded*, *evaluated*, then *simulated* until a terminal node is reached, and then the resulting value is *backpropagated*. If the trajectory fails, a *reflection* is generated and used as additional context for future trials. These operations are performed in succession until the budget is reached or the task is successful.

Algorithm 1 LATS$(s, p_\theta, p_V, p_{\text{ref}}, d, k, n, w, a, b)$

Require: Initial state s, action generator p_θ, value function p_V, reflection generator p_{ref}, number of generated actions n, depth limit L, number of roll-outs K, context c, exploration weight w, and value function weight λ
Initialize action space A, observation space O
Initialize the state-action value function $p_V : S \times A \mapsto \mathbb{R}$ and visit counter $N : S \mapsto \mathbb{N}$ to one
for $k \leftarrow 0, \ldots, K-1$ **do**
  for $t \leftarrow 0, \ldots, L-1$ **do**
    if s_t not terminal **then** ▷ Expansion & Simulation
      for $i \leftarrow 1, \ldots, n$ **do**
        Sample $a_t^{(i)} \sim p_\theta(s_t)$
        Get $o_t^{(i)}$ from environment, $s_{t+1}^{(i)} \leftarrow (c_t^{(i)}, o_t^{(i)}, a_t^{(i)})$, $c_{t+1}^{(i)} \leftarrow (o_t^{(i)}, a_t^{(i)})$
        Evaluate $V_t^{(i)} \sim \lambda * p_V(s_t^{(i)}) + (1-\lambda) * \text{SC}(s_t^{(i)})$ ▷ Evaluation
        $V(s_t) \leftarrow V_t^{(i)}$
        Add $s_t^{(i)}$ to children
      end for
    end if
    if s_t is terminal **then** ▷ Reflection
      Get r from environment
      if r not success **then**
        reflection $\leftarrow p_{\text{ref}}(c_t)$
        $c \leftarrow$ reflection
      end if
    end if
    $a_t \leftarrow \arg\max_{a \in e(s_t)} \left[V(s_t) + w\sqrt{\frac{\ln N(s_t)}{N(s_{t+1})}} \right]$ ▷ Selection
    Get corresponding o_t from memory, $s_{t+1} \leftarrow (c_t, o_t, a_t)$, $c_{t+1} \leftarrow (o_t, a_t)$
    $N(s_{t+1}) \leftarrow N(s_{t+1}) + 1$
    if a_t is an output action **then break**
  end for
  $T \leftarrow$ the actual number of steps
  for $t \leftarrow T-1, \ldots, 0$ **do** ▷ Backpropagation
    $V(s_t) \leftarrow \frac{V(s_t)(N(s_t)-1)+r}{N(s_t)}$
  end for
end for

引用：

1. Reflexion: Language Agents with Verbal Reinforcement Learning
2. Self Discover框架，万万想不到AI Agent还能这样推理
3. LATS:Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models