一、ToT

人类在做决策时的两种思维方式：快速、自动、无意识的模式（称为“系统1”）和慢速、经过深思熟虑、有意识的模式（称为“系统2”）。ToT框架试图模仿人类的这种双重处理过程，通过在语言模型中引入更深层次的规划和搜索机制，以提高问题解决的能力。

论文链接：https://arxiv.org/abs/2305.10601

1. 1、思维树的工作原理

ToT框架通过将问题解决过程表示为一棵树，其中每个节点代表问题的一部分解决方案，树枝则代表到达该节点的操作。ToT会主动维护这样一棵树，并在树上搜索不同的解决方案。

这个过程涉及到以下几个关键步骤：

1. **思维分解**：将问题分解为一系列连贯的思维步骤，每个步骤都是一个语言序列，作为问题解决的中介。
2. **思维生成**：为每个树状结构的状态生成潜在的思维步骤。
3. **状态评估**：评估每个状态的潜在价值，作为搜索算法的启发式标准。
4. **搜索算法**：使用不同的搜索算法（如广度优先搜索或深度优先搜索）来探索思维树，以找到最优解决方案。

2. 2、思维树的优势

ToT框架的优势在于它的通用性和模块化，它可以适应不同的问题特性、语言模型能力和资源约束。这意味着ToT可以应用于各种问题，而不需要对语言模型进行额外的训练。

3. 3、思维树的局限性

尽管ToT框架有其优势，但它也面临一些局限性，例如它依赖于语言模型的能力来生成和评估思维步骤，这受到模型本身限制的影响。此外，ToT的效率和效果可能取决于所使用的搜索算法和启发式方法的优化程度。

4. 4、ToT的主要流程

4.1. （1）generation

根据输入和当前节点，生成一定数量的候选子节点。文中给出了两种方法：

4.1.0.1. 1）Sample 方式

利用CoT的提示词重复调用大模型，生成多次回复作为候选节点。当思维空间丰富时（例如，创意写作），这种方法更有效。由于采样是独立的，因此这有助于增加候选思维之间的多样性。

4.1.0.1. 2）Propose 方式

依据propose prompt依次生成想法。当思维空间受到更多限制时（例如，数学计算等），这种方法更有效。可以避免同一个上下文生成重复的想法。

以24点游戏为例，**propose prompt**如下：prompt给定1-shot样例，期望大模型能按照样例生成多个下一阶段的想法。

```
propose_prompt = '''Input: 2 8 8 14
Possible next steps:
2 + 8 = 10 (left: 8 10 14)
8 / 2 = 4 (left: 4 8 14)
14 + 2 = 16 (left: 8 8 16)
2 * 8 = 16 (left: 8 14 16)
8 - 2 = 6 (left: 6 8 14)
14 - 8 = 6 (left: 2 6 8)
14 /  2 = 7 (left: 7 8 8)
14 - 2 = 12 (left: 8 8 12)
Input: {input}
Possible next steps:
'''
```

4.2. （2）evaluation

这一步是对所有候选子节点进行状态评估，评估出是否需要进一步在此基础上探索最终答案。作者提出两种评估方式：

4.2.0.1. 1）数值或分级评价

对每个状态，让大模型给出数值或分级的评价（例如：sure/likely/impossible），这样的评估方式不需要非常准确，只要能近似预估即可。让大模型给出分级评价的Prompt示例：

```
value_prompt = '''Evaluate if given numbers can reach 24
(sure/likely/impossible)
10 14
10 + 14 = 24
sure
11 12
11 + 12 = 23
12 - 11 = 1
11 * 12 = 132
11 / 12 = 0.91
impossible
4 4 10
4 + 4 + 10 = 8 + 10 = 18
4 * 10 - 4 = 40 - 4 = 36
(10 - 4) * 4 = 6 * 4 = 24
sure
4 9 11
9 + 11 + 4 = 20 + 4 = 24
sure
5 7 8
5 + 7 + 8 = 12 + 8 = 20
(8 - 5) * 7 = 3 * 7 = 21
I cannot obtain 24 now, but numbers are within a reasonable range
likely
5 6 6
5 + 6 + 6 = 17
(6 - 5) * 6 = 1 * 6 = 6
I cannot obtain 24 now, but numbers are within a reasonable range
likely
10 10 11
10 + 10 + 11 = 31
(11 - 10) * 10 = 10
```

```
10 10 10 are all too big
impossible
1 3 3
1 * 3 * 3 = 9
(1 + 3) * 3 = 12
1 3 3 are all too small
impossible
{input}
'''
```

4.2.0.1. 2）跨状态投票评价

让大模型在不同的状态间进行投票式选择，投票多次后选择得票率最高的选项。

```
vote_prompt = '''Given an instruction and several choices, decide which choice is
most promising. Analyze each choice in detail, then conclude in the last line
"The best choice is {s}", where s the integer id of the choice.
'''
```

4.3. （3）selection

根据打分结果，选择一定数量的候选子节点作为下一轮的父节点，然后再次循环。

论文中采用了两种搜索算法：

（1）BFS，广度优先算法：每一步中保留最有潜力的K个状态。

（2）DFS，深度优先算法：优先探索最有潜力的状态，直到得到最终结果，如果最终结果不可行，则回溯到父节点，继续探索。

二、Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models

论文链接：https://arxiv.org/abs/2310.04406

1. 1、摘要

本文提出了一种名为**LATS**（Language Agent Tree Search）的框架，该框架将语言模型在规划、行动和推理方面的优势结合起来，以增强决策能力。LATS借鉴了基于模型**强化学习**中常用的**蒙特卡罗树搜索**方法，并利用环境提供**外部反馈**，从而实现更明智和适应性更强的问题解决机制。

2. 2、主要内容

本文提出的LATS是一种基于蒙特卡罗树搜索（MCTS）的**推理决策框架**，旨在支持自然语言任务中的推理和决策。该框架通过将一个思考序列作为节点，使用**预训练的语言模型**来评估每个节点的价值，并根据环境反馈更新价值函数。同时，它还具有**自我反思**功能，可以从失败的轨迹中学习并提高其决策能力。

LATS是第一个结合了推理、决策和规划的框架。

Approach	Reasoning	Acting	Planning	Self Reflection	External Memory
CoT (Wei et al., 2022)	✓	×	×	×	×
ReAct (Yao et al., 2023b)	✓	✓	×	×	×
ToT (Yao et al., 2023a)	✓	×	✓	✓	✓
RAP (Hao et al., 2023)	✓	×	✓	×	✓
Self-Refine (Madaan et al., 2023)	✓	×	×	✓	×
Beam Search (Xie et al., 2023)	✓	×	×	✓	×
Reflexion (Shinn et al., 2023)	✓	✓	×	✓	✓
LATS (Ours)	✓	✓	✓	✓	✓

注：论文中把搜索算法的使用称为**规划**，将语言模型生成的反馈用于**自我反思**，将过去文本语境的存储视为**外部记忆**，以供将来对解决方案进行更新。

2.0.1. （1）方法改进

与传统的推理决策框架相比，LATS的主要改进在于：

1. 使用了**蒙特卡罗树**搜索算法，可以有效地探索可能的解决方案。
2. 利用了预训练的语言模型来评估节点的价值，从而更好地指导搜索过程。
3. 引入了自我反思机制，可以从失败的轨迹中学习并提高决策能力。
4. 引入了内部反思与外部条件反馈，将内外反馈条件作为记忆存储与利用

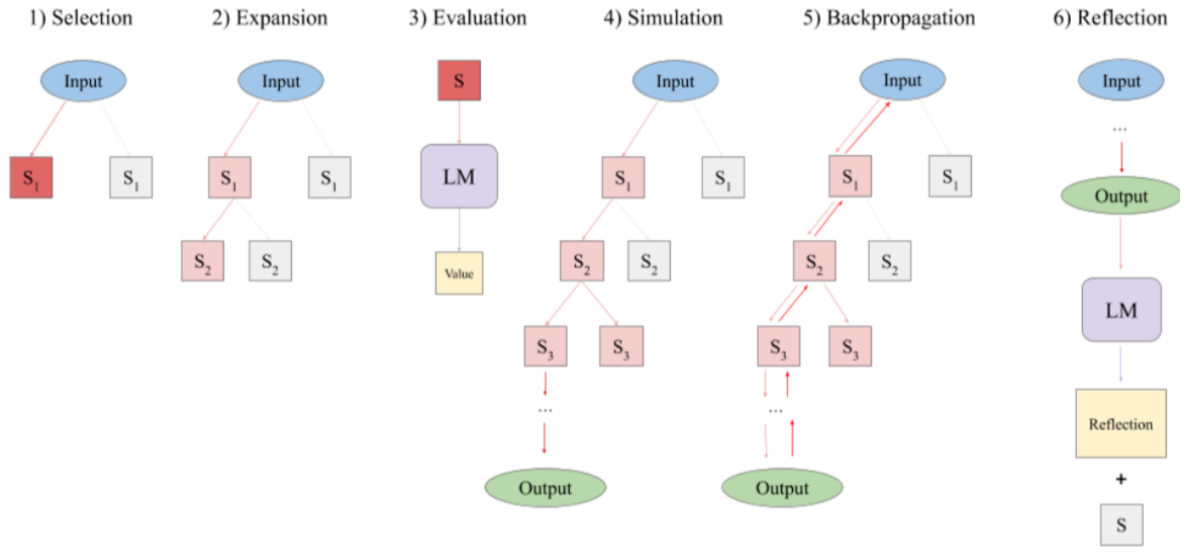
2.0.2. （2）解决的问题

本文主要解决了自然语言任务中的推理和决策问题。具体来说，它可以用于以下场景：

推理问题：当输入一个问题时，可以通过LATS生成一系列中间想法（思考序列），最终得到答案。

决策问题：当需要在多个选项之间做出选择时，LATS可以根据不同的情况生成不同的决策路径，并从中选择最优解。

2.0.3. （3）架构组成



1）选择

在树中，从根部（初始状态）开始，根据一定的规则（比如哪个分支最有可能通向正确答案），选择一个最有前途的分支继续前进。

2）扩展

探索这个分支，尝试从当前状态出发做出不同的行动，就像是在尝试不同的解题方法。
3）评估

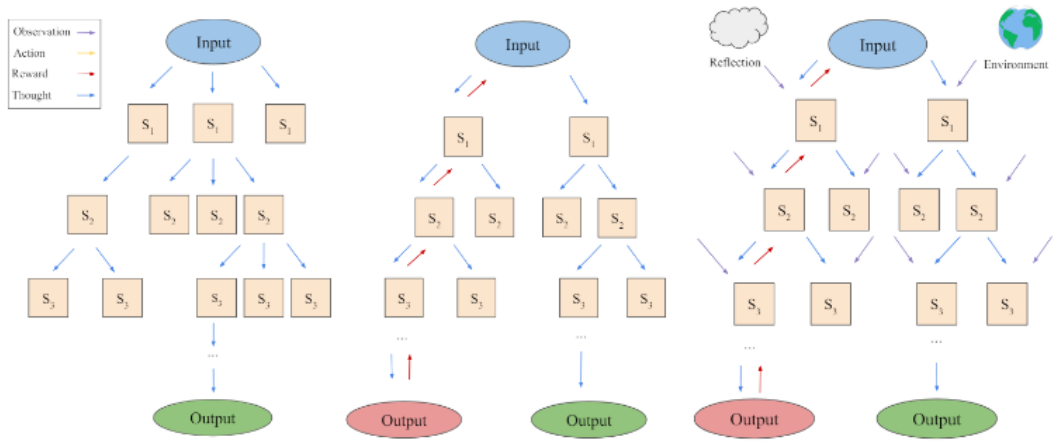
对每个可能的行动结果进行评估，就像是给每种解题方法打分，看看哪个更靠谱。
4）模拟

继续沿着选中的分支走，直到走到尽头，看看这个路径是否真的能解决问题。
5）反向传播

如果你找到了正确的解决方案，就将这个成功的信号传回到你之前经过的每个节点上，让它们也知道“这条路走得对”。
6）反思

如果你走错了路，没能解决问题，就停下来想一想错在哪里，怎样可以避免同样的错误。
重复这个过程多次，每次都尝试不同的路径，直到找到最好的解决方案。当你找到了一个满意的解决方案，或者尝试了很多次之后，就停止搜索。

2.0.3.1. LATS和ToT的区别：



a) Tree-of-Thoughts　b) Reasoning via Planning　c) Language Agent Tree Search

（1）搜索机制

LATS采用蒙特卡洛树搜索（MCTS）的变种，这是一种启发式搜索算法，通过构建决策树来平衡探索和利用。LATS在树搜索中使用每个节点代表一个状态，并通过选择、扩展、评估、模拟、反向传播和反思等操作来寻找最优解。

ToT使用深度优先搜索（DFS）或广度优先搜索（BFS）来探索多个推理路径。基于大模型评估的启发式来引导搜索，但通常不涉及树搜索中的动态规划或反向传播。

（2）外部反馈的使用

LATS特别强调使用外部环境的反馈来改进推理和决策。LATS利用环境交互和自我反思来增强模型的合理性，并使Agent能够从经验中学习。

ToT通过探索多个推理路径来增强语言模型的决策能力，但它主要依赖于语言模型内部的知识。

（3）自我反思

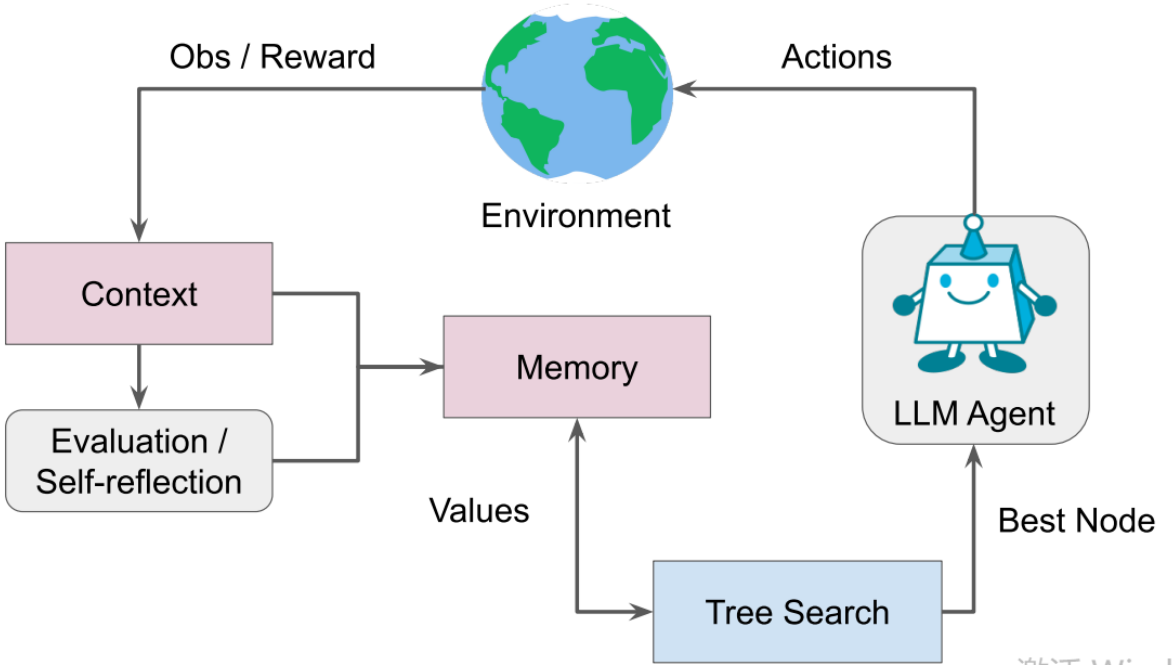
LATS包括一个反思操作，当遇到不成功的终端节点时，会生成一个反思，总结推理或行动过程中的错误，并提出更好的替代方案。这些反思被存储并作为额外的上下文信息，以提高未来尝试的性能。

ToT方法本身不包括自我反思的组件。它更多地关注于通过搜索算法探索不同的推理路径。

（4）灵活性和适应性

LATS设计为灵活和适应性强，可以根据不同环境和任务调整状态设计和树的维度。

ToT虽然能够探索多个推理路径，但它的适应性可能不如LATS，因为它不包括对外部反馈的整合。

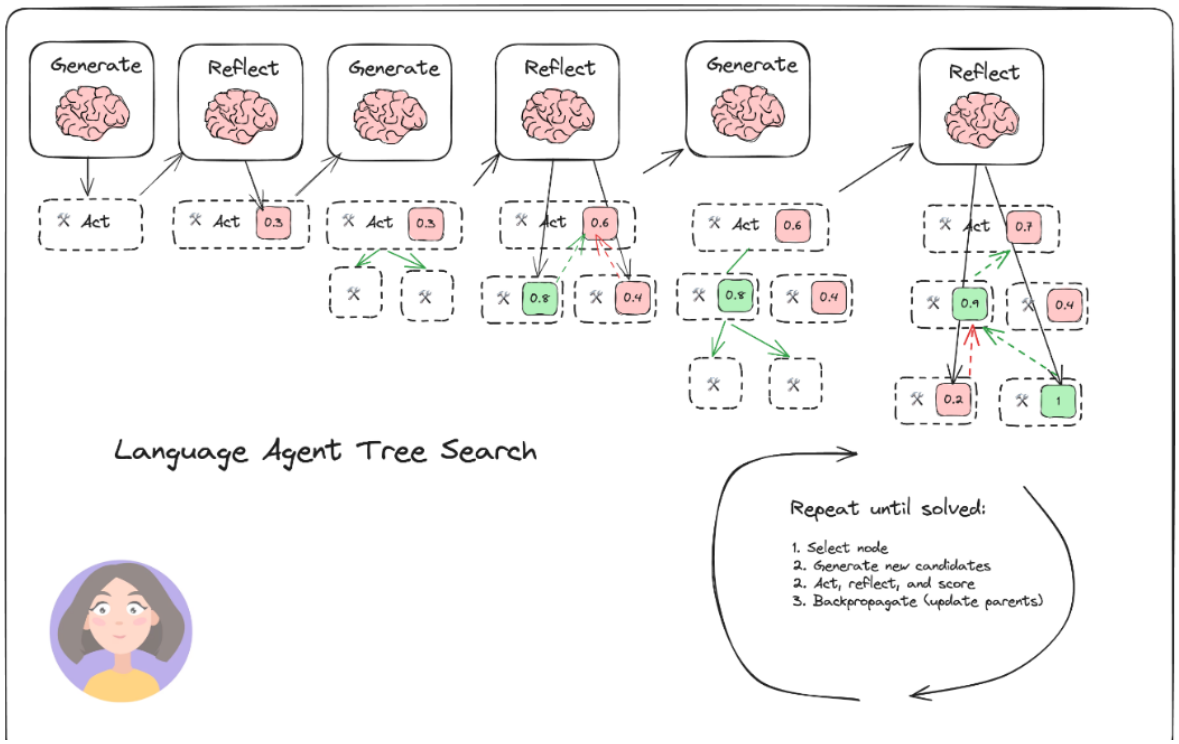


观察环境，收集相关信息作为决策的基础，利用收集到的观察结果，并结合上下文信息来更好地理解当前状态，访问其记忆，这可能包含先前的经验或状态信息，以辅助当前的决策，llm agent使用其内部的语言模型来处理信息，并生成可能的行动方案。

流程开始于代理从环境中获得观察（Obs），然后结合上下文（Context）和记忆（Memory）进行自我反思（Self-reflection）和评估（Evaluation），以确定行动（Actions）。代理根据其价值观（Values）选择最佳节点（Best Node）执行动作，并从环境中获得奖励（Reward）。这个过程是迭代的，代理不断通过树搜索（Tree Search）来优化其决策路径。

3. 3、实现

LATS的实现需要：选择、扩展、评估、模拟、反向传播和反思的过程。LangChain中的代码实现步骤如下：



完整代码参考：https://github.com/langchain-ai/langgraph/blob/main/examples/lats/lats.ipynb

https://blog.csdn.net/Attitude93/article/details/138491682

三、Self-discover

1. 1、摘要

SELF-DISCOVER框架的核心部分是自发现过程，它允许大型语言模型（LLMs）在没有明确标签的情况下，自主地为**特定任务生成推理结构**。核心的反思过程是对任务本身在更细粒度上进行思考和分析，而不是仅仅在高层次上考虑问题解决策略。SELF-DISCOVER框架包含两个主要阶段：自发现特定任务的推理结构、应用推理结构解决问题。

2. 2、主要内容

Self-discover分为两个基本阶段：

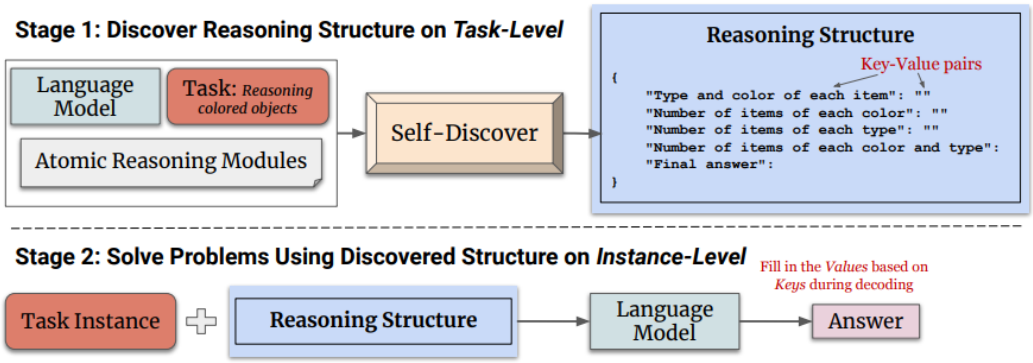


Figure 2. **Illustration of using SELF-DISCOVER for problem-solving**. Given a generative LM, task, and seed reasoning module descriptions, we guide LMs to generate a reasoning structure in *key-value* format to solve the task. Finally, models can follow the self-discovered structures to solve the every instance from the task by filling in the values in JSON step-by-step.

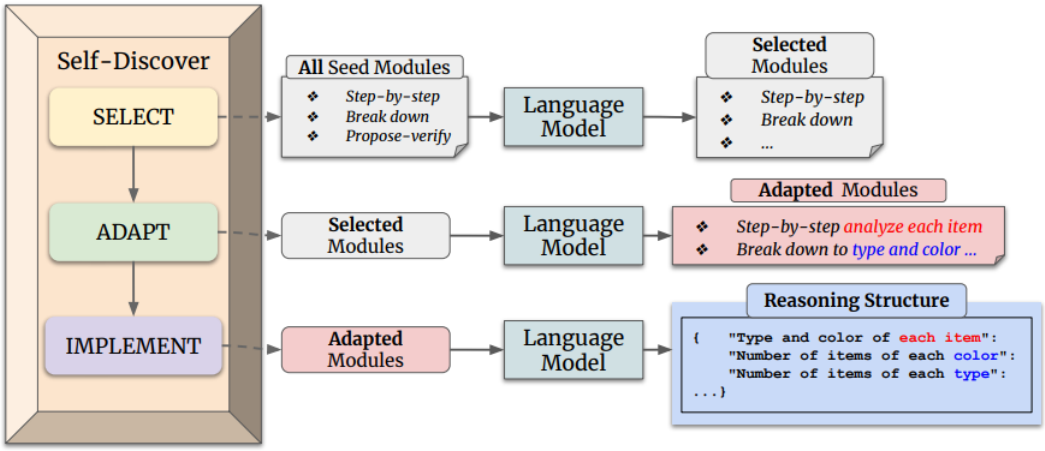


Figure 3. **Illustration of three actions of SELF-DISCOVER**. We use LMs to compose a coherent reasoning structure by selecting relevant modules, adapting to task-specific descriptions, and implement a reasoning structure in JSON.

2.0.4. 第一阶段：

1. **任务和原子推理模块（Task and Atomic Reasoning Modules）**：
    - 流程从左上角开始，这里有一个任务描述和一个原子推理模块的集合。原子推理模块是一些预定义的、描述性的问题解决启发式方法，例如“逐步思考”（step-by-step thinking）和“关键点思考”（critical thinking）。
2. **自我发现任务特定结构（Self-Discover Task-Specific Structures）**：

- 第一阶段（Stage 1）涉及三个动作：选择（SELECT）、适应（ADAPT）和实现（IMPLEMENT）。
  - **选择（SELECT）**：模型从原子推理模块集中选择与任务解决相关的模块。模型通过一个元提示（meta-prompt）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（告诉模型怎么去思考，激发内部知识）。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（ADAPT）**：结合上下文等信息，将选定的推理模块描述调整为更具体地适应手头的任务。这个过程是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」。
  - **实现（IMPLEMENT）**：将适应的推理模块描述实施为一个结构化的可操作计划（json键值对），以便按照结构解决任务。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，以帮助模型更好地将自然语言描述转化为结构化的推理计划。
3. **推理结构（Reasoning Structure）**：
   - 通过上述三个动作，生成一个推理结构，该结构以键值对的形式表示，类似于 JSON 格式，以提高可解释性和推理质量。

2.0.5. 第二阶段：

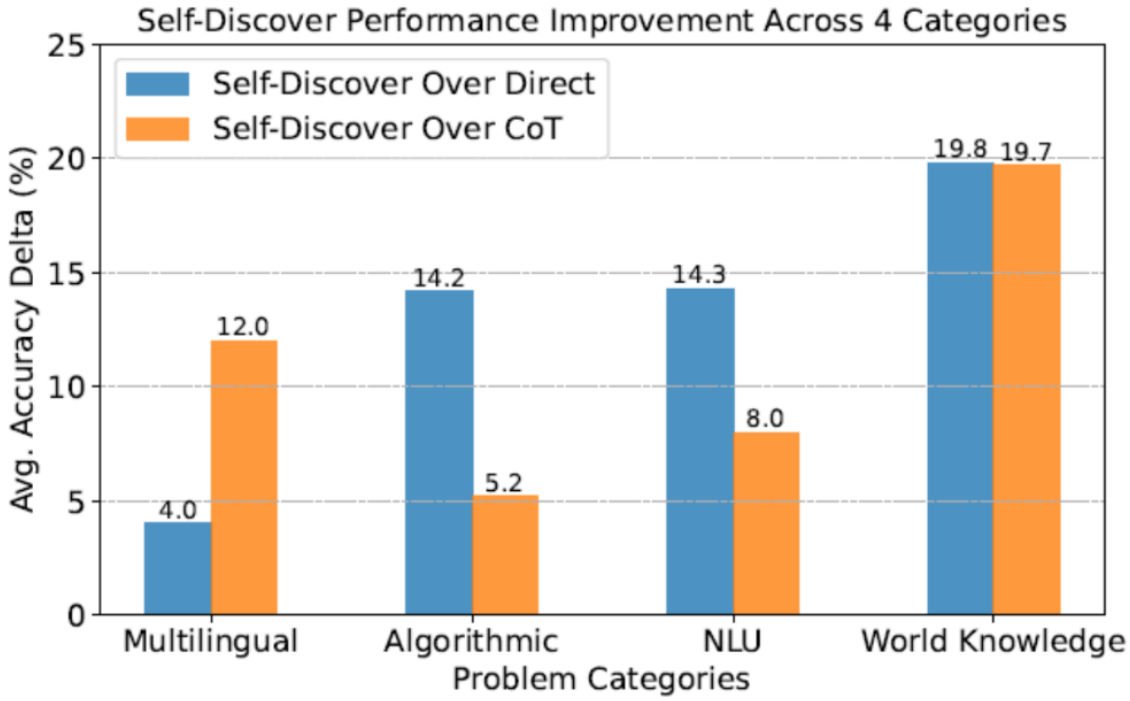
完成阶段一之后，模型将拥有一个专门为当前任务定制的推理结构。在解决问题的实例时，模型只需遵循这个结构，逐步填充JSON中的值，直到得出最终答案。

这个过程的关键在于，它允许模型在没有人类干预的情况下，自主地生成适合特定任务的推理结构，这不仅提高了模型的推理能力，而且提高了推理过程的可解释性。

1. **解决任务实例（Solve Problems Using Discovered Structure）**：
   - 第二阶段（Stage 2）使用在第一阶段发现的推理结构来解决任务的每个实例。模型被指示遵循推理结构，通过填充每个键的值来逐步得出最终答案。
2. **实例填充（Instance Filling）**：
   - 模型将推理结构应用于任务实例，根据结构中的键来填充值，完成对问题的推理和解答。
3. **输出答案（Answer）**：
   - 模型最终生成问题的答案。这个答案是基于自我发现的推理结构和对任务实例的分析得出的。

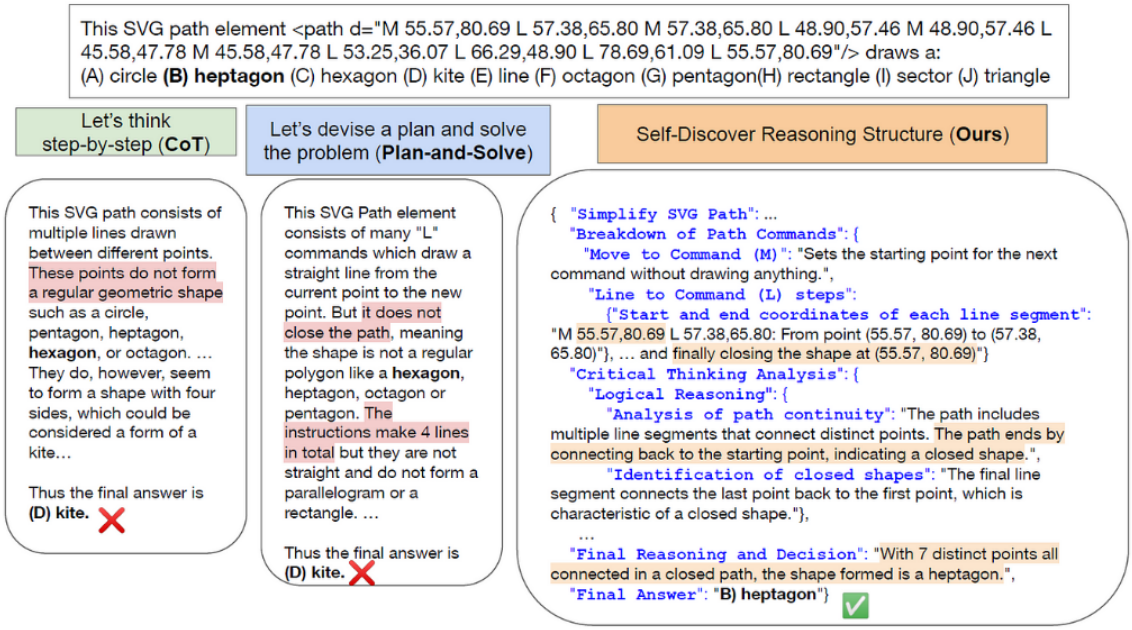
2.0.6. 结果

Google DeepMind的SELF-DISCOVER在提升PaLM 2-L和GPT-4等先进语言模型的推理能力方面取得了令人期待的结果，涵盖了各种领域的复杂推理任务的评估，包括BBH、T4D和MATH，与直接提示、思维链（CoT）和计划与解决（PS）等传统方法相比，展示了SELF-DISCOVER的有效性。



推理过程分析

在对BBH的几何形状任务的推理过程进行详细分析时，思维链和计划与解决的局限性变得明显。这两种方法错误地得出结论，认为路径不形成规则形状，错误地将其识别为非闭合形状。相比之下，SELF-DISCOVER的方法截然不同。它详细分析了每个线段及其坐标，运用逻辑推理推断出路径确实形成了闭合形状，因为它返回到起始坐标。这种方法ical的分解和分析使SELF-DISCOVER能够通过逻辑推理过程得出正确的结论。



四、 STORM

1. 1、摘要

本文研究如何应用大型语言模型从头开始编写扎根和有组织的长篇文章，其广度和深度与维基百科页面相当。这个未被充分探索的问题在写作前阶段提出了新的挑战，包括如何在写作前研究主题和准备大纲。我们提出了 STORM，这是一种通过检索和多视角提问来综合主题大纲的写作系统。STORM 通过以下方式对写作前阶段进行建模：（1） 在研究给定主题时发现不同的观点，（2） 模拟对话，其中携带不同观点的作者基于可信赖的互联网资源向主题专家提出问题，（3） 策划收集的信息以创建大纲。

网址：https://storm.genie.stanford.edu/

代码网址：https://github.com/stanford-oval/storm

演示视频：https://www.ixigua.com/7356996615321682467?utm_source=xiguastudio

STORM 是用于自动化知识管理过程的研究原型。

技术栈：

- dspy 一个用于算法优化LM提示和权重的框架
- You.com搜索API YOU APIs利用实时网络数据使LLMs和搜索体验更加真实和及时。

2. 2、维基百科式文章的挑战与解决方案

维基百科以其详尽的内容、丰富的视角和严谨的结构，成为了互联网上的知识宝库。然而，编写一篇高质量的维基百科文章需要深入研究和精心策划，包括广泛收集参考资料和精心制作大纲。这一繁琐的过程往往让许多作者望而却步。而STORM系统的出现，正是为了打破这一僵局。

2.1. （1）主要挑战

- 深入研究和计划：维基百科式的文章需要深入研究和计划，包括广泛收集参考资料和精心制作大纲。
- 现有工作的不足：现有的生成维基百科文章的工作往往绕过了写作前的研究和计划阶段。

2.2. （2）解决方案

- 模拟人类写作过程：STORM通过模拟人类写作过程中的**预写、起草和修订**阶段，自动化这一过程。
- 有效问题提问：特别是在预写阶段，通过有效的问题提问来自动化这一过程。

3. 3、STORM系统的工作流程

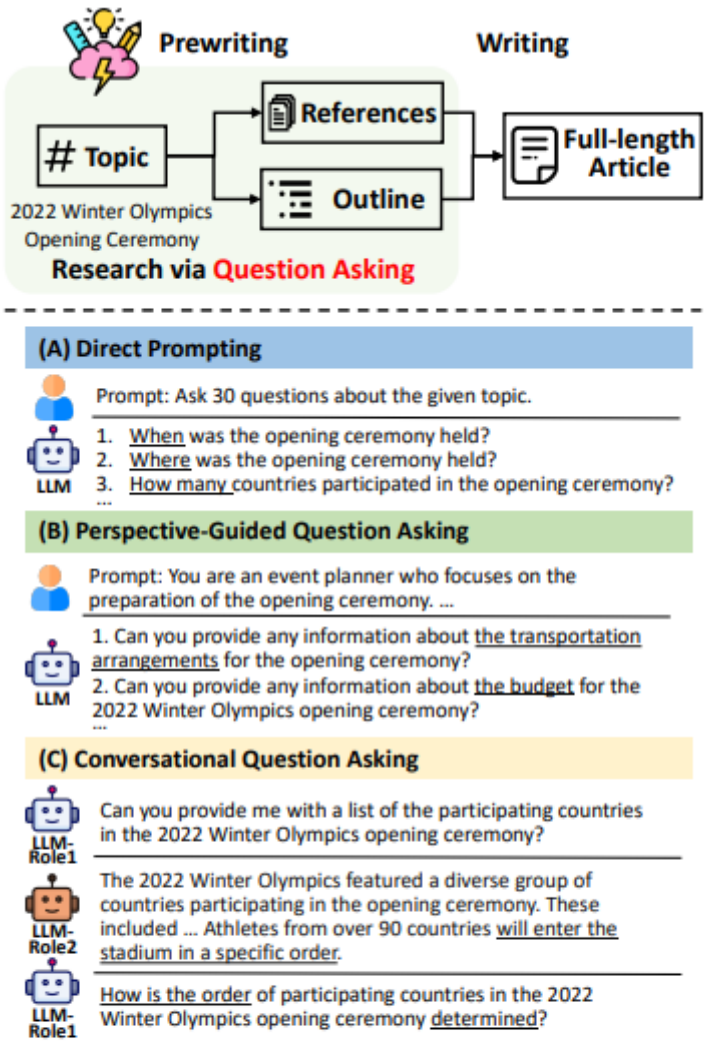


Figure 1: We explore writing Wikipedia-like articles from scratch, which demands a pre-writing stage before producing the article. In this stage, simpler approaches like Direct Prompting have limited planning capacity. In contrast, STORM researches the topic via perspective-guided question asking in simulated conversations.

1. **预写阶段（Prewriting）**：
    - 在实际写作开始之前，需要进行主题研究和大纲准备。
2. **参考资料（References）**：
    - 在撰写文章之前，需要搜集和整理相关的参考资料。
3. **主题（Topic）**：
    - 确定文章的主题，例如 "2022 Winter Olympics"。
4. **大纲（Outline）**：
    - 创建文章的大纲，包括各级标题和子标题。
5. **全文（Full-length Article）**：
    - 根据大纲和参考资料撰写完整的文章。
6. **研究通过提问（Research via Question Asking）**：
    - 使用提问的方式来进行主题研究，包括以下几种提问方法：

    a. **直接提示（Direct Prompting）**：
    - 直接向语言模型（LLM）提出关于给定主题的问题，例如询问开幕式的举办时间和地点。

b. **视角引导的提问（Perspective-Guided Question Asking）**：
   - 从特定视角提出问题，例如作为一个活动策划者关注开幕式的交通安排和预算。
c. **会话式提问（Conversational Question Asking）**：
   - 模拟对话形式提问，例如询问参与国家的列表和他们进入体育场的顺序。

7. **STORM 方法**:
   - STORM 通过模拟多角度提问的对话来研究主题，与简单的直接提示相比，STORM 的方法具有更强的规划能力。

自动化预写阶段：

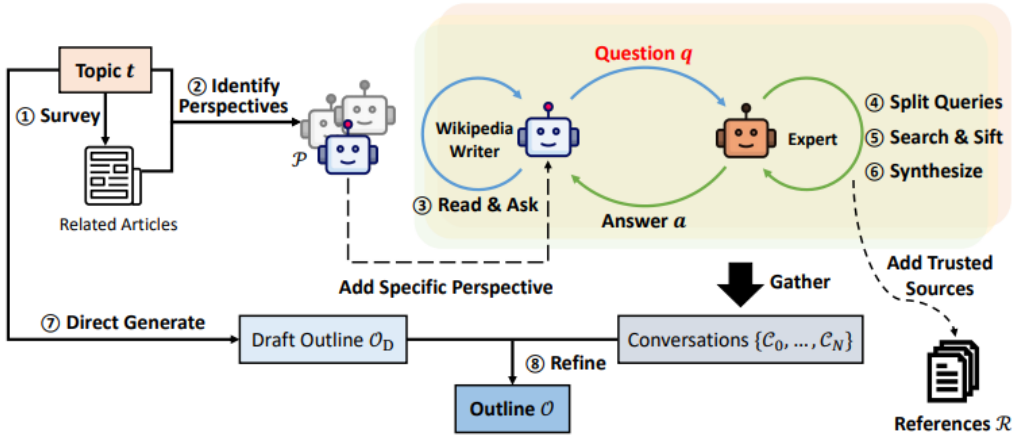


Figure 2: The overview of STORM that automates the pre-writing stage. Starting with a given topic, STORM identifies various perspectives on covering the topic by surveying related Wikipedia articles (①-②). It then simulates conversations between a Wikipedia writer who asks questions guided by the given perspective and an expert grounded on trustworthy online sources (③-⑥). The final outline is curated based on the LLM's intrinsic knowledge and the gathered conversations from different perspectives (⑦-⑧).

1. **起点（Starting Point）**：
   - 给定一个主题（Topic t），STORM 流程开始。
2. **识别视角（Identify Perspectives）**：
   - 通过调查相关维基百科文章，STORM 识别出涵盖该主题的不同视角。
3. **模拟对话（Simulate Conversations）**：
   - STORM 模拟一个维基百科作者（Writer）和基于可信在线资源的专家（Expert）之间的对话。作者根据给定的视角提出问题。
4. **提问与回答（Read & Ask）**：
   - 作者提出问题（Question q），专家根据可信赖的来源提供答案（Answer a）。
5. **搜索与筛选（Search & Sift）**：
   - 利用搜索结果来筛选和收集相关信息，以回答提出的问题。
6. **合成信息（Synthesize）**：
   - 将搜集到的信息综合起来，形成对话内容。
7. **添加特定视角（Add Specific Perspective）**：
   - 在提问过程中添加特定的视角，以丰富问题的深度和广度。
8. **收集对话（Gather Conversations）**：
   - 收集来自不同视角的模拟对话。
9. **直接生成草稿大纲（Direct Generate Draft Outline）**：
   - 利用 LLM（Large Language Model）直接生成基于主题的草稿大纲。
10. **完善大纲（Refine Outline）**：
    - 根据 LLM 的内在知识和从不同对话中收集的信息来完善大纲。

11. **收集参考资料（Gather References）**：
    - 将对话中提到的可信赖来源作为参考资料。
12. **最终大纲（Final Outline O）**：
    - 基于上述步骤，形成最终的详细大纲。

STORM系统的工作流程包括以下几个关键步骤：

1. 发现**不同视角**
    - 通过检索和分析与给定话题相似的维基百科文章，从多个来源和角度探索话题。
    - 确保内容的**全面性和深度**，发现研究话题时的多样视角。
2. **模拟对话**
    - 模拟作家向话题专家提出问题的对话过程。
    - 使用LLMs生成深入的问题，深化对话题的理解。
    - 这些对话基于互联网上的可信资源。
3. **创建大纲**
    - 基于收集到的信息和提出的问题，自动创建文章的大纲。
    - 组织文章结构，确保内容覆盖广度和深度。
4. 写作阶段
    - 生成带有引文的文本。
    - 逐节撰写完整的文章。

4. 4、STORM系统解决的主要问题

STORM系统旨在解决以下主要问题：

1. 写作前研究的自动化
    - 自动化写作前的研究过程，包括**话题研究、信息收集和大纲制作**。
    - 提高写作效率。
2. 多视角信息的整合
    - 从**不同视角**探索和理解信息，产生全面且深入的文章。
    - 通过**模拟对话式问题提问**，自动收集和整理话题相关信息。
3. 生成结构化的文章大纲
    - 利用检索到的信息和提出的问题自动创建大纲。
    - 帮助作者在写作过程中保持组织性和目标明确。
4. 提高文章质量
    - 通过自动化的写作前研究和大纲制作过程，提高文章的组织性和内容覆盖度。