

SELF-DISCOVER: 大语言模型自我构建推理结构

摘要

SELF-DISCOVER 是一个通用框架，用于让大语言模型 (LLMs) 自我发现任务内在的推理结构，以解决典型提示方法难以应对的复杂推理问题。该框架的核心是一个自我发现过程，在这个过程中，大语言模型会选择多个原子推理模块（如批判性思维和逐步思考），并将它们组合成一个明确的推理结构，供大语言模型在解码过程中遵循。与思维链 (CoT) 相比，SELF-DISCOVER 在诸如 BigBench-Hard、基于实际场景的智能体推理和 MATH 等具有挑战性的推理基准测试中，大幅提升了 GPT-4 和 PaLM 2 的性能，提升幅度高达 32%。此外，SELF-DISCOVER 的性能比推理密集型方法（如 CoT-Self-Consistency）高出 20% 以上，同时所需的推理计算量减少 10-40 倍。最后，论文证明了自我发现的推理结构在不同模型系列中具有通用性：从 PaLM 2-L 到 GPT-4，从 GPT-4 到 Llama2，并且与人类的推理模式具有共性。

关键词：SELF-DISCOVER；原子推理模块；推理结构；LLM；推理性能

1 引言

由 Transformer [6] 驱动的大语言模型 (LLM) [1] 在生成连贯文本和遵循指令 [2] 方面取得了令人瞩目的突破。为了提升大语言模型的推理和解决复杂问题的能力，人们从人类推理的认知理论中汲取灵感，提出了各种提示方法。例如，少样本和零样本思维链 (CoT) [4] 类似于人类逐步解决问题的方式；基于分解的提示 [10] 受到人类将复杂问题分解为一系列较小子问题并逐个解决的启发 [5]；回溯提示 [9] 则是受人类反思任务本质以推导通用原则的启发。然而，一个根本的局限是，每种技术本身都作为一个原子推理模块，对如何处理给定任务的过程做出了隐含的先验假设。相反，我们认为每个任务在高效解决问题所涉及的推理过程背后都有一个独特的内在结构。例如，最小到最大提示 [10] 在解决诸如符号操作和组合泛化等任务时，比思维链 [8] 要有效得多，这是由于这些任务的分解结构。

本文旨在自我发现每个任务独特的底层推理结构，同时在计算方面具有高效性。我们提出的方法 SELF - DISCOVER，受人类为解决问题而在内部设计推理程序的方式启发 [3]，如图 1 所示。给定一组用自然语言描述的原子推理模块（如“分解为子任务”和“批判性思维”）、一个大语言模型以及无标签的任务示例，SELF - DISCOVER 会组合出一个与任务内在相关的连贯推理结构（阶段 1），然后使用发现的结构来解决任务实例（阶段 2）。阶段 1 在任务层面运行，使用三个操作来引导大语言模型为任务生成推理结构。在阶段 2 的最终解码过程中，大语言模型只需遵循自我发现的结构即可得出最终答案。

与其他大语言模型推理方法相比，使用 SELF - DISCOVER 解决问题带来了几个好处。首先，发现的推理结构基于原子推理模块，受益于多个推理模块的优势，而不是应用像思维链这样的先验模块。其次，SELF - DISCOVER 在计算上效率高，因为它在任务层面仅需多进行 3 次推理步骤，同时比推理密集型的集成方法 [7] 表现更优。最后，发现的推理结构与任务内在相关，并且与优化提示 [12] 相比，能以更易解释的方式传达大语言模型对任务的洞察。

2 相关工作

2.1 Chain-of-Thought

思维链的概念最早由 Wei 等人 [8] 提出，CoT 通常被视为一种代表性的提示技术或提示策略，主要表现为引导大模型生成最终答案前进行逐步思考和推理，从而提高解决复杂任务的能力。因此，CoT 也可以视为是硅基大脑模仿人类进行分步推理，从而提升自身能力的一个思维过程。当我们自己在解决复杂推理任务时，我们也会将问题分解为多个中间步骤，然后一步步解决直到给出最终的答案。CoT 的目标也是赋予语言模型类似思维链的能力，从而解决复杂的问题。具体来说，在小样本 prompt 的示例中提供 CoT 的演示，可以产生思维链。作为促进语言模型推理的方法，CoT prompting 有一些吸引人的特性：

1. CoT 原则上来说可以让模型将多步任务分解为多个中间步骤，这意味着可以将额外的计算量分配给需要更多推理的步骤。
2. CoT 为模型的生成行为提供了可解释性。
3. CoT 适用各种推理任务，如数学、常识推理等，通用性好。
4. CoT 应用简单，只需要将 CoT 的示例序列包含到小样本 prompt 中，就可以提高模型的推理能力。

2.2 least-to-most prompting

让 LLM 将原来的问题分解为多个需要预先解决的 sub-questions，然后依次按顺序让 LLM 去解决这些 sub-questions，在解决每个 sub-question 的时候，LLM 可以看到之前的每个 sub-question 以及回复。它包含两个 stage，每个 stage 都是通过 few-shot prompt 来实现的，并且整个过程没有任何 model 被训练：

1. Decomposition：这个阶段的 prompt 包含固定的几个用于演示 decomposition 的 few-shot exemplars，然后跟着需要被 decomposed 的 question
2. Subproblem solving：这个阶段的 prompt 包含三个部分：
 - (1) 固定的几个用于演示 subproblem 如何被解决的 few-shot exemplars
 - (2) 先前已经被 LLM 回答了的 subquestions 以及对应的生成的回答
 - (3) 接下来需要被回答的 question

2.3 step-back prompting

先 step back 从更宏观的角度来看问题，让 LLM 对其有一个整体的把握，然后再回到 detail 上面让模型回答具体的问题。这是一个两步的 prompt 策略：

1. 抽象阶段：与直接回答问题不同，首先提示大语言模型提出一个关于更高层次概念或原理的泛化的“退一步”问题，并检索与该高层次概念或原理相关的事实。“退一步”问题对于每个任务都是独特的，以便检索最相关的事实。
2. 推理阶段：基于有关高层次概念或原理的事实，大语言模型可以推理出原始问题的解决方案。这被称为“基于抽象的推理”。

3 本文方法

3.1 本文方法概述

我们从人类如何利用先验知识和技能设计推理程序来解决问题中获得灵感 [3]。当我们面对一个新问题时，通常首先会在内部搜索过往经验中的哪些知识和技能可能有助于解决该问题。然后，我们会尝试将相关的知识和技能应用到这个任务中。最后，我们会将多个单独的技能 and 知识联系起来解决问题。我们将 SELF - DISCOVER 设计为如图 1 所示的两个阶段来执行这些步骤。

给定一个任务和一组代表高级问题解决启发式的推理模块描述（如“运用批判性思维”和“让我们逐步思考”），SELF - DISCOVER 的阶段 1 旨在通过元推理揭示解决此任务的内在推理结构。具体来说，我们使用三个元提示来引导大语言模型选择、调整并实施一个可操作的推理结构，无需标签或训练。由于可解释性以及关于遵循 JSON 格式能提升推理和生成质量的研究发现 [11]，我们将该结构格式化为类似于 JSON 的键值对。

阶段 1 在任务层面运行，这意味着我们只需为每个任务运行一次 SELF - DISCOVER。然后，在阶段 2 中，我们可以简单地使用发现的推理结构来解决给定任务的每个实例，方法是指示模型按照提供的结构填充每个键并得出最终答案。此部分对本文将要复现的工作进行概述，图的插入如图 1 所示：

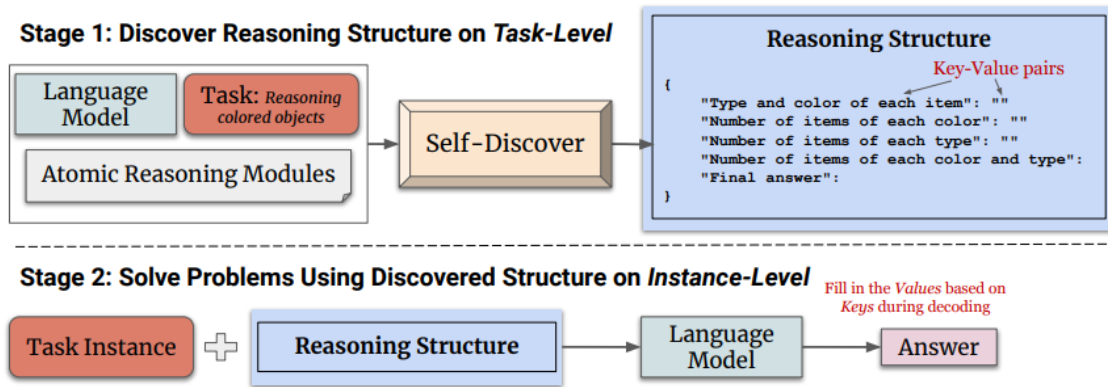


图 1. SELF-DISCOVER 框架

3.2 阶段 1：自我发现特定任务的结构

第一阶段包括三个操作：1) SELECT（选择），从推理模块描述集合中选择对解决任务有用的相关推理模块；2) ADAPT（调整），对所选推理模块的描述进行改写，使其更贴合当前任务；3) IMPLEMENT（实施），将调整后的推理描述实施为一个结构化的可操作计划，以便通过遵循该结构来解决任务。

SELECT 首先, 并非每个推理模块对每个任务都有帮助, 因此 SELF - DISCOVER 的第一阶段引导模型根据任务示例选择有用的模块。例如, “反思性思维”可能有助于在科学问题中寻找第一性原理理论, 而“创造性思维”则有助于为故事生成新颖的后续内容。给定原始的推理模块描述集合 D (如“批判性思维”和“将问题分解为子问题”) 以及一些无标签的任务示例 $t_i \in T$, SELF - DISCOVER 首先使用模型 \mathcal{M} 和元提示选择 p_S 对解决任务有用的推理模块子集 D_S :

$$D_S = \mathcal{M}(p_S \parallel D \parallel t_i) \quad (1)$$

ADAPT 由于每个推理模块提供了关于如何解决问题的一般性描述, SELF - DISCOVER 的下一步旨在使每个所选模块更贴合当前任务。例如, 将“将问题分解为子问题”调整为针对算术问题的“按顺序计算每个算术运算”。给定上一步中选择的推理模块子集 D_S , ADAPT 使用元提示 p_A 和生成模型 \mathcal{M} 对每个所选模块进行改写, 以生成调整后的推理模块描述 D_A :

$$D_A = \mathcal{M}(p_A \parallel D_S \parallel t_i) \quad (2)$$

IMPLEMENT 最后, 给定调整后的推理模块描述 D_A , SELF - DISCOVER 将推理模块实施为一个已实施的推理结构 D_I , 并针对每个步骤指定生成内容的说明。除了一个元提示 p_I , IMPLEMENT 还提供了另一个任务的人类编写推理结构作为示例 S_{human} , 以便更好地将自然语言描述转换为推理结构:

$$D_I = \mathcal{M}(p_I \parallel S_{human} \parallel D_A \parallel t_i) \quad (3)$$

3.3 阶段 2: 使用发现的结构处理任务

经过这三个阶段后, 我们得到了一个专门为我们需要解决的任务 T 调整的已实施推理结构 D_I 。然后, 我们可以简单地将推理结构附加到任务的所有实例上, 并提示模型遵循该推理结构生成答案 A :

$$A = \mathcal{M}(D_I \parallel t), \forall t \in T \quad (4)$$

4 复现细节

4.1 与已有开源代码对比

我参考了 GitHub 上一个复现 self-discover 的工作, 他的源代码如图 2 和图 3 所示, 以下代码是他在 self-discover 的 stage1 和 stage2 的复现工作。

```

# STAGE 1

def select_reasoning_modules(task_description, reasoning_modules):
    """
    Step 1: SELECT relevant reasoning modules for the task.
    """
    prompt = f"Given the task: {task_description}, which of the following reasoning modules are relevant? Do not elaborate on why.\n"
    selected_modules = query_openai(prompt)
    return selected_modules

def adapt_reasoning_modules(selected_modules, task_example):
    """
    Step 2: ADAPT the selected reasoning modules to be more specific to the task.
    """
    prompt = f"Without working out the full solution, adapt the following reasoning modules to be specific to our task:\n{selected_modules}"
    adapted_modules = query_openai(prompt)
    return adapted_modules

def implement_reasoning_structure(adapted_modules, task_description):
    """
    Step 3: IMPLEMENT the adapted reasoning modules into an actionable reasoning structure.
    """
    prompt = f"Without working out the full solution, create an actionable reasoning structure for the task using these adapted reasoning modules:\n{adapted_modules}"
    reasoning_structure = query_openai(prompt)
    return reasoning_structure

```

图 2. 参考代码 1

```

# STAGE 2

def execute_reasoning_structure(reasoning_structure, task_instance):
    """
    Execute the reasoning structure to solve a specific task instance.
    """
    prompt = f"Using the following reasoning structure: {reasoning_structure}\n\nSolve this task, providing your final answer: {task_instance}"
    solution = query_openai(prompt)
    return solution

```

图 3. 参考代码 2

我参考他的模板，尝试了不同的 prompt 去优化改善模型效果，这是最终效果最好的 prompt。如图 4所示。对 stage1 的三个动作，都设计了不同的 prompt，以此让模型达到最优效果。

SELECT	ADAPT	IMPLEMENT
<u>Select several reasoning modules that are crucial to utilize in order solve the given task:</u>	<u>Rephrase and specify each reasoning module so that it better helps solving the task:</u>	<u>Operationalize the reasoning modules into a step-by-step reasoning plan in JSON format:</u>
All reasoning module descriptions <ul style="list-style-type: none"> • Critical thinking: ... • Step-by-Step: ... • Propose and verify: 	SELECTED module descriptions: <ul style="list-style-type: none"> • Critical thinking: ... • Step-by-Step: 	Paired IMPLEMENT Step Demonstration <div>Reasoning description Example</div> <div>Reasoning Plan Example</div>
Task examples w/o answer: Example 1: ... Example 2: ...	Task examples w/o answer: Example 1: ... Example 2: ...	ADAPTED module description:
Select several modules that are crucial for solving the tasks above:	Adapt each reasoning module description to better solve the tasks:	Task examples w/o answer: ...
		Implement a reasoning structure for solvers to follow step-by-step and arrive at correct answers:

图 4. prompt

4.2 实验环境搭建

我是用 gpt-4o 大模型去进行实验，一张 A100 和一张 A800。服务器环境如图 5。

Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M. MIG M.
0	NVIDIA	A100 80GB PCIe	Off	00000000:01:00.0 Off		0
N/A	59C	P0	301W / 300W	17329MiB / 81920MiB	100%	Default Disabled
1	NVIDIA	A100 80GB PCIe	Off	00000000:22:00.0 Off		0
N/A	53C	P0	304W / 300W	79889MiB / 81920MiB	100%	Default Disabled
2	NVIDIA	A800 80GB PCIe	Off	00000000:41:00.0 Off		0
N/A	47C	P0	73W / 300W	77091MiB / 81920MiB	100%	Default Disabled
3	NVIDIA	A100 80GB PCIe	Off	00000000:61:00.0 Off		0
N/A	28C	P0	58W / 300W	70775MiB / 81920MiB	0%	Default Disabled
4	NVIDIA	A800 80GB PCIe	Off	00000000:81:00.0 Off		0
N/A	32C	P0	66W / 300W	69577MiB / 81920MiB	0%	Default Disabled
5	NVIDIA	A800 80GB PCIe	Off	00000000:A1:00.0 Off		0
N/A	53C	P0	351W / 300W	78393MiB / 81920MiB	100%	Default Disabled
6	NVIDIA	A100 80GB PCIe	Off	00000000:C1:00.0 Off		0
N/A	30C	P0	59W / 300W	8939MiB / 81920MiB	0%	Default Disabled
7	NVIDIA	A100 80GB PCIe	Off	00000000:E1:00.0 Off		0
N/A	48C	P0	327W / 300W	69295MiB / 81920MiB	100%	Default Disabled

图 5. 服务器环境

4.3 实验详述

函数 query_llm ：该函数用于向 OpenAI 的聊天完成 API 发送请求。while True 表明会不断尝试请求，直到成功。如图 6 在 try 块中：client.chat.completions.create 调用 OpenAI 的 API 来生成文本。model=os.environ["MODEL"] 从环境变量中获取要使用的模型。messages 是发送给模型的信息列表。temperature 控制生成文本的随机性，值越低越确定，越高越随机。max_tokens 限制生成文本的最大长度。n=1 表示生成一个响应。response.choices[0].message.content.strip() 从响应中提取生成的内容。在 except 块中：打印错误信息并使用 time.sleep(1) 暂停 1 秒，然后继续重试。本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析。

```
def query_llm(messages, max_tokens=2048, temperature=0.1):
    # Retry forever
    while True:
        try:
            response = client.chat.completions.create(
                model="gpt-4o",
                messages=messages,
                temperature=temperature,
                max_tokens=max_tokens,
                n=1,
            )

            content = response.choices[0].message.content.strip()

            return content
        except Exception as e:
            print("Failure querying the AI. Retrying...")
            time.sleep(1)
```

图 6. 函数 query_llm

函数 query_openai ：该函数接收一个 prompt，将其包装成 messages 列表（角色为 user），并调用 query_llm 函数进行查询。如图 7

```
def query_openai(prompt):
    messages = [
        { "role": "user", "content": prompt }
    ]
    return query_llm(messages)
```

图 7. 函数 query_openai

函数 main ：首先定义了一系列的推理模块列表。给出一个简单的任务示例：计算 Lisa 的苹果数量。依次调用 select_reasoning_modules、adapt_reasoning_modules、implement_reasoning_structure 和 execute_reasoning_structure 函数，并打印出各个阶段的结果，最终得到解决任务的最终结果。

```

# Example usage
if __name__ == "__main__":
    reasoning_modules = [
        一系列推理模块
    ]

    task_example = "John and Gary are playing a game. John spins a spinner numbered wi

    selected_modules = select_reasoning_modules(task_example, reasoning_modules)
    print("Stage 1 SELECT: Selected Modules:\n", selected_modules)

    adapted_modules = adapt_reasoning_modules(selected_modules, task_example)
    print("\nStage 1 ADAPT: Adapted Modules:\n", adapted_modules)

    reasoning_structure = implement_reasoning_structure(adapted_modules, task_example)
    print("\nStage 1 IMPLEMENT: Reasoning Structure:\n", reasoning_structure)

    result = execute_reasoning_structure(reasoning_structure, task_example)
    print("\nStage 2: Final Result:\n", result)

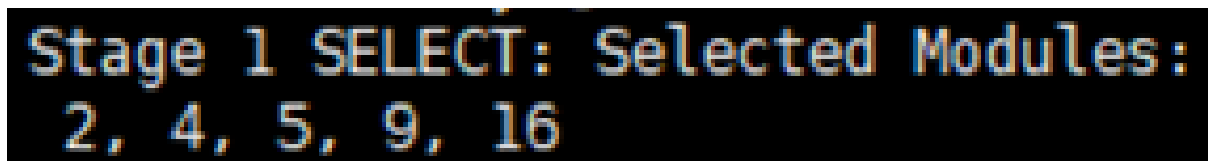
```

图 8. 函数 main

总结 该代码是一个使用 OpenAI API 进行任务推理和解决的程序。它将任务解决过程分为两个阶段，阶段 1 包括选择、适应和实现推理模块，阶段 2 执行推理结构以得到最终的任务解决方案。每个阶段都使用 query_openai 函数向 OpenAI 的 API 发送请求，将相应的信息封装成 prompt，并使用 query_llm 函数不断重试直到成功获取结果。程序的核心是利用自然语言处理能力，通过与 OpenAI 的 GPT 模型交互，将任务分解和解决，利用一系列预定义的推理模块逐步构建解决方案。

5 实验结果分析

提出一个问题 “Lisa has 10 apples. She gives 3 apples to her friend and then buys 5 more apples from the store. How many apples does Lisa have now?”，让 gpt-4o 自己生成推理模块，具体 stage1 的三个动作 select、adapt、implement 以及 stage2 的输出如下图 9、10、11。



Stage 1 SELECT: Selected Modules:
2, 4, 5, 9, 16

图 9. select


```

Stage 1 ADAPT: Adapted Modules:
To adapt the reasoning modules from the sequence task to the specific task involving Lisa and her apples,
we can break down the problem into similar logical steps:

1. **Initial State Identification:**
   - Sequence Task: Identify the starting number in the sequence (2).
   - Apple Task: Identify the initial number of apples Lisa has (10 apples).

2. **Action or Change Recognition:**
   - Sequence Task: Recognize the changes or operations applied to the sequence (e.g., adding numbers to get the next in the sequence).
   - Apple Task: Recognize the actions affecting the number of apples (Lisa gives away 3 apples and buys 5 more).

3. **Intermediate State Calculation:**
   - Sequence Task: Calculate the intermediate numbers in the sequence (e.g., 2 to 4, 4 to 5).
   - Apple Task: Calculate the intermediate state after each action:
     - After giving away 3 apples:  $10 - 3 = 7$  apples.
     - After buying 5 more apples:  $7 + 5 = 12$  apples.

4. **Final State Determination:**
   - Sequence Task: Determine the final number in the sequence (e.g., 16).
   - Apple Task: Determine the final number of apples Lisa has (12 apples).

By following these adapted reasoning modules, we can systematically solve the problem of determining how many apples Lisa has after the described actions.

```

图 10. adapt

```

Stage 1 IMPLEMENT: Reasoning Structure:
To solve the problem of determining how many apples Lisa has after the described actions, we can use the following reasoning structure:

1. **Initial State Identification:**
   - Start by identifying the initial number of apples Lisa has, which is 10 apples.

2. **Action or Change Recognition:**
   - Recognize the first action: Lisa gives away 3 apples.
   - Recognize the second action: Lisa buys 5 more apples.

3. **Intermediate State Calculation:**
   - Calculate the number of apples after the first action:
     - Lisa starts with 10 apples and gives away 3 apples:  $(10 - 3 = 7)$  apples.
   - Calculate the number of apples after the second action:
     - Lisa has 7 apples and buys 5 more:  $(7 + 5 = 12)$  apples.

4. **Final State Determination:**
   - Determine the final number of apples Lisa has, which is 12 apples.

By following these steps, we can conclude that Lisa has 12 apples after giving away 3 apples and buying 5 more.

```

图 11. implement

图 12详细列出了解决此问题的推理步骤：初始状态识别（Initial State Identification）：Lisa 一开始有 10 个苹果。动作或变化识别（Action or Change Recognition）：第一个动作：Lisa 送出 3 个苹果。第二个动作：Lisa 买入 5 个苹果。中间状态计算（Intermediate State Calculation）：送出 3 个苹果后： $10 - 3 = 7$ 个苹果。买入 5 个苹果后： $7 + 5 = 12$ 个苹果。最终状态确定（Final State Determination）：经过这些动作后，Lisa 有 12 个苹果。因此，最终得出结论：Lisa 现在有 12 个苹果。这段内容体现了一种结构化的问题解决方法，通过清晰地列出初始状态、动作、中间状态计算和最终状态确定，有条不紊地得出问题的答案。

```
Stage 2: Final Result:
To solve the problem, we can follow the reasoning structure provided:

1. **Initial State Identification:**
  - Lisa starts with 10 apples.

2. **Action or Change Recognition:**
  - First action: Lisa gives away 3 apples.
  - Second action: Lisa buys 5 more apples.

3. **Intermediate State Calculation:**
  - After giving away 3 apples:  $(10 - 3 = 7)$  apples.
  - After buying 5 more apples:  $(7 + 5 = 12)$  apples.

4. **Final State Determination:**
  - Lisa has 12 apples after these actions.

Therefore, Lisa has 12 apples now.
```

图 12. stage2

同时，我还对 math 数据集中的 algebra、counting_and_probability、geometry、intermediate_algebra、number_thory、prealgebra 六种类别的数据集进行测试。得出的结果与论文中类似。

6 总结与展望

这篇论文聚焦于提升大语言模型 (LLMs) 解决复杂推理问题的能力，提出了 SELF - DISCOVER 框架。该框架模拟人类解决问题的过程，分为两个阶段。阶段 1 通过 SELECT（选择相关推理模块）、ADAPT（调整模块描述）、IMPLEMENT（实施成可操作计划）三个动作，引导 LLMs 生成任务特定的推理结构；阶段 2 则使用该结构解决任务实例。结果显示，SELF - DISCOVER 显著提升了模型在复杂推理任务上的性能，尤其在需世界知识的任务中表现突出，同时计算效率更高，推理计算量大幅减少。

未来可以进一步探索如何改进框架流程，提升推理结构生成的准确性和效率，或增加更多原子推理模块以适应更多类型任务。尝试将 SELF - DISCOVER 应用到更多复杂场景和新兴领域，如自动驾驶、智能医疗诊断等，助力解决实际问题。基于 LLM 与人类推理的共性，研究如何更有效地实现人机协同推理，利用人类的领域知识和模型的计算能力共同攻克难题。

参考文献

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

- [2] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.
- [3] Allen Newell, John Calman Shaw, and Herbert A Simon. Elements of a theory of human problem solving. *Psychological review*, 65(3):151, 1958.
- [4] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.
- [5] George Polya. *How to solve it: A new aspect of mathematical method*, volume 34. Princeton university press, 2014.
- [6] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- [7] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- [8] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [9] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*, 2023.
- [10] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.
- [11] Pei Zhou, Aman Madaan, Srividya Pranavi Potharaju, Aditya Gupta, Kevin R McKee, Ari Holtzman, Jay Pujara, Xiang Ren, Swaroop Mishra, Aida Nematzadeh, et al. How far are large language models from agents with theory-of-mind?, 2023a. URL <https://arxiv.org/abs/2310.03051>, 2023.
- [12] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.