文章编号：1003-0077(2017)00-0000-00

基于程序结构树的多智能体协同数学推理框架

兰孟烨 杨燕

(华东师范大学 计算机科学与技术学院，上海 200062)

摘要：数学推理任务是人工智能模仿人类认知的一项重要任务。现有的方法将数学推理转换为程序推理的方法在小样本提示下取得了较好的效果，但仍然存在示例样本泛化性差以及多种类型的推理错误。为了提高大语言模型的综合推理能力，本文提出基于程序结构树的多智能体协同数学推理框架 PS-GPT，有效避免各类错误，提高大语言模型在数学推理任务上的准确性、可靠性和泛化性。PS-GPT 将推理过程分为推理阶段和优化阶段，采用三类智能体协同工作。推理阶段将程序生成分为四个子过程，每个子过程针对过程特征增强相应的技能，在过程技能提示下生成程序并得到多个方案；优化阶段则基于程序执行的反馈信息迭代进行子过程的反思和优化任务；最后，通过众数投票从优化后的程序执行输出的值中选出最终答案。实验结果表明，PS-GPT 方法在 GSM8K 和 MultiArith 上分别提高了 2.1% 和 0.5%，在不同难度水平的数学推理数据集上的综合能力提高了 1.2%。

关键词：程序推理；多智能体协同；数学推理

中图分类号：TP391 **文献标识码：**A

Program-structured Tree-based Mathematical Reasoning Framework with Multi-agent Collaboration

LAN Mengye, YANG Yan

(School of Computer Science and Technology, East China Normal University, Shanghai 200062, China)

Abstract： Mathematical reasoning is a crucial task in artificial intelligence, aiming to replicate human cognitive processes. Existing approaches that translate mathematical reasoning into program-based reasoning have shown promising results in few-shot learning scenarios. However, these methods still face challenges such as limited generalization and various types of reasoning errors. To enhance the comprehensive reasoning capabilities of LLMs, this paper proposes the Program-structured Tree-based Mathematical Reasoning Framwork with Multi-agent Collaboration (PS-GPT). This framework effectively addresses various types of errors, improving the accuracy, reliability, and generalization of LLMs in mathematical reasoning tasks. PS-GPT consists of two stages: the reasoning stage and the refinement stage, involving collaboration among three types of agents. In the reasoning stage, the program is generated through four procedures in reasoning stage, each associated with specific skills that enhance PS-GPT based on procedure features. Multiple programs are generated using prompts related to procedure-specific skills. In the refinement stage, reflection and refinement of the procedure is conducted iteratively according feedback from execution, and the final answer is selected by majority voting from values outputted through refined programs execution. Experimental results show that PS-GPT achieves improvements of 2.1% and 0.5% on the GSM8K and MultiArith datasets, respectively, while exhibiting an overall enhancement of 1.2% in comprehensive abilities across different difficulty levels of mathematical reasoning datasets.

Key words: program-based reasoning； multi-agent collaboration； mathematical reasoning

收稿日期：2024- 03-20；**定稿日期**：2024- 05-08

0 引言

数学推理是人工智能领域中一项具有挑战性的任务，要求模型具备复杂的认知能力，包括论据抽取，逻辑推理[1]等。随着大语言模型(Large Language Models，LLMs)时代的到来[2]，模型参数显著增加，探索各种提示技术完成特定数学任务成为研究热点。基于小样本提示的方法[3]在推理任务上表现出较好的效果。

思维链(Chain of Thought,CoT)[4]的方法引入了中间步骤启发模型思考推理过程，有效提升了大模型在数学推理上的表现。但 CoT 在数学运算过程常常出现计算错误。为了克服这一缺陷，研究工作将自然语言表达的数学问题转化为编程语言，例如 PAL[5]和 PoT[6]，借助程序解释器等外部工具执行运算。在程序和工具协同作用下，运算精度的提升让 LLMs 在数学推理问题上的整体表现显著提高。PoT 利用程序代替 CoT 中的自然语言完成推理，逐步引导模型生成解决问题的代码。为了进一步提高推理的可靠性，受 Zhou 等人[7]的工作启发，PoT 将原问题分解成子问题，生成每个子问题的代码，在解决每个子问题后得到原问题的解。这种方法不仅降低了推理的难度，且具有可解释性。但在解决更广泛和复杂的推理任务时，可能存在多种类型的推理错误，如图1(a)~(d)所示：由于长推理导致遗忘中间部分；问题表达冗余，受到无关信息干扰；简单问题复杂化问题；以及程序中变量使用前没有先定义[8]。

现有的方法在解决逻辑推理步骤的遗忘问题时，将原问题分解为多个子问题，子问题作为每次程序推理的注释内容保证推理的可解释性进而提高了方法的性能[9]。在解决无关信息干扰时，复述[10]方法通过重新理解忽略或者误解的内容，增强了模型对数学关系的理解能力。对于简单问题复杂化的问题，基于 LLMs 的标准提示方法和它对格式的敏感性[11]用答案变量名简化推理过程，完成用一步推理能够解决的任务。工作[12-14]提出将问题形式化为一组变量和方程的表达，程序通过定义未知变量提示，可以使用函数定义值待定的未知变量和声明式的数学表达式。然而，这些方法聚焦于单一能力的提升，而在程序推理过程中，而一个题目的错误可能由多类问题导致，无法用一种方法完全解决，可能需要同时用到提示工程、迭代优化等多种推理能力增强策略。此外，由于数据集上学习示例的选择和人工构建过程的不确定性，大部分方法难以泛化到其他数据集[15]。

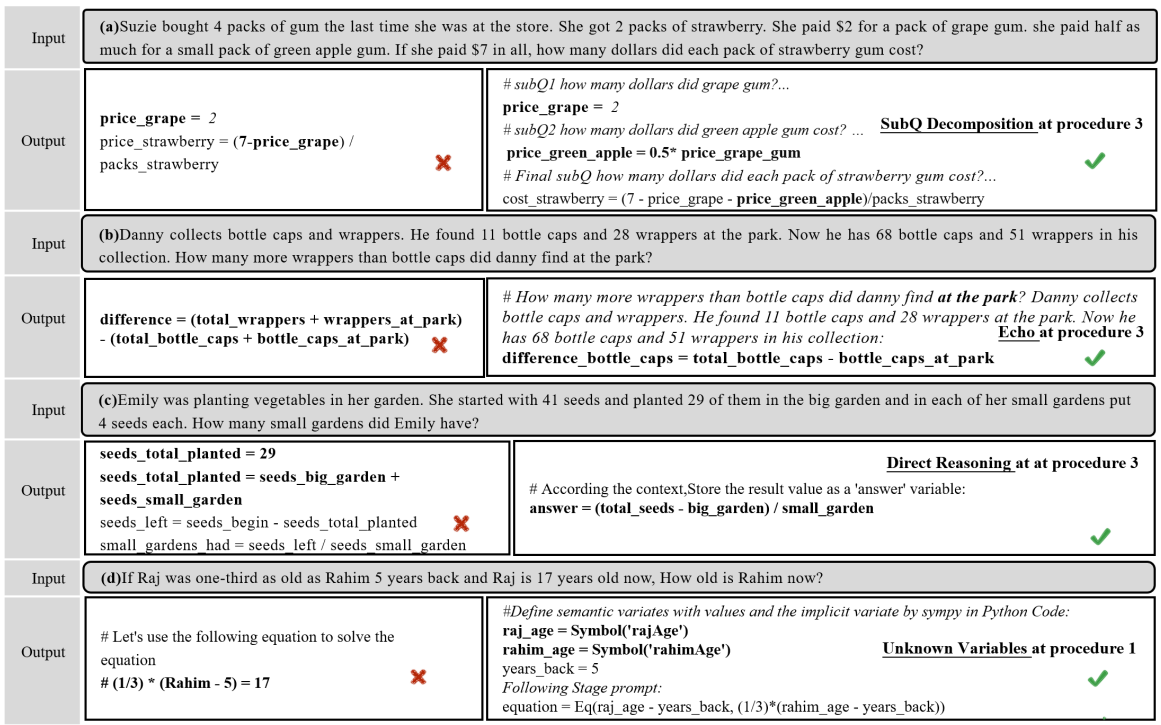

图 1：常见推理错误以及解决方法

因此，为了提高 LLMs 解决程序中常见的推理错误的综合能力，本文探究了数学推理问题生成程序的通用的模块化结构，在推理阶段定义了四个部分，在每个部分针对易出错的问题，使用技能提示以增强模型的推理和代码生成能力。例如，在逻辑推理过程中，可能存在推理步骤遗漏、无关信息干扰以及简单问题复杂化等错误，本文在这一子过程分别用子问题分解和复述以及答案变量名提示方法解决，如图 1(a)~图 1(c)所示。在变量定义子过程中，使用变量定义函数如 Python 中的 Symbol，定义未知变量，如图 1(d)。此外，为了提高推理的准确率和可靠性，受 React[16]和 Reflexion[17]以及 MetaGPT[18]等工作的启发，本文在子过程设置下，设计出基于 React 模式的多智能体协同框架。通过多智能体之间的通信和协同，能够可靠且高效地完成复杂的推理任务包括长步推理等[19]。

综上，本文提出了基于程序结构树的多智能体协同数学推理框架 (Program-structured Tree-based Mathematical Reasoning Framwork with Multi-agent Collaboration,PS-GPT)，是一种零样本提示下通用、可靠和综合能力提升的方案。本文将推理过程分两个阶段：基于技能提示的推理阶段和基于反思的优化阶段。在推理阶段中，技能提示指导推理过程顺序执行四个子过程定义的任务，构建程序结构树，得到多个方案。在优化阶段，执行程序方案得到结果或基于异常信息迭代优化每个方案的子过程代码。为了高效准确地完成每个阶段的任务，设置了提示智能体(Prompt Agent)、程序智能体(Program Agent)和反思智能体(Reflection Agent)协同工作，分别负责提示设计、代码生成、以及程序执行和反馈。本文工作的贡献点概括如下：

1.本文提出了基于程序结构树的多智能体协同的零样本提示数学推理方法 PS-GPT，将推理过程分为推理和优化两个阶段，通过提示智能体、程序智能体以及反思智能体和技能提高 LLMs 数学推理的通用性和可靠性。

2. 本文在推理阶段定义了与程序结构相关的四个子过程，通过多种技能增强模型推理并在不同技能提示下生成程序结构树中的多条推理路径，得到多个程序方案。在优化阶段，执行程序输出结果或利用解释器的异常信息进行子过程的迭代优化。最后，众数投票[20]从所有方案执行结果中选择出现次数最多的值作为最终答案，提高了方法的鲁棒性。

3.本文在不同难度水平的数学推理数据集上实验，结果表明，与其他提示方法相比，PS-GPT 方法在不同数据集上的综合表现提高了 1.2%，在 GSM8K 和 MultiArith 上分别提高了 2.1% 和 0.5%。在 SVAMP 和 AQuA 上实现和最佳性能相当的水平，说明了本文的方法适用于不同的数据集类型，且具有更好的综合能力。

本文组织结构如下：第 1 节介绍 LLMs 在数学推理任务上的相关工作；第 2 节介绍基于程序结构树的多智能体协同数学推理框架 PS-GPT，以及具体的子过程和智能体；第 3 节介绍实验设置及实验结果；第 4 节总结全文，并对未来工作进行展望。

1 相关工作

本节主要介绍 LLMs 在数学推理任务上提示技术、迭代优化以及通过智能体提高推理性能的工作。

基于 LLMs 在数学问题中展现的复杂推理能力，许多提示技术工作相继出现。CoT[4]引入了逐步推理的过程，启发模型思考中间过程，从而提高了 LLMs 的推理能力。为了规避 LLMs 在计算任务上的不足[21]，一些研究[5-6]将程序作为逻辑推理步骤的表达形式来替代 CoT，使用程序思维 PoT[6]逐步解决复杂任务。这种方法将数学问题中的基本运算转移到程序解释器上，从而提高了计算的准确度，增强了大型模型在算术运算任务上的表现。虽然程序能够使用变量赋值语句表示有数值的变量条件，但它们需要更特殊的函数来描述抽象的数学声明[12-14]，即包含未知变量的数学表达式。最后在外部符号解释器上执行计算。He-Yueya 等人[12]将数学问题形式化为线性方程组、增量地生成声明式解答。类似地，Liu 等人[22]提出 EoT，它在代数问题上取得了较好的表现，但在复杂或简单的算术运算问题上的表现不如

PoT。增强 CoT 的方法被证明在 PoT 上是有效的，例如 Gao[6]等人引入 Least-to-Most[7]的方法，程序通过解决多个分解的子问题得到最后的答案。提高了模型在数学推理任务上的表现。此外 Echoprompt[10]将以不同方式复述问题，来增强模型对问题的理解。虽然这些方法适用于程序推理，但没有考虑到自然语言缺少的结构规范。除了语义理解和逻辑推理等方面的能力增强，程序推理需要模型具备更复杂的能力以符合程序结构的语法要求。

现有的迭代优化方法中，Chen 等人[22]提出自我调试的方法，模型通过程序的执行结果并用自然语言解释生成的代码检查错误。此外，利用程序执行结果[24]的异常信息或反思[25]总结的错误迭代优化程序可以进一步提高代码质量。Miao 等人[25]提出，由于检查任务在大多数 LLMs 训练语料库中是一个不太常见的任务，LLMs 识别自己的分步推理错误在很大程度上是无效的。因此，SelfCheck[25]方法在零样本提示下对单步推理过程执行分阶段检查。虽然这一工作通过检查单步推理过程获取多个方案的投票权重的方法提高了预测结果的准确率，没有提出对单步错误改进。因此，在已有的优化方法中，零样本提示下程序在正确执行方面仍然存在挑战。

越来越多的工作结合提示技术和迭代优化方法提出基于 LLMs 的智能体概念，类比于人类的大脑它能够具有规划、记忆以及调用工具的能力[26]。例如，Yao 等人[16]提出 React 方法将结合搜索工具从外部获取知识帮助智能体从环境中获取观察，进行推理和行动。此外设计多个角色任务专一的智能体协同工作能进一步增强单个智能体的推理能力。Hong 等人[18]定义了产品经理、架构师等智能体，每个智能体在 React 模式下完成自动编程任务中的不同部分，通过多个智能体协同可靠地完成较复杂的编程任务[19]。

与之前的工作不同的是，本文提出的 PS-GPT，根据程序结构分为四个通用的子过程，将提示技术作为技能应用在子过程中。提示智能体根据技能设计提示进而指导程序生成。因此，PS-GPT 具备在零样本提示下解决不同推理错误的综合能力；在优化阶段，PS-GPT 通过反思智能体和程序智能体协同完成方案优化任务。反思智能体基于解释器和历史调试信息将文本描述的子过程任务作为反思提示并动态地调整进程，有效地指导程序智能体的优化。

2 PS-GPT 方法

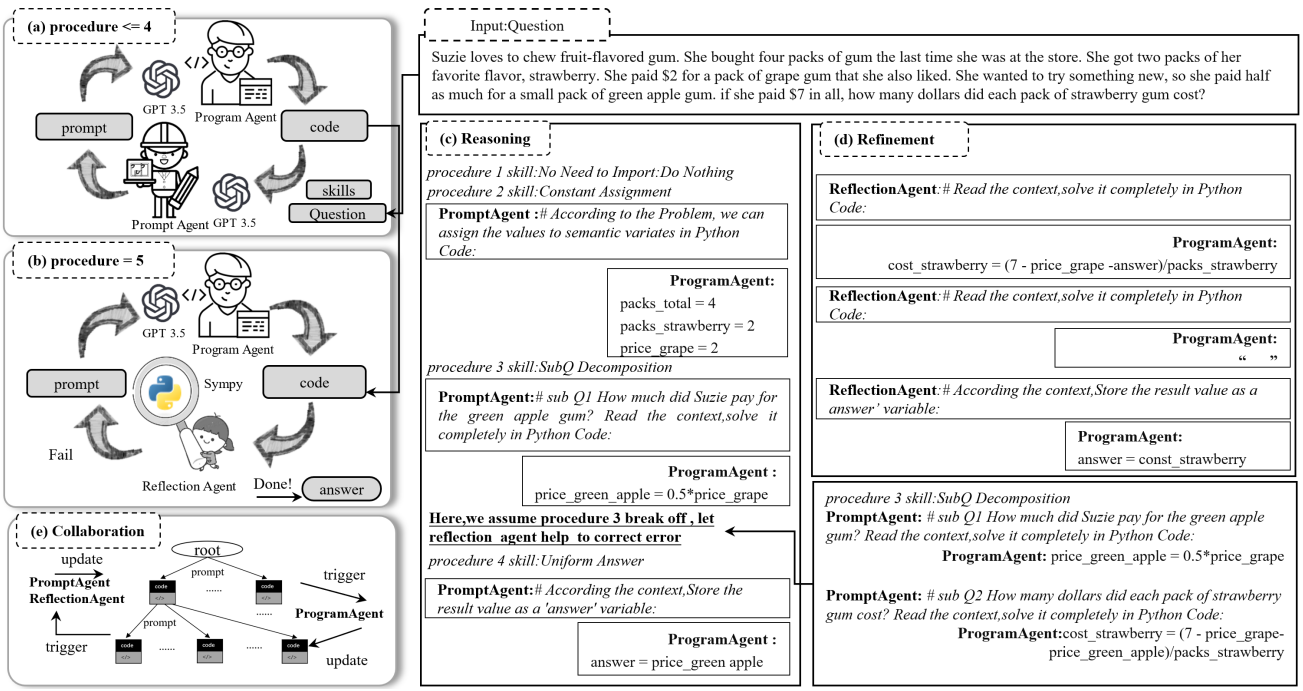

图 2：PS-GPT 中三个智能体在推理和优化阶段的工作过程

2.1 问题定义

给定自然语言描述的问题 Q，推理任务是从 Q 中解析出条件或者约束 ϕ，推导出目标变量 ɵ 的表达式，执行表达式生成答案α。[1]因此推导过程可以分为：约束解析，算术推理、程序执行。约束解析是从Q中抽取变量名并将变量之间关系的自然语言描述解析为满足程序规范的数学表达式 ϕ。算术推理在给定 ϕ 后，将中间代码$code_i$作为推理步骤，推理结束后确定κ个推理步骤$[code_i]_{i=1}^{\mathrm{K}}$，推理步骤以 ϕ 为开始，以目标变量 ɵ 作为结束。给定推理问题 Q，将前 i 个推理步骤作为上下文$C_{<i}$，参数化概率模型p_{LLM}基于零样本提示$prompt_i$生成 $code_i$，逐步推理得到程序方案 $Code$ 的过程如式(1)所示:

$$p_{(Code|Q,C)} = \prod_{i=1}^{\kappa} p_{LLM}\left(code_i|Q, C_{<i}, prompt_i\right) \quad (1)$$

即每个子任务完成需进行一次模型调用，下一个子任务可以访问历史子任务下的提示和推理步骤$C_{<i} = [(prompt_h, code_h)]_{h=1}^{i-1}$，模型在特定子任务提示 $prompt_i$ 和$C_{<i}$指导下生成 $code_i$ 作为第 i 个推理步骤，每个 $code_i$ 涉及一个简单可解决的运算操作并产生可在后续步骤中使用的中间变量。我们使用解释器执行 $Code$，返回最后一步$code_\kappa$中 ɵ 执行结果 α 作为答案。

2.2 总体框架

本文将程序推理过程分为推理和优化两个阶段。推理阶段有四个子过程：工具调用(Tool Usage)、变量定义(Variable Definition)、逻辑推理(Logic Reasoning)、答案返回(Answer Return)。每个子过程任务通过提示智能体和程序智能体协同完成，并设置了多个备选技能以尽可能避免可能出现的推理错误，增强模型的程序推理能力。每个子过程下，提示智能体根据不同技能设计提示，程序智能体根据推理上下文和提示生成中间代码片段。将每个子过程中的技能提示作为树的边，生成的代码作为树上的节点，形成了程序结构树。对树上每一条从根节点到叶节点的推理路径，反思智能体借助程序解释器执行相应的程序方案，输出数字答案或确定优化阶段的提示，程序智能体根据反思智能体提供的提示生成对应的子过程的代码，将优化的程序交给反思智能体，直到程序执行无异常信息或者达到最大反思优化的迭代次数。最后，使用众数投票从多个方案的程序执行结果中确定最终答案。

2.3 程序结构和技能

本文根据程序结构在推理阶段定义了四个子过程，分别为：工具调用、变量定义、逻辑推理和答案返回。其中工具调用子过程需确定问题需要使用的库或库函数；变量定义子过程根据题目的已知条件定义未知变量或已知变量；逻辑推理子过程中，基于已定义的变量逐步推理得到问题答案；答案返回子过程将答案变量约束为一致的变量名。

表 1 每个子过程设置的技能 *skills*

procedure	skill
1.Tool Usage	No Need to Import Scientific Tool
2.Variable Definition	No Need to Define Variables Unknown Variables Constant Assignment
3.Logic Reasoning	Echo SubQ Decomposition Code Reasoning Direct Reasoning
4.Answer Return	Uniform Answer

为了保证推理任务分子过程进行，本文设置了预定义的技能 *skills*，如表 1 所示。在工具调用子过程中，本文设置了两个技能，分别为调用科学工具(Scientific Tool)、无需导入(No Need to Import)，调用科学工具执行子过程下定义的库导入任务如 itertools，math；在变量定义子过程，设置了三项技能，分别为无需定义变量(No Need to Define Variables)，未知变量定义(Unknown Variables)，常量赋值(Constant Assignment)。常量赋值用于描述包含数值的条件，未知变量定义

的技能表达了没有用数值具体化的抽象条件；在逻辑推理子过程定义了复述(Echo)、子问题分解(SubQ Decomposition)、代码推理(Code Reasnoing)以及直接推理(Direct Reasoning)，其中代码推理可以保证逻辑推理子过程任务用代码完成，复述、子问题分解以及直接推理分别解决无关信息干扰、长步推理过程中步骤的遗漏、冗余推理步骤等问题。其中复述技能包括四类，分别是复合复述(Compound)、问句前置(Question First)、问题简化复述(Simple)、以及重复(Repetition)[10]；在答案返回子过程设置了答案统一(Uniform Answer)的技能，它使用答案变量名作为提示，约束了输出答案的命名。

2.4 提示智能体

提示智能体负责将技能转化为指导程序生成的提示。根据每个子过程的技能，提示智能体确定一组提示序列。这些提示序列分别与四个子过程任务相关的提示，引导程序智能体逐步推理得到完整的方案。提示有不同的部分：固有提示和可扩展提示，固有提示与具体问题无关，只和子过程任务相关，可用在多个数学问题类型；扩展提示根据子过程中常见的推理错误设置。在保证任务分子过程进行的前提下，根据技能设计其中一个或两个部分作为提示内容。

对于固有提示，提示智能体从表 2 中子过程和技能以及提示的对应关系确定一个描述子过程任务的提示内容，如式(2)所示：

$$prompt_i^* = \mathrm{Map}(skill_i^*) \quad (2)$$

$skill_i^*$是第 i 个子过程 p_i 在表 2 中对应的固有技能，根据这一技能映射得到固有提示部分 $prompt_i^*$，其中 $1 \le i \le 4$。

为了应对某些子过程常见的推理错误，本文设置了提示的可扩展部分，融合其他提示技术增强推理能力。此外，新的方法也可以作为扩展提示部分用于提高 LLMs 其他方面的能力，从而将不同推理错误的解决方法和能力统一。例如，直接推理技能在逻辑推理子过程提前使用下一个子过程的提示，将推理结果返回给答案变量。未知变量定义技能提示了程序语言中变量定义函数的名称。子问题分解或复述等技能需要完成分解和复述任务，生成与问题相关提示，如式(3)所示：

$$prompt_i^+ = \mathrm{LLM}(Q, skill_i^+) + \mathrm{Map}(skill_i^*) \quad (3)$$

Map 表示当前子过程 p_i 在表 2 所对应的技能提示$prompt_i^*$，LLM 表示利用 LLMs 基于增强技能$skill_i^+$和 Q 生成与具体问题相关的可扩展提示部分。例如，在图 2(c) procedure 3 所代表的逻辑推理子过程中，提示智能体生成和输入问题相关的子问题，然后将生成内容与代码推理技能的对应提示拼接。

表 2 技能和提示的映射关系 Map

procedure 1 skill：Scientific Tool
Use 'import' to help you used complex math tool such as itertools,math in Python Code:
procedure 2 skill：Constant Assignment
According to the Problem, we can assign the values to semantic variates in Python Code:
procedure 3 skill：Code Completion
Read the context,solve it completely in Python Code:
procedure 4 skill：Uniform Result
According the context, Store the result value as a answer variable:

根据提示设计过程的输入和输出，提示智能体设计过程概括为如式(4)所示：

$$prompt_{ij} = \mathrm{PromptAgent}(Q, skill_{ij}) \quad (4)$$

输入包括问题 Q 以及一组预定义技能 *skills*，如表 1 所示。它包含表 2 中的固有技能以及增强技能。在推理阶段，提示智能体将根据 Q 和子过程 p_i 下的某项技能 $skill_{ij}$ 包括 $skill_i^*$或者$skill_i^+$分别通过映射或生成得到提示$prompt_{ij}$，其中 $1 \le i \le 4$，具体流程如图 2(a)所示。

2.5 程序智能体

程序智能体根据子过程的技能提示或优化阶段的反馈提示生成代码。如算法 1 所示，在推理阶段，已完成的子过程以及使用的技能确定了程序结构树上某个节点的推理状态，程序智能体根据问题、推理状态的上下文以及提示设计者提供的提示，生成当前子过程代码。如式(5)所示：

$$code_{ij} = \mathrm{ProgramAgent}(Q, prompt_{ij}, context_{node}) \quad (5)$$

程序智能体得到提示后，提示作为上一子过程的节点 *node* 的分支并拓展出新的节点。新的节点根据 *node* 及其祖先节点提供的上下文

$context_{node}$、提示内容 $prompt_{ij}$ 以及输入问题 Q 执行提示中的推理任务，生成代码 $code_{ij}$。换句话说，在生成 $code_{ij}$ 时，历史子过程 p_1~p_{i-1} 生成的代码和会作为当前子过程 p_i 的上下文。在第一个子过程中，由于根结点 *root* 没有上下文，因此 $context_{node}$ = None。因此，生成代码作为新节点，每个提示作为节点的分支，连接新节点和它的父节点，构建出程序结构树。最后通过深度优先遍历获取每条从根节点到叶子节点的路径，得到多个的方案。

算法 1 程序结构树构建伪代码

算法 1 程序结构树构建.

输入:*Q*,*skills*
输出:*Codes* 程序结构树上的所有方案
1.初始化一颗空树，记录每层的节点和边
2.*Tree* ← InitialTree(*skills*)
3.顺序进行四个子过程推理任务，当前子过程 p_i
4.**for** all $i = 0 \rightarrow 4$ **do**
5. 从预定义的 *skills* 中找到子过程 *pi* 设置的技能 $skill_{ij}$
6. **for** all $skill_{ij}$ in $skill_i$ **do**
7. $prompt_{ij}$ ← PromptAgent(Q,$skill_{ij}$)
8. 从上一个子过程 p_{i-1} 的节点 *node* 上拓展新节点
9. **for** all *node* in p_{i-1} **do**
10. 观察 *node* 以及它所有祖先节点的代码 $context_{node}$
11. $code_{ij}$ ← ProgramAgent(Q,$prompt_{ij}$, $context_{node}$)
12. 生成的代码 $code_{ij}$ 作为新节点更新在树中
13. $prompt_{ij}$ 作为连接新节点和 *node* 的边
14. updateTree(*i*,*j*, *node*) ← $code_{ij}$, $prompt_{ij}$
15. **end for**
16. **end for**
17.**end for**
18.一条路径途径节点的代码组合成一个完整的 *Code*
19.*Codes* ← DFS(*Tree*)
20.**return** *Codes*

2.6 反思智能体

反思智能体负责执行程序并优化执行出错的程序。反思智能体执行方案，基于解释器的执行结果和历史调试信息提供反馈。执行成功的结果将作为候选答案，遇到异常时，将表 2 的固有提示作为反馈指导程序智能体优化。程序智能体根据提示反馈，重新进行子过程任务，如图 4 算法 2 所示。

每个程序方案 *Code* 都是程序结构树上一条从根节点到叶子节点之间的完整路径，程序执行后输出结果，或进一步根据提示优化方案。这一过程需要反思智能体和程序智能体协同工作，如式(6)~(7)所示：

$$prompt' = \text{ReflectionAgent}(Code, log) \quad (6)$$

$$Code = Code + \text{ProgramAgent}(Q, prompt', Code) \quad (7)$$

反思智能体基于解释器的和历史调试信息 *log* 反思得到提示$prompt'$。程序智能体再根据提示优化子过程得到更新后的 *Code*。

为了说明反思智能体的纠错过程，在图 2(c)中，假设推理阶段在执行第三个子过程任务时遗漏了第二个子问题，在程序执行错误的情况下，反思智能体根据解释器的异常信息进行反思，输出反思后得到的提示‘Read the context,solve it completely in Python Code:’并从逻辑推理子过程开始优化。程序智能体基于这一提示和 *Code* 上下文以及输入问题 Q，得到逻辑推理子过程的优化代码。然后将当前子过程优化后的代码和 *Code* 组合成新的方案通过解释器执行，若两次执行得到相同的错误或执行成功，就认为当前子过程的优化任务结束，并进行下一个子过程的反思优化，优化结束条件同上。通过这一方法可以根据历史调试信息动态调整程序的优化进程，如图 2(d)所示。

算法 2 反思优化算法

算法 2 反思优化算法.

输入: *Code*,*log*,*maxTimes*最大迭代次数
输出: *answer*
1.*prompt*,*answer* ← ReflectionAgent(*Code*,*log*)
2.**while** *maxTimes* > 0 **and** *answer* == None **do**
3. *Code* ← *Code* + ProgramAgent(Q,*prompt*,*Code*)
4. *prompt*,*answer* ← ReflectionAgent(*Code*,*log*)
5. *maxTimes* ←*maxTimes* − *1*
6.**end while**
7.**return** *answer*

2.7 多智能体协同

本文基于 React 模式循环推理、行动和观察过程。不同的是，本文使用不同智能体分别执行推理和行动并从程序结构树节点上获取观察，即上下文信息。此外，本文将推理过程分解为具体简单的任务，设置任务专一的角色进行推理和行动。

智能体交替工作，每一步中智能体分工进行提示设计和程序生成，提高了任务执行的准确性。具体来说，在推理阶段每个子过程中，提示智能体在技能指导下设计提示，程序智能体基于提示执行程序生成。设计的提示作为边或生成的代码作为节点更新到程序结构树，另一个智能体基于程序结构树的最新观察触发活动，如图 2(e)所示。在优化阶段，反思智能体和程序智能体将执行的方案作为观察的上下文，将反思后得到的提示或生成的代码更新在这一方案中，触发另一个智能体的活动。迭代进行这一过程，直到程序执行输出结果或达到最大优化次数。因此，程序智能体和提示智能体协同完成推理阶段任务构建程序结构树，针对树上每条路径对应的方案 *Code*，由反思智能体执行相应程序，根据执行结果得到 *answer* 或者设计提示与程序智能体协同完成子过程的代码优化任务。

技能指导下多智能体协同构建的程序结构树中每条推理路径或方案是唯一且具有可解释性。定义当前子过程为 i，假设第 i 个子过程选择了表 1 中技能集合 $skill_i$ 的技能 $\hat{skill_i}$，且第 1～*i-1* 个历史子过程中顺序选择的技能组合为 $\hat{skill}_{<i}$。因此，推理上下文是在 $\hat{skill}_{<i}$ 指导下智能体协同进行子过程提示和代码生成任务后形成的中间推理结果 $context = [(\hat{prompt_h}, \hat{code_h})]_{h=1}^{i-1}$。考虑到历史子过程下各自定义了的 $skill_i$ 下的多个技能，每个子过程的选择不同，得到不同的 $\hat{skill}_{<i}$，因此，推理上下文个数为 $|context|=\prod_{h=1}^{i-1}|skill_h|$。方案总数和推理阶段每个子过程 $skill_i$ 的关系可表示为 $|Codes| = \prod_{i=1}^{\mathrm{K}}|skill_i|$。其中K表示子过程的个数，在不同问题上，每个子过程允许每个提示针对子任务进一步分解，且在优化阶段会根据异常信息重新执行某一子过程提示和代码生成的子任务，因此定义在 2.1 节中的推理步骤个数 $\kappa \geq \mathrm{K}$。最后，众数投票从 *Codes* 的执行结果中选择出现次数最多的数值作为预测答案。

3 实验

本节介绍实验采用的数据集、实验设置、对比方法以及实验结果，并对实验结果进行分析。

3.1 数据集

参考 Liu 等人[22]的工作，本文选择了不同难度水平的数据集,分别是 AQuA、GSM8K、SVAMP、MultiArith 如表 3。其中# Data 表示数据集问题个数，# Step 表示正确推理的步数，* 表示由于数据格式不同带来下的粗略估计。

表 3 实验数据集

Dataset	# Data	# Steps
GSM8K [28]	1319	3.25
SVAMP[29]	1000	1.24
AQuA[30]	254	>= 3*
MultiArith[31]	600	2

3.2 实验设置

本文采用 Python 程序作为推理步骤的中间表达形式，并使用解释器执行计算任务。Sympy 库提供了处理这些数值问题的功能。本文对比了 GPT-3.5[1]上基于贪心解码策略的实验结果。为了公平起见，在使用 LLMs 时，本文设置了参数的温度值 Temperature 为 0，采样个数 sample 为 1。准确度 *Acc* 被作为评估指标如式(8)所示: 其中 *TQ* 表示预测答案和标签答案相等的问题数量，*DN* 表示数据集的问题总数。

$$Acc = \frac{TQ}{DN} \times 100\% \tag{8}$$

3.3 对比方法

为了验证 PS-GPT 方法的有效性，本文将以下方法作为对比基线。

CoT[27]：CoT 方法通过引入一系列中间步骤，引导模型一步步推理得到最终答案。

EoT[22]:该方法使用一组变量和方程的声明式地表达题目中的已有条件和推理关系，最后基于这些条件完成 CoT 的推理过程，并在解释器上执行计算和方程求解。

EchoPrompt[10]:该方法提出在零样本提示 CoT

[1] https://chat.openai.com

设置下，提示 LLMs 首先使用复合复述、问句前置、问题简化复述以及重复中的一种方式将题目重新描述，再结合原问题和生成的新的描述，用 CoT 方法推理得到最终答案。

SelfCheck[25]:该方法提出在零样本提示的分阶段的检查方案。通过减少检查错误和原始生成过程的相关性，LLMs 能够从自己的分步推理中识别错误，最后使用该检查方案对不同生成的答案进行加权投票，从而提高了最终的预测性能。

PoT[6]:该方法提出使用程序作为中间形式表示推理步骤，并将分解后的子问题作为程序注释，应用在 PoT 中。

由于本文的方法是在零样本提示设置下的实现的，因此将 CoT、Echoprompt、SelfCheck 在相同提示设置下的结果作为对比。但 PoT 分解子问题过程以及 EoT 变量定义过程依赖于数据集相关的示例，因此本文给出了小样本提示设置下的结果，如表 4。

3.4 实验结果

如表 4 所示，本文的方法在 GSM8K、MulriArith 上分别提高了 2.1%，0.5%，说明本文的方法提高了 LLMs 在不同难度数学任务上的推理能力。此外，在没有构建特定类型数据集上的学习样例情况下，本文的方法在 AQuA 上的准确率达到了 50.0%，说明方法能够解决不同数学问题类型包括算术运算和代数问题，具有一定的通用性。尽管复述技能 EchoPrompt 作为一个子过程的技能出现在部分方案中，但本文的方法在 SVAMP 上接近最佳性能，具有一定的鲁棒性。最后，本文的方法在四个数据集上实现了最高的平均性能，说明本文有效地利用了不同方法的先进性，综合提高了 LLMs 的能力。

CoT、EoT、Echoprompt 等均使用 CoT 提高了 LLMs 的性能。不同的是，EoT 方法增强了 CoT 解决声明式数学问题的能力，Echoprompt 方法先对问题进行复述再使用 CoT 推理，增强了方法对问题的理解能力以及对无关信息的鲁棒性。相比于 CoT 方法，本文在 GSM8K、SVAMP、MultiArith 数据集上分别提升了 3.6%、2.7%、5.9%，这验证了借助程序解释器完成计算能有效提高方法的性能。然而，在 AQuA 上的结果比 CoT 低 1.1%，通过分析发现，数据集的部分答案中包含程序无法输出的数学表达式或文字的答案，因此程序作为推理的表达形式制约了方法在这类问题上的性能。在没有这一限制的情况下，方法在 AQuA 上的表现可能介于 50.0%到 61.4%之间。对比 EoT，本文的方法在 GSM8K、SVAMP、MultiArith 上的表现分别提高了 15.5%、13.6%、20.7%，但在 AQuA 数据集上的差别较小。这一结果表明，EoT 方法在自定义数据集上具有独特优势，但针对一类问题的增强方法难以泛化到问题类型差异较大的非代数数据集上。此外， Echoprompt 只在 SVAMP 上取得和本文方法相近的最佳性能，在 GSM8K、MulriArith 上的结果，PS-GPT 分别高了 2.9%、3.0%。结合上述两点，可以得出的结论是：EoT、Echoprompt 等提示技术增强方法能够提高 LLMs 的单一能力，但本文的方法通过多技能的设置融合不同提示技术，能够提高了 LLMs 的综合表现。

表 4 基线方法和实验结果

Method（%）	GSM8K	SVAMP	AQuA	MultiArith	Avg.
CoT[10]	75.7	80.5	51.1	93.4	75.2
EoT[22]	63.8	69.6	46.7	78.6	64.7
EchoPrompt[10]	76.4	83.5	50.8	96.3	76.8
SelfCheck[25]	74.4	-	-	-	-
PoT[10]	77.2	79.5	49.2	98.8	76.2
PS-GPT	79.3	83.2	50.0	99.3	78.0

PoT 和 SelfCheck 方法在解决数学推理问题时，基于可执行的编程语言生成逐步推理过程。PoT 使用子问题分解的增强技术将问题分步解决，SelfCheck 生成多个方案，基于逐步的分阶段检查结果投票确定答案。综合 PoT、SelfCheck 在 GSM8K 上的结果可以发现，SelfCheck 的分阶段检查以及加权投票带来的优势不能弥补它在零样本提示下的缺陷。即使和小样本提示下的 PoT 方法相比，本文的方法基于零样本提示仍然在 GSM8K、SVAMP、AQuA、MultiArith 数据集上比 PoT 高了 2.1%、3.7%、0.8%、0.5%。这说明提示智能体将多种提示技术转换为固有提示和扩展提示，将 CoT 和 PoT 上的增强方法统一，并在多智能体协同机制下发挥了作用。

3.5 消融实验

本节为验证子过程、多技能以及优化阶段设置的有效性，进行了消融实验。考虑到调用 GPT-3.5 的成本，本文分别在每个数据集上随机采样了 100 条数据。

表 5 消融实验结果

Method (%)	GSM8K	SVAMP	AQuA	MultiArith
PS-GPT	**80.0**	**82.0**	**61.0**	**99.0**
w/o skills	62.0	59.0	33.0	96.0
w/o procedure	37.0	52.0	42.0	61.0
w/o Refinement	41.0	70.0	55.0	99.0

消融实验的结果如表 5 所示。"w/o skills"表示删除了使用多个技能设置，但保留了固有提示对应的技能；"w/o procedure"表示删除了子过程设置相关的固有提示，并将其他技能使用在一次提示生成过程中；"w/o Refinement"表示删除了优化阶段，四个子过程结束后直接输出程序执行结果。

3.5.1 多技能的有效性

为了验证每个子过程综合不同技能的有效性，本文进行了删除多个技能设置的消融实验。在这个实验中，每个子过程只使用固有提示对应的技能，基于相同的子过程性提示生成多个样本。具体地，为了获得和本文的方法相同数量的候选方案，在工具调用、变量定义以及逻辑推理子过程分别设置采样生成的样本个数为 2、3、7。实验结果显示，在删除了不同技能设置后，方法在 GSM8K、SVAMP 和 AQuA 数据集任务上分别下降了 18.0%、23.0%和 28.0%；而在简单的数据集任务 MultiArith 上保持相对稳定水平。

3.5.2 子过程的有效性

本文将程序生成分为多个子过程，将技能应用在子过程中，为了验证这一设计的作用，本文进行了删除子过程设置的实验。将未知变量定义、复述、子问题分解、直接推理这些技能应用在程序生成的整个过程，每个技能下设置采样生成 6 个样本，保证最后的方案数为 42。实验结果表明，删除子过程设置会严重影响 PS-GPT 在四个数据集上的表现，在 GSM8K、SVAMP、AQuA 和 MultiArith 四个数据集上分别下降了 43.0%、30.0%、19.0%和 38.0%。

3.5.3 优化阶段的作用

通过推理阶段的四个子过程，PS-GPT 可以得到完整的解决方案，但反思智能体和程序智能体协同完成反思以及优化任务，能进一步提高方法在复杂问题上的推理能力。为了验证优化阶段的有效性，本文将解释器上执行的结果直接输出。不进行优化的情况下，在推理难度更大的算术运算问题 GSM8K 上，方法的性能下降了 39.0%，且在 SVAMP、AQuA 上都有小幅度的下降。这说明优化阶段提高了代码质量，尤其在解决复杂算术运算任务上发挥了重要作用。

3.6 案例分析

如图 5(a)所示，在不引入复述技能的情况下，推理过程中忽略了问句中的状语修饰‘at the park’。使用复述技能后，在逻辑推理子过程能够注意到提问的范围，跳过了‘Now’时间状态下的无关信息，正确计算答案。说明解决推理错

误的方法能够有效地应用在推理的子过程中。

如图 5(b)所示，在逻辑推理子过程中采用了复述方法后，推理过程会考虑到先前定义的变量，并且能够利用这些信息进行后续的推理。四个子过程将推理任务分解后，预先进行工具调用以及变量定义这些必要的工作，再将技能应用于逻辑推理子过程，能够避免了无关子过程上的失误对技能作用的影响。

如图 5(c)所示，反思智能体使用代码推理技能的提示：‘Read the context,solve it completely in Python Code:’，提示程序智能体进行逻辑推理子过程中未完成的任务。若任务结束，则使用答案统一技能的提示：‘According the context, Store the result value as a answer variable:’，以确保任务被完整地解决。这一设置可以优化异常的方案，提高整个推理过程的鲁棒性。

4 总结与展望

本文提出了基于程序结构树的多智能体协同构建技能树的数学推理框架框架(PS-GPT)，设计推理任务分推理阶段和优化阶段完成。推理和优化的任务由提示智能体或反思智能体以及程序智能体协同完成。在推理阶段，任务分为四个子过程。由于本文将子问题分解、复述、未知变量定义等技能分别应用在相关的子过程中，指导提示智能体设计提示，程序智能体在生成代码时避免了不同推理错误，提高 LLMs 在数学推理任务上的综合能力。在优化阶段，反思智能体结合解释器和历史调试信息，控制反思优化的进程，输出结果或反馈提示。程序智能体根据提示生成代码，进一步优化解决方案。在零样本提示设置下，基于多智能体协同推理的方法提高了方法的可靠性和通用性，能够解决不同难度和类型的数学推理任务。

未来的研究可以扩展技能设置以及零样本提示的设计方法，以适应更多类型的数学推理任务。

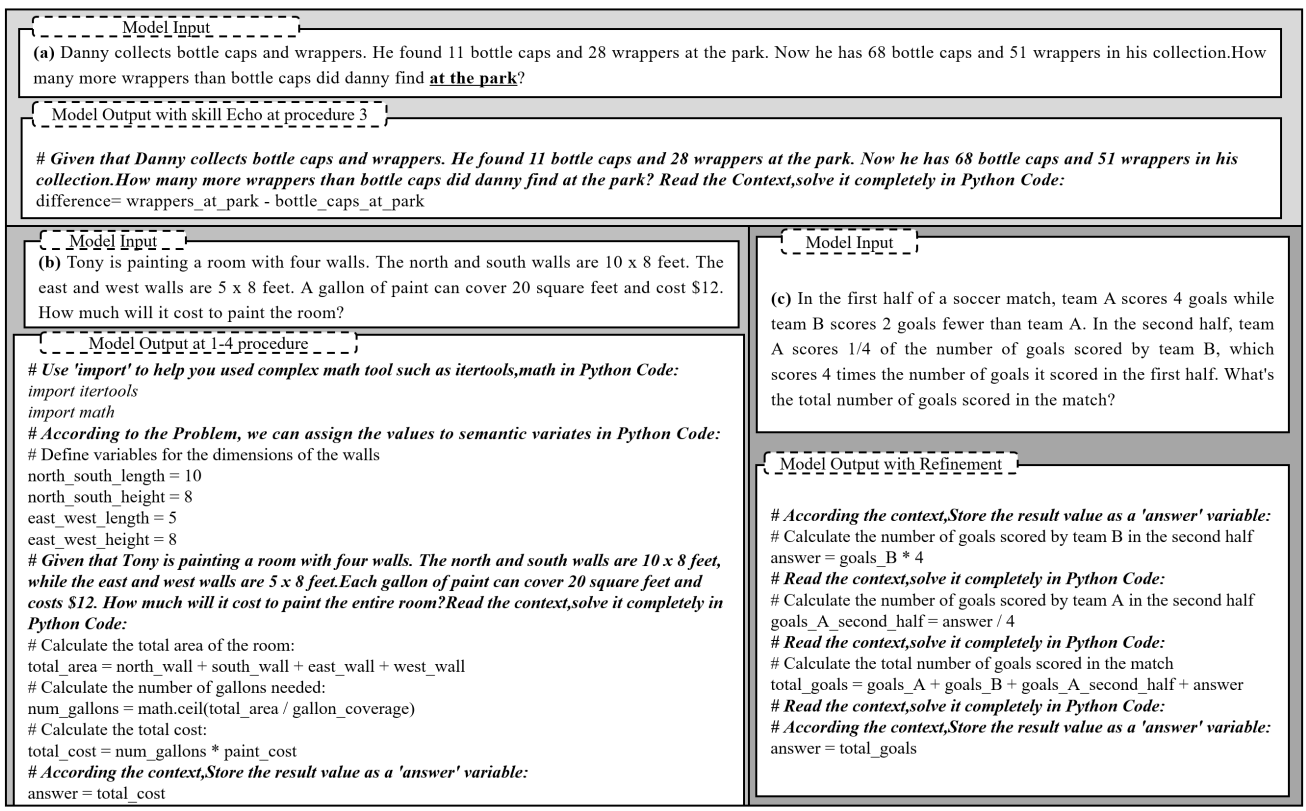

图 5：案例分析技能、子过程、优化阶段作用

参考文献

[1] Huang J, Chang K C C. Towards reasoning in large language models: A survey[J/OL]. arXiv preprint arXiv:2212.10403, 2022.

[2] Kaplan J, McCandlish S, Henighan T, et al. Scaling laws for neural language models[J/OL]. arXiv preprint arXiv:2001.08361, 2020.

[3] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in Neural Information Processing Systems, 2020, 33: 1877-1901.

[4] Wei J, Wang X, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models[J]. Advances in Neural Information Processing Systems, 2022, 35: 24824-24837.

[5] Gao L, Madaan A, Zhou S, et al. Pal: Program-aided language models[C]//International Conference on Machine Learning. PMLR, 2023: 10764-10799.

[6] Chen W, Ma X, Wang X, et al. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks[J]. Transactions on Machine Learning Research, 2023.

[7] Zhou D, Schärli N, Hou L, et al. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models[C]//The Eleventh International Conference on Learning Representations. 2022.

[8] Wang D, Dou L, Zhang W, et al. Exploring Equation as a Better Intermediate Meaning Representation for Numerical Reasoning of Large Language Models[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2024, 38(17): 19116-19125.

[9] Lyu Q, Havaldar S, Stein A, et al. Faithful Chain-of-Thought Reasoning[C]//The 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (IJCNLP-AACL 2023). 2023.

[10] Mekala R S R, Razeghi Y, Singh S. EchoPrompt: Instructing the Model to Rephrase Queries for Improved In-context Learning[C]//The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23. 2023.

[11] Raiyan S R, Faiyaz M N, Kabir S M J, et al. Math Word Problem Solving by Generating Linguistic Variants of Problem Statements[C]//Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop). 2023: 362-378.

[12] He-Yueya J, Poesia G, Wang R, et al. Solving Math Word Problems by Combining Language Models With Symbolic Solvers[C]//The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23. 2023.

[13] Ye X, Chen Q, Dillig I, et al. Satlm: Satisfiability-aided language models using declarative prompting[J]. Advances in Neural Information Processing Systems, 2024, 36.

[14] Yang S, Li X, Cui L, et al. Neuro-Symbolic Integration Brings Causal and Reliable Reasoning Proofs[J/OL]. arXiv preprint arXiv:2311.09802, 2023.

[15] Lu P, Qiu L, Chang K W, et al. Dynamic Prompt Learning via Policy Gradient for Semi-structured Mathematical Reasoning[C]//The Eleventh International Conference on Learning Representations. 2022.

[16] Yao S, Zhao J, Yu D, et al. ReAct: Synergizing Reasoning and Acting in Language Models[C]//The Eleventh International Conference on Learning Representations. 2022.

[17] Shinn N, Cassano F, Gopinath A, et al. Reflexion: Language agents with verbal reinforcement learning[J]. Advances in Neural Information Processing Systems, 2024, 36.

[18] Hong S, Zhuge M, Chen J, et al. MetaGPT: Meta Programming for Multi-Agent Collaborative Framework[C]//The Twelfth International Conference on Learning Representations. 2023.

[19] Zhang H, Du W, Shan J, et al. Building Cooperative Embodied Agents Modularly with Large Language Models[C]//The Twelfth International Conference on Learning Representations. 2023.

[20] Wang X, Wei J, Schuurmans D, et al. Self-Consistency Improves Chain of Thought Reasoning in Language Models[C]//The Eleventh International Conference on Learning Representations. 2022.

[21] Zhou A, Wang K, Lu Z, et al. Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification[C]//The Twelfth International Conference on Learning Representations. 2023.

[22] Liu T, Guo Q, Yang Y, et al. Plan, Verify and Switch: Integrated Reasoning with Diverse X-of-Thoughts[C]//The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23. 2023.

[23] Chen X, Lin M, Schaerli N, et al. Teaching Large Language Models to Self-Debug[C]//The 61st Annual Meeting Of The Association For Computational Linguistics. 2023.

[24] Madaan A, Tandon N, Gupta P, et al. Self-Refine: Iterative Refinement with Self-Feedback[C]//Thirty-seventh Conference on Neural Information Processing Systems. 2023.

[25] Miao N, Teh Y W, Rainforth T. SelfCheck: Using LLMs to Zero-Shot Check Their Own Step-by-Step Reasoning[C]//The Twelfth International Conference on Learning Representations. 2023.

[26] Xi, Zhiheng, et al. The Rise and Potential of Large Language Model Based Agents: A Survey [J/OL]. arXiv preprint arXiv:2309.07864 ,2023.

[27] Kojima T, Gu S S, Reid M, et al. Large Language Models are Zero-Shot Reasoners[C]//ICML 2022 Workshop on Knowledge Retrieval and Language Models, 2022, 35: 22199-22213.

[28] Jie Z, Lu W. Leveraging Training Data in Few-Shot Prompting for Numerical Reasoning[C]//Findings of the Association for Computational Linguistics: ACL 2023. 2023: 10518-10526.

[29] Patel A, Bhattamishra S, Goyal N. Are NLP Models really able to Solve Simple Math Word Problems?[C]//Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics, 2021.

[30] Ling W, Yogatama D, Dyer C, et al. Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems[C]//Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2017: 158-167.

[31] Roy S, Roth D. Solving General Arithmetic Word Problems[C]//Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015: 1743-1752.

兰孟烨(2001—)，硕士研究生，主要研究领域为自然语言处理。
E-mail：mengyelan@stu.ecnu.edu.cn

杨燕(1975—)，通讯作者，博士，副教授，主要研究领域为自然语言处理与认知计算。
E-mail：angel_yangyan@qq.com