《任务 3：基于大模型 API 的智能应用开发》

子任务 1、2

详细设计报告

https://github.com/Her-77/gpt-math.git

目录

第一章　系统实现架构

1.1　总体架构

本系统是为解答中学数学问题而设计的，它通过集成多个专用智能体（Agents）在内部的复杂架构实现问题的自动解答。系统的设计以用户友好和高效性为目标，确保能够快速准确地处理各类数学问题。系统的架构可以分为三个主要部分：前端用户界面、后端服务、以及智能体处理层。

1.1.1　前端用户界面

提供一个交互式的 Web 界面，用户可以通过这个界面提交数学问题，并接收系统生成的详细解答。这个界面是使用现代 Web 技术构建的，确保了良好的用户体验和跨平台兼容性。

1.1.2　后端服务

核心服务采用 Flask 框架搭建，负责处理来自前端的请求，调度各智能体进行任务处理，并将处理结果返回给前端展示。后端还管理系统的状态和数据，确保系统的稳定运行和数据的安全。

1.1.3　智能体处理层

系统的核心，包括多个专门设计的智能体，每个智能体负责处理特定的子任务，如问题解析、计算、推导、反思和解释等。智能体之间高度协作，通过复杂的逻辑和数据流动实现问题的解答。

1.2　智能体详细架构

智能体处理层是系统的核心，它由多个专用智能体组成，这些智能体共同协作解决数学问题。每个智能体都有特定的功能和责任。

1.2.1　思考智能体（Thinker Agent）

负责根据用户问题生成初步的解题思路和方法，是系统的起点。

1.2.2　反思智能体（Reflection Agent）

对初步思路进行反思，评估其可行性，并进行必要的调整和优化。

1.2.3　分析智能体（Analyzer Agent）

对反思后的思路进行深入分析，确保思路的完整性和准确性。

1.2.4　计算智能体（Computation Agent）

负责具体的数学计算，提供解题所需的数值和公式计算结果。

1.2.5　解释智能体（Explaner Agent）

生成详细的解题步骤和解释，帮助用户理解解答过程。

1.2.6 写作智能体（Writer Agent）

将解题过程整理成文档，以便输出给用户查看。

1.2.7 设计智能体（Designer Agent）

负责根据解题文档设计输出的版式和格式。

1.2.8 编辑智能体（Editor Agent）

对设计后的文档进行编辑，确保其格式美观，内容准确。

1.2.9 发布智能体（Publisher Agent）

将最终编辑好的文档发布到前端用户界面，供用户查看和下载。

1.3 系统架构图

智能体间的数据流和处理流程通过一系列定义好的逻辑路径进行，包括循环和条件分支，以确保处理的灵活性和精确性。这些智能体通过一个精心设计的图结构相互连接，允许数据和控制信息在智能体之间高效流动。

下图（图 1）为本项目系统架构图：

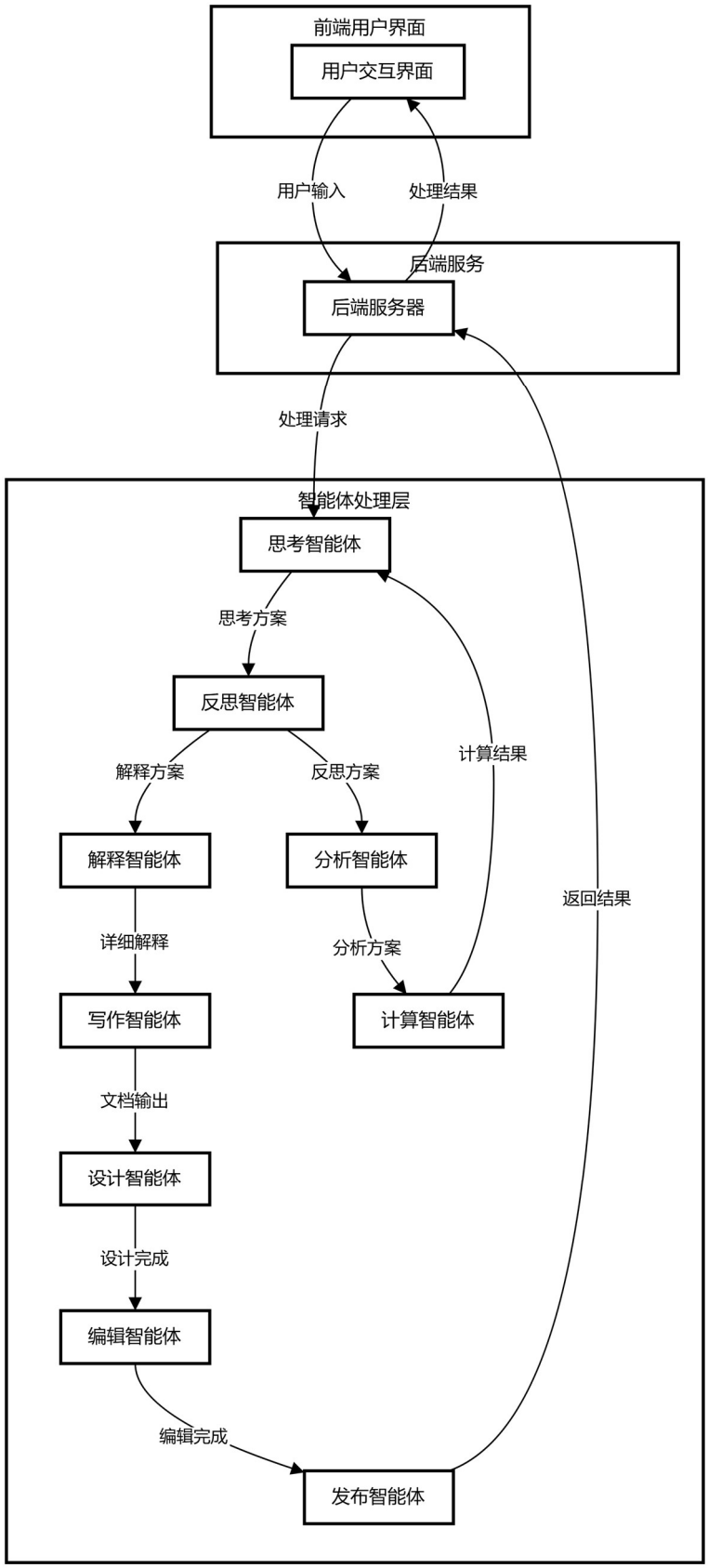

前端用户界面
用户交互界面
用户输入
处理结果
后端服务
后端服务器
处理请求
智能体处理层
思考智能体
思考方案
反思智能体
解释方案
反思方案
计算结果
解释智能体
分析智能体
返回结果
详细解释
分析方案
写作智能体
计算智能体
文档输出
设计智能体
设计完成
编辑智能体
编辑完成
发布智能体

图 1 系统架构图

第二章　各模块的模型和算法描述及实现细节

在这一章中，我们将详细讨论系统中各个智能体的模型、算法和实现细节。这些智能体共同构成了解答中学数学问题的核心处理层，每个模块都针对特定的子任务设计，确保了整个系统的高效性和准确性。

2.1　思考智能体（Thinker Agent）

2.1.1　功能描述

思考智能体负责生成解题的初步策略和方法。它基于当前的进展和反馈，提出下一步的解题思路，并判断是否已经得到最终答案。这一步确保系统能够灵活调整解题路径，逐步逼近正确答案。

2.1.2　设计依据

当我们使用 GPT-4 模型解答数学问题并直接生成结果时，发现会出现公式计算出错、提供解题思路不对以及解题过程中出现推导错误等问题（调用如下）。

为了解决上述问题、优化模型的解题思路，我们这里使用了 CoT 技术来实现分步解决复杂数学问题的方法，其设计依据和优点如下：

1. 在"content"中，我们赋予大模型专家角色以限制其输出内容和回答方式，同时，在 prompt 中明确说明大模型需要根据当前进行至的解题步骤和系统反馈来继续生成解答。

```
"role": "system",
"content": "You are a math expert. Your sole purpose is to
think about what to do next"
"for solving the question you are given based on current
progress and feedback.\n "
```

2. 在每一次反馈循环中，我们都向大模型提供了最终要解决的问题，并且提供前一步的思维结果、计算结果和反馈信息，使模型能够根据这些信息动态调整推理进程、不断进行修正推理路径，以提高问题的解决效率和准确性。

```
"content": f"Question: {problem}\n"
f"Previous thought: {previous_thought}\n"
f"Previous result: {previous_result}\n"
f"feedback : {feedback}\n"
f"Your task is to think about what to do next for solving the
question"
f"you are given based on current progress and feedback.\n "
```

3. 最终，我们还预定义了 JSON 格式，即通过提供 JSON 文件的输出示例，来保证大模型最终的标准输出形式，便于用于后续的调试和测试。同时，"nothing but"语句也能够保证模型最终只输出我们要求的标准化格式文件。

```
f"Please return nothing but a JSON in the following format:\n"
f"{sample_json}\n "
```

4. 在 think 函数中，我们调用了 GPT-4 的模型接口：

```
response = ChatOpenAI(model='gpt-4-0125-preview',
max_retries=3,
model_kwargs=optional_params).invoke(lc_messages).content
```

2.1.3 工作流程

1. 在流程实现中，我们使用了 progress_info，一个人为定义的字典结构（结构如下）作为用于跟踪和管理问题解决进度的关键数据结构：

```
progress_info = {
    "problem": "问题描述",
    "progress": {
        "thoughts": ["思维过程 1", "思维过程 2", ...],
        "progress_summary": ["进展摘要 1", "进展摘要 2", ...],
        "rethoughts": ["重新思考过程 1", "重新思考过程 2", ...]
    },
    "derivation_info": [
        {
            "goal": "目标",
            "conditions": "条件",
            "derivation_process": ["推导步骤 1", "推导步 2", ...],
            "computable_problem": "可计算的问题",
            "result": "结果",
            "feedback": "反馈信息"
        }
    ],
    "is_final": False  # 是否已经得到最终答案
}
```

2. 函数 run 为控制流程的主要部分，负责根据当前进展信息，调用 think 方法，并更新 progress_info，其基本逻辑如下：

(1) 获取当前问题和进展信息。

(2) 如果已有进展：

① 获取 previous thought 和 previous rethought；

```
previous_thought = progress.get("thoughts")[-1]
previous_rethoughts = progress.get("rethoughts")
```

② 调用 think 方法生成新的思考步骤；

```
thought = self.think(problem, previous_rethought,
previous_result, feedback)
```

③ 更新 progress_info 中的 thoughts、progress_summary 和 is_final。

```
        progress["thoughts"].append(thought["thought"])
        progress["progress_summary"].append(thought["progress_sum
mary"])
        progress_info["progress"] = progress
        progress_info["is_final"] = thought["is_final"]
```

(3) 如果没有进展：

① 调用 think 方法生成初始的思维步骤；

```
        thought = self.think(problem, None, None, None)
```

② 初始化 progress_info 中的 progress 字段。

```
        progress = {
            "thoughts": [thought["thought"]],
            "progress_summary": [thought["progress_summary"]]
        }
        progress_info["progress"] = progress
```

(4) 最终，返回更新后的 progress_info 结构。

2.1.4 实现代码

以下是思考智能体的具体实现代码：

```
from langchain_community.adapters.openai import
convert_openai_messages
from langchain_openai import ChatOpenAI
import json5 as json

sample_json = """
{
  "thought": what to do next in one sentence
  "progress_summary": current progress in short summary, None if
no progress
  "is_final": return 'True' if already got final answer, else
'False'
}
"""
class ThinkerAgent:
    def __init__(self):
        pass
    def think(self, problem: str, previous_thought: str | None,
previous_result: str | None, feedback: dict | None):

        prompt = [{
            "role": "system",
            "content": "You are a math expert. Your sole purpose
is to think about what to do next"
                       "for solving the question you are given
based on current progress and feedback.\n "
        }, {
            "role": "user",
            "content": f"Question: {problem}\n"
```

```
                        f"Previous thought: {previous_thought}\n"
                        f"Previous result: {previous_result}\n"
                        f"feedback : {feedback}\n"
                        f"Your task is to think about what to do
next for solving the question"
                        f"you are given based on current progress
and feedback.\n "
                        f"Please return nothing but a JSON in the
following format:\n"
                        f"{sample_json}\n "

        }]

        lc_messages = convert_openai_messages(prompt)
        optional_params = {
            "response_format": {"type": "json_object"}
        }

        response = ChatOpenAI(model='gpt-4o', max_retries=3,
model_kwargs=optional_params).invoke(lc_messages).content
        print(json.loads(response))
        return json.loads(response)

    def run(self, progress_info: dict):
        problem = progress_info.get("problem")
        progress = progress_info.get("progress")
        if progress is not None:
            # 获取之前最近的 thought 和 rethought，如果有 rethought 则使
用 rethought，否则使用 thought
            previous_thought = progress.get("thoughts")[-1]
            previous_rethoughts = progress.get("rethoughts")

            if previous_rethoughts is not None:
                previous_rethought = previous_rethoughts[-1]
                previous_result =
progress_info.get("derivation_info")[-1].get("result")
                feedback = progress_info.get("derivation_info")[-
1].get("feedback")
                thought = self.think(problem, previous_rethought,
previous_result, feedback)
            else:
                # 假设没有 rethought 则没经过计算过程，没有 result
                thought = self.think(problem, previous_thought,
None, None)

            progress["thoughts"].append(thought["thought"])
            progress["progress_summary"].append(thought["progress
_summary"])
            progress_info["progress"] = progress
            progress_info["is_final"] = thought["is_final"]
```

```
        else:
            thought = self.think(problem, None, None, None)
            progress = {
                "thoughts": [thought["thought"]],
                "progress_summary": [thought["progress_summary"]]
            }
            progress_info["progress"] = progress
        return progress_info
```

2.2 反思智能体（Reflection Agent）

2.2.1 功能描述

反思智能体负责对当前解题思路进行评估和优化。它通过分析当前进展和思路，判断思路的有效性，并在必要时提出更好的解决方案。这个智能体确保了解题过程中的思路和方法是最优的，从而提高解题的准确性和效率。

2.2.2 设计依据

反思智能体的设计初衷是为了确保系统能够不断优化解题思路。直接采用初步思路可能导致解答偏离正确方向，因此我们通过使用反思智能体来批判和改进当前思路，帮助系统逐步接近正确答案。

1. 我们依然使用与思考智能体相同的 prompt 设计，引导大模型判断当前思考步骤的正确性和可利用性，并基于当前思考进度得到更好的思考结果。

```
prompt = [{
            "role": "system",
            "content": "You are a math expert. Your sole purpose
is to criticize the usefulness of thought"
                       "for solving the problem and try to give a
better thought. However, if the thought is useful enought, do not
change it. \n "
        }, {
            "role": "user",
            "content": f"Problem: {problem}\n"
                       f"Progress: {progress_summary}\n"
                       f"Thought: {thought}\n"
                       f"Your task is to criticize the usefulness
of thought"
                       f"based on problem and current progress
and try to give a better thought.\n "
                       f"Please return nothing but a JSON in the
following format:\n"
                       f"{sample_json}\n"
        }]
```

2. 标准输出格式如下：

```
sample_json = """
```

```
{
  "rethought": what to do next in one sentence
}
"""
```

2.2.3 工作流程

反思智能体的工作流程如下：

1. 接收当前进展信息：从解题过程中获取当前的进展和思路。
2. 生成批判性评价：通过大语言模型对当前思路进行评价，判断其有效性。
3. 提出改进建议：如果当前思路不够有效，提出更好的解决方案；如果当前思路已经足够好，则保持不变。
4. 更新进展信息：将改进后的思路或确认有效的思路更新到解题进展中。

2.2.4 实现代码

以下是反思智能体的具体实现代码：

```
from langchain_community.adapters.openai import
convert_openai_messages
from langchain_openai import ChatOpenAI
import json5 as json

sample_json = """
{
  "rethought": what to do next in one sentence
}
"""

class ReflectionAgent:
    def __init__(self):
        pass

    def reflect(self, thought: str, problem: str,
progress_summary: str):
        prompt = [{
            "role": "system",
            "content": "You are a math expert. Your sole purpose
is to criticize the usefulness of thought"
                       "for solving the problem and try to give a
better thought. However, if the thought is useful enought, do not
change it. \n "
        }, {
            "role": "user",
            "content": f"Problem: {problem}\n"
                       f"Progress: {progress_summary}\n"
                       f"Thought: {thought}\n"
                       f"Your task is to criticize the usefulness
of thought"
```

```
                        f"based on problem and current progress
and try to give a better thought.\n "
                        f"Please return nothing but a JSON in the
following format:\n"
                        f"{sample_json}\n"
        }]

        lc_messages = convert_openai_messages(prompt)
        optional_params = {
            "response_format": {"type": "json_object"}
        }

        response = ChatOpenAI(model='gpt-4o', max_retries=3,
model_kwargs=optional_params).invoke(lc_messages).content
        print(json.loads(response))
        return json.loads(response)

    def run(self, progress_info: dict):
        problem = progress_info.get("problem")
        progress = progress_info.get("progress")

        rethought = self.reflect(progress["thoughts"][-1],
problem, progress["progress_summary"])

        if progress.get("rethoughts") is not None:
            progress["rethoughts"].append(rethought["rethought"])
        else:
            progress = {
                "rethoughts": [rethought["rethought"]]
            }
            progress_info["progress"].update(progress)
        return progress_info
```

2.3 分析智能体（Analyzer Agent）

2.3.1 功能描述

分析智能体负责对问题进行深入分析，提供详细的解题步骤，并判断是否可以直接得出最终答案。如果需要进一步计算，它将识别出可计算的问题并交给计算智能体处理。通过逐步分析和解答问题，分析智能体确保解题过程的严谨性和连贯性。

2.3.2 设计依据

分析智能体的设计初衷是为了确保系统能够对数学问题进行全面和系统的分析。在解题过程中，单纯的思路往往不足以解决复杂的问题，需要进行详细的步骤分析和推理，且大模型对于数学公式的计算能力有所欠缺。通过引入分

析智能体，系统能够逐步推进解题过程，并在必要时将问题分解为可计算的子问题，交由计算智能体处理。这种分步分析的方法能够提高解题的准确性和系统性。

1. 使用 prompt 如下：

```
prompt = [{
            "role": "system",
            "content": "Your are a math expert. Your sole purpose
is to try to analyze the problem step by step"
                       "based on current progress and your own
thought.\n"
        }, {
            "role": "user",
            "content": f"Problem: {problem}\n"
                       f"Progress_summary: {progress_summary}\n"
                       f"Your Thought: {thought}\n"
                       f"Your task is to analyze the problem step
by step"
                       f"based on current progress and your own
thought.\n"
                       f"If you can get the final answer without
computation, return nothing but a JSON in the following
format:\n"
                       f"{sample_json}\n "
                       f"However if you need to do computation,
return in the following format:\n"
                       f"{sample_json2}\n"
        }]
```

2. 输出该分析智能体的结果，可以得到下面两种输出：

```
sample_json = """
{
  "analysis": the process of your thinking
  "result": final answer.
}
"""

sample_json2 = """
{
  "analysis": the process of your thinking
  "computable_problem": computable problem after your thinking
}
"""
```

若问题无需进行计算，则直接输出最终结果；如果需要计算，则将问题转化为可计算的问题交由计算智能体进行计算。

2.3.3 工作流程

分析智能体的工作流程如下：

1. 接收当前进展信息：从解题过程中获取当前的进展和思路。
2. 生成分析步骤：通过大语言模型对当前问题和思路进行详细分析，逐步推理解题步骤。
3. 判断是否需要计算：如果能够直接得出最终答案，则返回最终答案；如果需要进一步计算，则识别出可计算的问题。
4. 更新进展信息：将分析结果或可计算的问题更新到解题进展中。

2.3.4 实现代码

以下是分析智能体的具体实现代码：

```
from langchain.adapters.openai import convert_openai_messages
from langchain_openai import ChatOpenAI
import json5 as json

sample_json = """
{
  "analysis": the process of your thinking
  "result": final answer.
}
"""

sample_json2 = """
{
  "analysis": the process of your thinking
  "computable_problem": computable problem after your thinking
}
"""


class AnalyzerAgent:
    def __init__(self):
        pass

    def analyze(self, problem: str, thought: str,
progress_summary: str):
        prompt = [{
            "role": "system",
            "content": "Your are a math expert. Your sole purpose
is to try to analyze the problem step by step"
                       "based on current progress and your own
thought.\n"
        }, {
            "role": "user",
            "content": f"Problem: {problem}\n"
                       f"Progress_summary: {progress_summary}\n"
                       f"Your Thought: {thought}\n"
```

```
                        f"Your task is to analyze the problem step
by step"
                        f"based on current progress and your own
thought.\n"
                        f"If you can get the final answer without
computation, return nothing but a JSON in the following
format:\n"
                        f"{sample_json}\n "
                        f"However if you need to do computation,
return in the following format:\n"
                        f"{sample_json2}\n"

        }]

        lc_messages = convert_openai_messages(prompt)
        optional_params = {
            "response_format": {"type": "json_object"}
        }

        response = ChatOpenAI(model='gpt-4-0125-preview',
max_retries=5,
model_kwargs=optional_params).invoke(lc_messages).content
        print(json.loads(response))
        return json.loads(response)

    def run(self, progress_info: dict):
        problem = progress_info.get("problem")
        progress = progress_info.get("progress")

        analysis = self.analyze(problem, progress["rethoughts"][-
1], progress["progress_summary"][-1])
        if "derivation_info" not in progress_info:
            progress_info["derivation_info"] = [analysis]
        else:
            progress_info["derivation_info"].append(analysis)
        return progress_info
```

2.4 计算智能体（Computation Agent）

2.4.1 功能描述

计算智能体负责对可计算的问题进行具体的数学计算和逻辑推理。它接收来自分析智能体分解出的可计算问题，通过执行 Python 代码计算得出答案，并将计算结果返回系统。这个智能体确保所有的数学计算都是准确和高效的。

2.4.2 设计依据

计算智能体的设计采用了 ReAct（Reasoning and Acting）技术。ReAct 技术是一种结合推理和行动的智能体设计方法。它通过结合语言模型的自然语言

处理能力和具体编程计算能力，实现了智能体在处理复杂问题时的高效性和准确性。

ReAct 技术简介

ReAct 技术是一种集成推理（Reasoning）和行动（Acting）的方法，使智能体能够通过自然语言推理和编程计算来解决复杂问题。ReAct 智能体通过以下步骤工作：

1. 推理（Reasoning）：智能体利用大语言模型进行问题的自然语言推理，生成初步的解决方案或步骤。
2. 行动（Acting）：智能体根据推理结果生成并执行相应的代码，获取计算结果。

为何采用 ReAct 技术

在计算智能体的设计中采用 ReAct 技术，主要是为了充分利用大语言模型的推理能力和编程计算的高效性。具体原因如下：

1. 处理复杂计算问题：许多数学问题需要通过编程计算才能得出准确答案，ReAct 技术使智能体能够在推理过程中生成并执行代码，处理复杂的计算问题。
2. 增强系统的灵活性：ReAct 技术使智能体能够灵活应对不同类型的问题，通过动态生成和执行代码，提高系统的适应性。例如使用 PythonREPLTool 需要用户准确严格按照规定格式输入，但是使用 ReAct 技术后用户可以直接以自然语言调用数学计算工具。

2.4.3 工作流程

计算智能体的工作流程如下：

1. 接收可计算问题：从解题进展信息中获取需要计算的问题。
2. 生成并执行计算代码：利用大语言模型生成 Python 代码，并在 Python REPL 环境中执行代码，获取计算结果。
3. 返回计算结果：将计算结果更新到解题进展信息中。

2.2.4 实现代码

以下是计算智能体的具体实现代码：

```
from langchain_experimental.tools import PythonREPLTool
from langchain.agents import AgentExecutor
from langchain import hub
from langchain.agents import create_react_agent
from langchain_openai import ChatOpenAI
```

```
instructions = """You are an agent designed to write and execute
python code to answer questions.
You have access to a python REPL, which you can use to execute
python code.
If you get an error, debug your code and try again.
Only use the output of your code to answer the question.
You might know the answer without running any code, but you
should still run the code to get the answer.
If it does not seem like you can write code to answer the
question, just return "I don't know" as the answer.
"""

prompt = base_prompt.partial(instructions=instructions)
class ComputationAgent:
    def __init__(self):
        self.llm = ChatOpenAI(model='gpt-4-0125-preview',
max_retries=3)
        self.tools = [PythonREPLTool()]
        self.agent = create_react_agent(self.llm, self.tools,
prompt)
        self.agent_executor = AgentExecutor(agent=self.agent,
tools=self.tools, verbose=False)

    def compute(self, computable_problem: str):
        computation_result = self.agent_executor.invoke({"input":
computable_problem})
        print(computation_result)
        return computation_result

    def run(self, progress_info: dict):
        if progress_info["derivation_info"][-
1].get("computable_problem") is None:
            return progress_info
        computable_problem = progress_info["derivation_info"][-
1]["computable_problem"]
        computation_result = self.compute(computable_problem)
        progress_info["derivation_info"][-1]["result"] =
computation_result
        return progress_info
```

2.5 解释智能体（Explainer Agent）

2.5.1 功能描述

写作智能体负责将解题过程和结果编写成一篇结构良好、内容引人入胜的文章。它根据解释智能体提供的材料，生成一篇完整的文章，包括标题、段落

和摘要。文章不仅要展示解题过程和结果，还要以通俗易懂的方式向读者传达解决问题的思路。

2.5.2 设计依据

解释智能体的设计目的是为了帮助用户深入理解解题过程。单纯的解题结果并不足以满足用户的需求，用户需要了解每一步操作的原因和背景知识。解释智能体通过利用大语言模型生成详细的解释和说明，帮助用户全面理解解题过程，提高用户的学习效果和对问题的掌握程度。

2.5.3 工作流程

解释智能体的工作流程如下：

1. 接收解题进展信息：从解题进展信息中获取问题描述、思路和推导过程。
2. 生成详细解释：通过大语言模型生成每一步解题操作的详细说明，包括操作步骤、动机、背景知识和数学推导过程。
3. 返回详细解释：将生成的详细解释更新到解题进展信息中。

2.2.4 实现代码

以下是解释智能体的具体实现代码：

```
from langchain.adapters.openai import convert_openai_messages
from langchain_openai import ChatOpenAI
import json5 as json

sample_json = """
{
  "problem": restate of the problem in short summary
  "key_steps": [
    {
      "step": what to do in one sentence,
      "motivation": goal of this step and benefits specific to
this problem,
      "background_knowledge": math theorem or techniques need to
know, None if nothing important,
      "process_and_result": exact math derivation process and
result, only output math expressions
    }
  ]
}
"""

class ExplainerAgent:
    def __init__(self):
        pass

    def explain(self, problem: str, rethoughts: [str],
derivation_info: dict):
        prompt = [{
```

```
            "role": "system",
            "content": "You are a math teacher expert in
explaining in simple yet accurate terms. "
                       "Your sole purpose is to explain how to
solve a math problem"
                       "and why doing each step based on the
solving process you are given.\n"
        }, {
            "role": "user",
            "content": f"Problem: {problem}\n"
                       f"Thinking process of solving the problem:
{rethoughts}"
                       f"Math derivation of each thought:
{derivation_info}\n"
                       f"Your task is to explain how to solve
this math problem"
                       f"and why doing each step based on the
solving process you are given.\n"
                       f"Please return nothing but a JSON in the
following format, each key step should correspond to one
thought\n"
                       f"{sample_json}\n "

        }]

        lc_messages = convert_openai_messages(prompt)
        optional_params = {
            "response_format": {"type": "json_object"}
        }

        response = ChatOpenAI(model='gpt-4-0125-preview',
max_retries=3,
model_kwargs=optional_params).invoke(lc_messages).content
        print(json.loads(response))
        return json.loads(response)

    def run(self, progress_info: dict):
        problem = progress_info.get("problem")
        progress = progress_info.get("progress")

        explanation = self.explain(problem,
progress["rethoughts"], progress_info["derivation_info"])
        progress_info["explanation"] = explanation
        return progress_info
```

2.6 写作智能体（Writer Agent）

2.6.1 功能描述

写作智能体负责将解题过程和结果编写成一篇结构良好、内容引人入胜的文章。它根据解释智能体提供的材料，生成一篇完整的文章，包括标题、段落和摘要。文章不仅要展示解题过程和结果，还要以通俗易懂的方式向读者传达解决问题的思路。

2.6.2 设计依据

没有写作智能体时，用户需要自己将解题步骤和结果组织成文章，这不仅增加了用户的负担，还可能导致信息表达不清晰，影响理解。有了写作智能体后，系统可以自动生成结构良好的文章，确保内容连贯清晰，并使用 Latex 格式展示数学公式，提高文章的专业性和可读性。

2.6.3 工作流程

写作智能体的工作流程如下：

1. 接收解释材料：从解题进展信息中获取解释智能体生成的详细解释。
2. 生成文章内容：通过大语言模型生成包含标题、段落和摘要的文章，确保内容连贯且易于理解。
3. 返回生成文章：将生成的文章更新到解题进展信息中，并返回供用户查看。

2.6.4 实现代码

以下是写作智能体的具体实现代码：

```
from langchain.adapters.openai import convert_openai_messages
from langchain_openai import ChatOpenAI
import json5 as json

sample_json = """
{
  "title": title of the article,
  "paragraphs": [
    "paragraph 1",
    "paragraph 2",
    "paragraph 3",
    "paragraph 4",
    "paragraph 5",
    ],
    "summary": "2 sentences summary of the article"
}
"""

class WriterAgent:
    def __init__(self):
        pass

    def write(self, material: dict):
        prompt = [{
```

```
            "role": "system",
            "content": "You are a excellent explanatory blog
writer. "
                       "Your sole purpose is to write a well-
written and engaging article about how to solve a math problem"
                       "based on the material provided to you.
The article should contain the final answer of the problem.\n"
                       "Your should using Latex for math formula"
        }, {
            "role": "user",
            "content": f"Material: {material}\n"
                       f"Your task is to write a well-written
article about how to solve a math problem"
                       f"based on the material provided to you.
The article should contain the final answer of the problem.\n"
                       f"Please return nothing but a JSON in the
following format.\n"#, the content should be chinese in UTF-8:\n"
                       f"{sample_json}\n "

        }]

        lc_messages = convert_openai_messages(prompt)
        optional_params = {
            "response_format": {"type": "json_object"}
        }

        response = ChatOpenAI(model='gpt-4-0125-preview',
max_retries=3,
model_kwargs=optional_params).invoke(lc_messages).content
        print(json.loads(response))
        return json.loads(response)

    def run(self, progress_info: dict):
        material = progress_info.get("explanation")
        article = self.write(material)
        return article
```

2.7 设计智能体（Designer Agent）

2.7.1 功能描述

设计智能体负责将写作智能体生成的文章转换为格式美观的 HTML 页面，并保存为文件。它根据 HTML 模板插入文章的标题、日期、图片和段落内容，生成最终的 HTML 文件供用户查看。

2.8.2 设计依据

设计智能体的设计目的是为了提高文章的可视化效果，使其更具吸引力。如果没有设计智能体，用户将只能看到纯文本的解题过程和结果，缺乏视觉上

的美感和专业性。有了设计智能体后，系统能够生成结构化且美观的 HTML 页面，增强用户体验，并使内容更具可读性和专业性。

2.7.3 工作流程

设计智能体的工作流程如下：

1. 加载 HTML 模板：从预定义的模板文件中读取 HTML 模板。
2. 生成 HTML 内容：将文章内容插入到 HTML 模板中，包括标题、日期、图片和段落内容。
3. 保存 HTML 文件：将生成的 HTML 内容保存为文件，供用户查看和下载。

2.7.4 实现代码

以下是设计智能体的具体实现代码：

```
import os
import re
from datetime import datetime
class DesignerAgent:
    def __init__(self, output_dir):
        self.output_dir = output_dir


    def load_html_template(self):
        relative_path = "../templates/article/index.html"
        dir_path = os.path.dirname(os.path.realpath(__file__))
        html_file_path = os.path.join(dir_path, relative_path)
        with open(html_file_path) as f:
            html_template = f.read()
        return html_template


    def designer(self, article):
        html_template = self.load_html_template()
        title = article["title"]
        date = datetime.now().strftime('%d/%m/%Y')
        #image = article["image"]
        paragraphs = article["paragraphs"]
        html_template = html_template.replace("{{title}}", title)
        html_template = html_template.replace("{{image}}",
"https://h7.alamy.com/comp/2AHHNH4/math-on-black-board-with-many-
solution-2AHHNH4.jpg")
        html_template = html_template.replace("{{date}}", date)
        for i in range(5):
            html_template = html_template.replace(f"{{paragraph{i
+ 1}}}", paragraphs[i])
        article["html"] = html_template
        article = self.save_article_html(article)
        return article
```

```
    def save_article_html(self, article):
        filename = re.sub(r'[\/:*?"<>| ]', '_', article["title"])
        filename = f"{filename}.html"
        path = os.path.join(self.output_dir, filename)
        with open(path, 'w') as file:
            file.write(article['html'])
        article["path"] = filename
        return article

    def run(self, article: dict):
        article = self.designer(article)
        return article
```

2.8 编辑智能体（Editor Agent）

2.8.1 功能描述

编辑智能体负责将多个文章按照预定义的布局模板整合成一个完整的 HTML 页面。它根据用户选择的布局模板，将每篇文章的标题、图片、摘要和链接插入到模板中，生成最终的报纸样式 HTML 页面。

2.8.2 设计依据

编辑智能体的设计目的是为了提高系统生成内容的展示效果，使得多个文章可以统一排版，方便用户浏览。如果没有编辑智能体，系统生成的文章将是分散的，缺乏整体布局和美观度。有了编辑智能体后，系统能够生成布局统一、结构清晰的 HTML 页面，增强用户体验和内容的专业性。

2.8.3 工作流程

编辑智能体的工作流程如下：

1. 加载 HTML 模板：从预定义的布局模板文件中读取 HTML 模板。
2. 生成文章 HTML：将每篇文章的内容插入到模板中，包括标题、图片、摘要和链接。
3. 生成报纸页面：将所有文章内容整合到一个 HTML 页面中，生成最终的报纸样式 HTML 页面。

2.8.4 实现代码

以下是编辑智能体的具体实现代码：

```
import os
from datetime import datetime
article_templates = {
    "layout_1.html": """
    <div class="article">
        <a href="{{path}}" target="_blank"><h2>{{title}}</h2></a>
        <img src="{{image}}" alt="Article Image">
        <p>{{summary}}</p>
```

```
    </div>
    """,
    "layout_2.html": """
    <div class="article">
        <img src="{{image}}" alt="Article Image">
        <div>
            <a href="{{path}}"
target="_blank"><h2>{{title}}</h2></a>
            <p>{{summary}}</p>
        </div>
    </div>
    """,
    "layout_3.html": """
    <div class="article">
        <a href="{{path}}" target="_blank"><h2>{{title}}</h2></a>
        <img src="{{image}}" alt="Article Image">
        <p>{{summary}}</p>
    </div>
    """,
}


class EditorAgent:
    def __init__(self, layout):
        self.layout = layout

    def load_html_template(self):
        template_path = os.path.join(os.path.dirname(__file__),
'..', 'templates', 'newspaper', 'layouts', self.layout)
        with open(template_path) as f:
            return f.read()

    def editor(self, articles):
        html_template = self.load_html_template()

        # Article template
        article_template = article_templates[self.layout]

        # Generate articles HTML
        articles_html = ""
        for article in articles:
            article_html = article_template.replace("{{title}}",
article["title"])
            article_html = article_html.replace("{{image}}",
"https://h7.alamy.com/comp/2AHHNH4/math-on-black-board-with-many-
solution-2AHHNH4.jpg")
            article_html = article_html.replace("{{summary}}",
article["summary"])
```

```
            article_html = article_html.replace("{{path}}",
article["path"])
            articles_html += article_html

        # Replace placeholders in template
        str_date = datetime.now().strftime('%d/%m/%Y')
        html_template = html_template.replace("{{date}}",
str_date)
        newspaper_html = html_template.replace("{{articles}}",
articles_html)
        return newspaper_html

    def run(self, articles):
        res = self.editor(articles)
        return res
```

2.9 发布智能体（Publisher Agent）

2.9.1 功能描述

发布智能体负责将生成的报纸样式 HTML 页面保存为文件，并提供文件的路径，供用户查看和下载。它确保生成的 HTML 页面能够被有效保存和访问。

2.9.2 设计依据

发布智能体的设计目的是为了实现系统生成内容的持久化和可访问性。如果没有发布智能体，生成的 HTML 页面将无法被保存和共享，用户无法方便地查看和下载生成的内容。通过发布智能体，系统能够将生成的内容保存为文件，提供文件路径，确保内容的持久性和可访问性。

2.9.3 工作流程

发布智能体的工作流程如下：

1. 保存 HTML 文件：将生成的报纸样式 HTML 页面保存为文件。
2. 返回文件路径：提供生成的 HTML 文件的路径，供用户查看和下载。

2.9.4 实现代码

以下是发布智能体的具体实现代码：

```
import os

class PublisherAgent:
    def __init__(self, output_dir):
        self.output_dir = output_dir

    def save_newspaper_html(self, newspaper_html):
        path = os.path.join(self.output_dir, "newspaper.html")
        with open(path, 'w') as file:
            file.write(newspaper_html)
        return path
```

```
def run(self, newspaper_html: str):
    newspaper_path = self.save_newspaper_html(newspaper_html)
    return newspaper_path
```

第三章　使用的数据资源说明

本项目中使用的数据资源包括中学数学问题与答案数据集。它们为多智能体协作的数学问题解答系统提供了必要的输入和参考。

3.1　数据集来源

3.1.1　APE210K Dataset

1. 这是一个由 Chenny0808 发布大规模且模板丰富的数学文字题数据集，包含 210,488 个问题和 56,532 个模板，适用于训练和评估数学文字题解题系统。
2. 数据集链接：APE210K Dataset on GitHub(GitHub)。

第四章　模块和系统测试方法和测试结果

我们分别进行了模块测试和系统测试。模块测试主要侧重于测试每个 agent 的功能能否达到预期目标，为此我们进行了多轮提示词调优。系统测试主要侧重于测试多个 agent 的协同是否正常，以及系统求解数学问题的整体准确率，为此我们选取了一些典型的数学问题，比较只使用 gpt4 和使用我们的系统求解的准确性。

4.1　模块测试

对于每个 agent 模块，我们都手动构造了输入，并测试其响应是否符合预期。我们实现的每一个 agent 模块，都有和下图类似的相应的单元测试代码，并对提示词进行了多轮优化。

```
if __name__ == "__main__":
    # 测试derivation_agent对第二轮输入的响应
    progress_info = {
        "problem": "如果5个连续奇数的乘积为135135，那么这5个数的和是多少",
        "progress": {
            "thoughts": ["find the five consecutive odd numbers whose product is 135135, and the
            "progress_summary": ["None", "Prime factorization of 135135 has been identified as 3
            "rethoughts": ["Consider prime factorization of 135135 to identify the pattern of fi
        },
        "derivation_info": [{
            "goal": "Sum = n + (n+2) + (n+4) + (n+6) + (n+8)",
            "conditions": "Product = n * (n+2) * (n+4) * (n+6) * (n+8) = 135135",
            "derivation_process": [
                "Prime factorization of 135135",
                "Identify pattern of five consecutive odd numbers from prime factors",
                "Compute sum of these five numbers"
            ],
            "computable_problem": "prime factorization of 135135",
            "result": 'Assumption: factor | 135135 \nAnswer: 3^3×5×7×11×13 (7 prime factors, 5 d
        }]
    }
    updated_progress_info = analyze_agent.run(progress_info)
    print(updated_progress_info)
```

4.2　系统测试

受限于 API 成本，我们并没有在大规模数据集上进行测试，而是选择了一些难度较大的、单纯使用大语言模型求解错误的数学问题，测验系统每个环节的运行是否符合预期以及整体求解的准确率。

首先，我们从 APE210K 中随机选取了二百多道数学问题，只利用 gpt-4-0125-preview 进行回答，得到将近一百道错题。然后我们从中挑选出一些具有代表性的错题，在我们的系统上进行测试，并比较系统的推导过程和最终结果。

这些错题（及系统测试的输入）放在了 **false_res.csv 文件夹** 中。

第五章　测试结果分析

5.1　模块测试

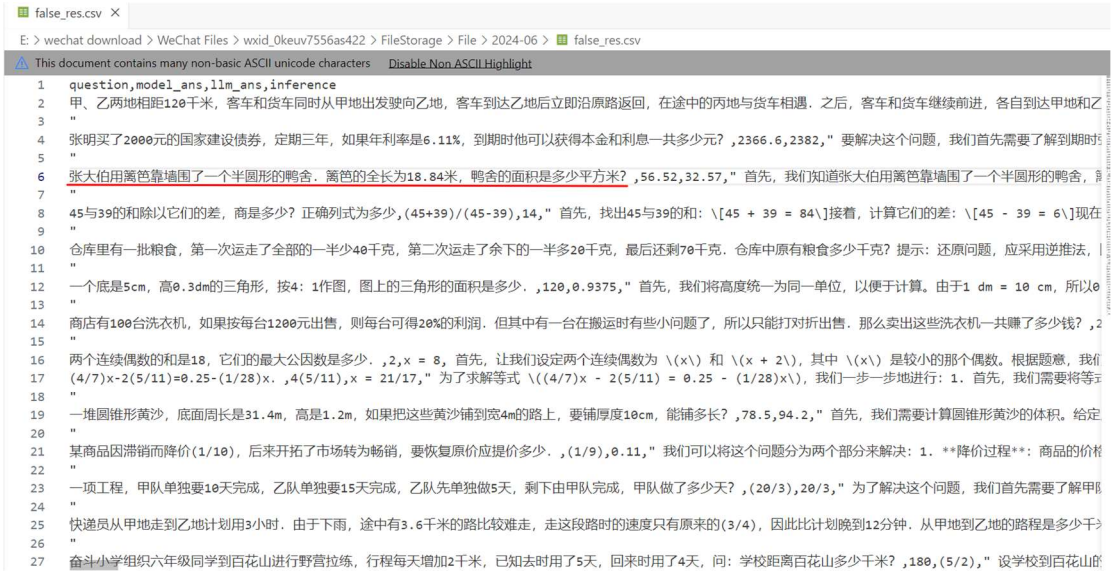
在模块测试的过程中，我们发现如下问题：

1. 提示词难以优化：修改提示词对模型的影响难以确定，并不是说得越详细越好，提示词的顺序对结果也有一定影响，因此提示词的优化主要依赖于反复调试，无法确定是否最优。
2. 模型输出格式不符合预期：为了让模型的输出便于解析，作为下一个模型的输入，我们需要模型的输出符合特定的格式。如果直接用语言说明，模型总会在细节处出现各种错误，最后我们选择向模型提供 json 格式的输出示例，能够较好地解决这一问题。

5.2　系统测试

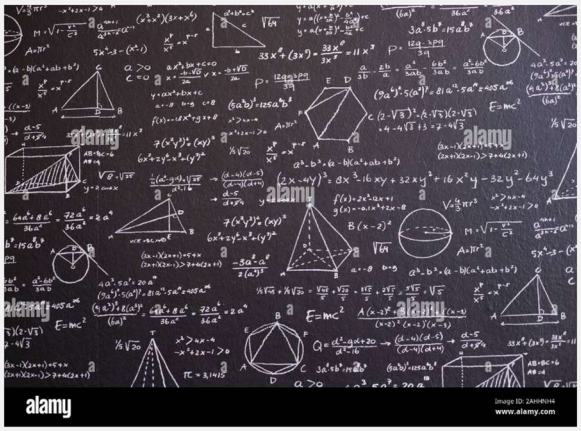
我们将以一道具体问题为例，展示我们系统的测试结果。其它问题的测试结果以 html 的格式统一放在 **output 文件夹** 中，可以直接用浏览器打开。

下图中划红线的问题是一道 gpt4 求解错误的问题，因为他混淆了一个概念，因为靠墙，篱笆的全长不包括直径。



经过多智能体的分析，我们的系统会将最终求解结果以页面的形式展示，如下图所示，包含生成讲解文章的标题和概要。

Calculating the Area of a Semi-Circular Duck Pen



This article explained the process of calculating the area of a semi-circular duck pen given the length of the fence against a wall. The final calculated area for the duck pen is approximately 56.55 square meters.

通过点进页面提供的链接，我们可以看到文章的具体内容，包括对于求解问题每一步骤的分析和解释，以及最终的求解结果。可以看到问题的最终结果是正确的。这并不是由于使用语言的不同，而是因为在呈现最终结果之前，系统进行了多步分析，矫正了自己的思路，并使用了外部工具。

When tasked with finding the area of a semi-circular duck pen that is enclosed by an 18.84-meter fence against a wall, the first instinct might be to wonder where to start. The key is realizing that the fence forms half of the circumference of what would be a full circle. This insight drives the entire solution process, which involves determining the full circle's circumference, solving for the radius, and finally calculating the area of the semi-circle.

To begin, we calculate the circumference of the entire circle if there were no wall, with the given 18.84 meters representing only half of this circumference. The formula for the circumference of a circle is given by $C = 2\pi r$, where r is the radius. Therefore, doubling the length of the fence gives us the full circle's circumference, which is $2 \times 18.84 \text{ meters} = 37.68 \text{ meters}$. This step sets the stage for finding the circle's radius.

With the full circumference calculated, the next step is to find the radius of the circle using the rearranged circumference formula, $r = \frac{C}{2\pi}$. Substituting the full circumference, we get $r = \frac{37.68}{2\pi} \approx 6 \text{ meters}$. This radius is crucial as it is used to determine the area of the circle.

To find the area of the semi-circle, we first compute the area of the full circle using the formula $A = \pi r^2$. Substituting the radius, the area of the full circle is $A_{full} = \pi \times 6^2 = 36\pi \approx 113.1 \text{ square meters}$. Since the problem deals with a semi-circle, the final step is to divide this area by two, giving $A_{semi} = \frac{113.1}{2} \approx 56.55 \text{ square meters}$.

This systematic approach to solving the problem, from doubling the fence length to find the full circumference, to using that in calculating the radius, and finally computing the area of the semi-circle, leads us to the solution efficiently and accurately. The area of the semi-circular duck pen is approximately 56.55 square meters.

在系统测试的过程中，我们发现如下问题：

1. 系统稳定性不够强：由于大模型的输出具有随机性，哪怕把 temperature 设置为零，在不同问题上的表现也未必符合我们的预期。因此系统可能会出现陷入死循环的情况，哪怕增加异常处理，也很难彻底解决这一问题，因为异常处理能够穷举的规则十分有限。
2. 系统响应时间较长：从产品的角度出发，系统必须在用户能够容忍的范围内提供结果，如果超出这个时间，用户体验就会变差，从而不会使用这一产品。我们的系统因为使用了较多的智能体，并且是以线性的方式进行串联，从而很难避免这一问题，也难以将流程改为并行，因为每个环节之间存在依赖关系。为此我们可以寻找特定的应用场景，例如将系统作为博客的生产工具，从而减少对响应时间的要求。

第六章　关于使用已有资源的说明

在本项目中，我们广泛利用了现有的公开数据资源和技术工具，以确保多智能体协作的中学数学问题解答系统的高效开发和准确性。以下是对这些资源的详细说明。

6.1　数据资源

6.1.1　Ape210k dataset：

6.2　技术资源

6.2.1　OpenAI GPT-4 API

用于自然语言处理和生成任务，包括生成数学问题的解答和解释。

6.2.2　LangChain

用于构建大语言模型应用的工具包，支持多种适配器和 API 调用。

6.2.3　Python REPL Tool

用于执行 Python 代码，进行计算和验证解答过程中的数学推导和计算。

6.3　现有资源的使用原则

在使用这些已有资源的过程中，我们遵循以下原则：

6.3.1　遵守开源协议

1. 所有使用的开源数据集和工具均严格遵守其对应的开源协议，确保合法合规使用。
2. 确保引用和标注所有使用的资源来源。

6.3.2　数据隐私与安全

1. 在处理和存储数据时，确保用户数据的隐私和安全。
2. 对数据进行适当的加密和保护，防止未经授权的访问和泄露。

6.3.3　高效利用资源

1. 最大程度地利用现有资源，避免重复劳动。
2. 通过使用高质量的开源工具和数据集，提升系统的性能和准确性。

6.3.4　持续更新与维护

1. 定期检查和更新所使用的资源，确保数据和工具的时效性和有效性。
2. 关注相关领域的最新研究和发展，及时引入新的技术和资源。

第七章　参考文献

1. ReAct: Synergizing Reasoning and Acting in Language Models
2. Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering
3. ExpertPrompting: Instructing Large Language Models to be Distinguished Experts
4. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models
5. MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning
6. AlphaGeometry: An Olympiad-level AI system for geometry
7. GitHub - ndg24/math-genius-ai: project aimed at solving elementary and middle school math problems using artificial intelligence