清华大学
高级机器学习
2024 年秋季学期

小作业-LLM 的高级推理

杨天行 2024310659 2025 年 1 月 11 日

1 实验任务

本实验的任务是探索和实现高级推理能力的语言模型，具体包括以下内容：

- **调研和总结高级推理方法**。对当前提升语言模型推理能力的技术进行调研，包括但不限于 Test Time Compute（测试时计算）、MCTS（蒙特卡洛树搜索）以及 OpenAI 的 O1 模型等方法，形成全面的技术总结。
- **选择并定义推理任务**。明确推理任务的范围和目标，以数学基础推理为例，通过合理的任务设计，检验模型的推理能力。
- **数据准备**。准备训练和测试所需的数据集，选用 Math 数据集进行训练，使用 Math500 数据集评估模型性能。
- **模型构建与算法优化**。加载并应用一个合适的预训练语言模型（如 GLM4）。根据所定义的任务需求，设计与优化用于提升推理能力的算法，包括多步推理、逻辑推理以及复杂决策等方面。
- **性能评估与对比分析**。对模型在验证集上的推理性能进行评估，使用多维指标（准确性、推理速度、复杂度）分析效果。对比不同推理方法的效果，并与基础模型进行对比分析，量化优化成果。

2 高级推理方法总结

测试时计算（Test-Time Compute，TTC）。TTC 是在模型推理阶段动态分配计算资源，以增强模型性能的方法。与传统的静态推理不同，TTC 允许模型在推理过程中执行额外的计算，以实现更深入的推理和实时适应新输入。这使得 AI 系统能够在面对复杂任务时，动态调整其计算策略，从而生成更精确和上下文相关的响应。

蒙特卡洛树搜索（Monte Carlo Tree Search，MCTS）。MCTS 是一种用于决策过程的搜索算法，特别适用于需要规划和多步骤推理的任务。在语言模型中，MCTS 可用于探索可能的推理路径，并通过模拟和评估不同的决策序列，选择最优的解决方案。例如，在数学推理任务中，MCTS 可与能量函数相结合，指导模型选择最有可能的正确推理路径，从而提高模型的准确性。

OpenAI 的 o1 模型是专为增强推理能力而设计的语言模型。与之前的模型相比，o1 在生成答案前会进行多步骤的“思考”，即生成一系列的推理步骤（链式思维），这使其在处理复杂的科学、数学和编程任务时表现出色。o1 的性能随着训练时和测试时计算量的增加而提升，体现了在推理过程中投入更多计算资源的重要性。

3 Baseline 代码分析

3.1 答案评估

作业 code repo 提供的 baseline 代码中实现了 default, 3-shot, 5-shot, g1 一共四种方法，在 MATH500 数据集上进行测试（由于时间有限，本次作业取其中前 100 条进行测试）。评测方法是使用不同方法输出答案文本后，提取其中最后一句作为当前方法的答案，使用`get_critic`函数评估答案是否正确，评估的方法是使用一个固定的 prompt 来评估自动评估答案，重复评估 4 次，得到一次正确的回答即视为正确回答问题。

```python
def get_critic(question, solution, answer, response, response_answer, model = "glm-4-flash"):
    query1 = PROMPT_TEMPLATE.format(
        problem=question,
        reference_answer=solution,
        assistant_ansiwer=response
    )
    critic1 = get_response(query1, model)
    rating1 = re.findall(r"\[\[(\d+)\]\]", critic1)
    query2 = PROMPT_TEMPLATE.format(
        problem=question,
        reference_answer=solution,
        assistant_ansiwer=response_answer
    )
    critic2 = get_response(query2, model)
    rating2 = re.findall(r"\[\[(\d+)\]\]", critic2)
    query3 = PROMPT_TEMPLATE.format(
        problem=question,
        reference_answer=answer,
        assistant_ansiwer=response
    )
    critic3 = get_response(query3, model)
    rating3 = re.findall(r"\[\[(\d+)\]\]", critic3)
    query4 = PROMPT_TEMPLATE.format(
        problem=question,
        reference_answer=answer,
        assistant_ansiwer=response_answer
    )
    critic4 = get_response(query4, model)
    rating4 = re.findall(r"\[\[(\d+)\]\]", critic4)
    return any("1" in sublist for sublist in [rating1, rating2, rating3, rating4])
```

3.2 default & 3/5-shot 方法

Baseline 里实现的 default 方法即为直接输入问题、相关主题、难度给大模型。

```python
    if method == "default":
```

```
        prompt = f"Question: {question}\nSubject: {subject}\nLevel: {level}\nAnswer:"
        return get_response(prompt, model)
```

3/5 shot 方法的实现则为在训练集（MATH）中找到对应数量的 QA 对（要求主题、难度与当前问题一致）添加进入当前 prompt，作为 few-shot learning 进行测试。

```
def get_few_shot_prompt(subject, level, num = 3):
    id = 0
    prompt = ""
    file_subject = ""
    if subject == "Precalculus":
        file_subject = "precalculus"
    elif subject == "Number Theory":
        file_subject = "number_theory"
    elif subject == "Intermediate Algebra":
        file_subject = "intermediate_algebra"
    elif subject == "Prealgebra":
        file_subject = "prealgebra"
    elif subject == "Algebra":
        file_subject = "algebra"

    for i in range(30000):
        data_dir = os.path.join(parent_dir, f"data/math/train/{file_subject}/")
        data_filepath = os.path.join(data_dir, f"{i}.json")
        if os.path.exists(data_filepath):
            with open(data_filepath, "r") as f:
                data = json.load(f)
                if data["level"] == "Level " + str(level):
                    prompt += f"Question: {data['problem']}\nSolution:
                        {data['solution']}\n\n"
                    id += 1
        if id == num:
            break

    return prompt
```

3.3 g1 方法

Baseline 实现的 g1 方法是一个简单的类 CoT 方法。该方法有两种输入的 prompt，第一种是 regular prompt，要求 LLM 推理一次（最多推理 25 次），每次输出包含当前推理的 title，内容，以及在 continue/final step 之间做选择；另一种是 final answer prompt，要求 LLM 总结之前的推理记录并给出最终答案。

```
    steps = []
    step_count = 1
    total_thinking_time = 0

```

```
    while True:
        start_time = time.time()
        step_data = analyze_response(messages, model=model)

        print(f"Step {step_count}: {step_data} \n")

        end_time = time.time()
        thinking_time = end_time - start_time
        total_thinking_time += thinking_time

        if step_data is not None:
            steps.append((f"Step {step_count}: {step_data['title']}",
                step_data['content'], thinking_time))

            messages.append({"role": "assistant", "content": json.dumps(step_data)})

            if step_data['next_action'] == 'final_answer': # Maximum of 25 steps to
                prevent infinite thinking time. Can be adjusted.
                break

        if step_count == 25:
            break
        step_count += 1
```

4 pysolver

在本次实验中，我实现了一个简单的 test-time compute 方法，pysolver。其思路是将让 LLM 求解数学问题，换为让 LLM 写一个 python 程序来求解该问题，并运行该 python 程序来得到该问题的解。该方法的 prompt 如下：

> Write a Python program to solve the following math problem. You can call various library functions, including numpy, scipy, panda, torch, etc. The Python program needs to be able to output intermediate results and final results, and the program needs to have comments. Notice: Only output the code.

pysolver 的执行过程是生成代码后，从中提取出 python 代码的部分，建立一个 subprocess 执行代码（同时设置最大运行时间为 30 秒），如果代码运行出错则重新执行，最多重复 5 次。代码关键实现如下：

```
def execute_python_code(code: str, timeout: int = 30):
    open("tmp.py", "w").write(code)
    try:
        # Run the Python subprocess with a timeout
        result = subprocess.run(
            ["python", "tmp.py"],
            text=True,
            capture_output=True,
```

```
            timeout=timeout
        )
        return {
            "success": result.returncode == 0,
            "output": result.stdout if result.returncode == 0 else result.stderr
        }
    except subprocess.TimeoutExpired as e:
        return {
            "success": False,
            "output": f"Timeout: The script took longer than {timeout} seconds to
                execute."
        }
    except Exception as e:
        return {
            "success": False,
            "output": f"Error: {str(e)}"
        }
    finally:
        if os.path.exists("tmp.py"):
            os.remove("tmp.py")


def extract_python_code(text):
    code_blocks = re.findall(r"```python\n(.*?)```", text, re.DOTALL)
    return "\n\n".join(code_blocks)
```

5 实验效果比较

我们取 MATH500 的前 100 个问题作为测试集，比较 default, 3-shot, 5-shot, g1, pysolver 一共 5 种方法。测试结果如下：

	1-shot	3-shots	5-shots	g1	pysolver
Acc Rate	0.74	0.75	0.80	0.65	0.78

最后是在样例问题 “how many r in strawberry” 下，g1 和 pysolver 方法的对比测试。

```
(base) ytx24@h12:/flash2/aml/ytx24/newproj/AML/proj_4/o1$ python g1.py
Step 1: {'title': "Counting the Occurrences of 'r' in 'Strawberry'", 'content': "We need to count the number of times the letter 'r' appears in the word 'strawberry'. This is a straightforward task
 involving a simple examination of the word.", 'next_action': 'continue'}

Step 2: {'title': 'Identifying Key Information', 'content': "The problem is asking for the number of letter 'r' occurrences in the word 'strawberry'. This involves counting the specific letter with
in the given word.", 'next_action': 'continue'}

Step 3: {'title': 'Visual Inspection of the Word', 'content': "Upon visually examining the word 'strawberry', we can count the letter 'r' appearing twice in the sequence. This is a quick method tha
t does not require complex reasoning.", 'next_action': 'continue'}

Step 4: {'title': "Counting the Letter 'r' in 'Strawberry'", 'content': "The word 'strawberry' contains 8 letters. We will now count how many of these letters are the letter 'r'.", 'next_action': '
continue'}

Step 5: {'title': "Verifying Count of 'r'", 'content': "To ensure accuracy, I will now recount the letter 'r' in the word 'strawberry' to confirm that there are indeed two occurrences.", 'next_acti
on': 'continue'}

Step 6: {'title': "Visual Count of 'r' in 'Strawberry'", 'content': "By visually scanning the word 'strawberry', we can see that the letter 'r' appears twice. This is the count we need for our answ
er.", 'next_action': 'final_answer'}

step: ("Step 1: Counting the Occurrences of 'r' in 'Strawberry'", "We need to count the number of times the letter 'r' appears in the word 'strawberry'. This is a straightforward task involving a s
imple examination of the word.", 6.597428798675537)
step: ('Step 2: Identifying Key Information', "The problem is asking for the number of letter 'r' occurrences in the word 'strawberry'. This involves counting the specific letter within the given w
ord.", 1.8095853328704834)
step: ('Step 3: Visual Inspection of the Word', "Upon visually examining the word 'strawberry', we can count the letter 'r' appearing twice in the sequence. This is a quick method that does not req
uire complex reasoning.", 1.7937326431274414)
step: ("Step 4: Counting the Letter 'r' in 'Strawberry'", "The word 'strawberry' contains 8 letters. We will now count how many of these letters are the letter 'r'.", 1.6639165878295898)
step: ("Step 5: Verifying Count of 'r'", "To ensure accuracy, I will now recount the letter 'r' in the word 'strawberry' to confirm that there are indeed two occurrences.", 1.7290527820587158)
step: ("Step 6: Visual Count of 'r' in 'Strawberry'", "By visually scanning the word 'strawberry', we can see that the letter 'r' appears twice. This is the count we need for our answer.", 1.591603
5175323486)
step: ('Final Answer', "There are 2 occurrences of the letter 'r' in the word 'strawberry'.", 1.303633451461792)
total_time: 16.488953113555908
final_answer: There are 2 occurrences of the letter 'r' in the word 'strawberry'.
```

图 1: g1 测试结果

```
(base) ytx24@h12:/flash2/aml/ytx24/newproj/AML/proj_4/o1$ python pysolver.py
# Define the word to analyze
word = "strawberry"

# Initialize a counter for the letter 'r'
count_r = 0

# Loop through each character in the word
for char in word:
    # Check if the character is 'r'
    if char == 'r':
        # Increment the counter if it is
        count_r += 1

# Output the intermediate result
print(f"Number of 'r' in 'strawberry': {count_r}")

# Final result
final_result = count_r

========================================
Number of 'r' in 'strawberry': 3
```

图 2: pysolver 测试结果