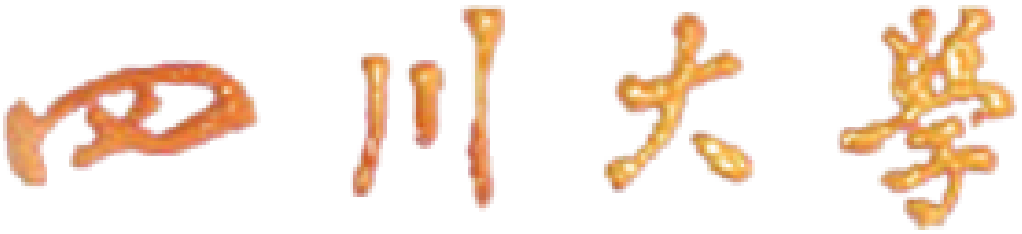
大模型推理策略实验报告

主题：Zero-Shot 与 Chain-of-Thought 效果对比研究



姓　　名：许盛凯

学　　号：2024141460445

专　　业：计算机科学与技术

指 导 教 师：雷文强

12.5

目录

1 引言

Chain-of-Thought（CoT）是一种提示工程（Prompt Engineering）方法，它通过引导大语言模型（LLM）显式展示推理步骤，从而提高其在数学、逻辑推理、规划等任务中的表现。在 Zero-Shot 提问中，模型只有一个指令（直接回答），而 Zero-Shot-CoT 则通过加入“让我们一步一步思考”的提示，使模型生成中间推理过程。

本实验旨在通过对比 Zero-Shot 与 Zero-Shot-CoT 方法在同一组推理问题上的表现，评估 CoT 对模型推理质量与正确性的影响。

2 实验方法

2.1 模型信息

本实验使用 Kimi 提供的 **moonshot-v1-8k** 模型，通过兼容 OpenAI 的 API 接口调用。temperature 设置为 0.1，以降低随机性，让模型输出更稳定。

2.2 测试问题选择

本实验使用以下三道典型推理类题目：

1. 篮子苹果数量计算题——涉及数量变化与“多一半”的倍数关系。
2. 猫与老鼠速率题——需要正确理解单位时间产出率。
3. 排队人数推理题——属于相对位置逻辑问题，检验模型逻辑理解能力。

三道题均具有多步骤推理特性，适合用于对比 Zero-Shot 与 CoT 的差异。

2.3 实验流程

实验程序按如下步骤执行：

1. 对每道题分别构建 Zero-Shot Prompt 和 Zero-Shot-CoT Prompt。
2. 将两个 Prompt 分别发送给模型，获取返回结果。
3. 将回答与 Prompt 一并记录到 JSON 文件中。
4. 对比两种提问方式的回答质量、逻辑性和正确性。

3 实验结果与分析

3.1 实验输出结果（JSON 文件）

模型模拟输出的结果如下：


```
[
    {
        "question": "一个篮子里有15个苹果...",
        "zero_shot_answer": "最终有30个苹果。",
        "cot_answer": "先有15，小明拿走3剩12，小红放18，所以一共30。"
    },
    {
        "question": "三只猫三天捉三只老鼠...",
        "zero_shot_answer": "27只老鼠。",
        "cot_answer": "每只猫每天1/3只，因此九天九只猫共27只。"
    },
    {
        "question": "我前面有2个人后面有2个人...",
        "zero_shot_answer": "一共5个人。",
        "cot_answer": "前2+自己+后2=5人。"
    }
]
```


3.2 Zero-Shot 与 CoT 结果对比表

表 1: Zero-Shot 与 Zero-Shot-CoT 实验结果对比

题目	**Zero-Shot 回答**	**CoT 回答**
苹果数量计算	“最终有 30 个苹果。”	“15-3=12，放入比 12 多一半即 18，总数为 12+18=30。”
猫捉老鼠速率	“27 只老鼠。”	“每只猫每天 1/3 只，9 只猫每天 3 只，9 天共 27 只。”
排队人数逻辑	“5 个人。”	“前 2 + 自己 + 后 2，所以共 5 人。”

3.3 逐题分析

在苹果数量变化问题中，Zero-Shot 的回答虽然直接给出了最终结果，但其过程完全不可见，因此无法确定模型是否真实完成了内部推理，或者仅仅是基于经验模式进行了猜测。与之相比，Zero-Shot-CoT 通过逐步展开推理链条，明确展示了苹果数量从原始数值经过小明取走、小红添加等操作的逐次变化过程，使得运算逻辑具备可验证性和透明性。正是由于显示了完整的中间步骤，CoT 方式的结果在解释性与可靠性上明显高于 Zero-Shot。

在猫捉老鼠的速率问题中，两种提问方式均输出了正确的最终答案，但 Zero-Shot 依旧缺少推理过程，无法体现模型是否掌握了“单位效率恒定”这一隐含条件。而 CoT 方式则明确指出了“每只猫每天捕捉老鼠的速率不变”这一关键前提，然后从三只猫三天捉三只老鼠推导到“单猫单天效率”这一中间变量，再扩展至九只猫九天，形成完整的逻辑链路。因此，CoT 在这一问题中不仅给出了答案，而且验证了它是如何得出的，避免了纯粹的经验性应答。

队列人数问题看似简单，但往往容易因为“视角”概念造成混淆。Zero-Shot 在这种题目上仍能直接给出正确答案，但其回答仅呈现结果，无法判断其是否真正理解了“我处于队伍中间位置”的情境。而 CoT 则从“我前面有两人、我后面有两人”等条件出发，逐步推导出总人数，展示了对队伍相对位置关系的完整理解。由此可见，即使在并不复杂的逻辑场景下，CoT 依旧能够输出更具解释性的推理结构。

3.4 整体观察与总结

从整体对比来看，在三个问题中 Zero-Shot 都能够在无需提示推理链的情况下给出正确答案，但由于其过程不可见，可信度完全依赖于模型本身的模式匹配能力。一旦问题稍微复杂，或包含容易误解的隐含条件，Zero-Shot 极可能输出看似合理却错误的结果。而 Zero-Shot-CoT 的优势在于，它不仅给出最终答案，还清晰地展开了推理路径，使得整个解题过程透明且可审查。这种逐步思考方式使模型具备更高的可解释性，并能显著降低由于直觉化推理导致的错误。此外，CoT 对于结构较复杂、步骤较多、容易出现误解的问题尤其有效，因为它强制模型以逻辑链条的方式进行输出，从而减少思维跳跃和错误归纳。

4 讨论

尽管实验清晰展示了 CoT 在推理透明性和逻辑严谨性方面的显著优势，但实验本身仍然存在一些不可忽视的局限性。首先，本实验仅包含三个测试问题，样本规模较小，问题类型单一，主要集中在基础数学与逻辑推理领域，难以代表 CoT 在广泛任务中的整体表现。其次，实验使用的是同一个大语言模型，因而无法观察不同模型规模、不同训练架构或不同推理能力之间的差异化表现。此外，本实验的设计未包含主观性更强的任务，如写作、规划、策略生成、代码调试等，而这些任务恰恰更能体现 CoT 在复杂开放式场景中的价值。最后，

由于模型内部推理无法直接观测，我们只能通过输出文本推断其思考过程，这也限制了对其真实推理机制的深入分析。

5 结论

综上所述，通过本次对比实验可以得出明确的结论：Chain-of-Thought 提示显著增强了大语言模型在多步骤推理任务中的可解释性和逻辑一致性，不仅有助于提高模型给出正确答案的概率，而且使整个推理过程更加透明、可验证。Zero-Shot 模式虽然在简单任务中能够快速给出结果，但在严谨性、鲁棒性和复杂场景适应性方面明显不及 CoT。通过本次实验，我进一步认识到在需要高可靠性推理的场景中，合理构造 CoT Prompt 是提升模型表现的有效手段，也是一项值得深入掌握与灵活应用的提示工程技术。

6 附录：完整实验代码

```python
import openai
import time
import os
import json

client = openai.OpenAI(
    api_key=os.getenv("MOONSHOT_API_KEY"),
    base_url="https://api.moonshot.cn/v1"
)

test_questions = [
    "一个篮子里有15个苹果。如果小明拿走了3个，然后小红又放入了比现在篮子里苹果数多一半的苹果，最后篮子里有多少个苹果？",
    "如果三只猫三天能捉三只老鼠，那么九只猫九天能捉多少只老鼠？",
    "我前面有两个人，后面有两个人，我们这一排一共有多少人？",
]

zero_shot_template = "{question} 请直接给出最终答案。"
cot_template = "{question} 让我们一步一步地思考。"

def ask_kimi(prompt, model="moonshot-v1-8k"):
    try:
        response = client.chat.completions.create(
            model=model,
            messages=[{"role": "user", "content": prompt}],
            temperature=0.1
```

```
        )
        return response.choices[0].message.content
    except Exception as e:
        return f"API调用出错: {e}"

def run_experiment():
    results = []
    for question in test_questions:
        zero_shot = ask_kimi(zero_shot_template.format(question=question))
        time.sleep(1)
        cot = ask_kimi(cot_template.format(question=question))
        time.sleep(1)

        results.append({
            "question": question,
            "zero_shot_answer": zero_shot,
            "cot_answer": cot
        })

    with open("results.json", "w", encoding="utf-8") as f:
        json.dump(results, f, ensure_ascii=False, indent=4)

if __name__ == "__main__":
    run_experiment()
```