

# Chain-of-Thought 对比实验报告

## 基础信息

- 课程名称：<课程名称>
- 姓名：<姓名>
- 学号：<学号>
- 实验时间：2025-12-15T05:18:01.563773

## 引言

Chain-of-Thought (CoT) 提示通过要求模型逐步给出推理过程，可以帮助大模型显式梳理复杂问题的中间逻辑。与直接提问相比，CoT 在数学、逻辑、代码调试等任务中经常能显著提升正确率。

本实验目的：通过对比Zero-Shot与Zero-Shot-CoT，验证Chain-of-Thought提示在数学/逻辑推理题目上的增益。

## 实验方法

- 模型：gpt-4o-mini (temperature=0.1)
- 提问方式：同一问题分别使用 Zero-Shot 与 Zero-Shot-CoT 两种提示。
- 测试集：至少 3 道需要多步推理的问题，覆盖算术、比例与空间关系。
- 实现：cot\_experiment.py 顺序调用 API，记录完整回答并判定对错。

## 结果与分析

本次共测试 3 道题。Zero-Shot 正确 1 次，准确率 33%；CoT 正确 3 次，准确率 100%。CoT 在样本中实现了更稳定的正确率。

问题 1 分析：多步算术题，容易在'比现在多一半'的语义上出错。

问题 1: 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？

- Zero-Shot 提示: 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？请直接给出最终答案。

- Zero-Shot 回答: 15个苹果拿走3个剩12个。比现在多一半理解为在12的基础上再增加一半(6个)，所以最后是18个苹果。

- Zero-Shot 是否正确: False

- CoT 提示: 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？让我们一步一步地思考。

- CoT 回答: 我们一点点来：1) 原来15个，拿走3个还剩12个；2) "比现在多一半"表示增加的数量是当前数量的1.5倍，即 $12 \times 1.5 = 18$ 个新苹果被放入；3) 篮子里已有12个，再加18个得到30个。因此最终有30个苹果。

- CoT 是否正确: True

问题 2 分析：比例推理题，考察单位速率是否被正确抽象。

问题 2: 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？

- Zero-Shot 提示: 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？请直接给出最终答案。

- Zero-Shot 回答: 三只猫三天捉三只，说明一只猫三天捉一只，相当于每天三分之一只。九只猫九天就是 $9 \times 9/3 = 27$ ，所以是27只。

- Zero-Shot 是否正确: True

- CoT 提示: 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？让我们一步一步地思考。

- CoT 回答: 拆解：一只猫三天=1只鼠 → 单位速率  $1/3$  只/天。九只猫每天捕到  $9 \times 1/3 = 3$  只。九天共  $3 \times 9 = 27$  只老鼠。答案：27。

- CoT 是否正确: True

问题 3 分析：空间关系题，Zero-Shot常把答案误判为4。

问题 3: 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 提示: 我前面有两个人，后面有两个人，我们这一排一共有多少人？请直接给出最终答案。

- Zero-Shot 回答: 前面2个后面2个，所以连同我一共4个人。

- Zero-Shot 是否正确: False

- CoT 提示: 我前面有两个人，后面有两个人，我们这一排一共有多少人？让我们一步一步地思考。

- CoT 回答: 排成一列：X 表示我。前面有两个人=A,B；后面有两个人=C,D。队形是 A,B,X,C,D，总人数 5。

- CoT 是否正确: True

## 讨论与局限

- 1) 题目数量有限，无法覆盖更复杂的推理。
- 2) 正确性判定仅基于字符串匹配，未融入人类评审。
- 3) 实验依赖单一模型配置，尚未比较不同模型的表现。

## 结论

在给定样本中，Zero-Shot-CoT 始终给出正确答案，而直接提问在第1、第3题出现错误。CoT 提示通过显式推理步骤帮助模型纠正歧义，显示出在多步推理任务中的一致优势。

## 附录：完整代码

```
"""Zero-Shot vs Zero-Shot-CoT experiment runner.
```

```
This script calls an LLM twice per question: once with a direct (zero-shot) prompt, and once with an additional Chain-of-Thought cue. Results are logged to per-run folders so they can be used directly in reports.
```

```
"""
```

```
from __future__ import annotations
```

```

import argparse
import dataclasses
import datetime as dt
import json
import os
import sys
import time
from pathlib import Path
from typing import Any, Dict, Iterable, List, Optional

try: # Import lazily so that unit tests without openai installed still run.
    from openai import OpenAI
except Exception: # pragma: no cover - best effort fallback during dev.
    OpenAI = None # type: ignore

@dataclasses.dataclass
class QuestionSpec:
    """Container for a single evaluation question."""

    prompt: str
    ground_truth: str
    rationale: str

DEFAULT_QUESTIONS: List[QuestionSpec] = [
    QuestionSpec(
        prompt=(
            "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？"
        ),
        ground_truth="30",
        rationale="多步算术题，容易在'比现在多一半'的语义上出错。",
    ),
    QuestionSpec(
        prompt="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？",
        ground_truth="27",
        rationale="比例推理题，考察单位速率是否被正确抽象。",
    ),
    QuestionSpec(
        prompt="我前面有两个人，后面有两个人，我们这一排一共有多少人？",
        ground_truth="5",
        rationale="空间关系题，Zero-Shot常把答案误判为4。",
    ),
]

def parse_args(argv: Optional[Iterable[str]] = None) -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description=(
            "运行 Zero-Shot 与 Zero-Shot-CoT 对比实验，自动保存原始回答与评估"
        )
    )
    parser.add_argument("--model", default="gpt-4o-mini", help="要调用的模型名")

```

```
parser.add_argument(
    "--temperature", default=0.1, type=float, help="采样温度 (0-2)"
)
parser.add_argument(
    "--cot-suffix",
    default="让我们一步一步地思考。",
    help="附加在问题末尾的CoT提示语",
)
parser.add_argument(
    "--zero-shot-suffix",
    default="请直接给出最终答案。",
    help="附加在Zero-Shot prompt后的补充语",
)
parser.add_argument(
    "--max-retries",
    type=int,
    default=3,
    help="API失败时的最大重试次数",
)
parser.add_argument(
    "--retry-wait",
    type=float,
    default=3.0,
    help="重试之间等待的秒数 (指数退避起点)",
)
parser.add_argument(
    "--pause",
    type=float,
    default=1.0,
    help="两次提问之间的等待秒数，避免触发限速",
)
parser.add_argument(
    "--output-dir",
    default="outputs",
    help="存放实验结果的根目录 (若不存在将自动创建)",
)
parser.add_argument(
    "--questions-file",
    help="自定义问题 JSON 文件，结构符合 QuestionSpec",
)
parser.add_argument(
    "--base-url",
    default=None,
    help="可选：自定义API base url (硅基流动、代理等)",
)
parser.add_argument(
    "--replay-file",
    help="使用离线结果复现实验，跳过真实API调用",
)
parser.add_argument(
    "--dry-run",
    action="store_true",
    help="使用内置模板生成假数据，用于调试流程",
)
```

```

parser.add_argument(
    "--save-markdown",
    action="store_true",
    help="额外导出一份Markdown格式的对比表",
)
parser.add_argument(
    "--tag",
    default=None,
    help="自定义run标签，方便对应实验记录",
)
return parser.parse_args(list(argv) if argv is not None else None)

```

```

def load_questions(path: Optional[str]) -> List[QuestionSpec]:
    if not path:
        return DEFAULT_QUESTIONS
    file_path = Path(path)
    data = json.loads(file_path.read_text("utf-8"))
    questions: List[QuestionSpec] = []
    for item in data:
        questions.append(
            QuestionSpec(
                prompt=item["prompt"],
                ground_truth=item.get("ground_truth", ""),
                rationale=item.get("rationale", ""),
            )
        )
    return questions

```

```

def build_client(base_url: Optional[str] = None) -> Any:
    api_key = os.getenv("OPENAI_API_KEY")
    if not api_key:
        raise EnvironmentError(
            "OPENAI_API_KEY 未设置。请通过环境变量配置后再运行。"
        )
    if OpenAI is None:
        raise RuntimeError(
            "openai SDK 未安装。请运行 'pip install openai' 后重试。"
        )
    return OpenAI(api_key=api_key, base_url=base_url)

```

```

def run_chat_completion(
    client: Any,
    model: str,
    prompt: str,
    temperature: float,
    max_retries: int,
    retry_wait: float,
) -> str:
    attempt = 0
    wait = retry_wait
    last_error: Optional[Exception] = None

```

```

while attempt <= max_retries:
    try:
        response = client.chat.completions.create(
            model=model,
            messages=[{"role": "user", "content": prompt}],
            temperature=temperature,
        )
        return response.choices[0].message.content # type: ignore[index]
    except Exception as exc: # pragma: no cover - network failure path
        last_error = exc
        if attempt == max_retries:
            break
        sleep = wait * (2**attempt)
        print(
            f"API 调用失败 (attempt {attempt + 1}/{max_retries + 1}): {exc}\n"
            f"等待 {sleep:.1f}s 后重试...",
            file=sys.stderr,
        )
        time.sleep(sleep)
        attempt += 1
    raise RuntimeError(f"API 调用失败: {last_error}")

```

```

def normalize_text(s: str) -> str:
    return "".join(ch.lower() for ch in s if ch.isalnum())

```

```

def is_answer_correct(answer: str, ground_truth: str) -> Optional[bool]:
    if not ground_truth:
        return None
    normalized_truth = normalize_text(ground_truth)
    normalized_answer = normalize_text(answer)
    if not normalized_truth:
        return None
    return normalized_truth in normalized_answer

```

```

def build_prompt(question: str, suffix: str) -> str:
    suffix = suffix.strip()
    if suffix and not question.endswith(suffix):
        return f"{question.strip()} {suffix}".strip()
    return question.strip()

```

```

def load_replay_data(path: str) -> Dict[str, Dict[str, str]]:
    data = json.loads(Path(path).read_text("utf-8"))
    lookup: Dict[str, Dict[str, str]] = {}
    for item in data.get("records", []):
        lookup[item["question"]] = {
            "zero_shot": item["zero_shot_answer"],
            "cot": item["cot_answer"],
        }
    return lookup

```

```

def simulate_answer(question: QuestionSpec, mode: str) -> str:
    # Deterministic mock response for dry-run workflows.
    intro = "[模拟回答]"
    if mode == "cot":
        return (
            f"{intro} 让我们一步一步地分析：\n1. 问题：{question.prompt}\n"
            f"2. 推理……所以答案是 {question.ground_truth or '未知'}。"
        )
    return f"{intro} {question.prompt} => {question.ground_truth or '未知'}"

def now_stamp() -> str:
    return dt.datetime.now().strftime("%Y%m%d-%H%M%S")

def ensure_dir(path: Path) -> None:
    path.mkdir(parents=True, exist_ok=True)

def format_markdown(results: List[Dict[str, Any]], metadata: Dict[str, Any]) -> str:
    lines = ["# Zero-Shot vs Zero-Shot-CoT", "", "## 元信息", ""]
    for key, value in metadata.items():
        lines.append(f"- **{key}**: {value}")
    lines.extend(["", "## 结果对比", ""])
    header = (
        "| 序号 | 问题 | Zero-Shot 摘要 | CoT 摘要 | Zero-Shot 正确 | CoT 正确 |"
    )
    lines.append(header)
    lines.append("| --- | --- | --- | --- | --- | --- |")
    for idx, item in enumerate(results, start=1):
        zero = item["zero_shot_answer"].replace("\n", "<br>")
        cot = item["cot_answer"].replace("\n", "<br>")
        zero_ok = item.get("zero_shot_correct")
        cot_ok = item.get("cot_correct")
        lines.append(
            f"| {idx} | {item['question']} | {zero} | {cot} | {zero_ok} | {cot_ok} |"
        )
    return "\n".join(lines)

def save_results(
    results: List[Dict[str, Any]],
    output_dir: Path,
    metadata: Dict[str, Any],
    save_markdown: bool,
) -> Path:
    ensure_dir(output_dir)
    run_folder = output_dir / metadata["run_id"]
    ensure_dir(run_folder)
    result_path = run_folder / "results.json"
    payload = {"metadata": metadata, "records": results}
    result_path.write_text(json.dumps(payload, ensure_ascii=False, indent=2), "utf-8")
    if save_markdown:

```

```

md_path = run_folder / "results.md"
md_path.write_text(format_markdown(results, metadata), "utf-8")
return run_folder

```

```

def main(argv: Optional[Iterable[str]] = None) -> None:
    args = parse_args(argv)
    questions = load_questions(args.questions_file)
    replay_lookup = load_replay_data(args.replay_file) if args.replay_file else {}
    client = None
    if not args.dry_run and not replay_lookup:
        client = build_client(args.base_url)
    run_id = args.tag or now_stamp()
    results: List[Dict[str, Any]] = []
    for idx, question in enumerate(questions, start=1):
        print(f"\n=== 问题 {idx}: {question.prompt}")
        zero_prompt = build_prompt(question.prompt, args.zero_shot_suffix)
        cot_prompt = build_prompt(question.prompt, args.cot_suffix)

        if replay_lookup:
            replay = replay_lookup.get(question.prompt)
            if not replay:
                raise KeyError(f"replay 数据中缺少问题: {question.prompt}")
            zero_answer = replay["zero_shot"]
            cot_answer = replay["cot"]
        elif args.dry_run:
            zero_answer = simulate_answer(question, "zero")
            cot_answer = simulate_answer(question, "cot")
        else:
            zero_answer = run_chat_completion(
                client,
                args.model,
                zero_prompt,
                args.temperature,
                args.max_retries,
                args.retry_wait,
            )
            time.sleep(args.pause)
            cot_answer = run_chat_completion(
                client,
                args.model,
                cot_prompt,
                args.temperature,
                args.max_retries,
                args.retry_wait,
            )
        zero_correct = is_answer_correct(zero_answer, question.ground_truth)
        cot_correct = is_answer_correct(cot_answer, question.ground_truth)
        result_entry = {
            "question": question.prompt,
            "rationale": question.rationale,
            "ground_truth": question.ground_truth,
            "zero_shot_prompt": zero_prompt,
            "cot_prompt": cot_prompt,

```



```

        "zero_shot_answer": zero_answer,
        "cot_answer": cot_answer,
        "zero_shot_correct": zero_correct,
        "cot_correct": cot_correct,
    }
    results.append(result_entry)
    print("- Zero-Shot 是否正确:", zero_correct)
    print("- CoT 是否正确:", cot_correct)
    time.sleep(args.pause)

metadata = {
    "run_id": run_id,
    "model": args.model,
    "temperature": args.temperature,
    "cot_suffix": args.cot_suffix,
    "zero_shot_suffix": args.zero_shot_suffix,
    "question_count": len(questions),
    "timestamp": dt.datetime.now().isoformat(),
}
if args.replay_file:
    metadata["replay_file"] = args.replay_file
output_dir = save_results(
    results,
    Path(args.output_dir),
    metadata,
    save_markdown=args.save_markdown,
)
print(f"\n实验完成，结果已保存至: {output_dir}")

if __name__ == "__main__":
    main()

```