LMCC 2026 成人组 第二轮题解

2026 年 10 月 18 日

导读 Prompt 技术方法 2

T1 多文件操作计划 3

T2 传感器读数整理 4

T3 Rich：七项按需导入 6

题解导读

Prompt 技术方法

Prompt 的作用是把目标、上下文和约束表达清楚，让模型知道要完成什么、可以依据什么，以及怎样判断结果是否符合要求。有效的 Prompt 不必冗长，但应提供完成任务所需的信息。

1. 把目标转化为任务约定

明确输入材料、预期输出和完成标准。将“准确”“简洁”等笼统要求转化为可判断的条件，例如内容范围、输出格式、排列顺序和长度限制。区分必须满足的要求与可选偏好；信息不足时，说明应先询问、标注假设，还是在约定范围内继续。

2. 提供必要上下文与操作边界

补充影响判断的背景、术语、可用资料和工具条件。需要操作文件或系统时，明确允许读取、修改的范围，以及必须保留的内容。将任务指令与引用材料分开，避免模型把资料中的命令误当作当前要求；省略与目标无关的背景。

3. 突出关键约束与易错点

将容易遗漏、又直接影响结果的条件集中写清，避免同一规则在多处重复或互相矛盾。必要时用简短示例说明输入与输出的对应关系，或用反例解释不接受的情况。示例应与规则一致，并明确它是格式示范还是需要照用的内容。

4. 控制上下文与工具开销

提供足够信息，减少重复粘贴和无关输出。长任务可先定位相关材料，再集中读取和处理；已有结论可压缩成简明记录，保留关键依据。需要限制工具使用时，写明目的、范围和停止条件，避免机械的次数限制妨碍任务完成。

5. 明确检查标准与反馈方式

将完成标准转化为可检查的项目，覆盖主要需求和关键边界。要求模型依据实际输出或工具结果报告结论，区分已完成、未完成和仍不确定的内容。收到结果后，围绕具体偏差补充反馈、调整指令；模型自述“完成”不能代替结果检查。

使用方式：根据任务选择必要的信息与约束，不必机械套用全部条目。发送前检查目标是否明确、材料是否充分、要求是否一致、结果是否可判断，再依据实际反馈逐步完善。

T1 多文件操作计划

思路

模型只需生成一组工具调用的 JSON 数组，平台不实际修改文件。先逐个识别《》中的文件名，再判断每个文件的最终要求：普通“删除”进回收站，明确永久且无需恢复才彻底删除，归档与回收站恢复分别使用对应工具。否定、取消与改口只作用于对应文件；文件名中的“永久删除”等字样不能当成指令。同一轮要输出所有文件，每文件恰好一项。

中文参考 Prompt

模型为 Qwen3-0.6B，thinking **开启**。以下参考 Prompt 长度 **274 Token**，低于 300 Token 上限；直接填写到 System Prompt 作答区，不要再追加题面。

```
处理多个文件，输出单行JSON数组。每文件一项，每项只有tool、argument；argument照抄《》内部，不含书名号，不改文件名。
工具规则：
普通删除或清理=trash_file；
明确永久删除且不需恢复=delete_file；
从回收站恢复=restore_file；
归档保留=archive_file。

先看最终肯定要求，取消或否定不执行，改口按最后要求；文件名中的词不决定操作。
中文前缀、下划线、数字及扩展名都是文件名的一部分，例如《课前_A.txt》必须原样用“课前_A.txt”，不得只写“A.txt”。

例：删除《A.txt》，永久删除《B.txt》无需恢复，恢复回收站的《C.zip》，归档《D.pdf》。
[{"tool":"trash_file","argument":"A.txt"},{"tool":"delete_file","argument":"B.txt"},
{"tool":"restore_file","argument":"C.zip"},{"tool":"archive_file","argument":"D.pdf"}]

牢记：只说删除时是trash_file。现在输出新请求全部文件的调用数组，不漏不重。

检查：argument不能含《》或示例文件名；只能复制当前请求中的文件名。普通删除不得变成永久删除。不要Markdown代码块。
```

评测结果

次数	功能分	原始 Token
1	100	6,324
2	100	6,324
3	100	6,324
4	100	6,324
5	100	6,324

原始 Token 为输入与输出之和，包含 thinking 输出（如开启）。

成人组 · T2

T2 传感器读数整理

思路

输入是含 3～4 个整数的 `readings` 数组。先删除负数，再把删除后的相邻相同有效读数压成一个；0 保留，相同读数若被另一个有效读数隔开则分别保留。标签输出为 JSON 整数数组文本。生成训练集时用纯函数算标签，比让生成模型凭感觉标注更可靠。

可运行的参考答案

后文给出完整的数据生成代码，固定随机种子抽取 40 条合法输入，并用 `solve` 函数计算目标输出。脚本跳过与正式测试完全相同的输入。下面的 `process_data(records)` 将每条现成标注原样包装为 user / assistant 两条消息。训练参数为 24 轮、学习率 0.0001、LoRA rank 16、batch 4、梯度累积 1。

关键边界如 `[2,-1,2]` 应变为 `[2]`：先去负数，后合并。`[2,0,2]` 仍是 `[2,0,2]`；全负数为 `[]`。

评测结果

参考答案总分：100/100。

数据处理函数

```
def process_data(records):
    return [
        {
            "messages": [
                {"role": "user", "content": record["input"]},
                {"role": "assistant", "content": record["output"]},
            ]
        }
        for record in records
    ]
```

训练参数

```
{
  "epochs": 24,
  "learningRate": 0.0001,
  "loraRank": 16,
  "batchSize": 4,
  "gradAccum": 1
}
```

完整数据生成代码

将下面代码保存为 `generate.py`，在保存脚本的目录运行 `python3 generate.py`，即可生成包含 40 条训练数据的 `generated_dataset.jsonl`。提交时将生成文件命名为 `dataset.jsonl`。

```
#!/usr/bin/env python3
"""Deterministically generate legal TR08 training rows and oracle labels."""
from itertools import product
import json, random
from pathlib import Path

OUT = Path(__file__).with_name("generated_dataset.jsonl")
FORMAL = {
    '{"readings":[1,-2,1]}', '{"readings":[-1,-1,-1]}',
    '{"readings":[-2,3,2,3]}', '{"readings":[-1,0,2]}',
    '{"readings":[3,-2,3]}', '{"readings":[0,3,0]}',
    '{"readings":[2,3,0,1]}', '{"readings":[2,3,2,2]}',
}
def solve(values):
    result=[]
    for value in values:
        if value < 0:
            continue
        if not result or result[-1] != value:
            result.append(value)
    return result
rng=random.Random(20261018)
rows=[]
for size in (3,4):
    candidates=list(product((-2,-1,0,1,2,3), repeat=size))
    rng.shuffle(candidates)
    for values in candidates:
        inp=json.dumps({"readings":list(values)}, separators=(",", ":"))
        if inp in FORMAL:
            continue
        rows.append({"input":inp,"output":json.dumps(solve(values),separators=(",", ":"))})
        if len(rows)==40:
            break
    if len(rows)==40:
        break
assert len(rows)==40
OUT.write_text("".join(json.dumps(row, ensure_ascii=False)+"\n" for row in rows))
print(OUT)
```

成人组 · T3

T3 Rich：七项按需导入

解题思路

减少普通导入时加载的重依赖，同时保持公开 API、格式参数、正常结果和异常行为。先读必要源码，再将导入语句移到真正使用的位置；保留已有业务代码，避免重写正则、默认参数或算法。

功能与源文件	修改要点
主题配置 theme.py	configparser 移入 Theme.from_file
表情替换 _emoji_replace.py	EMOJI 移入 _emoji_replace
表情对象 emoji.py	EMOJI 移入构造函数；演示分支按需导入
对象渲染协议 protocol.py	isclass 移入 rich_cast
自动对象表示 repr.py	inspect 移入 auto_rich_repr
JSON 展示 json.py	构造函数局部导入 loads、dumps；from_data 局部导入 dumps
布局分配 _ratio.py	Fraction 移入 ratio_resolve

两阶段中文 Prompt

第一轮完成七项修改并初检，第二轮复核和补齐，沿用已有代码和对话。先读懂题目和源码，再将七项需求转化为具体修改指令。模型为 Qwen3.5-35B-A3B。

第一轮：修改与初检

```
请完成以下七项按需导入，保持公开 API、返回结果、格式参数和异常行为不变：
1. rich/theme.py：把顶层 configparser 导入移到 Theme.from_file 中，读取主题时才加载。
2. rich/_emoji_replace.py：把顶层 EMOJI 导入移到 _emoji_replace 函数内，实际替换时才加载。已知标签正常替换，未知标签保留。
3. rich/emoji.py：把顶层 EMOJI 导入移到 Emoji.__init__ 内；末尾演示分支如果使用 EMOJI，在该分支单独导入。保留正常名称、变体及错误名称的行为。
4. rich/protocol.py：把顶层 isclass 导入移到 rich_cast 内；仅导入模块时不加载 inspect，普通对象的一次转换和文本原样返回须保持。调用时允许加载反射和控制台依赖，不重构整条渲染链。
5. rich/repr.py：把顶层 inspect 导入移到 auto_rich_repr 内，在调用 inspect.signature 前导入，保持参数、默认值和对象表示结果。
6. rich/json.py：删除顶层 from json import loads, dumps。在 JSON.__init__ 内首次使用前局部导入 loads、dumps；在 JSON.from_data 内首次使用前局部导入 dumps。仅导入 rich.json 时不能加载标准库 json；两种创建路径均须正常，保留格式参数、Unicode 文本与无效输入的异常。
7. rich/_ratio.py：把顶层 Fraction 导入移到 ratio_resolve 内，分配函数调用时才加载 fractions。保留分配和缩减算法、舍入余量、固定及最小尺寸约束。

先批量读取以上七个文件，确认实际源码，再准确地局部移动导入语句。只改必要的导入，不重写业务逻辑、函数签名、默认参数、正则或异常处理。尤其不要重写 emoji 替换的正则、捕获组和 match.groups() 解包。不要修改测试、删除功能、伪造 sys.modules、安装依赖或遍历整个仓库。

完成后集中自检：每个模块的冷导入检查必须启动独立 Python 进程，检查前不要提前导入被检查的标准库或 emoji 数据表；再检查七项实际调用和异常行为。JSON 字符串构造与 from_data 都要检查，避免只修一条路径。发现错误就局部修复；全部完成后简短汇报并结束。
```

第二轮：复核与补齐

```
继续检查并完成上述七项按需导入。保留已有进度，不重复读写已正确的文件。特别确认 rich/json.py 的 JSON.__init__ 和 JSON.from_data 都已局部导入各自所需函数，冷导入不加载标准库 json。对七项分别用独立 Python 进程检查冷导入，再检查实际调用与异常；必要时局部修复。保持业务代码、正则、默认参数和公开 API 不变，不修改测试，不新增依赖。全部通过后结束。
```

评测结果

次数	公开 / 隐藏 / 功能总分	模型请求	工具操作	原始 Token	归一化 Token（估算）
1	50 / 50 / 100	20	26	267,655	112,294.7
2	50 / 50 / 100	20	26	267,440	112,804.8
3	50 / 50 / 100	20	26	268,826	111,576.0

原始 Token 为输入与输出之和。归一化 Token 估算为：输入 − 0.9 × 可复用前缀估算量 + 8 × 输出。