LMCC-2025 第二轮试题解析 - 成人组

重要说明

在考试过程中，考生只能看到 data 文件夹中 **不含 .final 后缀** 的自测数据文件：

- `test_data_T1.jsonl` - T1 题目的自测数据（5个学生）
- `test_data_T2.jsonl` - T2 题目的自测数据（6道题）
- `similarity_test_T2.jsonl` - T2 相似度测试的自测数据（20个查询）

最终评测会使用**含 .final 后缀**的测试数据（如 `test_data_T1.final.jsonl`），考生在考试期间**无法看到这些文件**。

由于最终测试数据与自测数据不同，考生**不应在 Prompt 中包含自测数据的具体内容**。合理的 Prompt 设计应该：

- ✅ 具有**通用性**，能够处理各种不同格式的输入
- ✅ 包含**多样化的示例**，而非仅针对自测数据
- ✅ 强调**核心方法和步骤**，而非死记硬背特定答案
- ❌ 避免针对自测数据进行过度优化
- ❌ 避免在 Prompt 中写死自测数据中出现的特定数字或格式

这样才能确保解决方案在最终评测中也能获得良好的表现。

考题概览

本次认证考试包含两道题目，均考察大语言模型的提示工程（Prompt Engineering）能力和实际应用能力。

- **T1：数学作业批改助手** - 考察模型的文本解析、答案提取和判断能力（50分）
- **T2：数学作业解答助手** - 考察向量检索、RAG系统实现和结构化输出能力（50分）

总分：100分

T1：数学作业批改助手

题目背景

实现一个智能数学作业批改助手，能够：

1. 读取学生作业内容（5道数学题）
2. 识别每道题的学生答案
3. 判断答案正确性（与标准答案比对）
4. 输出标准JSON格式的批改结果

任务要求

需要在 `submission.py` 中实现两个函数：

1. **build_system_prompt()** - 定义 system prompt，引导模型理解批改任务
2. **build_generation_parameters()** - 配置模型生成参数

题目难点

1. **答案格式多样化**：学生答案包含大量干扰信息
   - 示例：`"47（这个太简单了吧）"` → 核心答案：`47`
   - 示例：`"我觉得是325？刚才同桌说337我不信😤"` → 核心答案：`325`
2. **精确的答案比对**：需要从文本中提取核心数字并与标准答案比对
3. **标准JSON输出**：严格的输出格式要求

代码块

```
{
  "student_id": "student_001",
  "judgements": [true, false, true, true, false]
}
```

解题思路

1. System Prompt 设计

核心要点：

- **明确任务定位**：让模型理解自己是"数学作业批改助手"
- **清晰的输入输出说明**：通过具体示例展示输入格式和输出格式
- **提供完整的示例**：包含学生ID、题目、答案（带干扰信息）的完整示例
- **强调执行步骤**：
  a. 识别题目和学生答案
  b. 计算标准答案并与学生答案比对
  c. 严格按照JSON格式输出

参考实现：

代码块

```
def build_system_prompt() -> str:
    return """
你是一个数学作业批改任务的高级助手，拥有超强的判断作业是否正确的能力。
这个任务的核心在于提炼和比对关键信息。

## 输入格式：
    - 输入包含学生id和该学生的题目答案
    - [提供具体的输入案例]

## 输出格式：
    - 标准JSON格式
    - 包含student_id、judgements两个字段
    - [提供具体的输出格式说明]

## 难点说明：
    - 答案格式多样化，需要提取核心数字
    - 包含干扰信息（如表情、文字说明等）
    - 准确识别学生答案，准确计算标准答案
    - 精准比对答案，避免判断出错

## 执行步骤：
    1、获得题目信息，精准识别出学生题目和答案
    2、准确计算标准答案，将学生答案与标准答案比对
    3、严格按照输出格式进行输出
"""
```

2. 生成参数配置

关键参数推荐：

- **enable_thinking: True** - **关键设置！**开启思考模式让模型显式展示推理过程
- **max_new_tokens: 1024** - 给予足够的生成空间，确保能完整输出JSON和推理过程

关键技巧

1. **Thinking模式的重要性：**
    - 让模型显式展示"识别答案 → 计算标准答案 → 比对判断"的完整思考链
    - 大幅提高答案准确率，减少幻觉
2. **Prompt设计技巧：**
    - 使用结构化的Prompt（输入格式、输出格式、难点说明、执行步骤）

- 提供完整的示例（包含干扰信息的真实案例）
- 明确强调"精准比对"，避免模型产生幻觉

3. **答案提取提示**：
   - 强调提取"核心数字"而非完整文本
   - 提醒模型忽略表情符号、文字说明等干扰信息

评分标准

- **每道题 2 分，共 50 分**（5个学生 × 5道题 × 2分）
- 评测方式：模型输出的 `judgements` 数组与期望结果完全匹配才得分
- 必须满足：
  a. JSON格式正确
  b. 包含 `student_id` 和 `judgements` 两个字段
  c. `judgements` 包含5个布尔值
  d. 所有判断与期望一致

T2：数学作业解答助手

题目背景

实现一个基于 **RAG（检索增强生成）** 的数学解题助手：

1. 使用 Embedding 模型从题库（100道题）中检索最相似的题目
2. 参考这些相似题目的解法来解答当前问题
3. 输出包含解题思路和最终答案的标准JSON格式

任务要求

需要在 `submission.py` 中实现五个函数：

1. **compute_similarity()**- 计算余弦相似度（**单独评测，20分**）
2. **retrieve_relevant_problems()** - 从题库检索最相似的题目
3. **build_user_message()** - 组装用户消息（包含检索内容和当前题目）
4. **build_system_prompt()** - 定义 system prompt（RAG版本）
5. **build_generation_parameters()** - 配置生成参数

题目难点

1. **余弦相似度实现**：正确实现向量相似度计算。
2. **有效的检索**：从100道题中检索到真正相关的题目
3. **消息组装**：合理组织检索到的相似题目信息
4. **RAG Prompt设计**：引导模型学习参考例子的解题方法
5. **结构化输出**：输出标准JSON格式，answer字段只包含数字

解题思路

1. 余弦相似度计算（20分）

关键点：必须完整实现余弦相似度公式

余弦相似度计算向量之间的夹角余弦值：

代码块

```
def compute_similarity(query_embedding: torch.Tensor, doc_embedding:
torch.Tensor) -> float:
    # 完整的余弦相似度公式
    dot_product = query_embedding @ doc_embedding
    norms = query_embedding.norm() * doc_embedding.norm()
    similarity = dot_product / norms
    return similarity.item()
```

公式说明：

- 完整公式：`cos(θ) = (A · B) / (||A|| * ||B||)`
- `A · B`：向量点积
- `||A||`：向量A的L2范数（模长）
- `||B||`：向量B的L2范数（模长）

或者使用 PyTorch 内置函数：

代码块

```
import torch.nn.functional as F

def compute_similarity(query_embedding: torch.Tensor, doc_embedding:
torch.Tensor) -> float:
    return F.cosine_similarity(query_embedding, doc_embedding, dim=0).item()
```

2. 检索函数实现

实现步骤：

代码块

```python
def retrieve_relevant_problems(query, problem_bank, embedding_model,
                               tokenizer, top_k=3, get_embedding_func=None):
    # 步骤1：获取查询的embedding
    query_embedding = get_embedding_func(query, embedding_model, tokenizer)

    # 步骤2：计算所有题目的embedding
    problem_embeddings = []
    for prob in problem_bank:
        emb = get_embedding_func(prob['problem'], embedding_model, tokenizer)
        problem_embeddings.append(emb)

    # 步骤3：计算相似度
    similarities = []
    for prob_idx, prob_emb in enumerate(problem_embeddings):
        sim = compute_similarity(query_embedding, prob_emb)
        similarities.append((sim, prob_idx))

    # 步骤4：排序并返回top_k
    similarities.sort(key=lambda x: x[0], reverse=True)

    retrieved = []
    for i in range(top_k):
        retrieved.append(problem_bank[similarities[i][1]])
    return retrieved
```

3. 用户消息组装

设计思路：将检索到的相似题目清晰地展示给模型：

代码块

```python
def build_user_message(problem: str, retrieved_problems: List[Dict]) -> str:
    user_msg = f"当前要解答的题目是{problem}，从题库中检索到相似的题目列表内容是：\n"

    for i, prob in enumerate(retrieved_problems, 1):
        relevant_problem_item = (
            f"  {i}. 题目 ID={prob['id']}: {prob['problem']}。"
            f"计算过程：{prob['explanation']}。"
            f"正确答案：{prob['answer']}。\n"
        )
        user_msg = f"{user_msg} {relevant_problem_item}"

    return user_msg
```

要点：

- 明确标识"当前题目"和"参考题目"
- 为每个参考题目提供完整信息（题目、解析、答案）
- 使用清晰的格式便于模型理解

4. System Prompt 设计（RAG版本）

与T1的区别：

- 强调"参考检索到的相似题目"
- 引导模型学习相似题目的解题方法
- 输出包含 `reasoning` 和 `answer` 两个字段

参考实现：

代码块

```
def build_system_prompt() -> str:
    return """你是一名高级的小学数学助手。非常擅长做数学推理计算，
    在计算过程中需要明确输出思考链路。

    ## 输入格式：
        一个数学题目的字符串
        比如：一个数除以5商是12余3，这个数是多少？

    ## 输出格式：
        必须以 JSON 格式输出，包含 reasoning 和 answer 两个字段
        比如：{
            "reasoning": "根据除法关系，被除数 = 除数 × 商 + 余数。
                         这个数 = 5 × 12 + 3 = 60 + 3 = 63",
            "answer": "63"
        }

    ## 难点：
        精确给出正确答案，答案必须只包含数字，不包含单位
        严格按照输出格式进行输出

    ## 实现步骤：
        1、提取出数学题目
        2、根据检索到的相似题目作为参考，推理计算得到正确答案
        3、严格按照输出格式进行输出，思考计算过程写在reasoning字段中，
            正确答案写在answer字段中
    """
```

5. 生成参数配置

关键参数推荐：

- **enable_thinking: True** - 开启思考模式
- **max_new_tokens: 1024** - 给予足够的生成空间

关键技巧

1. **余弦相似度的正确实现：**
   - 必须计算向量的范数（模长）
   - 可以手动实现或使用 PyTorch 内置函数
   - 确保返回值在 [-1, 1] 范围内
2. **RAG的核心价值：**
   - 通过相似题目提供解题思路和方法
   - 即使是小模型也能学习参考例子完成任务
3. **检索质量的重要性：**
   - 检索到的题目质量直接影响最终答案质量
   - 需要正确实现相似度计算
4. **结构化输出：**
   - 明确区分 `reasoning`（推理过程）和 `answer`（最终答案）
   - `answer` 只包含数字，不含单位

评分标准

第一部分：相似度计算测试（20分）

- 20 个查询，每个查询对应题库中的一道题
- 检查 `compute_similarity()` 函数能否找到最相似的题目
- 每题1分，共20分

第二部分：RAG 问答测试（30分）

- 6 道数学题，每题5分
- 判定规则：
  - ✅ 答案完全正确得 5 分
  - ❌ JSON格式错误或答案不正确得 0 分

总分：50分