面向推荐重排任务的提示词工程探索报告

纪嘉仪
上海交通大学

所有实验代码、日志和测试可在 GitHub 获取

引言

推荐系统是现代互联网应用的核心技术，能够根据用户历史行为提供个性化内容推荐，在缓解信息过载、提升用户体验方面发挥重要作用。随着大型语言模型（LLM）的快速发展，将LLM应用于推荐系统重排阶段展现出独特优势，包括对文本内容的深刻理解、利用海量世界知识进行语义推理的能力，以及零样本/少样本学习和结果解释性等。

近期研究表明，利用LLM进行推荐结果重排可以提升个性化效果，但同时高度依赖人工设计的提示词(prompt)。如何通过提示词工程有效引导LLM理解用户偏好并生成高质量的推荐排序，成为一个值得深入探索的问题。已有工作尝试优化提示词策略以提升重排性能，报告显示经过优化的提示词可使推荐排序的 `NDCG@10` 指标相比基线提升5.61%至20.68%。

本项目围绕提示词工程在电影推荐重排任务中的应用展开。我们使用提供的验证集数据(`val.jsonl`)模拟场景：

- 每条数据包含一个用户ID
- 该用户按观看时间排序的历史电影列表(item_list)
- 召回阶段得到的候选电影列表(`candidates` ，其中包含用户实际下一部观看的电影 `target_item`)

目标是设计提示词引导LLM对候选列表进行重排序，产出一个按推荐强度从高到低排列的电影ID列表，使实际观看的电影尽可能靠前。我们采用 OpenRouter 提供的 `deepseek/deepseek-chat-v3-0324:free` 模型（OpenAI Chat Completions接口）进行实验，设置 `temperature=0` 以保证结果稳定。模型输出的推荐列表通过计算NDCG@10与真实目标进行对比评估，NDCG@10越接近1表示排序质量越高。

我们在验证集上分别测试了三种不同的提示词策略，并对比它们的表现和特点，最终确定了最佳方案。以下将介绍提示词的设计思路和结构，并详细讨论不同版本提示词的尝试过程、实验结果和优化分析。

提示词设计策略与实现

LLM提示词采用OpenAI Chat API所需的消息列表格式，即由多个 `{"role": ..., "content": ...}` 字典组成的列表。其中常用的角色包括"system"（系统指令）、"user"（用户输入）和"assistant"（助手示例）。通过精心组织系统和用户消息，可以引导模型扮演特定角色、遵循指定风格完成任务。本项目中，我们需要提示模型阅读用户的历史观影列表和候选电影列表，然后给出排序后的推荐结果（仅包含电影ID列表）。提示词设计的要点包括：

明确角色和任务

在系统消息中告知模型其身份（如“电影推荐助手”）和需要完成的任务（根据用户历史喜好对候选电影排序）。

提供必要信息

在用户消息中呈现用户的历史记录和待排序的候选列表，使模型有足够上下文进行推理。

指令输出格式

明确要求模型输出排序后的电影ID列表，并尽量不包含多余解释，以便后续解析。为了稳定解析过程，我们要求输出为Python列表格式（如 `[id1, id2, ...]` ），方便使用正则或JSON解析提取ID序列。

控制模型行为

可选地，通过系统消息或示例对话控制模型先进行一定的分析推理，再给出最终答案，从而提高排序准确性。

接下来小节将详细介绍我们探索的三种提示词版本及其效果。

策略一：直接指令提示策略

设计思路：

首先我们尝试最直接的提示设计，即单轮对话直接给出排序任务指令。这一策略中，我们仅提供一个简洁的系统消息和用户消息。

- 系统消息将模型设定为“电影推荐系统助手”，明确任务是根据用户历史喜好对候选电影排序。
- 用户消息则列出用户看过的电影列表和候选电影列表，直接要求模型按用户可能喜欢的程度排序候选电影并输出对应的ID列表。
- 我们特别强调输出格式要求模型“仅以Python列表形式给出电影ID”。这样做是为了防止模型输出多余的文字，方便我们后续用代码提取结果列表。

提示结构：

- **System（系统角色）**：说明模型身份和任务要求，例如：

```
"你是一个专业的电影推荐系统。"
      "根据用户的观影历史，你需要将候选电影列表重新排序，使得最可能被用户喜欢的电影排在前面。"
      "请考虑电影的题材、风格、主题、导演、演员等多个因素，找出最适合用户的电影。"
      "为了确保推荐有效，请只从候选电影列表中进行推荐，不要添加任何列表以外的电影。"
      "输出格式要求：按照推荐优先级输出电影名称和ID的列表，使用JSON数组格式，每个元素包含电影名称和ID，不要包含其他说明文字，例如:
```

- **User（用户角色）**：提供具体数据和请求，例如：

```
f"用户ID: {user_id}\n\n"
f"用户观看过的电影列表:\n{watched_movies_text}\n\n"
f"候选电影列表，请根据用户可能的喜好对以下电影名称进行排序:\n{candidate_movies_text}\n\n"
"请直接输出按推荐优先级排序的电影名称和ID列表，使用JSON数组格式，每个元素包含电影名称和ID，不要包含其他解释。"
```

在这个提示中，所有信息和指令都在单轮对话中给出，模型需要直接据此完成任务。

实验结果：

- 我们使用验证集对策略一进行了测试。对于每条验证数据，通过 `construct_prompt` 生成上述格式的消息列表，调用DeepSeek-V3模型API获取排序结果，再用 `parse_output` 提取模型输出的ID序列，与真实目标进行NDCG@10计算。策略一在验证集上的平均NDCG@10约为0.4812（满分1.0）。这一成绩比随机排序（理论上NDCG@10约0.33左右）有明显提升，表明LLM已经能够利用用户历史偏好对候选集进行一定程度的重排。然而，我们也观察到几点不足：模型有时会 **倾向推荐热门或知名度更高**的电影，而不完全基于用户的个性化历史；此外，在个别样本中，模型的输出格式偶尔不够严格（例如带有解释文本或缺少列表括号），需要 `parse_output` 进行额外处理。这些问题促使我们考虑改进提示方式，以进一步提升模型对用户兴趣的把握和输出规范性。

策略二：思维链引导的提示策略

设计思路：

- 针对策略一中模型对用户兴趣理解不够深入的问题，我们引入了思维链 (Chain-of-Thought, CoT) 提示方法。思维链提示通过要求模型在给出最终答案前先进行分步的推理分析，能让模型更充分地利用其知识和推理能力，从而提高复杂任务的表现。在我们的任务中，思维链提示的实现方式是在提示中增加引导，使模型先分析用户历史的偏好特征以及候选电影的属性匹配，再产出排序结果。具体而言，我们将提示调整为多轮对话格式，分两步进行：
- 引导模型分析：在前两轮，让模型先输出对用户偏好的分析和对候选列表的初步评价。例如我们在用户消息中增加一句请求：“请首先解释用户可能喜欢哪些类型的电影，以及候选电影中有哪些匹配这些偏好的特征。然后给出最终的推荐排序。” 模

型回应这一消息时，会以助手身份先给出一段分析性的回答，列出用户历史偏好的主题/类型，以及逐一讨论候选电影与这些偏好的契合程度。
- 引导模型输出最终列表：紧接着，我们以第三条用户消息要求模型给出最终的排序结果，只保留电影ID列表格式。比如第三轮用户消息可以简洁地说："现在仅输出最终排序的电影ID列表（Python列表格式），不要解释。"

通过上述两轮对话，我们期望模型先"思考"再回答。在实际构造 `messages` 时，这可以体现在提供一个示例对话：第一条用户消息要求分析，接着用一个 `assistant` 角色消息填写一个"占位"的分析过程（这相当于 `few-shot` 示例，见下一策略），然后第二条用户消息要求列表，最终模型以 `assistant` 角色产生最终答案。这种设计迫使模型在生成最终列表前经过一个分析步骤，有助于提升推荐准确性。需要注意，我们在提示中明确第二步只需输出ID列表，以确保最终输出易于解析。

提示结构： 多增加两轮交互，大致如下：

- **System**：

```
"# Role:\n"
    "你是一名电影推荐系统的重排助手。你的任务是根据用户观影历史的风格类型，从候选电影中推荐最符合其偏好的作品，基于用戶的历史行为序列

    "# Profile:\n"
    "你具备电影类别分析和推荐能力，能够理解用户的喜好，并评估电影之间的相似度。\n"

    "# Background:\n"
    "用户提供了历史电影列表，以及候选电影列表，电影ID没有关联关系，需要你根据用户偏好的电影风格对候选电影进行排序。\n"

    "## Goals:\n"
        "1. 分析用户历史观影记录中的电影类别喜好。\n"
        "2. 评估每个候选电影与用户偏好的相似程度。\n"
        "3. 输出按相似度排序的最终推荐电影列表（JSON格式）。\n"

    "## Constraints:\n"
        "- 只考虑以下电影风格类别进行分析: Animation, Comedy, Children's, Musical, Romance, Adventure, Action, Th
        "- 最终答案仅包含候选列表中的电影，且以JSON数组格式输出电影ID列表，不附加多余解释。\n"

    "## Workflows:\n"
        "1. **用户偏好分析**: 统计用户历史中各类别出现频次，识别用户最喜欢的类别。\n"
        "2. **候选相似度评估**: 比较每个候选电影的类别与用户偏好类别的重合程度，判断匹配度（高/中/低）。\n"
        "3. **结果输出**: 根据匹配度对候选电影进行排序，输出匹配度最高的电影名称和ID列表（JSON）。"
```

- **User (Round 1)**：
  提出需要模型先分析用户偏好。

```
    # 用户基本信息
    f"用户ID: {user_id}\n\n"
    f"用户观看过的电影列表:\n{watched_movies_text}\n\n"
    # 步骤引导
    "请按照以下步骤进行分析：\n"
    "1. 分析每部电影属于哪些风格类别，并制作风格归属表格\n"
    "2. 统计各风格类别的出现频率，并绘制风格偏好频率表格\n"
    # 数据输出格式要求
    f"3. 请基于这些类别计算电影类别偏好频率数据：{genres}\n"
    "4. 将类别偏好频率输出为JSON格式，包含类别名称和对应的偏好频率\n"
    # 指定输出格式
    "请在分析结果最后将用户偏好数据以JSON格式输出，格式必须为：\n"
    "```json\n{\"类别偏好\": {\"Animation\": 频率1, \"Comedy\": 频率2, ...}}\n```\n"
    "注意：这个JSON数据将用于后续分析，请确保其格式正确且包含所有类别名称。"
```

- **Assistant (Round 1)**：模型实际生成时，这部分是它的分析内容。
- **User (Round 2)**：
  分析候选电影并与用户偏好匹配。

```
    # 候选电影列表
    f"候选电影列表:\n{candidate_movies_text}\n\n"
    # 类别参考信息
```

```
        f"电影风格类别参考: {', '.join(genres)}\n\n"
        # 分析步骤
        "请按以下步骤分析候选电影：\n"
        "1. 分析每部候选电影属于哪些风格类别，并制作风格归属表格\n"
        "2. 基于上轮输出的JSON格式用户类别偏好数据(包含类别名称和频率的对象)，计算各候选电影与用户偏好的相似度\n"
        "3. 根据电影类别与用户偏好匹配程度计算相似度得分，并为每部电影分配相似度值(最高为1.0)\n"
        "4. 根据相似度对候选电影进行降序排序（相似度越高排序越靠前），并在结果中包含相似度分数\n"
```

- **Assistant (Round 2)**：模型实际生成时，这部分是它的分析内容。
- **User (Round 3)**：
  要求输出最终的JSON格式的推荐列表.

```
        "### Step 3: 推荐结果输出\n"
            "请根据以上分析，从候选电影中选出最终的推荐列表，按照匹配度从高到低排序，以 JSON 数组格式输出电影名称、ID列表。\n\n"
            "要求：只输出JSON，不要添加任何解释。"
        "格式示例：[{\"name\": \"电影1\", \"id\": 123}]\n"
```

- **Assistant (Final Response)**：模型最终输出ID列表。

我们实现中尝试了两种方式：

- (a) 不提供示例，让模型自主在前两轮生成分析，再根据第三轮指令给列表；
- (b) 在提示中包含一个人工设计的示例分析作为Few-shot（详见策略三），然后让模型模仿这种格式。

方法(a)利用了模型自身的推理能力，但存在不确定性；方法(b)提供示例可能稳定输出风格。两者均属于思维链引导的思路。

实验结果： 策略二在验证集上的平均NDCG@10提升到了约 `0.5029`. 相比直接指令提示，性能有显著提高，证明了思维链提示的有效性。这一策略下，模型往往会先给出一段针对用户历史的分析，例如“用户观看的影片多为科幻动作类，因此可能对类似题材感兴趣。候选名单中电影X是科幻片，符合用户口味…”，然后在第三轮只报出ID列表。通过这些分析，我们观察到模型对用户兴趣的把握更加准确，推荐命中实际目标影片的概率提高，NDCG分数因此上升。不过也存在开销和风险：模型生成的分析增加了交互轮数，在每次API调用中 `Tokens` 消耗更多；同时如果第一轮分析未能正确识别用户偏好，可能仍会影响最终排序。但总体而言，思维链引导有效增强了模型的推理深度，使推荐结果更符合用户兴趣。

策略三：少样本示例提示策略

设计思路： 在策略二基础上，我们进一步尝试在提示中加入少样本示例（Few-shot），通过提供一个完整的示例对话来示范如何进行推荐排序。 `Few-shot` 提示可以让模型从示例中学习任务格式和逻辑，从而在处理实际问题时更准确。我们的 `Few-shot` 示例精心设计为一个类似的用户历史和候选列表，以及相应的分析和排序结果。这一示例突出展示了如何根据用户历史推断偏好并据此对候选电影排序。

在具体实现上，我们将示例以第一轮用户-助手对话的形式置于提示最前面：系统消息后，首先来一组示例（一个“虚拟”用户消息及对应的“助手”回复）。助手回复包含两个部分：对偏好的分析+最终的推荐列表。这相当于手把手给模型演示了一遍。当实际用户的问题（实际要排序的那条数据）作为新的用户消息出现时，模型已经有了示例可参考，会倾向于沿用类似的回答结构和策略。

提示结构： 以一个示例为前导，之后再接真实提问：

- **System**：

```
         "# Role:\n"
        "你是一名电影推荐系统的重排助手。你的任务是根据用户观影历史的风格类型，从候选电影中推荐最符合其偏好的作品，基于用戶的历史行为序列

        "# Profile:\n"
        "你具备电影类别分析和推荐能力，能够理解用户的喜好，并评估电影之间的相似度。\n"

        "# Background:\n"
        "用户提供了历史电影列表，以及候选电影列表，电影ID没有关联关系，需要你根据用户偏好的电影风格对候选电影进行排序。\n"
```

```
"## Goals:\n"
    "1. 分析用户历史观影记录中的电影类别喜好。\n"
    "2. 评估每个候选电影与用户偏好的相似程度。\n"
    "3. 输出按相似度排序的最终推荐电影列表（JSON格式）。\n"

"## Constraints:\n"
    "- 只考虑以下电影风格类别进行分析: Animation, Comedy, Children's, Musical, Romance, Adventure, Action, Th
    "- 最终答案仅包含候选列表中的电影，且以JSON数组格式输出电影ID列表，不附加多余解释。\n"

"## Workflows:\n"
    "1. **用户偏好分析**: 统计用户历史中各类别出现频次，识别用户最喜欢的类别。\n"
    "2. **候选相似度评估**: 比较每个候选电影的类别与用户偏好类别的重合程度，判断匹配度（高/中/低）。\n"
    "3. **结果输出**: 根据匹配度对候选电影进行排序，输出匹配度最高的电影名称和ID列表（JSON）。"
```

- **User (Example)**：
  示例用户对话 (Few-shot example)

```
"用户ID: example_user_123\n\n"
"用户观看过的电影列表:\n"
"- Liar Liar (ID: 123)\n"
"- The Mask (ID: 456)\n"
"- Mrs. Doubtfire (ID: 789)\n"
"- Toy Story (ID: 246)\n"
"- The Lion King (ID: 357)\n\n"
"候选电影列表:\n"
"- Ace Ventura: Pet Detective (ID: 111)\n"
"- Forrest Gump (ID: 222) \n"
"- The Shawshank Redemption (ID: 333) \n"
"- Aladdin (ID: 444) \n"
"- Terminator 2: Judgment Day (ID: 555) "
```

- **Assistant (Example)**：
  示例助手对话 (Few-shot example)

```
"### 分析用户观影风格偏好\n\n"
 "首先，我将分析每部电影的风格类别:\n\n"
 "| 电影 | 风格类别 |\n"
 "|------|------------|\n"
 "| Liar Liar | Comedy |\n"
 "| The Mask | Comedy |\n"
 "| Mrs. Doubtfire | Comedy |\n"
 "| Toy Story | Animation, Children's, Comedy |\n"
 "| The Lion King | Animation, Children's, Musical |\n\n"
 "用户明显偏好喜剧类(Comedy)电影，其次是动画(Animation)和儿童(Children's)类电影。\n\n"
 "### 分析候选电影和用户偏好的匹配度\n\n"
 "分析每部候选电影的风格类别:\n\n"
 "| 电影 | 风格类别 | 与用户偏好匹配 |\n"
 "|------|------------|------------|\n"
 "| Ace Ventura: Pet Detective | Comedy | 高  |\n"
 "| Forrest Gump | Comedy, Drama, Romance | 中  |\n"
 "| The Shawshank Redemption | Drama | 低  |\n"
 "| Aladdin | Animation, Children's, Comedy | 高  |\n"
 "| Terminator 2 | Action, Sci-Fi, Thriller | 低 |\n\n"
 "根据匹配程度去除匹配度低的候选电影，留下10个匹配度高和中的电影继续分析：\n\n"
 "1. Aladdin (ID: 444) \n"
 "2. Ace Ventura: Pet Detective (ID: 111) -\n"
 "3. Forrest Gump (ID: 222) \n"
 "4. The Shawshank Redemption (ID: 333)\n"
 "5. Terminator 2 (ID: 555) \n\n"
 "在这10个电影里继续考虑电影的题材、主题、导演、演员等多个因素，找出最适合用户的电影，输出匹配度最高的电影名称和ID列表排序（JSON
 "[{\"name\": \"Aladdin\", \"id\": 444}]\n"
```

- **User**：
  分析候选电影并与用户偏好匹配。

```
        # 用户基本信息
        f"用户ID: {user_id}\n\n"
        f"用户观看过的电影列表:\n{watched_movies_text}\n\n"
        # 候选电影列表
        f"候选电影列表:\n{candidate_movies_text}\n\n"
        # 类别参考信息
        f"电影风格类别参考: {', '.join(genres)}\n\n"
        # 步骤引导
        "让我们一步步思考！请按照以下步骤进行分析：\n"
        "1. 分析每部电影属于哪些风格类别，并制作风格归属表格\n"
        "2. 用户的电影风格偏好分析\n"
        "3. 分析每部候选电影属于哪些风格类别，评估相似程度，去除相似度低的，只留下10个候选\n"
        "4. 考虑电影的题材、主题、导演、演员等多个因素，找出最适合用户的电影，输出匹配度由高到低的电影名称和ID列表排序（JSON）"
        "[{\"name\": \"Aladdin\", \"id\": 444}]\n"
```

示例的选择需要代表性且清晰。我们构造了一个虚拟用户历史偏好明显的案例，例如用户看了多部喜剧片，候选列表里混合了喜剧和非喜剧电影，我们在示例回答中将喜剧类电影的ID放在前列并给出解释。这样模型可以从中学习到“依据类型匹配来排序”的思路。在示例输出部分，我们刻意采用严格的格式：最后答案部分就是纯粹的Python列表字符串。

实验结果： 策略三取得了平均NDCG@10约 `0.5410` 的成绩，相比策略二进一步提升了约4个百分点。这说明Few-shot示例提供的范例作用让模型对任务的理解和执行更加到位。一方面，模型几乎100%遵循了我们希望的格式输出列表，再也没有解析失败的情况；另一方面，推荐准确率也略有提升，我们推测这是因为示例教会了模型更好地将用户历史偏好映射到候选排序中。例如，在若干包含多类型混杂的候选列表里，策略三的模型能更可靠地把与用户偏好类型一致的影片排在前面。需要注意的是， `Few-shot` 提示增加了提示长度，在OpenAI接口中意味着更多的token开销；但鉴于验证集每条样本调用都是独立的，对总消耗影响有限。此外，示例需要 carefully 构造，否则不恰当的示例可能误导模型。但在本次实验中，我们提供的示例在绝大多数情况下对模型起到了正向指导作用。

不同策略效果比较与分析

经过以上三种提示词策略的尝试，我们将它们的验证集结果汇总如下：

策略	NDCG\@10	log
策略一（直接指令）	≈0.4812	test1.log
策略二（思维链）	≈0.5029	test7.2.log
策略三（少样本示例）	≈0.5410	test10.2.log

从结果来看，策略二和策略三相对于直接指令都有明显优势。其中，思维链提示（策略二）通过鼓励模型进行中间推理，大幅提高了排序准确性；在此基础上加入示例（策略三）又略有提升，并确保了输出格式的一致性。 策略一的劣势主要在于模型可能欠推理：它虽然知道要排序，但未必充分利用了用户历史信息，往往给出大众化的排序，导致一些用户真正喜爱的冷门电影可能被排得较低。同时，对于输出格式，只靠一次指令模型有时不够严格遵守。 策略二通过显性推理引导，让模型在回答中表现出对用户兴趣的理解。 `Chain-of-Thought` 的介入相当于让模型自己解释“为什么推荐这些电影”，在这个过程中模型会调用其关于电影类型、风格的知识，从而更有依据地调整排序。这符合直觉：让模型“想明白了再做”，结果就更可靠。确实，我们看到策略二下模型往往准确找到用户喜欢的类型，例如针对热爱科幻的用户会把科幻候选片排在最前。这一策略显著提升了NDCG。当然，代价是生成的内容更多，每次调用消耗增加，但在可接受范围。 策略三进一步提供了示例示范，使模型对任务的预期输出有了明确模板可参考。示例中的解释和排序逻辑被模型部分内化后，减少了它在真实任务中走弯路的可能。这种 `few-shot` 学习对于复杂任务尤其有效。我们注意到策略三对某些边缘案例有所帮助：例如用户历史偏好不明显时，策略三的模型也能借鉴示例中“结合常识进行推荐”的做法，给出合理的排序。然而示例策略的提升幅度有限，说明在策略二的基础上，模型已经相当了解任务，示例更多地是锦上添花，保证稳定性和格式规范。 综上，策略二和策略三均表现优秀，均显著优于直接指令法。这符合我们对提示词优化的预期：更丰富的上下文和指令可以激发LLM的推理潜能，提升推荐准确率。另一方面，也说明了LLM在推荐任务中的潜力——尽管没有传统用户画像和复杂模型训练，仅凭提示词和内置知识，LLM也能达到不错的推荐效果。

最终方案选择：
综合考虑推荐性能和提示词复杂度，我们选择**策略三（少样本示例提示策略）**作为最终的方案。主要原因包括：

- 最高的客观效果：策略三在NDCG@10上略胜一筹，排名第一，尽可能保证了在隐藏测试集上取得更高的分数。
- 输出稳定性：该策略下模型输出格式规范统一，便于解析，不会因格式问题影响评分。此外，多轮对话提示降低了模型产生无关内容的概率。
- 方案创新性：从主观评价角度，策略三融合了思维链和少样本示例两种提示工程技巧，具有一定创新性和合理性。它让模型既能解释又能模仿示例完成任务，具备良好的可解释性和可控性，满足评价标准中对提示词合理性和可解释的要求。
- 开销可接受：虽然提示包含示例较长，但考虑到我们使用的DeepSeek模型对每条样本调用次数有限，且官方提供了一定的免费额度，总体开销在可控范围内。相较性能提升，这是值得的权衡。

最后，我们提交的 `construct_prompt` 函数会生成如策略三所述的消息列表，而 `parse_output` 函数能够从模型回答中准确提取出电影ID列表用于评估。通过这一系列探索，我们验证了提示词工程在推荐重排任务中的价值：精心设计的提示能够有效引导LLM利用其内含的世界知识和推理能力，完成传统推荐模型难以处理的个性化排序任务，在无需额外训练的情况下取得令人满意的性能提升。今后，随着LLM能力的提升和更多提示技巧的出现，我们有理由相信基于提示词的推荐系统将在个性化和多样性等方面取得更大的突破。

Bonus：英文提示词效果分析

作为额外实验，我们将最优的中文提示词策略（策略三）完全翻译为英文版本，并在相同测试集上进行了验证。结果显示英文提示取得了平均NDCG@10 `0.5857` 的成绩，相比中文版本的 `0.5410` 提升了约 `8.26%`。

关键发现：

1. **性能提升**：10个测试样本中有3个达到完美分数(NDCG@10=1.0)，表明英文提示在某些情况下能更准确地理解用户偏好
2. **稳定性保持**：所有样本输出均符合JSON格式要求，解析成功率达100%
3. **最佳案例**：样本2、3、6实现完美排序，样本5、7也获得较高分数(0.6309)

原因分析：

1. 电影领域的专业术语和分类体系多以英文为主，模型对英文电影相关概念理解更准确
2. LLM在英文语料上训练更充分，语义理解能力更强
3. 英文表达通常更为直接，减少了语义歧义

应用建议：

1. 对于国际化推荐系统，可优先考虑英文提示词
2. 在电影、科技等英文主导领域，英文提示可能效果更佳
3. 可尝试混合语言提示策略，关键指令使用英文，辅助说明使用本地语言

这一实验证实了提示词语言选择也是影响模型性能的重要因素，为多语言场景下的提示工程提供了新的优化方向。

个人思考：思维树提示（Tree of Thoughts, ToT）与懒惰提示（Lazy Prompting）

思维树提示（Tree of Thoughts, ToT）

对于需要探索或策略性前瞻的复杂任务，传统或简单的提示技术往往难以胜任。Yao 提出了 **“思维树”（Tree of Thoughts, ToT）**框架，作为对思维链（Chain-of-Thought, CoT）提示方法的推广，旨在鼓励对作为通用问题求解中间步骤的“思想”进行探索。

ToT 会维护一棵“思想树”，其中的每个“思想”都代表一段连贯的语言序列，作为解决问题的中间步骤。这种方法使语言模型能够在解决问题的过程中，通过有意的推理步骤自我评估所取得的中间进展。随后，模型生成和评估“思想”的能力会与搜索算法（例如广度优先搜索和深度优先搜索）相结合，从而实现带有前瞻和回溯的系统化思想探索。

- **核心思想** ：对 CoT 的进一步泛化和扩展。CoT 遵循单一线性思维链，而 ToT 允许 LLM 同时探索多个不同的推理路径，形成一个“思维树”。模型可以在树的不同节点上评估中间“想法”的价值，并决定扩展哪些分支，或回溯到更有希望的路径。

- **机制** ：维护一个由“想法”（连贯的文本序列，代表解决问题的中间步骤）组成的树状结构。模型可以进行广度或深度优先的探索，并使用评估机制（可能由 LLM 自我评估或外部反馈）来指导搜索。
- **优势** ：特别适用于需要广泛探索和复杂规划的任务，传统线性推理难以解决。

Hulbert (2023) 将 ToT 框架的主要概念简化为一段简短的“ **ToT 提示词** ”，指导 LLM 在一次提示中对中间思维做出评估。例如：

```
假设三位不同的专家来回答这个问题。
所有专家都写下他们思考这个问题的第一个步骤，然后与大家分享。
然后，所有专家都写下他们思考的下一个步骤并分享。
以此类推，直到所有专家写完他们思考的所有步骤。
只要大家发现有专家的步骤出错了，就让这位专家离开。
请问...
```

```
Imagine three different experts are answering this question.
All experts will write down 1 step of their thinking,
then share it with the group.
Then all experts will go on to the next step, etc.
If any expert realises they're wrong at any point then they leave.
The question is...
```

懒惰提示（Lazy Prompting）

“懒惰提示”是与传统“给足上下文”做法相对的高阶技巧：先抛出一个简短甚至粗略的指令，快速观察 LLM 的反应，再决定是否补充信息。它能节省时间的前提是，你必须能立刻判断输出优劣，然后有针对性地增补背景。

最常见的应用场景是调试代码——开发者往往直接把长篇错误日志贴进模型，或只写一句 “Edit this: …”、“sample dotenv code” 等极短提示。模型通常明白你想要理解报错、修复缺陷或回忆写法，并能生成相当可用的答案；若答复有误，你也能迅速识别并微调提示，引导模型修正。

然而，懒惰提示并非万能。若没有额外上下文，模型几乎不可能给出好结果——例如只给不完整的程序规格，或你明确想指定某款 PDF-to-text 工具（如 LandingAI 的 **Agentic Doc Extraction** ）——就必须在提示里写明细节。若验证输出正确性本身耗时巨大（例如必须完整跑代码才能发现bug），也应在一开始提供更充分的背景，以提高一次成功率。

> • 仅在能够快速迭代、易于人工评估的场景下使用懒惰提示（如网页或聊天界面）。
>
> • 对于批量 API 调用、代码嵌入等无法逐条检视输出的情况，应避免懒惰提示，改用“足量上下文”策略。
>
> • 在掌握“足量上下文”标准后，再尝试“最小上下文”以（Lazy Prompting）确保产出质量。

“懒惰提示”的命名灵感来自计算机科学中的“惰性求值”：只有在确实需要某个结果时才调用函数；同理，我们只在确有必要时向提示中逐步添补细节。该概念由 **aisuite** 合作者 Rohit Prasad 提出，并在实践中印证了其高效价值。