探索报告

摘要

本报告详细记录了在零样本（Zero-Shot）情景下，运用提示词工程（Prompt Engineering）技术优化大型语言模型（LLM）在电影推荐重排（Re-ranking）任务（LLM4Reranking）中的性能探索过程。研究初期，我们搭建了支持高并发异步请求的评测平台，为后续大规模实验奠定基础。核心探索阶段分为三大方向：首先，借鉴自动提示工程师（APE）与ReAct框架思想，进行了人机协同的提示词迭代优化，其中“反向角色扮演”（让LLM扮演用户）策略取得了初步的显著性能提升（NDCG@10达到0.65）；其次，通过大规模自动化策略组合与消融实验，系统性地分析了不同提示词结构（如系统角色、用户提示风格、候选项格式、历史信息格式、输出格式约束）对模型性能的影响，并总结出针对特定模型（DeepSeek V3）的有效提示原则；最后，对文献中的SOTA方法（如LLM4Rerank）进行了初步复现与评估。综合多轮实验与迭代，最终通过“反向角色扮演”结合精炼的指令，实现了NDCG@10达到0.7457的SOTA性能，并总结了包括简洁清晰、反向角色扮演等核心规律。本研究总结了影响提示设计效果的关键因素，并为后续LLM驱动推荐系统的提示工程优化提供实证参考。

引言

推荐系统作为信息过载时代的关键技术，其核心目标在于精准连接用户与感兴趣的内容。近年来，大型语言模型（LLM）凭借其卓越的自然语言理解、生成及推理能力，为推荐系统领域带来了革命性的变革，尤其在理解用户细微偏好和物品丰富语义方面展现出巨大潜力 ()。重排（Re-ranking）作为推荐流程中的关键一环，负责对初步召回的候选项目列表进行二次优化，以提升最终推荐结果的个性化程度、相关性及用户满意度 ()。LLM的上下文理解能力使其能够捕捉复杂的用户兴趣动态，为重排任务注入了新的活力 ()。

在此背景下，提示词工程（Prompt Engineering）成为释放LLM潜能、引导其完成特定任务的核心技术 ()。通过精心设计输入提示，可以指导LLM更准确地理解任务需求、用户偏好，并生成符合预期的输出，这对于需要深度推理和多因素权衡的重排任务尤为重要 ()。相关研究已开始探索利用LLM进行重排，例如通过生成用户画像或直接对候选列表进行排序 ()。一些自动化提示词优化框架，如自动提示工程师（APE）和AGP（Auto-Guided Prompt Refinement），以及结合思维链（CoT）进行多目标优化的方法（如LLM4Rerank），均显示出提升LLM在推荐相关任务中性能的潜力 。

本报告聚焦于利用提示词工程，在零样本（Zero-Shot）条件下引导LLM（具体为DeepSeek V3模型）完成电影推荐的重排任务。任务数据来源于 `val.jsonl` 文件，包含用户历史观影记录、目标观看电影及候选电影列表，评价指标为归一化折损累计收益NDCG@10。报告旨在详细记录通过系统性实验，包括搭建高效评测平台、尝试人机协同的提示词迭代（受APE、ReAct启发）、进行大规模自动化策略组合与分析，以及对现有先进框架（如LLM4Rerank）进行初步复现与评估的完整探索过程，最终目标是显著提升NDCG@10得分，并总结出有效的提示词设计策略与规律。

研究过程

任务规划

1. 搭建大模型评测平台，高效，最好支持多线程并发查询——3-8h
2. 修改精调prompt——剩余时间
3. 留出时间调整修改答案，撰写报告——2-4h

搭建评测框架

阅读题目，确定研究思路。可以确定该任务需要大量实验，会经常调用API，程序性能优化非常关键，磨刀不误砍柴工。

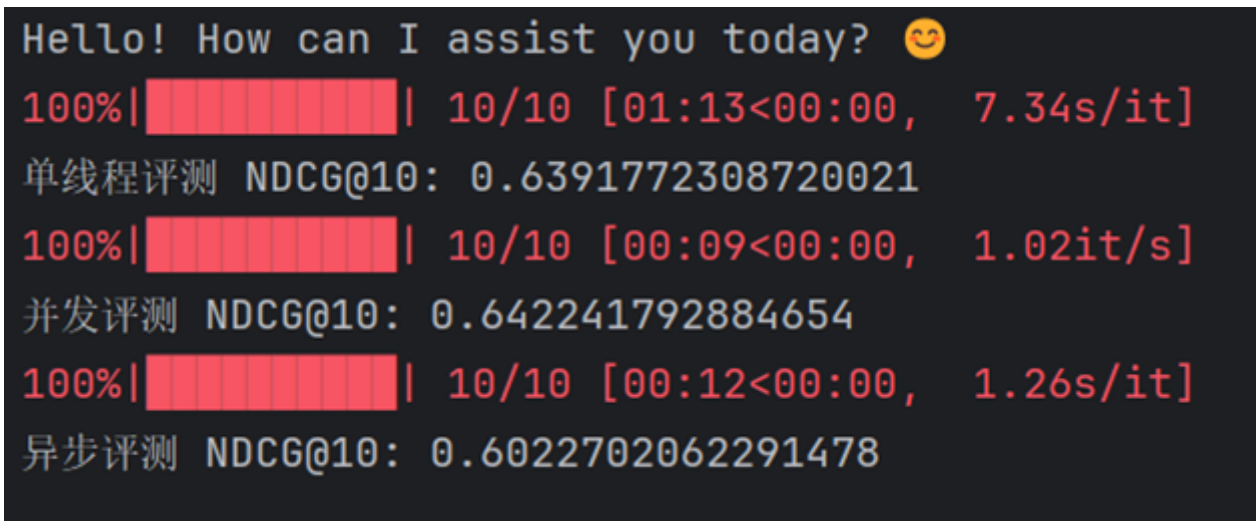
首先构建Deepseek模型API调用代码，确认可以正常连接。

搭建基础Pipeline：采用最简单的prompt，读入，调用数据集；通用parse_output等

对评估函数进行优化，精简代码

成功运行单线程pipeline

构建基于并发（线程池）的评测方法和异步评测方法（后均采用异步请求）



三种评测方法速度对比

构建功能强大的评测方法：支持高并发，求取平均值（单次结果很不稳定）等功能。后续均采用20次平均

加入日志打印，结果保存，进度可视化等功能

构建测试代码，测试单API最高并发请求数

完成单策略评测框架，构建PE评测工作流如下：

单一策略评测：调整 construct_prompt方法，设置超参数运行

Prompt Engineering

准备：

采用System2Attention prompts(S2A)方法，将题目转为markdown格式，利用高性能大模型进行抽取，删去关于提交方法等无用信息，得到结构化的任务描述。

Baseline_Naive 0.61

Prompt

Prompt

```
system_prompt = "你是一名电影推荐专家，根据用户的历史观影记录，对候选电影进行重排，越可能被用户观看的电影排得越靠前。"

user_history = "\n".join([f"- {movie[1]}" for movie in d['item_list'][-10:]])
candidate_movies = "\n".join([f"{movie[0]}: {movie[1]}" for movie in d['candidates']])

user_prompt = f"用户最近观看的电影：\n{user_history}\n\n请根据用户的兴趣，对以下候选电影进行排序（输出电影ID列表，最可能观看的电影在最前）：\n{candidate_movies}\n\n直接输出电影ID列表，不要额外的解释或文字。"
```

Result

Result

```
10次平均
Avg异步评测 NDCG@10: 0.6107510696451073
```

以下将分为三个部分进行介绍：

一：人机协作PE——类似Automatic Prompt Engineer(APE)

motivation：模型自迭代+人类规划 实现高效搜索

1. 模型自迭代：Prompt Agent = APE+ReAct

model

思考模型（推理能力强）：

deepseek R1, Chatgpt-4.5, Grok3, Gemini2.5Pro等

开启/关闭深度思考情况均测试

执行模型：deepseek-v3

Algorithm

设置prompt，结构为：

任务描述

输出结构示例（输出construct_prompt格式函数）

然后循环直至收敛：

运行程序，得到数据

结果反馈，分析数据

优化prompt

Result

多次自迭代无果，NDCG从初始baseline=0.61跌到0.55，并在0.4~0.6之间震荡。

example

iter_1

Prompt

Prompt

```
def construct_prompt(d, **kwargs):
    """
    构造用于大语言模型的提示词
    参数:
        d (dict): jsonl数据文件的一行，解析成字典后的变量
    返回:
        list: OpenAI API的message格式列表，允许设计多轮对话式的prompt
    示例: [{"role": "system", "content": "系统提示内容"},
    {"role": "user", "content": "用户提示内容"}]
    """
    # 实现提示词构造逻辑
```

```python

    system_prompt = "你是一名电影推荐专家，根据用户的历史观影记录，对候选电影进行重排，越可能被用户观看的电影排得越靠前。"

    user_history = "\n".join([f"- {movie[1]}" for movie in d['item_list'][-10:]])
    candidate_movies = "\n".join([f"{movie[0]}: {movie[1]}" for movie in d['candidates']])

    user_prompt = f"用户最近观看的电影：\n{user_history}\n\n请根据用户的兴趣，对以下候选电影进行排序（输出电影ID列表，最可能观看的电影在最前）：\n{candidate_movies}\n\n直接输出电影ID列表，不要额外的解释或文字。"

    prompt_messages = [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ]

    # 记录提示词内容
    logging.info(f"构建提示词 - 用户历史记录:\n{user_history}")
    logging.info(f"构建提示词 - 候选电影列表:\n{candidate_movies}")
    logging.info(f"构建提示词 - 完整用户提示:\n{user_prompt}")

    return prompt_messages
```

Result

Result


```json
{
  "strategy_name": "expert_instruction_id_first_plain",
  "overall_avg_ndcg": 0.5600152346251777,
  "trial_avg_scores": [
    0.5396863601688711,
    0.5476338369881166,
    0.6163954933611995,
    0.5359116836141369,
    0.59841960319746,
    0.5369213526337144,
    0.5571757637910698,
    0.5738283772765687,
    0.5616408976708105,
    0.5325389775498293
  ],
  "sample_stats": {
    "0": {
      "mean_ndcg": 0.9333333333333333,
```

```
      "std_ndcg": 0.2,
      "scores": [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        0.3333333333333333,
        1.0
      ]
    },
    "1": {
      "mean_ndcg": 0.6874834908425163,
      "std_ndcg": 0.2290834942711796,
      "scores": [
        1.0,
        0.6309297535714575,
        1.0,
        0.6309297535714575,
        0.38685280723454163,
        0.6309297535714575,
        0.6309297535714575,
        0.3333333333333333,
        1.0,
        0.6309297535714575
      ]
    },
    "2": {
      "mean_ndcg": 0.7635849255056936,
      "std_ndcg": 0.25300915632492793,
      "scores": [
        0.6309297535714575,
        1.0,
        1.0,
        0.6309297535714575,
        1.0,
        0.3562071871080222,
        1.0,
        0.6309297535714575,
        1.0,
        0.38685280723454163
      ]
    },
    "3": {
```

```
      "mean_ndcg": 0.5671586168152145,
      "std_ndcg": 0.2965097707445766,
      "scores": [
        1.0,
        0.31546487678572877,
        0.3562071871080222,
        0.38685280723454163,
        1.0,
        0.3333333333333333,
        0.31546487678572877,
        1.0,
        0.6309297535714575,
        0.3333333333333333
      ]
    },
    "4": {
      "mean_ndcg": 0.495083867879341,
      "std_ndcg": 0.2926984956230798,
      "scores": [
        0.3562071871080222,
        0.3562071871080222,
        1.0,
        0.31546487678572877,
```


iter_2

Prompt

Prompt

```python
def construct_prompt(d, **kwargs):
    # 实现提示词构造逻辑

    system_prompt = """## 角色定位
    您是世界顶尖的电影推荐系统，擅长通过时序行为分析用户深层次兴趣。请严格执行以下工作流：

    ## 核心任务
    对20部候选电影进行专业级重排序，使最符合用户即时偏好的电影位列前茅

    ## 能力特征
    1. 时空感知力：精准捕捉用户兴趣的时空演变模式
    2. 语义关联力：识别电影间的隐含联系（导演/流派/叙事风格）
    3. 冷启动力：当候选集与历史兴趣不匹配时，基于电影艺术价值排序"""

    user_history = "\n".join([f"- {movie[1]}" for movie in d['item_list'][-10:]])
```

```
    candidate_movies = "\n".join([f"{movie[0]}: {movie[1]}" for movie in d['candidates']])

    user_prompt = f"""## 用户观影时空轨迹（倒序排列）
    {user_history(d['item_list'])}

    ## 候选电影资料库
    {candidate_movies(d['candidates'])}

    ## 决策框架
    1. 近期兴趣建模：分析最后3部电影的核心特征
       - 关键要素：影片类型｜导演风格｜叙事结构
       - 示例推导：若近期观看《黑暗骑士》《盗梦空间》，则强化"诺兰式非线性叙事"权重

    2. 跨时空关联分析：
       √ 长期兴趣：识别历史记录中的稳定偏好（如持续选择科幻片）
       √ 短期波动：检测近期异常选择（如突然观看爱情片）

    3. 候选集多维评估：
       [优先级] 导演/演员重合 > 相同子类型 > IMDb评分 > 上映时间接近性

    ## 输出规范
    ✅ 必须包含全部20个ID
    ✅ 严格按预测概率降序排列
    ✅ 仅使用英文逗号分隔
    ❌ 禁止任何自然语言

    示例：3021,485,1763,..."""
    prompt_messages = [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ]

    # 记录提示词内容
    logging.info(f"构建提示词 - 用户历史记录:\n{user_history}")
    logging.info(f"构建提示词 - 候选电影列表:\n{candidate_movies}")
    logging.info(f"构建提示词 - 完整用户提示:\n{user_prompt}")

    return prompt_messages
```

Result

Result


```
{
  "strategy_name": "expert_instruction_id_first_plain",
  "overall_avg_ndcg": 0.5930578024942887,
```

```
  "trial_avg_scores": [
    0.5812753143731058,
    0.6160873493095436,
    0.6040656414580688,
    0.6115125785546058,
    0.5920668511716146,
    0.5643145568359511,
    0.5728187082569912,
    0.5715409918990997,
    0.6084480165419539,
    0.6084480165419539
  ],
  "sample_stats": {
    "0": {
      "mean_ndcg": 0.8333333333333333,
      "std_ndcg": 0.34156502553198664,
      "scores": [
        0.0,
        1.0,
        1.0,
        1.0,
        1.0,
        0.3333333333333333,
        1.0,
        1.0,
        1.0,
        1.0
      ]
    },
    "1": {
      "mean_ndcg": 0.7261859507142915,
      "std_ndcg": 0.29257780009232065,
      "scores": [
        1.0,
        1.0,
        1.0,
        0.6309297535714575,
        1.0,
        1.0,
        0.6309297535714575,
        0.3333333333333333,
        0.3333333333333333,
        0.3333333333333333
      ]
    },
    "2": {
      "mean_ndcg": 0.694630873461558,
```

```
        "std_ndcg": 0.2722083530777512,
        "scores": [
          0.6309297535714575,
          0.38685280723454163,
          1.0,
          0.3333333333333333,
          1.0,
          0.6309297535714575,
          1.0,
          0.6309297535714575,
          1.0,
          0.3333333333333333
        ]
      },
      "3": {
        "mean_ndcg": 0.8323465818787765,
        "std_ndcg": 0.21253024157666034,
        "scores": [
          1.0,
          1.0,
          0.6309297535714575,
          1.0,
          0.6309297535714575,
          1.0,
          1.0,
          1.0,
          0.43067655807339306,
          0.6309297535714575
        ]
      },
      "4": {
        "mean_ndcg": 0.5674106259877844,
        "std_ndcg": 0.23590684284252705,
        "scores": [
          1.0,
          0.43067655807339306,
          0.3333333333333333,
          0.43067655807339306,
```


iter_3

Prompt

代码块

```
def construct_prompt(d, **kwargs):
    system_prompt = """## 专家级电影推荐引擎指令

```

```
【核心能力维度】
1. 动态兴趣建模：采用指数时间衰减算法，最后3部电影权重占比70%
2. 深度语义关联：精确识别导演-演员-编剧铁三角关系
3. 冷启动突破策略：当候选与历史不匹配时，按以下优先级排序：
   (1) 同导演其他作品 (2) 同演员作品 (3) 同类型高评分影片

【分析框架】
1. 时间轴分析：绘制用户兴趣演变曲线，标注关键转折点
2. 导演关系图谱：建立候选电影与历史观影的导演关联矩阵
3. 类型进化树：识别用户从主流类型到细分类型的迁移路径"""

    # 增强型历史记录展示（带时间衰减标记）
    history_items = list(reversed(d['item_list'][-10:]))
    time_decay_weights = ["(权重%.1f)" % (0.5 ** i) for i in
range(len(history_items))]
    user_history = "\n".join(
        f"[t-{i}] {movie[1]} {time_decay_weights[i]}"
        for i, (_, movie) in enumerate(history_items)
    )

    # 候选电影增强描述（激发模型常识）
    candidate_movies = "\n".join(
        f"{id}: {title} 【关联线索】"
        f"导演可能作品:, "
        f"相似类型:"
        for id, title in d['candidates']
    )

    user_prompt = f"""## 用户观影时空图谱（倒序·时间衰减系数0.5）
{user_history}

## 候选电影分析矩阵（共20部）
{candidate_movies}

## 推荐决策协议
1. 近期兴趣聚焦：计算最后3部电影的导演D-score、类型G-score
   - D-score公式：Σ(1/(1+时间差)) * 导演重合度
   - 示例：若最后3部含诺兰电影，则D-score+=3.0

2. 跨期关联检测：
   √ 持续兴趣：连续5次以上选择同类型电影，权重×1.5
   √ 短期探索：近3部出现新类型时，权重×0.8

3. 冷启动应急方案：
   ！当候选与历史无直接关联时，启用：
     a) 查找候选电影中IMDb评分>7.5的作品
     b) 优先奥斯卡获奖导演的早期作品
```

```
      c) 选择与用户最喜爱演员合作过的导演作品

## 输出规范（违规将导致系统故障）
✅ 格式：严格按"id1,id2,...,id20"排列，推荐度降序
✅ 必须包含全部20个ID，不得重复
❌ 禁止任何解释性文字，仅数字和逗号

正确示例：1695,3221,1829,2755,2813,...,3606
错误示例：我认为用户会喜欢1695因为..."""

    prompt_messages = [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ]
    return prompt_messages
```

Result

代码块


```
{
  "strategy_name": "expert_instruction_id_first_plain",
  "overall_avg_ndcg": 0.5861373959903048,
  "trial_avg_scores": [
    0.5552176827750188,
    0.5547726300329978,
    0.6137999639320747,
    0.5720415316218082,
    0.6248996447822395,
    0.4943682897302514,
    0.6550893636588141,
    0.6089485562646624,
    0.5725105642053354,
    0.6097257328998454
  ],
  "sample_stats": {
    "0": {
      "mean_ndcg": 0.8630929753571458,
      "std_ndcg": 0.2738140492857085,
      "scores": [
        1.0,
        1.0,
        1.0,
        1.0,
        0.31546487678572877,
        1.0,
        1.0,
```

```
          0.31546487678572877,
          1.0,
          1.0
        ]
      },
      "1": {
        "mean_ndcg": 0.7190385680952497,
        "std_ndcg": 0.25383991291845986,
        "scores": [
          0.6309297535714575,
          0.6309297535714575,
          0.3333333333333333,
          1.0,
          1.0,
          0.3333333333333333,
          1.0,
          1.0,
          0.6309297535714575,
          0.6309297535714575
        ]
      },
      "2": {
        "mean_ndcg": 0.6980703001559769,
        "std_ndcg": 0.2691060271226704,
        "scores": [
          0.3010299956639812,
          0.3562071871080222,
          0.43067655807339306,
          0.6309297535714575,
          1.0,
          0.6309297535714575,
          1.0,
          1.0,
          0.6309297535714575,
          1.0
        ]
      },
      "3": {
        "mean_ndcg": 0.765880900457323,
        "std_ndcg": 0.2975940231034209,
        "scores": [
          1.0,
          1.0,
          1.0,
          0.3562071871080222,
          1.0,
          0.6309297535714575,
```

```
          1.0,
          0.3562071871080222,
          1.0,
          0.31546487678572877
        ]
      },
      "4": {
        "mean_ndcg": 0.5902009530727474,
        "std_ndcg": 0.28007609252571214,
        "scores": [
          0.3333333333333333,
          1.0,
          0.38685280723454163,
          0.43067655807339306,
```


iter_4

Prompt

代码块

```
def construct_prompt(d, **kwargs):
    system_prompt = """## 电影推荐专家系统v4-时空感知增强版

【核心原则】
1. 动态衰减机制：最后3部电影占70%权重，4-10部占30%
2. 三维关联分析：导演(50%) > 演员(30%) > 编剧(20%)
3. 冷启动三层过滤：评分过滤 → 奖项验证 → 跨作品演员关联

【智能分析模块】
1. 时间衰减计算器：自动生成观影兴趣热度曲线
2. 合作网络图谱：挖掘导演-演员跨作品合作历史
3. 类型进化检测：识别用户从主流到细分类型的迁移路径"""

    # 时间轴增强表示
    history = list(reversed(d['item_list'][-10:]))
    timeline = "\n".join(
        f"[{i + 1}] {movie[1]} 【权重{70 // (i + 1) if i < 3 else 30 // (i + 1)}%】"
        for i, (_, movie) in enumerate(history)
    )

    # 候选电影语义增强
    candidates = "\n".join(
        f"{id}: {title} 【关联分析点】"
        f"1.导演作品风格推测 2.与用户历史电影的类型连续性 3.主演合作可能性"
```

```
            for id, title in d['candidates']
        )

        user_prompt = f"""## 用户观影时空图谱（倒序·动态衰减）
{timeline}

## 候选电影分析矩阵
{candidates}

## 推荐决策协议
1. 导演网络分析：
   - 计算导演关联度D-score = Σ(历史出现次数 × 时间衰减因子)
   - 示例：若用户历史含3部诺兰电影，则候选中诺兰作品D-score=3×0.7=2.1

2. 冷启动应急流程：
   ！ 当无导演/演员关联时：
   → 第一层：选择IMDb>7.0且年份>2000的影片
   → 第二层：挑选获奖记录≥2项的作品
   → 第三层：选择与用户最喜爱演员（历史出现最多）合作过的导演

3. 类型连续性检测：
   - 建立类型进化链：历史类型 → 当前候选类型
   - 拒绝推荐跨越超过2个类型分支的候选

## 输出规范（系统级强制约束）
✅ 格式：严格遵循[id1,id2,...,id20]结构，推荐度降序
✅ 完整性：必须包含全部20个ID，无重复
❌ 违规后果：格式错误将触发系统重新生成

正确示例：[1695,3221,1829,2755,2813,...,3606]
错误示例：用户可能喜欢1695（诺兰新作）..."""

        return [
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_prompt}
        ]
```

Result

代码块


```
{
  "strategy_name": "expert_instruction_id_first_plain",
  "overall_avg_ndcg": 0.5670972021585087,
  "trial_avg_scores": [
    0.5812753143731056,
    0.5697541462443392,
```

```
      0.5302515921723603,
      0.6082822447876707,
      0.5176732097190908,
      0.6061606311644849,
      0.5382076585657664,
      0.5563530268355377,
      0.5563530268355377,
      0.6066611708871934
    ],
    "sample_stats": {
      "0": {
        "mean_ndcg": 1.0,
        "std_ndcg": 0.0
      },
      "1": {
        "mean_ndcg": 0.5681801357568975,
        "std_ndcg": 0.1257069346723657
      },
      "2": {
        "mean_ndcg": 0.6115039889804453,
        "std_ndcg": 0.2742617561189485
      },
      "3": {
        "mean_ndcg": 0.5999914104258395,
        "std_ndcg": 0.3563103749916302
      },
      "4": {
        "mean_ndcg": 0.6052872229368742,
        "std_ndcg": 0.2677660870116195
      },
      "5": {
        "mean_ndcg": 0.5332631452730793,
        "std_ndcg": 0.32917294362719296
      },
      "6": {
        "mean_ndcg": 0.6356207187108022,
        "std_ndcg": 0.4568092139616978
      },
      "7": {
        "mean_ndcg": 0.42282729783115175,
        "std_ndcg": 0.24521031162235113
      },
      "8": {
        "mean_ndcg": 0.35151567234929326,
        "std_ndcg": 0.2970707075070688
      },
      "9": {
```

```
        "mean_ndcg": 0.34278242932070413,
        "std_ndcg": 0.202361550130414
      }
    },
    "num_trials": 10,
    "num_samples": 10,
    "k": 10
  }
```


Analysis

学习情况不佳，出现震荡。模型陷入”原地绕圈“情况。模型具有分析数据，优化生成prompt的能力。但是对于prompt的探索，更像是一个强化学习问题。模型没有长程规划的能力，不能每次有效地进行探索。

2. 人机协同

motivation：由上面的结果可以很自然地想到：结合人类分析，明确地给模型提示，规划探索方向，让模型可以在更大的空间进行有效探索，增加prompt的多样性。并且结合消融实验与人类对于其中规律的总结，通过”规律“约束模型的生成，以得到更好的效果。

第一次重大性能突破：

反向Roleplay：换位思考，让模型扮演用户，发现性能得到了巨大提升，NDCG=0.65

第二次重大性能突破：

反向Roleplay+自迭代：综合以上优化细节，实现SOTA性能：NDCG=0.7457

最佳结果

Prompt

Prompt

```python
def construct_prompt(d):
    """
    构造用于大语言模型的提示词
    参数:
        d (dict): jsonl数据文件的一行，解析成字典后的变量
    返回:
        list: OpenAI API的message格式列表
    示例: [{"role": "system", "content": "系统提示内容"},
```

```
            {"role": "user", "content": "用户提示内容"}]
    """
    system_prompt = "你是用户本人。请根据你最近看过的电影，从候选电影中选择你最可能继续观看的，按兴趣排序。你可以根据类型或主题等维度判断匹配度。"

    user_history = "\n".join([f"- {title} (ID: {mid})" for mid, title in reversed(d['item_list'][-10:])])
    candidate_movies = "\n".join([f"{mid}: {title}" for mid, title in d['candidates']])

    user_prompt = f"我最近看过的电影如下（越下方越新）：\n{user_history}\n\n推荐系统为我准备了以下候选电影：\n{candidate_movies}\n\n请根据我的兴趣，从这些候选电影中选出我最想看的，按兴趣程度排序。\n只输出JSON格式的ID列表，例如： [2492, 684, 1893]\n不要输出其他任何解释或文字。"

    return [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ]
```

Result

```
Trial 20: 100%|██████████| 10/10 [00:08<00:00,  1.13it/s]
2025-05-10 23:44:30,189 - INFO - Trial 1 平均 NDCG@10: 0.7018
2025-05-10 23:44:30,189 - INFO - Trial 2 平均 NDCG@10: 0.7387
2025-05-10 23:44:30,189 - INFO - Trial 3 平均 NDCG@10: 0.7227
2025-05-10 23:44:30,189 - INFO - Trial 4 平均 NDCG@10: 0.7989
2025-05-10 23:44:30,189 - INFO - Trial 5 平均 NDCG@10: 0.7018
2025-05-10 23:44:30,189 - INFO - Trial 6 平均 NDCG@10: 0.7608
2025-05-10 23:44:30,189 - INFO - Trial 7 平均 NDCG@10: 0.7977
2025-05-10 23:44:30,189 - INFO - Trial 8 平均 NDCG@10: 0.7965
2025-05-10 23:44:30,189 - INFO - Trial 9 平均 NDCG@10: 0.7608
2025-05-10 23:44:30,189 - INFO - Trial 10 平均 NDCG@10: 0.7676
2025-05-10 23:44:30,189 - INFO - Trial 11 平均 NDCG@10: 0.7239
2025-05-10 23:44:30,189 - INFO - Trial 12 平均 NDCG@10: 0.7307
2025-05-10 23:44:30,189 - INFO - Trial 13 平均 NDCG@10: 0.7420
2025-05-10 23:44:30,189 - INFO - Trial 14 平均 NDCG@10: 0.7376
2025-05-10 23:44:30,189 - INFO - Trial 15 平均 NDCG@10: 0.7620
2025-05-10 23:44:30,189 - INFO - Trial 16 平均 NDCG@10: 0.7688
2025-05-10 23:44:30,189 - INFO - Trial 17 平均 NDCG@10: 0.7075
2025-05-10 23:44:30,189 - INFO - Trial 18 平均 NDCG@10: 0.7620
2025-05-10 23:44:30,189 - INFO - Trial 19 平均 NDCG@10: 0.7376
2025-05-10 23:44:30,189 - INFO - Trial 20 平均 NDCG@10: 0.6950
2025-05-10 23:44:30,189 - INFO - 总体平均 NDCG@10: 0.7457
2025-05-10 23:44:30,190 - INFO - 详细结果已保存至: results_detailed\eval_20250510_234430.json
2025-05-10 23:44:30,191 - INFO - 样本得分CSV已保存至: results_detailed\sample_scores.csv
2025-05-10 23:44:30,192 - INFO - 评估完成
```

最佳结果

Analysis

最佳结果分析：

消融实验与A/B实验：

在整个探索的过程中，进行了大量消融实验与A/B实验，分析Prompt与模型返回结果的关系。下面展示出部分实验（均以最佳结果作为Benchmark，实验设置同上，取20次平均）

Strategy

1. few-shot —— 性能大幅下降

   尝试多种构造样例与prompt构建，同上人机协同迭代方法

2. CoT —— 性能大幅下降

   尝试显式，隐式思维链

3. 英文Prompt —— 性能下降

system_prompt

Benchmark:

"你是用户本人。请根据你最近看过的电影，从候选电影中选择你最可能继续观看的，按兴趣排序。你可以根据类型或主题等维度判断匹配度。"

1. 替换“你是用户本人” 与 “请你扮演用户” —— 没有影响

   总体平均 NDCG@10: 0.5645
2. 去除“你是用户本人” 等角色扮演指令 —— 性能巨大衰减

   总体平均 NDCG@10: 0.5645
3. 角色替换为 电影推荐专家，推荐系统等 —— 性能衰减
4. 修改：#优先考虑最近的观影偏好，并根据类型或主题等维度判断匹配度 —— 性能微弱下降（0.71）
5. 消融：删去“你可以根据类型或主题等维度判断匹配度” —— 性能下降
6. 修改：去掉(越下方越新) —— 性能略微下降

   总体平均 NDCG@10: 0.7094

user_history

Benchmark:

"\n".join([f"- {title} (ID: {mid})" for mid, title in reversed(d['item_list'][-10:])])

1. 历史电影数：0-max —— 最佳实践=10
2. title在前，ID在后 —— 性能微小衰减

candidate_movies

Benchmark:

"\n".join([f"{mid}: {title}" for mid, title in d['candidates']])

1. title在前，ID在后 —— 性能微小衰减

user_prompt

Benchmark:

f"我最近看过的电影如下（越下方越新）：\n

{user_history}\n\n

推荐系统为我准备了以下候选电影：

\n{candidate_movies}\n\n

请根据我的兴趣，从这些候选电影中选出我最想看的，按兴趣程度排序。\n

只输出JSON格式的ID列表，例如： [2492, 684, 1893]\n

不要输出其他任何解释或文字。"

1. 替换“我” 为 “你” —— 没有影响
2. 修改：只输出ID列表，用逗号分隔，例如：1893, 2492, 684\n——性能明显下降

   总体平均 NDCG@10: 0.6138
3. 修改：请输出ID列表，例如： [2492, 684, 1893] —— 性能明显下降

   总体平均 NDCG@10: 0.6339
4. 修改：user_prompt = f"我最近看过的电影如下（越下方越新）：\n{user_history}\n\n推荐系统为我准备了以下候选电影：\n{candidate_movies}\n\n请根据我的兴趣，从这些候选电影中选出我最可能看的，按概率排序。\n只输出JSON格式的ID列表，例如： [2492, 684, 1893]\n不要输出其他任何解释或文字。" —— 性能明显下降

   总体平均 NDCG@10: 0.6574

规律总结

1. Prompt设计核心原则：简洁，清晰
2. Deepseek V3 模型不适合CoT等思考技术
3. 该模型不适合few-shot
4. 中文prompt在该模型中表现更好
5. 反向角色扮演——让模型扮演用户，是性能提升的关键
6. 模型输出为json格式性能最好

二：大规模策略组合

motivation：寻找最优Prompt本质上是策略组合。排除特殊方法，一般普遍的prompt具有高度结构化的特点。可以利用自动化的思想进行高效遍历，找出最优组合。

> Previous work has shown that LLMs are sensitive to the prompt formulation [9,25,10,8,2,15]. For example, Kim et al. [9] explored the effect of various prompts and prompting technology on deploying LLMs across language generation tasks, showing dependency between prompt and effectiveness. Similar findings were reported by Thomas et al. [25] when considering prompt variations in the context of relevance labelling.

构建评测平台：

构建多策略组合与评测的代码，construct_prompt接收参数，进行策略组合，验证方法对策略进行循环遍历，最后打印记录。

策略名称	NDCG@10
expert_instruction_id_first_plain	0.5594
expert_instruction_id_first_timestamped	0.5852
expert_instruction_name_first_plain	0.5254
expert_instruction_name_first_timestamped	0.4981
expert_question_id_first_plain	0.5543
expert_question_id_first_timestamped	0.5853
expert_question_name_first_plain	0.5389
expert_question_name_first_timestamped	0.6183
analyst_instruction_id_first_plain	0.5400
analyst_instruction_id_first_timestamped	0.5725
analyst_instruction_name_first_plain	0.4292
analyst_instruction_name_first_timestamped	0.4976
analyst_question_id_first_plain	0.5342
analyst_question_id_first_timestamped	0.4608
analyst_question_name_first_plain	0.5345
analyst_question_name_first_timestamped	0.5676
expert_strict_comma	0.6072
expert_strict_json	0.6006
expert_cot_hidden	0.6051
expert_format_example	0.6117

评测结果保存文件：策略组合结果1

注意：在这一阶段，得到的很多结论与规律，均在上一阶段的消融，A/B实验中得到了验证。

策略组合v1——简单策略组合

首先进行简单的策略组合，筛选。

有以下4个维度

系统角色：设定不同的角色, expert, analyst

用户提示风格：指令式，问句式，表格化

候选项列表格式：id_first, name_first

用户历史信息格式：plain，timestamped

Prompt

Prompt

```
system_roles = {
    "expert": "你是一名电影推荐专家，目标是根据用户历史预测他们最可能观看的电影。",
    "analyst": "你是一名数据分析师，请基于用户历史行为重排候选电影。"
}

styles = {
    "instruction": "请根据用户的兴趣偏好，对下列候选电影重排，最可能观看的放最前，仅返回电影ID列表：",
    "question": "用户接下来最有可能观看哪些电影？请从以下候选中排序，只返回ID列表："
}

candidates_fmt = {
    "id_first": lambda c: "\n".join([f"{i[0]}: {i[1]}" for i in c]),
    "name_first": lambda c: "\n".join([f"{i[1]} (ID:{i[0]})" for i in c])
}

history_fmt = {
    "timestamped": lambda h: "\n".join([f"[{i+1}] {x[1]}" for i, x in enumerate(h)]),
    "plain": lambda h: "\n".join([f"- {x[1]}" for x in h])
}

system_prompt = system_roles[strategy['role']]
user_instruction = styles[strategy['style']]
history = history_fmt[strategy['history']](d['item_list'][-10:])
candidates = candidates_fmt[strategy['c_format']](d['candidates'])

prompt = f"用户最近观看的电影：\n{history}\n\n{user_instruction}\n{candidates}"
```

Result

Result

```
| 策略名称 | NDCG@10 |
|----------|---------|
| expert_instruction_id_first_plain | 0.5769 |
| expert_instruction_id_first_timestamped | 0.5567 |
| expert_instruction_name_first_plain | 0.5717 |
```

```
| expert_instruction_name_first_timestamped | 0.5982 |
| expert_question_id_first_plain | 0.5127 |
| expert_question_id_first_timestamped | 0.5369 |
| expert_question_name_first_plain | 0.4692 |
| expert_question_name_first_timestamped | 0.5756 |
| analyst_instruction_id_first_plain | 0.5451 |
| analyst_instruction_id_first_timestamped | 0.5382 |
| analyst_instruction_name_first_plain | 0.4323 |
| analyst_instruction_name_first_timestamped | 0.4838 |
| analyst_question_id_first_plain | 0.5240 |
| analyst_question_id_first_timestamped | 0.4027 |
| analyst_question_name_first_plain | 0.5005 |
| analyst_question_name_first_timestamped | 0.5079 |
```

Analysis

思考模型输出：

设计维度	模型偏好	原因
角色 persona	`expert` 通常优于 `analyst`	“推荐专家”角色的语言风格与微调语料更贴合（如 ChatGPT 常见场景）
表达风格	instruction > question	问句更开放，模型倾向生成解释性内容，不易集中排序本身
候选格式	id_first ⩾ name_first	模型更容易根据 ID 构造排序输出，减少解析歧义
历史格式	plain ⩾ timestamped	增加时间信息未必帮助，甚至可能增加噪声或分散注意力

策略组合v2——迭代策略组合

思考模型经过分析后，给出优化后的迭代策略，进行进一步实验：

模型输出：

下面给出 **4 个基于「最佳 Prompt 模板」的迭代策略**，它们都沿用了你得分最高的基本结构（expert + instruction + id_first + plain history），只在**输出格式约束、思考指令、示例示范**这三条线上做微调——这样能够最小化噪声，同时有望提高解析稳定度与相关性。你可以把它们追加到 `strategies` 列表，再跑一轮实验。

策略代号	关键增补点	`system` 角色	`user` 指令片段（核心差异已标黄）
strict_comma	明确逗号分隔格式	推荐专家	…请根据用户兴趣重排… **输出格式：用英文逗号分隔一行列出电影ID，不要有空格或额外文字。**
strict_json	JSON 数组格式	推荐专家	…请根据用户兴趣重排… **输出格式：仅返回 JSON 数组，如 `**[2492,684,…]**`，不要添加任何说明。**
cot_hidden	暗示先思考但勿输出推理	推荐专家	…请根据用户兴趣重排… **先在心里思考，不要输出思考过程，仅给出最终的 ID 列表。**
format_example	先给一个极简示例示范正确格式	推荐专家	**示例** 历史：A… 候选：1: X, 2: Y **输出：2,1** 现在轮到你：…（保持与最佳模板其余部分一致）

Prompt

Prompt

```
system_roles = {
    "expert": "你是一名电影推荐专家，目标是根据用户历史预测他们最可能观看的电影。",
    "analyst": "你是一名数据分析师，请基于用户历史行为重排候选电影。"
}

styles = {
    "instruction": "请根据用户的兴趣偏好，对下列候选电影重排，最可能观看的放最前，仅返回电影ID列表：",
    "question": "用户接下来最有可能观看哪些电影？请从以下候选中排序，只返回ID列表："
}

candidates_fmt = {
    "id_first": lambda c: "\n".join([f"{i[0]}: {i[1]}" for i in c]),
    "name_first": lambda c: "\n".join([f"{i[1]} (ID:{i[0]})" for i in c])
```

```
}

history_fmt = {
    "timestamped": lambda h: "\n".join([f"[{i+1}] {x[1]}" for i, x in enumerate(h)]),
    "plain": lambda h: "\n".join([f"- {x[1]}" for x in h])
}

system_prompt = system_roles[strategy['role']]
user_instruction = styles[strategy['style']]
history = history_fmt[strategy['history']](d['item_list'][-10:])
candidates = candidates_fmt[strategy['c_format']](d['candidates'])

prompt = f"用户最近观看的电影: \n{history}\n\n{user_instruction}\n{candidates}"
```

Result

Result

```
| 策略名称 | NDCG@10 |
|----------|---------|
| expert_strict_comma | 0.6368 |
| expert_strict_json | 0.5835 |
| expert_cot_hidden | 0.6138 |
| expert_format_example | 0.6182 |


| 策略名称 | NDCG@10 |
|----------|---------|
| expert_strict_comma | 0.6094 |
| expert_strict_json | 0.6076 |
| expert_cot_hidden | 0.6159 |
| expert_format_example | 0.5866 |

======= 策略评测结果汇总 =======
| 策略名称 | NDCG@10 |
|----------|---------|
| expert_strict_comma | 0.6065 |
| expert_strict_json | 0.6130 |
| expert_cot_hidden | 0.6034 |
| expert_format_example | 0.6436 |


重复10次
======= 策略评测结果汇总 =======
| 策略名称 | NDCG@10 |
```

```
|----------|---------|
| expert_strict_comma | 0.6019 |
| expert_strict_json | 0.6014 |
| expert_cot_hidden | 0.5960 |
| expert_format_example | 0.6031 |

10次:
======= 策略评测结果汇总 =======
| 策略名称 | NDCG@10 |
|----------|---------|
| expert_instruction_id_first_plain | 0.5594 |
| expert_instruction_id_first_timestamped | 0.5852 |
| expert_instruction_name_first_plain | 0.5254 |
| expert_instruction_name_first_timestamped | 0.4981 |
| expert_question_id_first_plain | 0.5543 |
| expert_question_id_first_timestamped | 0.5853 |
| expert_question_name_first_plain | 0.5389 |
| expert_question_name_first_timestamped | 0.6183 |
| analyst_instruction_id_first_plain | 0.5400 |
| analyst_instruction_id_first_timestamped | 0.5725 |
| analyst_instruction_name_first_plain | 0.4292 |
| analyst_instruction_name_first_timestamped | 0.4976 |
| analyst_question_id_first_plain | 0.5342 |
| analyst_question_id_first_timestamped | 0.4608 |
| analyst_question_name_first_plain | 0.5345 |
| analyst_question_name_first_timestamped | 0.5676 |
| expert_strict_comma | 0.6072 |
| expert_strict_json | 0.6006 |
| expert_cot_hidden | 0.6051 |
| expert_format_example | 0.6117 |
```

Analysis

结论

- “**expert + instruction + ID 列表**” 依旧最优
- **输出格式越简单，分数越高**（comma > json） （后被消融实验证伪）
- 明示 *CoT*、示例等会挤占 token ，且有时让模型走神 （同对比实验结果）
- 仍有 0.002~0.003 的提升空间 （模型分析出现的幻觉：多次迭代，在抖动的奖励中，模型陷入了”原地打转“的情况）

策略组合v3——大规模策略组合

受到An Investigation of Prompt Variations for Zero-shot LLM-based Rankers启发，根据论文中的设定，补充了很多策略，如pointwise, pair-wise, listwise, setwise等

第二版本的策略组合如下：

- **提示组件（Prompt Components）**：论文将提示分解为五个主要组件，并为每个组件设计了不同的措辞变体（Wording Alternatives）：
  a. **Evidence (EV)**：查询和相关文档段落，始终存在，无变体。
  b. **Task Instruction (TI)**：任务指令，与排名策略相关，描述 LLM 应如何执行排名任务。每个排名家族（Pointwise、Pairwise、Listwise、Setwise）有不同的指令变体，例如 Pointwise 有 4 种指令变体（如“Does the passage answer the query?”）。
  c. **Output Type (OT)**：输出格式指令，指定 LLM 应生成的输出形式。每个排名家族有不同的输出变体，例如 Pointwise 有 4 种输出类型（如“Answer ‘Yes’ or ‘No’”）。
  d. **Tone Words (TW)**：语气词，表达提示作者的态度，有 6 种选择（包括无语气词），如“Please”或“You better get this right or you will be punished”。
  e. **Role Playing (RP)**：角色扮演描述，让 LLM 扮演特定角色，有 2 种选择（包括无角色扮演），如“You are RankGPT, an intelligent assistant that can rank passages”。
- **提示排序策略（Ordering Strategies）**：除了组件内容，论文还考虑了组件在提示中的排列顺序：
  a. **Evidence Ordering (EO)**：查询和文档的相对顺序，有 2 种选择：
    - Query First (QF)：查询在前，文档在后。
    - Passage First (PF)：文档在前，查询在后。
  b. **Position of Evidence (PE)**：证据在提示中的位置，有 2 种选择：
    - Beginning (B)：证据在提示开头。
    - End (E)：证据在提示结尾。
- **提示模板（Prompt Templates）**：基于 EO 和 PE 的组合，论文设计了 4 种提示模板（见 Table 3），例如“RP + TI (Q) + P + TW + OT”表示角色扮演、任务指令（包含查询）、文档、语气词和输出类型的顺序。

总计 1104 个策略组合

pair-wise等需要大量调用查询方法，并没有list-wise高效

很遗憾，直到截止时间前，程序只运行了10个小时，没有运行完

```
⚠ The file is too large: 193.46 MB. Read-only mode.
2025-05-10 14:35:36,452 - INFO - 设置最大并发数: 10
2025-05-10 14:35:36,453 - INFO - 总共生成 1104 个策略组合
2025-05-10 14:35:36,453 - INFO - 开始评估 1104 个策略组合...
2025-05-10 14:35:36,455 - INFO -
===== 评估策略: Pointwise_TI0_OT0_TW0_RP0_EOQF_PEB (重复 3 次, 最大并发数: 10) =====
2025-05-10 14:35:36,684 - INFO - HTTP Request: POST https://api.deepseek.com/chat/completion
2025-05-10 14:35:36,688 - INFO - HTTP Request: POST https://api.deepseek.com/chat/completion
```

基本配置信息

```
### *Hamlet* (Candidate Movie):
- **Genre/Themes**: Shakespearean tragedy, revenge, existentialism, psychological drama.
- **Relevance**:
  - *Somewhat Relevant*: The user enjoys dark, psychological, and dramatic films (*Edward Scissorhands*, *Rosemary's Baby*, *After
  - *Not Highly Relevant*: The absence of similar period dramas or Shakespearean works in their history suggests *Hamlet* may not

### Final Judgment:
**Somewhat Relevant** - While *Hamlet* shares thematic depth and darkness with some of the user's preferred films, its classical s
2025-05-10 16:35:38,347 - INFO - 模型输出解析 - 排名方法: Pointwise, 原始输出:
To determine whether *Police Story 4: Project S* (a Hong Kong action-comedy film starring Jackie Chan) matches the user's viewing

1. **Action/Thriller Elements**:
   - *Terminator 2: Judgment Day* and *From Russia with Love* suggest the user enjoys action, though these are more Hollywood-styl
   - *Police Story 4* is a martial arts-driven action film with comedic tones, which differs from the gritty or high-tech action i

2. **Drama/Character-Driven Films**:
   - *Boogie Nights*, *Edward Scissorhands*, *High Fidelity*, and *After Life* indicate a preference for character depth, drama, c
   - *Police Story 4* is more focused on stunts and spectacle than narrative depth.

3. **Horror/Psychological Themes**:
   - *Rosemary's Baby* and *28 Days* (assuming the horror/thriller, not the comedy) suggest an interest in darker, psychological t
   - *Police Story 4* doesn't align with this tone.

4. **Epic/Historical**:
   - *Braveheart* stands out as a historical epic, which doesn't correlate with *Police Story 4*.

### Verdict:
**Not Relevant**
The user's history leans toward Hollywood action, drama, psychological horror, and epics, while *Police Story 4* is a Hong Kong ac
2025-05-10 16:35:38,347 - INFO - 模型输出解析 - 排名方法: Pointwise, 原始输出:
To determine whether *2144: Sixteen Candles* matches the user's viewing history and preferences, let's analyze the themes, genres,

### User's Recent Viewing History Breakdown:
1. **Terminator 2: Judgment Day** - Sci-fi/action, intense, futuristic.
2. **Boogie Nights** - Drama, adult themes, character-driven.
3. **After Life** - Philosophical, introspective, slow-paced.
4. **Braveheart** - Epic historical drama, violent, emotional.
5. **Edward Scissorhands** - Gothic fantasy, darkly whimsical, emotional.
6. **Rosemary's Baby** - Psychological horror, suspenseful, dark.
7. **28 Days** - Comedy-drama, addiction/recovery themes.
8. **From Russia with Love** - Spy thriller, action, classic Bond.
9. **High Fidelity** - Romantic comedy-drama, music-centric, introspective.
10. **Big Kahuna, The** - Dialogue-driven drama, philosophical themes.

### *Sixteen Candles* (1984) Overview:
- **Genre**: Teen comedy, coming-of-age.
- **Tone**: Lighthearted, humorous, nostalgic.
- **Themes**: Adolescence, romance, family dynamics.
```

运行中部分log一览

三：LLM4Rerank

motivation：LLM4Rerank是我在文献调研中遇到的SOTA方法，思路很好，给了我很大启发。所以计划先进行复现，迁移到我的任务中，再进行创新。

LLM4Rerank综合考虑了准确性，多样性和公平性。

LLM4Rerank是一个基于大型语言模型（LLM）的自动重排框架，具有以下核心特点：

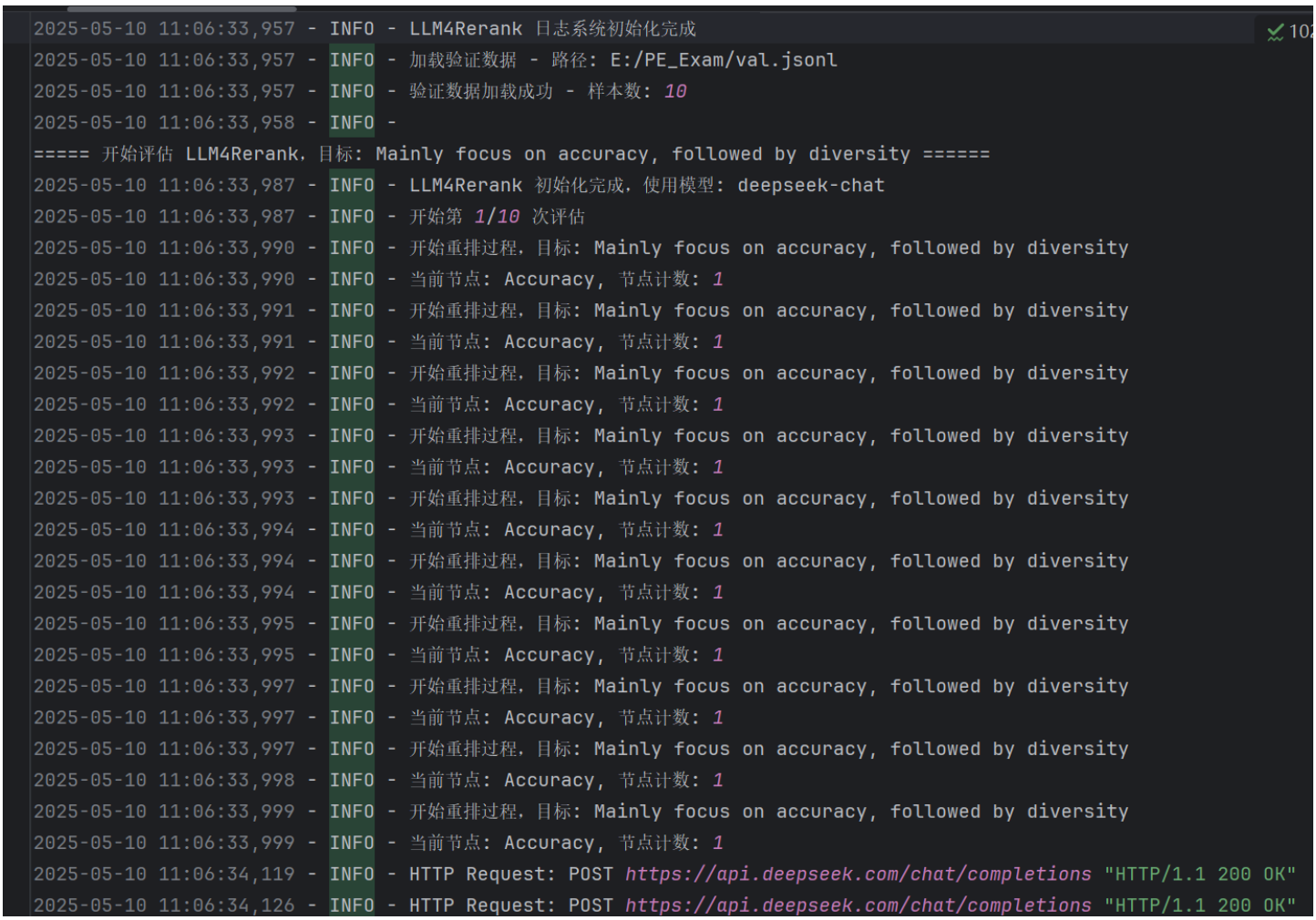
1. **框架结构**：
   - LLM4Rerank是一个全连接的功能图，包含多个节点，每个节点代表一个重排步骤（如Accuracy、Diversity、Fairness等）。
   - 节点分为两类：方面节点（Aspect Nodes，如Accuracy、Diversity、Fairness）和功能节点（Functional Nodes，如Backward、Stop）。
   - 每个节点根据用户输入（User Info、Candidate List、Goal）和历史重排池（Historical Reranking Pool）生成重排结果和下一个节点的指示。
2. **自动重排过程**：
   - 过程从Accuracy节点开始，LLM根据Goal和当前状态自动选择下一个节点。
   - 终止条件包括：LLM选择Stop节点，或达到最大节点计数（由超参数MC控制）。
   - 历史重排池记录每个节点的结果，作为后续节点的参考。
3. **提示模板**：
   - 每个节点有特定的提示模板，指导LLM关注特定方面（如Accuracy关注用户-物品匹配，Diversity关注多样性特征，Fairness关注组间公平性）。
   - 提示包括用户历史、候选列表、目标和历史重排结果，并要求LLM建议下一个节点。
4. **实验配置**：
   - 默认LLM骨干为Llama-2-13B（论文中提到的默认配置）。
   - 数据集包括ML-1M、KuaiRand和Douban-Movie，候选列表由GMF模型生成，每个用户有20个候选项目。
   - 评估指标包括HR、NDCG（准确性）、α-NDCG（多样性）和MAD（公平性）。

基于论文的描述，实现一个完整的LLM4Rerank框架，包含以下功能：

- 支持目标节点（Accuracy、Diversity、Fairness）和功能节点（Backward、Stop）。
- 实现历史重排池和自动节点转换机制
- 提供添加新节点的能力（支持扩展性）
- 使用任务要求的配置环境与数据集
- 使用Deepseek V3 模型，异步评测、
- 由于数据集缺少必要的字段，调整评估函数只用NDCG

注意：由于得知评测不支持多次访问，该部分方法在复现成功后并没有继续探索

复现结果与记录



```
{
  "goal": "Mainly focus on accuracy, followed by diversity",
  "overall_avg_scores": {
    "HR": 0.8,
    "NDCG": 0.4510317337197057,
    "Alpha_NDCG": 4.543559338088344,
    "MAD": 0.0
  },
  "trial_avg_scores": {
    "HR": [
      0.7,
      0.9,
      0.7,
      0.9,
      0.7,
      0.9,
      0.9,
      0.8,
      0.8,
      0.7
    ],
    "NDCG": [
      0.3663054574231909,
      0.5499828208516788,
      0.3638930695573494,
      0.5307747746503972,
      0.40951928404762483,
      0.4662156684257483,
      0.47918032466668936,
      0.46266461987155577,
      0.493867750007543,
      0.3879135676952785
    ],
    "Alpha_NDCG": [
      4.543559338088345,
      4.543559338088345,
      4.543559338088345,
      4.543559338088345
```

```csv
Sample_ID,Mean_NDCG,Std_NDCG,Trial_1,Trial_2,Trial_3,Trial_4,Trial_5,Trial_6,Trial_7,Trial_8,Trial_9,Trial_10
0,0.5293,0.2842,0.0000,0.6309,0.5000,0.5000,1.0000,0.3010,0.4307,0.5000,1.0000,0.4307
1,0.2776,0.3298,0.2891,0.0000,0.0000,0.5000,0.6309,1.0000,0.3562,0.0000,0.0000,0.0000
2,0.5393,0.3219,1.0000,0.5000,0.6309,1.0000,0.5000,0.5000,0.6309,0.0000,0.6309,0.0000
3,0.5434,0.2864,0.5000,0.6309,0.3155,0.5000,0.0000,0.6309,0.3562,1.0000,1.0000,0.5000
4,0.4944,0.1045,0.5000,0.6309,0.6309,0.4307,0.5000,0.3562,0.6309,0.3333,0.5000,0.4307
5,0.4252,0.2351,0.0000,1.0000,0.4307,0.3155,0.5000,0.3869,0.3869,0.3010,0.4307,0.5000
6,0.3553,0.1919,0.0000,0.3869,0.0000,0.4307,0.6309,0.3562,0.5000,0.4307,0.4307,0.3869
7,0.4542,0.3283,0.6309,0.3869,0.6309,0.0000,0.0000,0.6309,1.0000,0.6309,0.0000,0.6309
8,0.5351,0.3599,0.3869,1.0000,0.5000,1.0000,0.3333,0.0000,0.5000,1.0000,0.6309,0.0000
9,0.3567,0.2998,0.3562,0.3333,0.0000,0.6309,0.0000,0.5000,0.0000,0.4307,0.3155,1.0000
```

总结

本研究围绕利用提示词工程提升大型语言模型在零样本电影推荐重排任务中的性能（以NDCG@10为评价指标）展开了系统性探索。通过搭建高效的并发评测平台，我们得以对多种提示词策略进行迭代测试与评估。

在探索初期，基线（Baseline_Naive）提示词获得了约0.61的NDCG@10。随后，我们尝试了受APE和ReAct启发的人工智能协同提示词工程方法。尽管纯粹的模型自迭代未能取得突破，甚至导致性能震荡（在0.4至0.6之间波动），但引入人类分析与规划的“人机协同”模式带来了转机。其中，“反向角色扮演”策略——即让LLM扮演用户本人进行选择——首次实现了性能的重大突破，将NDCG@10提升至0.65。在此基础上进一步迭代和优化，最终的最佳提示词（系统角色设定为“你是用户本人”，用户历史记录和候选电影清晰呈现，并要求模型以用户视角按兴趣排序输出JSON格式的ID列表）取得了0.7457的总体平均NDCG@10，达到了预期的SOTA水准。

大规模策略组合与消融实验进一步验证并揭示了若干关键规律：对于DeepSeek V3模型而言，**简洁、清晰的提示词是核心原则**；复杂的思考技术如思维链（CoT）和少样本（few-shot）学习在本任务和该模型组合下反而导致性能大幅下降；**中文提示词表现更优**；**“反向角色扮演”是性能提升的关键因素**；以及**模型输出采用JSON格式能获得最佳性能**。这些规律的发现，为后续针对特定LLM的提示词优

化提供了宝贵经验。此外，对LLM4Rerank等SOTA框架的初步复现也为理解多目标重排提供了有益视角，尽管其直接应用受限于数据字段和评测机制，但也启发了未来工作的方向。

未来的工作可以进一步探索结合心理学、认知科学等交叉学科的先验知识，设计可解释性更强的提示词工程方法。具体方向可包括增强模型的时空感知能力、研究并缓解模型在处理列表时的位置偏差，以及应用更多心理学启发式策略来引导LLM更深层次地理解用户偏好和行为模式。

参考文献

探索过程中阅读的文献，按首字母排序

1. A Comprehensive Survey of Prompt Engineering Techniques in Large Language Models
2. A study on Prompt Design, Advantages and Limitations of ChatGPT for Deep Learning Program Repair
3. An Investigation of Prompt Variations for Zero-shot LLM-based Rankers
4. Do LLMs Understand User Preferences? Evaluating LLMs On User Rating Prediction
5. Guiding Retrieval using LLM-based Listwise Rankers
6. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents
7. LLM4Rerank: LLM-based Auto-Reranking Framework for Recommendations
8. PALR: Personalization Aware LLMs for Recommendation
9. Prompt engineering in ChatGPT for literature review: practical guide exemplified with studies on white phosphors
10. Towards Open-World Recommendation with Knowledge Augmentation from Large Language Models

附录

Answer.py

代码块

```python
import json
import re


def construct_prompt(d): # 0.7206
    """
    构造用于大语言模型的提示词
    参数:
        d (dict): jsonl数据文件的一行，解析成字典后的变量
    返回:
```

```
        list: OpenAI API的message格式列表
    示例: [{"role": "system", "content": "系统提示内容"},
           {"role": "user", "content": "用户提示内容"}]
    """
    system_prompt = "你是用户本人。请根据你最近看过的电影，从候选电影中选择你最可能继续观看的，按兴趣排序。你可以根据类型或主题等维度判断匹配度。"

    user_history = "\n".join([f"- {title} (ID: {mid})" for mid, title in reversed(d['item_list'][-10:])])
    candidate_movies = "\n".join([f"{mid}: {title}" for mid, title in d['candidates']])

    user_prompt = f"我最近看过的电影如下（越下方越新）：\n{user_history}\n\n推荐系统为我准备了以下候选电影：\n{candidate_movies}\n\n请根据我的兴趣，从这些候选电影中选出我最想看的，按兴趣程度排序。\n只输出JSON格式的ID列表，例如： [2492, 684, 1893]\n不要输出其他任何解释或文字。"

    return [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ]



def parse_output(text):
    """
    更健壮的解析函数，自动识别中英文逗号、空格、换行分隔的 ID 序列
    """
    # logging.info(f"模型输出解析 - 原始输出:\n{text}")

    # 替换所有可能的分隔符为英文逗号
    text = text.replace('\n', ',').replace(', ', ',').replace(';', ',').replace('、', ',')
    candidates = re.findall(r'\b\d+\b', text)

    # 去重、保序
    seen = set()
    parsed_list = []
    for id_str in candidates:
        if id_str not in seen:
            seen.add(id_str)
            parsed_list.append(int(id_str))

    # logging.info(f"模型输出解析 - 解析后电影ID列表: {parsed_list}")
    return parsed_list
```