项目介绍：基于大语言模型（LLM）的个性化推荐排序

项目目标

本项目旨在通过大语言模型（LLM）对电影推荐结果进行 **重排序（reranking）**，基于用户的历史喜好和评分，优化电影推荐的准确性和排序效果。

项目背景

- 传统的电影推荐系统通常依赖于基于内容的过滤、协同过滤或混合方法来生成初步的推荐列表。然而，这些方法往往忽略了用户的具体偏好以及电影之间更细粒度的关联。
- 使用GPT Based方法可以通过自然语言处理用户历史的喜好数据，为推荐系统提供更个性化、符合用户心理模型的排序结果。

方法概述

1. **数据准备**：
   - 从用户历史评分数据中提取出 **喜欢** 和 **不喜欢** 的电影，构建每个用户的喜好画像。用户的喜好不仅仅通过评分体现，还可以通过评论和标签等其他方式进一步丰富。
2. **候选电影召回**：
   - 使用传统的推荐算法（如 **协同过滤** 或 **基于内容的推荐**）生成一个初步的候选电影列表。传统方法依赖用户的历史评分数据，可能会根据相似用户的行为或相似电影的特征进行推荐，但未能考虑用户的动态兴趣和电影之间的深层次关联。
3. **大模型输入**：
   - 将用户的历史喜好信息与候选电影一同输入 **大语言模型（如 GPT）**。该模型能够通过处理用户喜好与电影内容，推断每部电影的 **喜欢度** 和推荐理由。
   - GPT 可以从自然语言中理解用户的潜在偏好，比如通过对用户历史评分的上下文进行分析，判断用户对某种类型电影的潜在兴趣。
4. **重新排序**：
   - 基于大模型输出的 **喜欢** 和 **不喜欢** 结果，对候选电影进行重新排序。模型通过分析电影与用户偏好的契合度，以及每个推荐理由的相关性，生成更符合用户当前兴趣的推荐结果。

- 与传统协同过滤相比，这一过程考虑了更多维度的特征，如情感分析、电影背景、用户的情感偏好等，使推荐更加精准。

5. **评估**：
   - 使用 **准确率（Precision）**、**召回率（Recall）** 和 **NDCG**（Normalized Discounted Cumulative Gain）等指标评估排序效果，展示模型优化的效果。通过比较模型在不同阶段的表现，能够直观反映大语言模型如何提升推荐系统的精度和覆盖率。

数据集

- **用户评分数据**：包含用户对电影的评分记录，每条记录包括用户ID、电影ID、评分和喜好（喜欢/不喜欢）。
- **电影数据**：包含电影的基本信息（如电影名称、发行年份、类型等）。

数据集介绍

该数据集是 **MovieLens 100k** 数据集，包含用户对电影的评分信息。数据集广泛用于推荐系统研究，包含以下几部分数据：

1. **用户数据（`u.user`）**：
   - **用户ID（`user id`）**：唯一标识每个用户。
   - **年龄（`age`）**：用户的年龄（通过年龄段分类）。
   - **性别（`gender`）**：用户的性别（男性或女性）。
   - **职业（`occupation`）**：用户的职业类型。
   - **邮政编码（`zip code`）**：用户的邮政编码。
2. **电影数据（`u.item`）**：
   - **电影ID（`item id`）**：唯一标识每部电影。
   - **电影标题（`title`）**：电影的名称。
   - **上映日期（`release date`）**：电影的首次上映日期。
   - **视频发布日期（`video release date`）**：电影的发行视频日期（如果有）。
   - **IMDb链接（`IMDb URL`）**：电影在IMDb上的链接。

- **电影类型**：该数据集包含了电影的多个类别标签，例如：动作、冒险、动画、儿童、喜剧、犯罪等。

3. **评分数据（`u.data`）**：
   - **用户ID（`user id`）**：唯一标识每个用户。
   - **电影ID（`item id`）**：唯一标识每部电影。
   - **评分（`rating`）**：用户对电影的评分，评分范围从 `1` 到 `5`，越高表示越喜欢。
   - **时间戳（`timestamp`）**：评分产生的时间，记录为 `Unix` 时间戳（自1970年1月1日起的秒数），经过转换后表示为日期时间。
4. **评分标记（`like`）**：
   - 通过对评分数据进行处理，将用户评分大于 `3` 的电影标记为“喜欢”（`like = True`），否则标记为“不喜欢”（`like = False`）。

数据集的目标：

- 本数据集用于电影推荐系统的开发与评估，通常目的是根据用户的历史评分，预测其可能喜欢的电影。
- 数据集包含了用户与电影的交互历史，通过分析用户的评分行为，可以为用户推荐他们未观看过的电影。

数据集大小：

- **用户数**：包含了 `100,000` 个电影评分记录，这些评分来自于 `943` 个用户。
- **电影数**：数据集包含 `1682` 部电影。
- **评分范围**：每个评分都在 `1` 到 `5` 之间，代表了用户对电影的喜好程度。

名词介绍

1. 准确率（Precision）

准确率 衡量的是推荐列表中的电影中有多少是用户实际喜欢的。它反映了推荐系统的 **准确性**，即推荐给用户的电影中有多少电影是用户喜欢的。

公式：

$$
\text{Precision} = \frac{\text{推荐电影中用户喜欢的电影数}}{\text{推荐的电影总数}}
$$

举个例子：

假设推荐系统为用户推荐了 5 部电影，推荐结果如下：

- 推荐电影：`[Movie A, Movie B, Movie C, Movie D, Movie E]`
- 用户喜欢的电影：`[Movie A, Movie D]`

那么：

- 推荐电影中用户喜欢的电影有 2 部（`Movie A`，`Movie D`）。
- 总共推荐了 5 部电影。

因此，准确率为：

$$
\text{Precision} = \frac{2}{5} = 0.4
$$

即推荐的电影中有 **40%** 是用户实际喜欢的。

---

2. 召回率（Recall）

召回率 衡量的是用户所有喜欢的电影中，有多少出现在推荐列表中。它反映了推荐系统的 **全面性**，即推荐系统是否能够覆盖用户真正喜欢的电影。

公式：

$$
\text{Recall} = \frac{\text{推荐的用户喜欢的电影数}}{\text{用户实际喜欢的电影总数}}
$$

举个例子：

假设用户实际上喜欢 4 部电影，推荐系统推荐了 5 部电影，推荐结果如下：

- 推荐电影：`[Movie A, Movie B, Movie C, Movie D, Movie E]`
- 用户喜欢的电影：`[Movie A, Movie D, Movie F, Movie G]`

那么：

- 推荐的用户喜欢的电影有 2 部（`Movie A`，`Movie D`）。
- 用户实际喜欢的电影总数为 4 部（`Movie A`，`Movie D`，`Movie F`，`Movie G`）。

因此，召回率为：

$$
\text{Recall} = \frac{2}{4} = 0.5
$$

即推荐列表中的 **50%** 是用户实际喜欢的电影。

总结：

- **准确率（Precision）** 衡量的是推荐的电影中有多少是用户实际喜欢的。推荐系统的准确率越高，意味着推荐的电影越符合用户的兴趣。
- **召回率（Recall）** 衡量的是用户喜欢的电影中有多少出现在推荐列表中。召回率越高，意味着推荐系统能够覆盖更多用户喜欢的电影。

在推荐系统的优化过程中，我们通常需要平衡这两个指标。提高准确率可能会降低召回率，因为系统推荐的电影会更加精确，但可能会错过一些用户喜欢的电影。而提高召回率可能会增加不相关或不喜欢的电影，降低准确率。因此，通常需要找到一个平衡点，确保推荐既有较高的准确性，也能覆盖更多用户喜欢的电影。

```
# !unzip sample_data/ml-100k.zip -d sample_data
```

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import warnings
from pylab import mpl, plt

# best font and style settings for notebook
warnings.filterwarnings('ignore')
sns.set_style("white")
mpl.rcParams['font.family'] = 'MiSans'
```

```
import pandas as pd
import numpy as np
import os
import time
from datetime import datetime
import json

dir = 'sample_data/ml-100k'
col_names = ['user id', 'item id', 'rating', 'timestamp']
data = pd.read_csv(os.path.join(dir, 'u.data'), delimiter='\t', names=col_names, header=None)
data['timestamp'] = data['timestamp'].apply(lambda x: datetime.fromtimestamp(x))

with open(os.path.join(dir, 'u.item'), encoding="ISO-8859-1") as f:
    movie = pd.read_csv(f, delimiter='|', header=None)

movie.columns = ['item id', 'title', 'release date', 'video release date', 'IMDb URL', 'unknown', 'Action', 'Adventure',
                 'Animation', 'Children\'s', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
                 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']

with open(os.path.join(dir, 'u.user'), encoding="ISO-8859-1") as f:
    user = pd.read_csv(f, delimiter='|', header=None)

user.columns = ['user id', 'age', 'gender', 'occupation', 'zip code']

ratings = data.merge(movie[['item id', 'title']], on='item id')
```

```
ratings['like'] = ratings['rating'] > 3
```

```
ratings.sort_values(by=['user id'], ascending=[True]).head(10)
```

	user id	item id	rating	timestamp	title	like
66567	1	55	5	1997-09-24 11:44:48	Professional, The (1994)	True
62820	1	203	4	1997-11-03 15:30:31	Unforgiven (1992)	True
10207	1	183	5	1997-09-24 11:37:42	Alien (1979)	True
9971	1	150	5	1997-10-15 13:09:56	Swingers (1996)	True
22496	1	68	4	1997-09-24 11:44:48	Crow, The (1994)	True
9811	1	201	3	1997-11-03 15:42:40	Evil Dead II (1987)	False
9722	1	157	4	1997-10-15 13:21:58	Platoon (1986)	True
9692	1	184	4	1997-09-24 11:49:16	Army of Darkness (1993)	True
9566	1	210	4	1997-11-03 15:41:49	Indiana Jones and the Last Crusade (1989)	True
9382	1	163	4	1997-09-24 11:40:42	Return of the Pink Panther, The (1974)	True

```
train_ratio = 0.9
train_size = int(len(ratings) * train_ratio)
ratings_train = ratings.sample(train_size, random_state=42)
ratings_test = ratings[~ratings.index.isin(ratings_train.index)]
```

Recall (MF)

```
# pip install implicit
```

```
from scipy.sparse import csr_matrix

n_users = ratings_train['user id'].max()
n_item = ratings_train['item id'].max()
ratings_train_pos = ratings_train[ratings_train['like']]
ratings_test_pos = ratings_test[ratings_test['like']]

row = ratings_train_pos['user id'].values - 1
col = ratings_train_pos['item id'].values - 1
data = np.ones(len(ratings_train_pos))
user_item_data = csr_matrix((data, (row, col)), shape=(n_users, n_item))
```

```
import implicit

# initialize a model
model = implicit.als.AlternatingLeastSquares(factors=50, random_state=42)

# train the model on a sparse matrix of user/item/confidence weights
model.fit(user_item_data)
```

```
  0%|          | 0/15 [00:00<?, ?it/s]
```

传统方法-协同过滤的处理过程

```
# 导入用于计算DCG和NDCG的评估函数
from sklearn.metrics import dcg_score, ndcg_score
```

1. `precision_k`：计算准确率（Precision）@k

```
def precision_k(actuals, recs, k=5):
    return len(set(recs[0:k]).intersection(set(actuals))) / k
```

- **功能**：计算推荐结果中前 `k` 个电影中，有多少是用户实际喜欢的电影的比例，即准确率（Precision）。
- **参数**：
  - `actuals`：用户实际喜欢的电影的列表。
  - `recs`：推荐系统为该用户推荐的电影列表。
  - `k`：关注的前 `k` 个推荐（默认为 5）。
- **解释**：
  - 首先，`recs[0:k]` 获取推荐的前 `k` 个电影，`set(recs[0:k])` 将其转换为集合以去重。
  - `actuals` 是用户实际喜欢的电影列表。
  - `set(recs[0:k]).intersection(set(actuals))` 找到推荐列表和实际喜欢的电影的交集，即推荐的用户实际喜欢的电影。
  - 准确率是交集的大小除以 `k`，表示推荐的电影中有多少比例是用户喜欢的。

2. `recall_k`：计算召回率（Recall）@k

```
def recall_k(actuals, recs, k=5):
    return len(set(recs[0:k]).intersection(set(actuals))) / len(actuals)
```

- **功能**：计算推荐结果中，用户实际喜欢的电影有多少被推荐的比例，即召回率（Recall）。
- **参数**：
  - `actuals`：用户实际喜欢的电影的列表。
  - `recs`：推荐系统为该用户推荐的电影列表。
  - `k`：关注的前 `k` 个推荐（默认为 5）。

- **解释**：
  - 与准确率不同，召回率衡量的是用户喜欢的电影中有多少被推荐系统包含。
  - 通过计算推荐电影列表和实际喜欢电影的交集大小，并除以用户实际喜欢的电影数，得到召回率。

---

3. `dcg_k`：计算DCG (Discounted Cumulative Gain) @k

```
def dcg_k(actuals, recs, k=5):
    relevance = np.array([[float(i in actuals) for i in recs[0:k]]])
    score = k - np.arange(k)
    return dcg_score(relevance, score.reshape(1, -1), k=k)
```

- **功能**：计算 **Discounted Cumulative Gain (DCG)**，衡量推荐列表中电影的排名是否符合用户的偏好。DCG 考虑了推荐结果的排名顺序，排名靠前的电影对用户的影响更大。
- **参数**：
  - `actuals`：用户实际喜欢的电影的列表。
  - `recs`：推荐系统为该用户推荐的电影列表。
  - `k`：关注的前 `k` 个推荐（默认为 5）。
- **解释**：
  - **Relevance**：根据电影是否在 `actuals`（用户喜欢的电影）列表中，将推荐电影的相关性设为 1（喜欢）或 0（不喜欢）。
  - **Score**：为每个推荐电影分配一个分数，越靠前的电影得分越高，通常是 `k-1` 到 `0`，表示从推荐列表顶部到底部的递减。
  - `dcg_score` 函数计算实际推荐的 DCG 值，评分越高的电影排名靠前，DCG 会更高。

---

4. `ndcg_k`：计算NDCG (Normalized Discounted Cumulative Gain) @k

```
def ndcg_k(actuals, recs, k=5):
    relevance = np.array([[float(i in actuals) for i in recs[0:k]]])
    score = k - np.arange(k)
    return ndcg_score(relevance, score.reshape(1, -1), k=k)
```

- **功能**：计算 **Normalized Discounted Cumulative Gain (NDCG)**，与 DCG 相似，但 NDCG 会进行归一化，确保不同大小的推荐列表之间可以比较。
- **参数**：
  - `actuals`：用户实际喜欢的电影的列表。
  - `recs`：推荐系统为该用户推荐的电影列表。
  - `k`：关注的前 `k` 个推荐（默认为 5）。
- **解释**：
  - **Relevance** 和 **Score** 的计算方法与 DCG 相同。
  - `ndcg_score` 会计算归一化后的 DCG，NDCG 值越高说明推荐列表的顺序越符合用户的实际偏好，推荐系统的排序质量越高。

5. `recall_stage`：生成推荐电影

```
def recall_stage(model, user_id, user_item_data, ratings_train, N):
    filter_items = ratings_train[ratings_train['user id'] == user_id]['item id'].values
    filter_items = filter_items - 1
    user_id = user_id - 1

    recs, scores = model.recommend(user_id,
                                   user_item_data[user_id],
                                   filter_items=filter_items,
                                   N=N_recall)
    recs = recs.flatten() + 1
    return recs
```

- **功能**：生成推荐电影列表。该函数利用推荐模型根据用户历史数据生成推荐，并且排除了用户已经评分的电影。
- **参数**：
  - `model`：推荐模型（例如基于协同过滤、矩阵分解的模型）。
  - `user_id`：目标用户的 ID。
  - `user_item_data`：包含用户–物品交互信息的稀疏矩阵。
  - `ratings_train`：用户历史评分数据。
  - `N`：需要推荐的电影数量。
- **解释**：
  - 根据用户的历史评分，生成推荐列表 `recs`，并排除用户已经评分的电影。
  - 推荐列表返回的是用户最可能喜欢的电影，推荐数量为 `N_recall`。

6. `evaluate`：评估推荐效果

```
def evaluate(user_id, ratings_test_pos, recs, k=5):
    actuals = ratings_test_pos[ratings_test_pos['user id'] == user_id]['item id'].values
    return precision_k(actuals, recs, k), recall_k(actuals, recs, k), dcg_k(actuals, recs, k)
```

- **功能**：评估推荐系统的效果，计算 **准确率**、**召回率** 和 **NDCG**。
- **参数**：
  - `user_id`：目标用户的 ID。
  - `ratings_test_pos`：测试集，包含用户喜欢的电影数据。
  - `recs`：推荐系统为该用户推荐的电影列表。
  - `k`：评估的前 `k` 个推荐（默认为 5）。
- **解释**：
  - 该函数计算并返回用户的 **Precision@k**、**Recall@k** 和 **NDCG@k**，用于衡量推荐系统的质量。

```
from sklearn.metrics import dcg_score, ndcg_score


def precision_k(actuals, recs, k=5):
    return len(set(recs[0:k]).intersection(set(actuals))) / k


def recall_k(actuals, recs, k=5):
    return len(set(recs[0:k]).intersection(set(actuals))) / len(actuals)


def dcg_k(actuals, recs, k=5):
    relevance = np.array([[float(i in actuals) for i in recs[0:k]]])
    score = k - np.arange(k)
    return dcg_score(relevance, score.reshape(1, -1), k=k)


def ndcg_k(actuals, recs, k=5):
```

```
    relevance = np.array([[float(i in actuals) for i in recs[0:k]]])
    score = k - np.arange(k)
    return ndcg_score(relevance, score.reshape(1, -1), k=k)


def recall_stage(model, user_id, user_item_data, ratings_train, N):
    filter_items = ratings_train[ratings_train['user id'] == user_id]['item id'].values
    filter_items = filter_items - 1
    user_id = user_id - 1

    recs, scores = model.recommend(user_id,
                                   user_item_data[user_id],
                                   filter_items=filter_items,
                                   N=N_recall)
    recs = recs.flatten() + 1
    return recs


def evaluate(user_id, ratings_test_pos, recs, k=5):
    actuals = ratings_test_pos[ratings_test_pos['user id'] == user_id]['item id'].values
    return precision_k(actuals, recs, k), recall_k(actuals, recs, k), dcg_k(actuals, recs, k)
```

```
# recommend items for a user
N_recall = 30
user_id = 1
recs = recall_stage(model, user_id, user_item_data, ratings_train, N_recall)
evaluate(user_id, ratings_test_pos, recs, 20)
```

```
(0.25, 0.2, 2.0491745551794502)
```

Ranking (GPT)

```
# !pip install langchain openai == 0.27.0
```

GPT Based 排序方法过程

在该项目中，GPT 或类似的语言模型用于 **重新排序**（re-ranking）推荐结果的过程通常如下：

1. 召回阶段 (Recall Stage)

在召回阶段，通过基于协同过滤、矩阵分解、或者其他推荐算法（如 ALS、KNN 等）为用户生成一个初步的推荐列表。例如，假设你从推荐系统中召回了 30 个电影。

2. 准备用户历史数据

- 在重新排序之前，需要将用户的历史评分数据（如喜欢和不喜欢的电影）传递给语言模型，以便生成个性化的推荐排序。这些数据将作为模型的输入，帮助模型理解用户的偏好。
- 具体来说，用户的喜好会被转化为 **"用户喜欢的电影"** 和 **"用户不喜欢的电影"** 的列表，作为模型的输入。

3. GPT 进行重新排序

- **输入**：用户喜欢和不喜欢的电影列表，以及候选电影（即召回的初步推荐列表）。例如：
  - 喜欢的电影：`[The Terminator, Aliens, Matrix]`
  - 不喜欢的电影：`[Fast & Furious, Die Hard]`
  - 候选电影列表：`[Alien, Chinatown, Speed, True Lies, The Piano]`

  这些信息将传递给 GPT（或其他语言模型），模型的任务是判断每个候选电影对该用户的适配程度。
- **输出**：模型会返回每个候选电影是否符合用户的喜好，并为每个电影提供一个解释。例如：

```
[
  {
    "title": "Alien (1979)",
```

```
    "like": true,
    "explanation": "The person likes sci-fi and horror films like 'Aliens (1986)' and 'The Terminator (1984)', and 'Alien' is a classic in the same genre."
  },
  {
    "title": "Chinatown (1974)",
    "like": true,
    "explanation": "The person enjoys classic films like 'The Godfather (1972)' and 'Citizen Kane (1941)', and 'Chinatown' is a critically acclaimed noir film that fits their taste."
  },
  {
    "title": "Speed (1994)",
    "like": false,
    "explanation": "The person dislikes action films like 'Under Siege (1992)' and 'The Rock (1996)', and 'Speed' is a high-octane action film that may not appeal to them."
  }
]
```

在这个例子中，GPT 会对每个候选电影生成是否喜欢（`like: true/false`）的标签，并且提供一个解释，阐述为什么它认为这个电影符合或不符合用户的喜好。

4. 根据模型输出重新排序

- **排序**：根据 GPT 的输出，对候选电影进行排序。模型的输出提供了每个电影是否符合用户偏好的概率（如 `like: true` 或 `false`），你可以按 **喜欢的概率** 对电影进行排序。
- **最终排序**：重新排序后的推荐列表将包含电影，按用户最可能喜欢的顺序排列，优先显示更符合用户口味的电影。

5. 输出

- 返回经过 GPT 排序后的推荐列表，这个列表中的电影按符合用户偏好的程度排列。例如，模型可能会把 `"Alien (1979)"` 排在推荐列表的最前面，因为它符合用户的偏好，而将 `"Speed (1994)"` 排在后面，因为用户不喜欢动作片。

总结

GPT 通过以下步骤进行重新排序：

1. 接收用户的历史数据（喜欢和不喜欢的电影）以及初步召回的候选电影。

2. 使用语言模型判断每个候选电影是否符合用户的偏好，并给出解释。
3. 根据模型输出的 **喜欢/不喜欢** 标志和解释，为推荐列表中的电影重新排序，确保最符合用户兴趣的电影排在最前面。

这个过程使得推荐结果更加个性化和精准，通过语言模型的强大理解能力，优化了推荐系统的排序质量。

```
import os
from langchain.chat_models import ChatOpenAI
from langchain.prompts import ChatPromptTemplate
from langchain.chains import LLMChain

# 设置 DeepSeek API 密钥和地址
os.environ["OPENAI_API_KEY"] = "*******"  # ✅ 请替换为你的真实密钥
os.environ["OPENAI_API_BASE"] = "https://api.deepseek.com"

# 初始化 DeepSeek LLM 模型（兼容 OpenAI 格式）
llm = ChatOpenAI(
    model_name="deepseek-chat",  # 模型名（也可为 deepseek-coder 等）
    openai_api_base=os.environ["OPENAI_API_BASE"],  # API地址
    openai_api_key=os.environ["OPENAI_API_KEY"],  # 密钥
    temperature=0.0
)

# 构建 Prompt 模板
prompt = ChatPromptTemplate.from_template(
    """The person has a list of liked movies: {movies_liked}. \
The person has a list of disliked movies: {movies_disliked}. \
Tell me if this person likes each of the candidate movies: {movies_candidates}. \
Return a list of boolean values and explain why the person likes or dislikes.

<< FORMATTING >>
Return a markdown code snippet with a list of JSON object formatted to look like:
{{
    "title": string \\ the name of the movie in candidate movies
    "like": boolean \\ true or false
    "explanation": string \\ explain why the person likes or dislikes the candidate movie
}}

REMEMBER: Each boolean and explanation for each element in candidate movies.
```

```
REMEMBER: The explanation must relate to the person's liked and disliked movies.
"""
)

# 构建 LLM Chain
chain = LLMChain(llm=llm, prompt=prompt)
```

```
def ranking_stage(chain, user_id, ratings_train, pre_recs, movie, batch_size=10):
    few_shot = ratings_train[(ratings_train['user id'] == user_id)]
    if len(few_shot) ≥ 300:
        few_shot = few_shot.sample(300, random_state=42)
    recall_recs = movie.set_index('item id').loc[pre_recs].reset_index()

    movies_liked = ','.join(few_shot[few_shot['like']]['title'].values.tolist())
    movies_disliked = ','.join(few_shot[~few_shot['like']]['title'].values.tolist())

    n_batch = int(np.ceil(len(recall_recs) / batch_size))
    candidates = recall_recs[['item id', 'title']]
    result_json = []

    for i in range(n_batch):
        candidates_batch = candidates.iloc[i * batch_size: (i + 1) * batch_size]
        movies_candidates = ','.join(candidates_batch['title'].values.tolist())
        result = chain.run(movies_liked=movies_liked, movies_disliked=movies_disliked,
                           movies_candidates=movies_candidates)
        result_list = result.replace('\n', '').replace('},', '}\n,').split('\n,')
        result_json_batch = [json.loads(i) for i in result_list]
        result_json = result_json + result_json_batch

    result_rank = pd.DataFrame.from_dict(result_json)
    result_rank['item id'] = recall_recs['item id'].values
    result_rank = pd.concat([result_rank[result_rank['like']], result_rank[~result_rank['like']]])

    return result_rank
```

```
import numpy as np
```

```python
import pandas as pd


def debug_ranking_stage_raw_output(chain, user_id, ratings_train, pre_recs, movie, batch_size=10):
    few_shot = ratings_train[ratings_train['user id'] == user_id]
    if len(few_shot) ≥ 300:
        few_shot = few_shot.sample(300, random_state=42)

    recall_recs = movie.set_index('item id').loc[pre_recs].reset_index()

    movies_liked = ','.join(few_shot[few_shot['like']]['title'].values.tolist())
    movies_disliked = ','.join(few_shot[~few_shot['like']]['title'].values.tolist())

    n_batch = int(np.ceil(len(recall_recs) / batch_size))
    candidates = recall_recs[['item id', 'title']]
    raw_outputs = []

    for i in range(n_batch):
        candidates_batch = candidates.iloc[i * batch_size: (i + 1) * batch_size]
        movies_candidates = ','.join(candidates_batch['title'].values.tolist())

        print(f"\n--- Batch {i + 1}/{n_batch} ---")
        print(f"[Candidates]: {movies_candidates}")

        result = chain.run(
            movies_liked=movies_liked,
            movies_disliked=movies_disliked,
            movies_candidates=movies_candidates
        )

        # print(f"[Raw LLM Output]:\n{result}\n")
        raw_outputs.append({
            'batch_index': i,
            'candidates': candidates_batch,
            'raw_text': result
        })

    return raw_outputs
```

```python
import json
import re


def extract_json_array(text):
    # 匹配 JSON 列表内容（中间可能换行）
    match = re.search(r"\[\s*\{.*?\}\s*\]", text, re.DOTALL)
    if match:
        try:
            return json.loads(match.group(0))
        except json.JSONDecodeError as e:
            print(f"[解析失败] JSONDecodeError: {e}")
    else:
        print("[警告] 没有找到 JSON 列表")
    return []


def parse_rank_result_from_raw_outputs(raw_outputs):
    result_json_all = []

    for batch in raw_outputs:
        raw_text = batch['raw_text']
        candidates_df = batch['candidates'].reset_index(drop=True)

        # 提取 JSON 数组字符串并解析
        try:
            json_array = extract_json_array(raw_text)
        except Exception as e:
            print(f"[ERROR] 解析失败: {e}")
            json_array = []

        # 加上原来的 item id（标题顺序匹配即可）
        for idx, item in enumerate(json_array):
            item['item id'] = candidates_df.loc[idx, 'item id']

        result_json_all.extend(json_array)
```

```
    # 转为 DataFrame，并按 'like' 优先排序
    result_rank = pd.DataFrame(result_json_all)
    result_rank = pd.concat([
        result_rank[result_rank['like']],
        result_rank[~result_rank['like']]
    ])

    return result_rank
```

```
raw_outputs = debug_ranking_stage_raw_output(chain, user_id, ratings_train, recs, movie)
```

```
D:\mambaforge\envs\ml\lib\site-packages\langchain_core\_api\deprecation.py:117: LangChainDeprecationWarning: The function `run` was deprecated in LangChain 0.1.0 and will be removed in 0.2.0. Use invoke instead.
  warn_deprecated(
```

```
--- Batch 1/3 ---
[Candidates]: Alien (1979),Chinatown (1974),Close Shave, A (1995),Birds, The (1963),City of Lost Children, The (1995),Speed (1994),Dead Poets Society (1989),True Lies (1994),Piano, The (1993),Leaving Las Vegas (1995)
[Raw LLM Output]:
```json
[
    {
        "title": "Alien (1979)",
        "like": true,
        "explanation": "The person likes sci-fi and horror movies like 'Aliens (1986)' and 'The Terminator (1984)', which are in the same genre and directed by James Cameron, who also directed 'Aliens'. Additionally, the person enjoys Ridley Scott's 'Blade Runner (1982)', suggesting an appreciation for his work."
    },
    {
        "title": "Chinatown (1974)",
        "like": true,
```
```

```
        "explanation": "The person enjoys classic films like 'The Godfather (1972)' and 'Citizen Kane (1941)', and 'Chinatown' is a critically acclaimed neo-noir film that fits their taste for well-regarded cinema. They also like 'The Sting (1973)', another classic from the same era."
    },
    {
        "title": "Close Shave, A (1995)",
        "like": true,
        "explanation": "The person likes animated and quirky films like 'Wallace & Gromit: The Best of Aardman Animation (1996)' and 'Toy Story (1995)'. 'A Close Shave' is part of the Wallace & Gromit series, which aligns with their preferences for clever and humorous animation."
    },
    {
        "title": "Birds, The (1963)",
        "like": true,
        "explanation": "The person enjoys classic horror/thriller films like 'Psycho (1960)' and 'The Shining (1980)', and 'The Birds' is another Hitchcock masterpiece in the same vein. Their dislike of 'A Nightmare on Elm Street (1984)' doesn't extend to more psychological horror, which they seem to appreciate."
    },
    {
        "title": "City of Lost Children, The (1995)",
        "like": true,
        "explanation": "The person likes visually distinctive and unconventional films like 'Delicatessen (1991)' and 'The Haunted World of Edward D. Wood Jr. (1995)'. 'The City of Lost Children' shares a similar surreal and artistic style, making it likely they would enjoy it."
    },
    {
        "title": "Speed (1994)",
        "like": false,
        "explanation": "The person dislikes action-heavy, mainstream blockbusters like 'Twister (1996)' and 'The Rock (1996)'. 'Speed' falls into this category, and their preference leans more toward critically acclaimed or unique films rather than pure action thrillers."
    },
    {
        "title": "Dead Poets Society (1989)",
        "like": true,
        "explanation": "The person enjoys dramatic and emotionally resonant films like 'The Shawshank Redemption (1994)' and 'Good Will Hunting (1997)'. 'Dead Poets Society' fits this mold with its inspirational and heartfelt storytelling."
    },
```

```
    {
        "title": "True Lies (1994)",
        "like": false,
        "explanation": "The person dislikes over-the-top action films like 'Die Hard 2 (1990)' and 'The Long Kiss Goodnight (1996)'. 'True Lies' is a high-energy action-comedy that doesn't align with their preference for more subdued or critically praised films."
    },
    {
        "title": "Piano, The (1993)",
        "like": true,
        "explanation": "The person appreciates artistic and emotionally complex films like 'The Unbearable Lightness of Being (1988)' and 'Remains of the Day (1993)'. 'The Piano' is a critically acclaimed drama that fits their taste for nuanced storytelling."
    },
    {
        "title": "Leaving Las Vegas (1995)",
        "like": true,
        "explanation": "The person likes intense, character-driven dramas like 'Dead Man Walking (1995)' and 'Taxi Driver (1976)'. 'Leaving Las Vegas' is a gritty, emotional film that aligns with their appreciation for raw and powerful performances."
    }
]
```

```
--- Batch 2/3 ---
[Candidates]: That Thing You Do! (1996),Bob Roberts (1992),Hamlet (1996),Schindler's List (1993),Muriel's Wedding (1994),GoodFellas (1990),Star Trek: First Contact (1996),Strictly Ballroom (1992),Cinema Paradiso (1988),Adventures of Priscilla, Queen of the Desert, The (1994)
[Raw LLM Output]:
```json
[
    {
        "title": "That Thing You Do! (1996)",
        "like": true,
        "explanation": "The person enjoys lighthearted and comedic films like 'Austin Powers: International Man of Mystery (1997)' and 'While You Were Sleeping (1995)', and 'That Thing You Do!' fits this tone. It's also a music-themed film, and the person liked 'The Blues Brothers (1980)', which suggests an appreciation for music-driven narratives."
    },
    {
```
```

```
        "title": "Bob Roberts (1992)",
        "like": true,
        "explanation": "The person likes satirical and politically charged films like 'Quiz Show (1994)' and 'The Candidate (1972)', and 'Bob Roberts' is a political satire. Additionally, the person enjoys Tim Robbins' work, as they liked 'The Shawshank Redemption (1994)' and 'The Hudsucker Proxy (1994)'."
    },
    {
        "title": "Hamlet (1996)",
        "like": false,
        "explanation": "The person dislikes Shakespeare adaptations like 'Much Ado About Nothing (1993)' and 'Richard III (1995)'. While they appreciate dramatic films, 'Hamlet' might be too heavy or traditional for their taste, given their preference for more modern or unconventional storytelling."
    },
    {
        "title": "Schindler's List (1993)",
        "like": true,
        "explanation": "The person appreciates intense, well-crafted dramas like 'The Shawshank Redemption (1994)' and 'Dead Man Walking (1995)'. 'Schindler's List' is a critically acclaimed historical drama, which aligns with their liking for serious, impactful films."
    },
    {
        "title": "Muriel's Wedding (1994)",
        "like": true,
        "explanation": "The person enjoys quirky, character-driven comedies like 'The Full Monty (1997)' and 'Strictly Ballroom (1992)'. 'Muriel's Wedding' is a similarly offbeat and heartfelt comedy, which fits their taste."
    },
    {
        "title": "GoodFellas (1990)",
        "like": true,
        "explanation": "The person likes crime dramas and films with strong direction, such as 'The Godfather (1972)' and 'Pulp Fiction (1994)'. 'GoodFellas' is a classic crime film that aligns with their appreciation for gritty, well-made narratives."
    },
    {
        "title": "Star Trek: First Contact (1996)",
        "like": true,
        "explanation": "The person is a fan of the 'Star Trek' franchise, having liked 'Star Trek IV: The Voyage Home (1986)' and 'Star Trek VI: The Undiscovered Country (1991)'. 'First Contact' is a well-regarded entry in the series, so it's likely they would enjoy it."
```

```
    },
    {
        "title": "Strictly Ballroom (1992)",
        "like": true,
        "explanation": "The person enjoys quirky, feel-good films like 'The Full Monty (1997)' and 'Muriel's Wedding (1994)'. 'Strictly Ballroom' is a charming, offbeat comedy with a romantic edge, which fits their preferences."
    },
    {
        "title": "Cinema Paradiso (1988)",
        "like": true,
        "explanation": "The person appreciates heartfelt, nostalgic films like 'The Princess Bride (1987)' and 'Jean de Florette (1986)'. 'Cinema Paradiso' is a touching, sentimental film about love for cinema, which aligns with their taste."
    },
    {
        "title": "Adventures of Priscilla, Queen of the Desert, The (1994)",
        "like": true,
        "explanation": "The person enjoys unconventional and visually striking films like 'Delicatessen (1991)' and 'The Nightmare Before Christmas (1993)'. 'Priscilla' is a vibrant, eccentric road movie, which fits their appreciation for unique storytelling."
    }
]
```

```
--- Batch 3/3 ---
[Candidates]: Star Trek: Generations (1994),In the Line of Fire (1993),Rosencrantz and Guildenstern Are Dead (1990),Brazil (1985),Heathers (1989),When We Were Kings (1996),One Flew Over the Cuckoo's Nest (1975),Mighty Aphrodite (1995),Cool Hand Luke (1967),Shallow Grave (1994)
[Raw LLM Output]:
```json
[
    {
        "title": "Star Trek: Generations (1994)",
        "like": true,
        "explanation": "The person likes several Star Trek movies (e.g., Star Trek IV, Star Trek VI) and sci-fi films (e.g., Contact, Blade Runner), so they are likely to enjoy this entry in the franchise."
    },
    {
```
```

```
        "title": "In the Line of Fire (1993)",
        "like": true,
        "explanation": "The person enjoys thrillers and action films (e.g., Die Hard, The Fugitive), and this movie fits that genre well. They also like Clint Eastwood films (e.g., Unforgiven)."
    },
    {
        "title": "Rosencrantz and Guildenstern Are Dead (1990)",
        "like": false,
        "explanation": "The person dislikes some Shakespeare adaptations (e.g., Much Ado About Nothing) and this film's abstract, theatrical style may not align with their preferences for more straightforward narratives."
    },
    {
        "title": "Brazil (1985)",
        "like": true,
        "explanation": "The person enjoys dark, satirical, and visually distinctive films (e.g., Delicatessen, Monty Python movies), and Brazil fits this style. They also like Terry Gilliam's work (e.g., 12 Monkeys)."
    },
    {
        "title": "Heathers (1989)",
        "like": true,
        "explanation": "The person likes dark comedies and unconventional films (e.g., Pulp Fiction, Fargo), and Heathers aligns with this taste. They also enjoy 1980s movies with unique tones."
    },
    {
        "title": "When We Were Kings (1996)",
        "like": true,
        "explanation": "The person appreciates documentaries (e.g., Hoop Dreams, Crumb) and this acclaimed documentary about Muhammad Ali would likely appeal to them."
    },
    {
        "title": "One Flew Over the Cuckoo's Nest (1975)",
        "like": true,
        "explanation": "The person likes classic, critically acclaimed dramas (e.g., The Godfather, 12 Angry Men), and this film fits that category. They also enjoy Jack Nicholson's work (e.g., The Shining is disliked, but they like other Nicholson films)."
    },
    {
        "title": "Mighty Aphrodite (1995)",
        "like": false,
```

```
        "explanation": "The person dislikes some Woody Allen films (e.g., not listed in liked movies) and may not connect with this lighter, romantic comedy style compared to their preference for darker or more intense films."
    },
    {
        "title": "Cool Hand Luke (1967)",
        "like": true,
        "explanation": "The person enjoys classic films (e.g., The Graduate, Citizen Kane) and Paul Newman's work (e.g., The Sting is liked). This film's rebellious tone and strong performances align with their tastes."
    },
    {
        "title": "Shallow Grave (1994)",
        "like": true,
        "explanation": "The person likes dark, suspenseful films (e.g., Reservoir Dogs, Pulp Fiction) and this early Danny Boyle film fits that style. They also enjoy crime thrillers with twists."
    }
]
```

```python
rank_result = parse_rank_result_from_raw_outputs(raw_outputs)
rank_recs = rank_result['item id'].values
# rank_recs
p, r, ndcg = evaluate(user_id, ratings_test_pos, rank_recs, k=5)
```

输出结果解读：

```
((0.25, 0.2, 2.049),  # 初步召回结果
 (0.30, 0.24, 2.370)) # rerank 后结果
```

指标	协同过滤	GPT Based 方法	变化
P@20	0.25	0.30	提升
R@20	0.20	0.24	提升
NDCG@20	2.049	2.370	提升

结论：

- **准确率** 提升 +20% (0.25 → 0.30)
- **召回率** 提升 +20% (0.20 → 0.24)
- **NDCG** 提升，推荐更准确

LLM reranking 提升了推荐质量，正确推荐的电影更多且排序更靠前。

```
k = 20
evaluate(user_id, ratings_test_pos, recs, k), evaluate(user_id, ratings_test_pos, rank_recs, k)
```

```
((0.25, 0.2, 2.0491745551794502), (0.3, 0.24, 2.34713553777021))
```

```
!jupyter nbconvert --to markdown llm_recs_2.ipynb
```

```
[NbConvertApp] Converting notebook llm_recs_2.ipynb to markdown
[NbConvertApp] Writing 34187 bytes to llm_recs_2.md
```