这里选择使用自定义的 MLP 来处理，因为想不到调用什么 model 比较好
这个 MLP 实际上只对于当前用户的关系网络中的人喜欢的电影进行一个统计

```
import pandas as pd
import numpy as np
from collections import defaultdict
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset

train = pd.read_csv('/root/data/input_data04/NS-2025-04-data/train_label.csv')
trust = pd.read_csv('/root/data/input_data04/NS-2025-04-data/user_trust.csv')
test = pd.read_csv('/root/data/input_data04/NS-2025-04-data/test_input.csv')
all_movies = sorted(train['movieID'].unique())
movie2idx = {mid: i for i, mid in enumerate(all_movies)}
num_movies = len(all_movies)

user_like_movies = defaultdict(set)
for row in train.itertuples():
    if row.movieRating >= 4:
        user_like_movies[row.userID].add(row.movieID)
trust_dict = defaultdict(list)
for row in trust.itertuples():
    trust_dict[row.trustorID].append(row.trusteeID)
def get_social_vector(uid):
    vec = np.zeros(num_movies, dtype=np.float32)
    for t in trust_dict.get(uid, []):
        for mid in user_like_movies.get(t, []):
            vec[movie2idx[mid]] += 1
    return vec

X, y = [], []
for row in train.itertuples():
    uid = row.userID
    mid = row.movieID
    label = int(row.movieRating >= 4)
    social_vec = get_social_vector(uid)
    movie_vec = np.zeros(num_movies, dtype=np.float32)
    movie_vec[movie2idx[mid]] = 1
    X.append(np.concatenate([social_vec, movie_vec]))
    y.append(label)
X = np.stack(X)
y = np.array(y)

class MLP(nn.Module):
```

```
    def init(self, input_dim):
        super().init()
        self.net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 2)
        )
    def forward(self, x):
        return self.net(x)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = MLP(X.shape[1]).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
criterion = nn.CrossEntropyLoss()


dataset = TensorDataset(torch.tensor(X, dtype=torch.float32), torch.tensor(y,
dtype=torch.long))
loader = DataLoader(dataset, batch_size=256, shuffle=True)
for epoch in range(10):
    model.train()
    for xb, yb in loader:
        xb, yb = xb.to(device), yb.to(device)
        optimizer.zero_grad()
        out = model(xb)
        loss = criterion(out, yb)
        loss.backward()
        optimizer.step()

    model.eval()
    with torch.no_grad():
        X_tensor = torch.tensor(X, dtype=torch.float32, device=device)
        y_tensor = torch.tensor(y, dtype=torch.long, device=device)
        logits = model(X_tensor)
        preds = logits.argmax(dim=1)
        acc = (preds == y_tensor).float().mean().item()
    print(f"Epoch {epoch+1}, Train Acc: {acc:.4f}")


def predict_one(uid):
    social_vec = get_social_vector(uid)
    movie_mat = np.eye(num_movies, dtype=np.float32)
    social_mat = np.repeat(social_vec[None, :], num_movies, axis=0)
    X_pred = np.concatenate([social_mat, movie_mat], axis=1)
    x_tensor = torch.tensor(X_pred, dtype=torch.float32, device=device)
```

```
    with torch.no_grad():
        probs = torch.softmax(model(x_tensor), dim=1)[:, 1].cpu().numpy()
    top10_idx = np.argsort(-probs)[:10]
    top10 = [all_movies[i] for i in top10_idx]
    return (uid, top10)
results = []
for row in test.itertuples():
    results.append(predict_one(row.userID))


with open('results.csv', 'w') as f:
    f.write('userID\trank\n')
    for uid, top10 in results:
        f.write(f"{uid}\t{str(top10)}\n")
```