相关训练过程

1. 读取数据集

In [42]:
```
import pandas as pd

ratings = pd.read_csv(
    "../datasets/ml-1m/ratings.dat",
    sep="::",
    engine="python",
    names=['userId', 'movieId', 'rating', 'timestamp']
)
movies = pd.read_csv(
    "../datasets/ml-1m/movies.dat",
    sep="::",
    engine="python",
    names=['movieId', 'title', 'genres']
)
df = pd.merge(ratings, movies[['movieId', 'title']], on='movieId')
```

2. 使用留出法制作训练集和测试集

In [43]:
```
import pandas as pd
import numpy as np

def split_low_memory(df, test_ratio=0.1, min_items=5, seed=42):
    rng = np.random.default_rng(seed)
    test_indices = []

    for uid, group in df.groupby("userId"):
        if len(group) < min_items:
            continue

        test_size = max(1, int(len(group) * test_ratio))
        chosen = rng.choice(group.index, size=test_size, replace=False)
        test_indices.extend(chosen)

    test_indices = set(test_indices)
    mask = df.index.isin(test_indices)
    return df[~mask], df[mask]


train_df, test_df = split_low_memory(df)
print("train:", len(train_df), "test:", len(test_df))
```

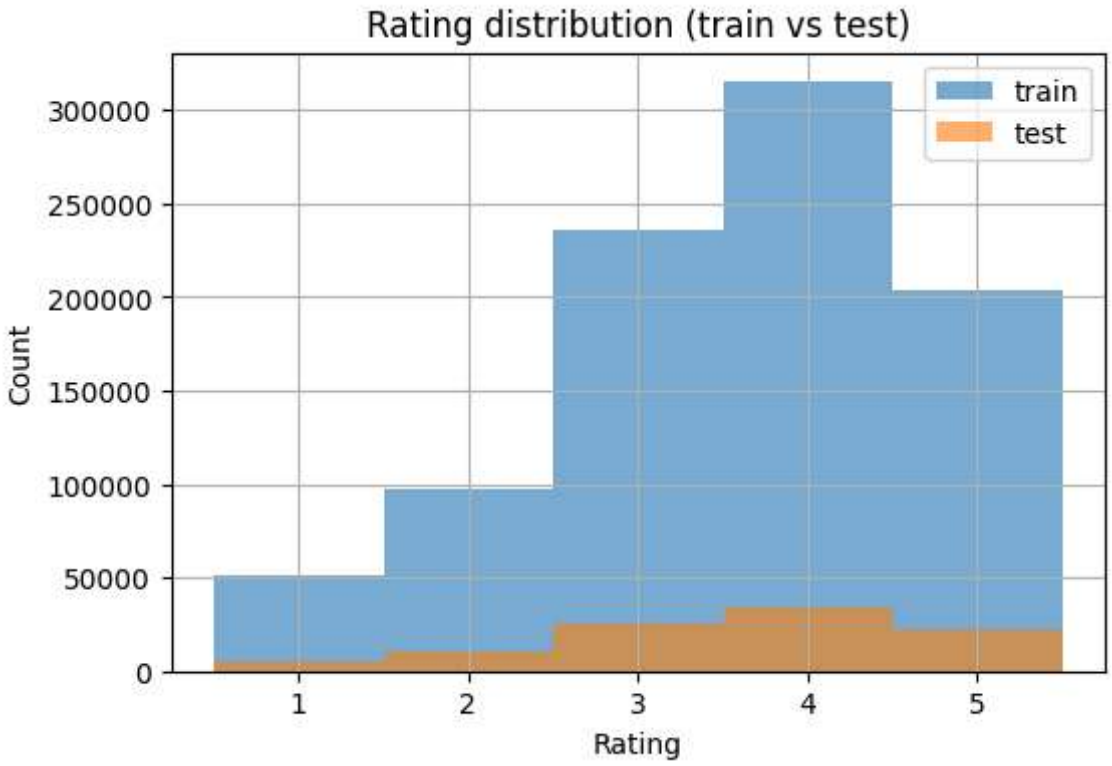
```
train: 902826 test: 97383
```

回收df，减少内存占用

In [44]:
```
import gc
del df

gc.collect()
```

Out[44]: 52

In [77]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))
train_df["rating"].hist(bins=[0.5, 1.5, 2.5, 3.5, 4.5, 5.5], alpha=0.6, label="t
test_df["rating"].hist(bins=[0.5, 1.5, 2.5, 3.5, 4.5, 5.5], alpha=0.6, label="te
plt.xlabel("Rating")
plt.ylabel("Count")
plt.title("Rating distribution (train vs test)")
plt.legend()
plt.show()
```



3. 建立训练集的稀疏矩阵，同时将测试集转换为（用户，电影，评分）元组的列表

In [45]:
```
from scipy.sparse import csr_matrix
import pandas as pd


def build_train_test(train_df, test_df):
    # category 创建映射
    user_cat = train_df["userId"].astype("category")
    movie_cat = train_df["movieId"].astype("category")

    u_train = user_cat.cat.codes.values
    m_train = movie_cat.cat.codes.values
    r_train = train_df["rating"].values

    user_categories = user_cat.cat.categories
    movie_categories = movie_cat.cat.categories

    # 测试集映射到训练集空间
    u_test = pd.Categorical(test_df["userId"], user_categories).codes
    m_test = pd.Categorical(test_df["movieId"], movie_categories).codes
```

```
    r_test = test_df["rating"].values

    # 过滤掉 -1（训练集没出现）
    mask = (u_test != -1) & (m_test != -1)
    test_tuples = list(zip(u_test[mask], m_test[mask], r_test[mask]))

    mat = csr_matrix((r_train, (u_train, m_train)), shape=(len(user_categories),

    return mat, test_tuples
train_sparse, test_tuples = build_train_test(train_df, test_df)
print("train sparse shape:", train_sparse.shape)
print("test tuples:", len(test_tuples))
```

```
train sparse shape: (6040, 3693)
test tuples: 97370
```

4. 将训练集进行svd分解防止在后续训练过程中爆内存

In [65]:
```
from sklearn.preprocessing import normalize
from sklearn.decomposition import TruncatedSVD


def build_user_vectors(train_mat,dim=128):
    svd = TruncatedSVD(n_components=dim, random_state=42)
    X = svd.fit_transform(train_mat.astype("float32"))
    X = normalize(X, axis=1)
    return X

def build_item_vectors(train_mat,dim=128):
    svd = TruncatedSVD(n_components=dim, random_state=42)
    X = svd.fit_transform(train_mat.T.astype("float32"))
    X = normalize(X, axis=1)
    return X

train_user_vectors = build_user_vectors(train_sparse)
train_item_vectors = build_item_vectors(train_sparse)
print("train user vectors shape:", train_user_vectors.shape)
print("train item vectors shape:", train_item_vectors.shape)
```

```
train user vectors shape: (6040, 128)
train item vectors shape: (3693, 128)
```

5. 通过计算用户之间的余弦相似度,进而聚合用户，然后利用相似用户评分的均值得到预测评分

In [47]:
```
from sklearn.metrics.pairwise import cosine_similarity

def predict_rating(user_vectors, train_mat, user_id, movie_id, k=20):
    user_vector = user_vectors[user_id].reshape(1, -1)
    sim = cosine_similarity(user_vector, user_vectors).flatten()

    # 排除自己
    sim[user_id] = -1

    # 取前 k 个相似用户
    top_k_users = np.argpartition(sim, -k)[-k:]
    top_k_sims = sim[top_k_users]
```

```
    # 计算加权平均评分
    ratings = train_mat[top_k_users, movie_id].toarray().flatten()
    mask = ratings > 0
    if np.sum(mask) == 0:
        return 0.0  # 没有相似用户评分，返回默认值

    weighted_ratings = ratings[mask] * top_k_sims[mask]
    prediction = np.sum(weighted_ratings) / np.sum(top_k_sims[mask])
    return prediction

# 测试预测
user_id, movie_id, true_rating = test_tuples[0]
pred_rating = predict_rating(train_user_vectors, train_sparse, user_id, movie_id
print(f"User {user_id}, Movie {movie_id}, True Rating: {true_rating}, Predicted
```

```
User 0, Movie 2155, True Rating: 5, Predicted Rating: 4.31
```

In [100…

```
import time
import inspect
from functools import wraps


def calculate_time(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        # 解析实参，拿到 pred_func（如果有）
        sig = inspect.signature(func)
        bound = sig.bind_partial(*args, **kwargs)
        pred = bound.arguments.get("pred_func", None)
        if pred is not None and pred.__name__ == "<lambda>":
            function_name = "ann"
        elif pred is None:
            function_name = "pred_rating_by_user_cf"
        else:
            function_name = pred.__name__

        start = time.perf_counter()
        result = func(*args, **kwargs)
        end = time.perf_counter()

        if pred is not None:
            print(f"{func.__name__} (pred_func={function_name}) took {end - star
        else:
            print(f"{func.__name__} took {end - start:.4f} seconds")
        return result

    return wrapper
```

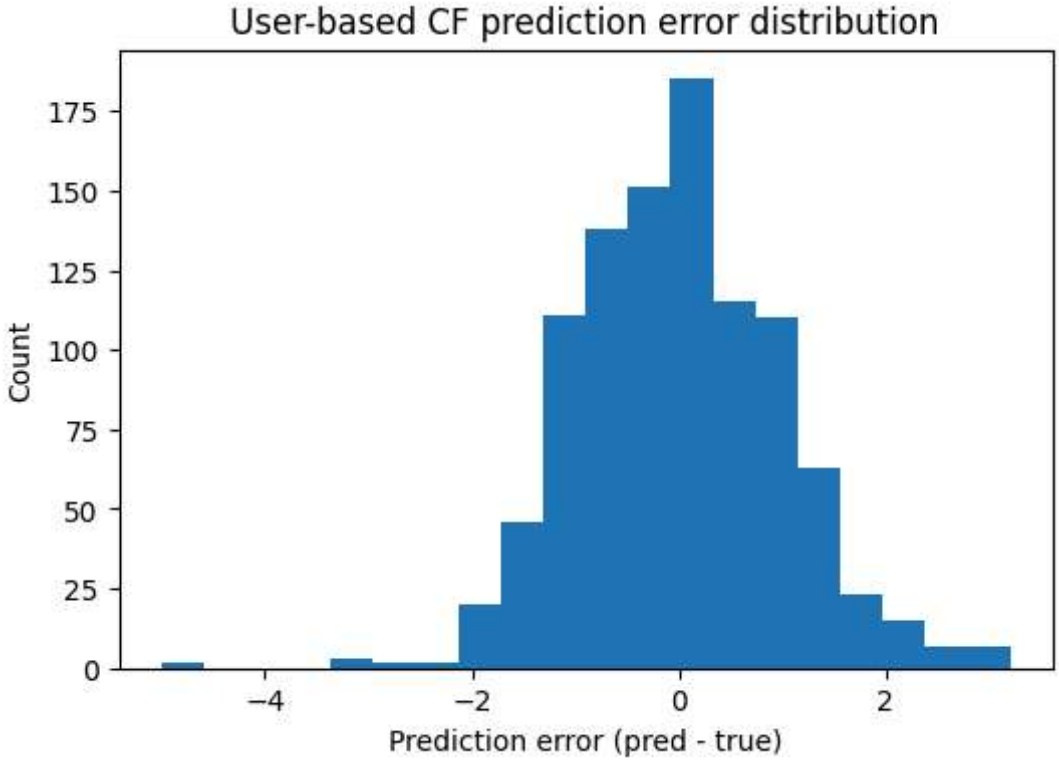
6. 计算通过余弦相似度进行预测的准确率

In [101…

```
@calculate_time
def evaluate(user_vectors, test_tuples, top_k=10, pred_func=predict_rating):
    hits = 0
    total = len(test_tuples)
    errors = []

    for user_id, movie_id, true_rating in test_tuples:
        pred_rating = pred_func(user_vectors, train_sparse, user_id, movie_id, k
        errors.append(pred_rating - true_rating)
```

```
        if abs(pred_rating - true_rating) < 0.5:
            hits += 1

    accuracy = hits / total if total > 0 else 0
    return accuracy, errors

accuracy, errors = evaluate(train_user_vectors, test_tuples[:1000], top_k=100)
print(f"Evaluation Accuracy: {accuracy:.4f}")
plt.figure(figsize=(6, 4))
plt.hist(errors, bins=20)
plt.xlabel("Prediction error (pred - true)")
plt.ylabel("Count")
plt.title("User-based CF prediction error distribution")
plt.show()
```

```
evaluate took 2.0786 seconds
Evaluation Accuracy: 0.3950
```



7. 计算通过余弦相似度进行预测的均方根误差

In [102…

```
@calculate_time
def rmse(user_vectors, test_tuples, k=100, pred_func=predict_rating):
    squared_errors = []

    for user_id, movie_id, true_rating in test_tuples:
        pred_rating = pred_func(user_vectors, train_sparse, user_id, movie_id, k
        squared_errors.append((pred_rating - true_rating) ** 2)

    mse = np.mean(squared_errors)
    return np.sqrt(mse)
rmse_value = rmse(train_user_vectors, test_tuples[:1000], k=100)
print(f"RMSE: {rmse_value:.4f}")
```

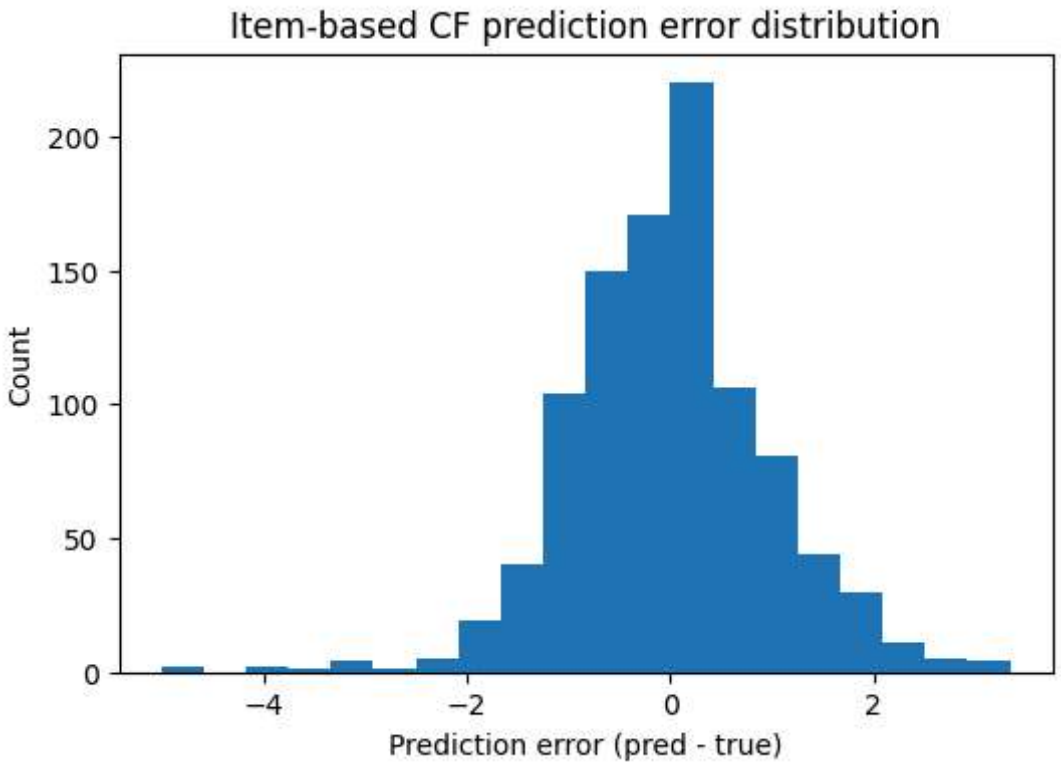
```
rmse took 1.9960 seconds
RMSE: 0.9888
```

8. 基于物品的协同过滤

In [103…

```python
def predict_rating_byitem(item_vectors, train_mat, user_id, movie_id, k=20):
    item_vector = item_vectors[movie_id].reshape(1, -1)
    sim = cosine_similarity(item_vector, item_vectors).flatten()

    # 排除自己
    sim[movie_id] = -1

    # 取前 k 个相似物品
    top_k_items = np.argpartition(sim, -k)[-k:]
    top_k_sims = sim[top_k_items]

    # 计算加权平均评分
    ratings = train_mat[user_id, top_k_items].toarray().flatten()
    mask = ratings > 0
    if np.sum(mask) == 0:
        return 0.0  # 没有相似物品评分，返回默认值

    weighted_ratings = ratings[mask] * top_k_sims[mask]
    prediction = np.sum(weighted_ratings) / np.sum(top_k_sims[mask])
    return prediction

# 测试预测
user_id, movie_id, true_rating = test_tuples[0]
predicted_rating = predict_rating_byitem(train_item_vectors, train_sparse, user_
print(f"User {user_id}, Movie {movie_id}, True Rating: {true_rating}, Predicted
accuracy, errors = evaluate(train_item_vectors, test_tuples[:1000], top_k=100, p
print(f"Evaluation Accuracy (Item-based): {accuracy:.4f}")
rmse_value = rmse(train_item_vectors, test_tuples[:1000], k=100, pred_func=predi
print(f"RMSE (Item-based): {rmse_value:.4f}")
plt.figure(figsize=(6, 4))
plt.hist(errors, bins=20)
plt.xlabel("Prediction error (pred - true)")
plt.ylabel("Count")
plt.title("Item-based CF prediction error distribution")
plt.show()
```

```
User 0, Movie 2155, True Rating: 5, Predicted Rating: 4.00
evaluate (pred_func=predict_rating_byitem) took 1.3674 seconds
Evaluation Accuracy (Item-based): 0.4390
rmse (pred_func=predict_rating_byitem) took 1.3875 seconds
RMSE (Item-based): 0.9708
```



9. 建立hnsw进行查询时需要的索引

In [ ]:
```
import hnswlib


def build_ann_index(X, ef=200, M=128):
    dim = X.shape[1]
    num_users = X.shape[0]
    index = hnswlib.Index(space="cosine", dim=dim)
    index.init_index(max_elements=num_users, ef_construction=ef, M=M)
    index.add_items(X)
    index.set_ef(ef)
    return index

ann_index = build_ann_index(train_user_vectors)
print("ANN index built.")
```

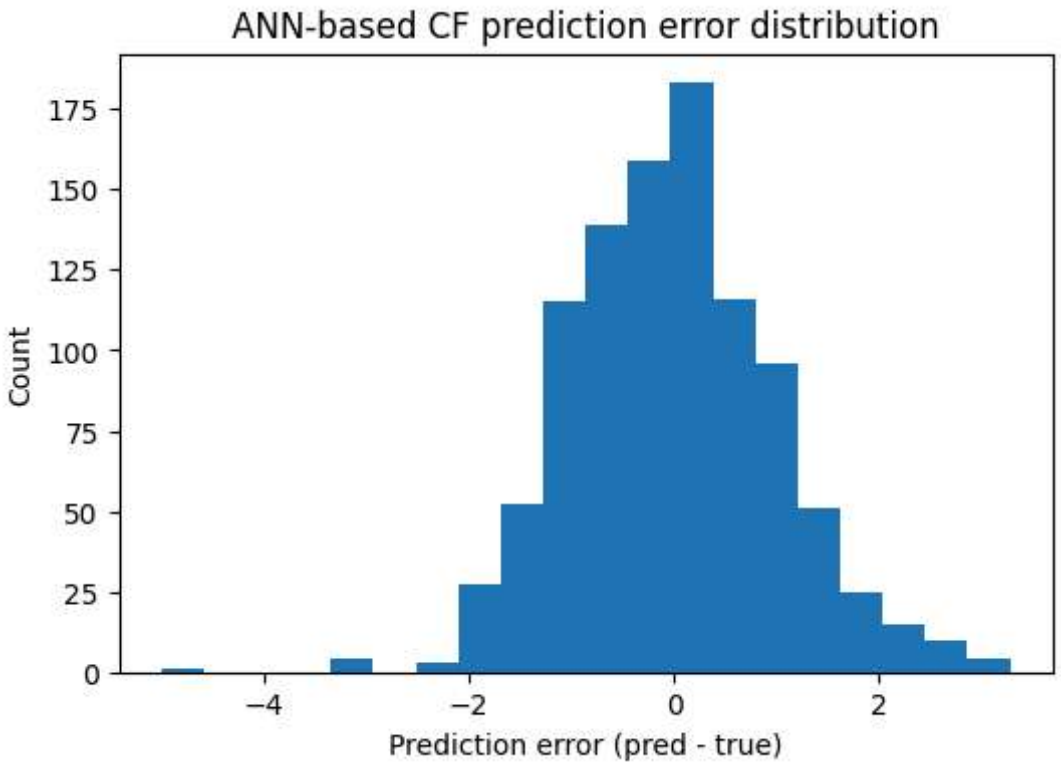
```
ANN index built.
```

8. 基于用户的协同过滤使用hnsw进行优化

In [104…
```
def get_similar_users(ann_index, user_vector, k=20):
    labels, distances = ann_index.knn_query(user_vector, k=k)
    # cosine 距离 ∈ [0, 2]，相似度 = 1 - distance
    sims = 1 - distances[0]
    users = labels[0]
    return users, sims

def predict_rating_ann(ann_index, user_vectors, train_mat, user_id, movie_id, k=
    user_vector = user_vectors[user_id].reshape(1, -1)
    top_k_users, top_k_sims = get_similar_users(ann_index, user_vector, k=k)

    # 排除自己
    mask = top_k_users != user_id
```

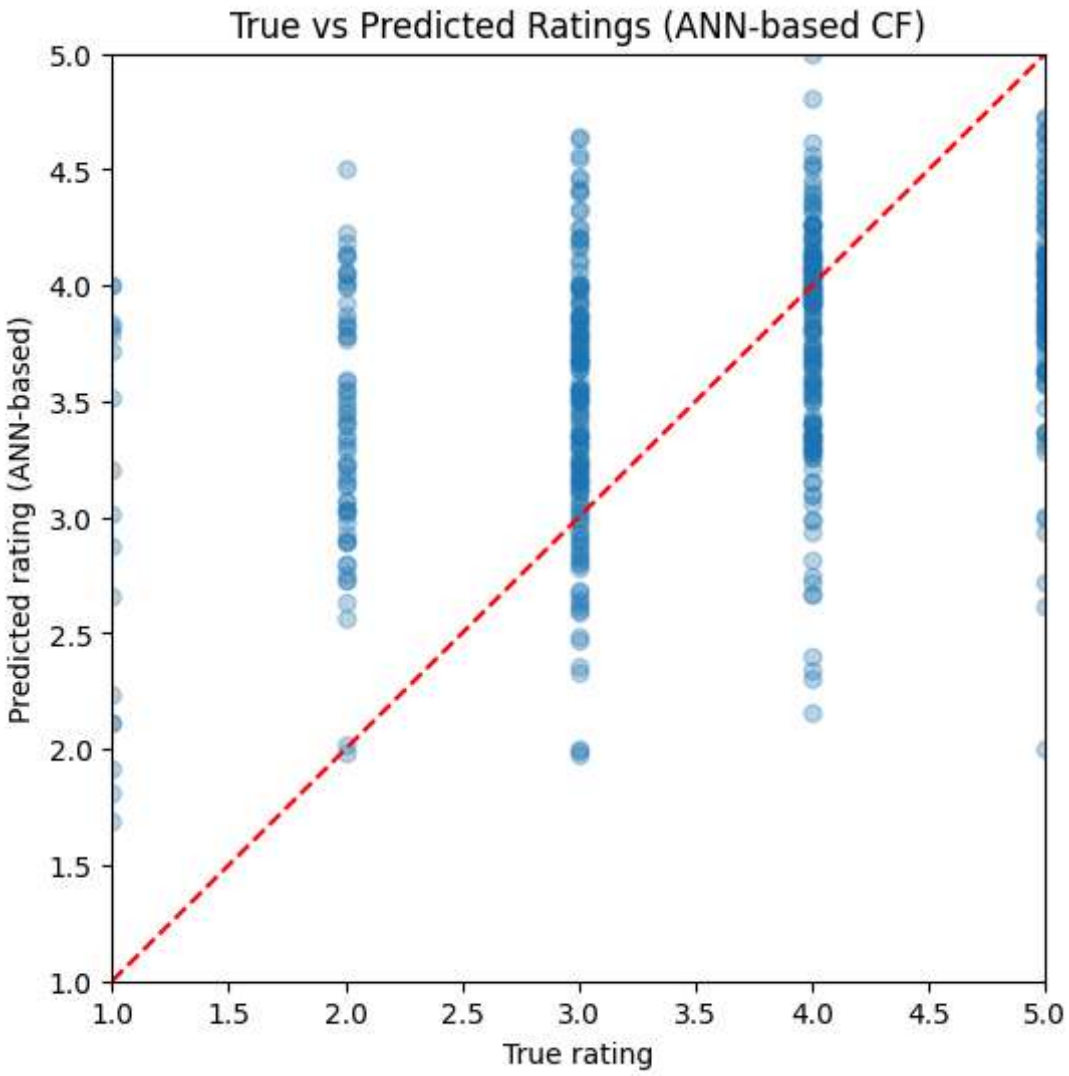
```
    top_k_users = top_k_users[mask]
    top_k_sims = top_k_sims[mask]

    # 计算加权平均评分
    ratings = train_mat[top_k_users, movie_id].toarray().flatten()
    mask = ratings > 0
    if np.sum(mask) == 0:
        return 0.0  # 没有相似用户评分，返回默认值

    weighted_ratings = ratings[mask] * top_k_sims[mask]
    prediction = np.sum(weighted_ratings) / np.sum(top_k_sims[mask])
    return prediction

# 测试预测
user_id, movie_id, true_rating = test_tuples[0]
predicted_rating = predict_rating_ann(ann_index, train_user_vectors, train_spars
print(f"User {user_id}, Movie {movie_id}, True Rating: {true_rating}, Predicted
accuracy, errors = evaluate(
    train_user_vectors,
    test_tuples[:1000],
    top_k=100,
    pred_func=lambda uv, tm, uid, mid, k: predict_rating_ann(ann_index, uv, tm,
)
print(f"Evaluation Accuracy (ANN-based): {accuracy:.4f}")
rmse_value = rmse(
    train_user_vectors,
    test_tuples[:1000],
    k=100,
    pred_func=lambda uv, tm, uid, mid, k: predict_rating_ann(ann_index, uv, tm,
)
print(f"RMSE (ANN-based): {rmse_value:.4f}")
plt.figure(figsize=(6, 4))
plt.hist(errors, bins=20)
plt.xlabel("Prediction error (pred - true)")
plt.ylabel("Count")
plt.title("ANN-based CF prediction error distribution")
plt.show()
```

```
User 0, Movie 2155, True Rating: 5, Predicted Rating: 4.13
evaluate (pred_func=ann) took 0.2343 seconds
Evaluation Accuracy (ANN-based): 0.3860
rmse (pred_func=ann) took 0.2335 seconds
RMSE (ANN-based): 0.9917
```



In [82]:

```
true_vals = []
pred_vals = []

for user_id, movie_id, true_rating in test_tuples[:500]:
    pred = predict_rating_ann(ann_index, train_user_vectors, train_sparse, user_
    true_vals.append(true_rating)
    pred_vals.append(pred)

plt.figure(figsize=(6, 6))
plt.scatter(true_vals, pred_vals, alpha=0.3)
plt.plot([1, 5], [1, 5], "r--")  # 对角线
plt.xlabel("True rating")
plt.ylabel("Predicted rating (ANN-based)")
plt.title("True vs Predicted Ratings (ANN-based CF)")
plt.xlim(1, 5)
plt.ylim(1, 5)
plt.show()
```



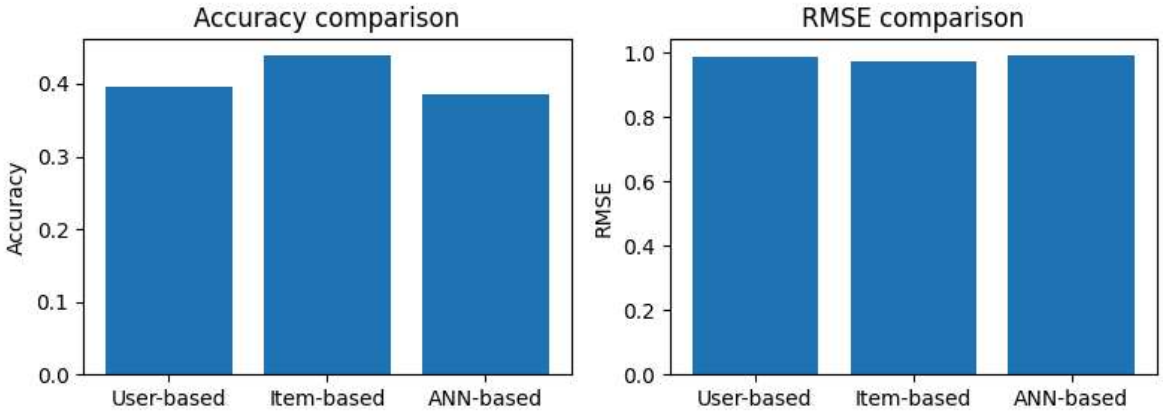
9. 对三种方法计算得到的准确率进行对比分析（这里可以看到三者的准确率是差不多的，但是使用ANN优化后速度要快不少）

In [105…

```
methods = ["User-based", "Item-based", "ANN-based"]

acc_user, _ = evaluate(train_user_vectors, test_tuples[:1000], top_k=100)
rmse_user = rmse(train_user_vectors, test_tuples[:1000], k=100)

acc_item, _ = evaluate(train_item_vectors, test_tuples[:1000], top_k=100, pred_f
rmse_item = rmse(train_item_vectors, test_tuples[:1000], k=100, pred_func=predic

acc_ann, _ = evaluate(
    train_user_vectors,
    test_tuples[:1000],
    top_k=100,
    pred_func=lambda uv, tm, uid, mid, k: predict_rating_ann(ann_index, uv, tm,
)
rmse_ann = rmse(
    train_user_vectors,
    test_tuples[:1000],
    k=100,
    pred_func=lambda uv, tm, uid, mid, k: predict_rating_ann(ann_index, uv, tm,
)

accuracies = [acc_user, acc_item, acc_ann]
```

```
rmses = [rmse_user, rmse_item, rmse_ann]

plt.figure(figsize=(8, 3))
plt.subplot(1, 2, 1)
plt.bar(methods, accuracies)
plt.ylabel("Accuracy")
plt.title("Accuracy comparison")

plt.subplot(1, 2, 2)
plt.bar(methods, rmses)
plt.ylabel("RMSE")
plt.title("RMSE comparison")
plt.tight_layout()
plt.show()
```

```
evaluate took 2.0337 seconds
rmse took 1.9784 seconds
evaluate (pred_func=predict_rating_byitem) took 1.3492 seconds
rmse (pred_func=predict_rating_byitem) took 1.3358 seconds
evaluate (pred_func=ann) took 0.2310 seconds
rmse (pred_func=ann) took 0.2314 seconds
```



9. 计算用户对所有电影的评分，并得到其中的topk个电影

In [106…

```
def recommend_topk(train_mat, user, Krec=50):
    user_vector = train_user_vectors[user].reshape(1, -1)
    top_k_users, top_k_sims = get_similar_users(ann_index, user_vector, k=100)
    watched_set = set(train_mat[user].nonzero()[1])

    score_dict = {}
    for similar_user, sim in zip(top_k_users, top_k_sims):
        user_ratings = train_mat[similar_user].toarray().flatten()
        for movie_id, rating in enumerate(user_ratings):
            if rating > 0 and movie_id not in watched_set:
                if movie_id not in score_dict:
                    score_dict[movie_id] = 0.0
                score_dict[movie_id] += rating * sim

    ranked_movies = sorted(score_dict.items(), key=lambda x: x[1], reverse=True)
    topk = [movie_id for movie_id, score in ranked_movies[:Krec]]
    return topk

topk = recommend_topk(train_sparse, user=0, Krec=10)
print("Top-K recommendations for user 0:", topk)
```

```
Top-K recommendations for user 0: [354, 2155, 1893, 572, 998, 1185, 33, 1115, 189
9, 1173]
```

10. 进行召回率的计算（这里召回率表示在预测喜欢的电影占用户真正喜欢电影的比例（具体指在5393部电影上选50部模型认为用户喜欢的电影/5393部电影中用户真正喜欢的电影））

In [107…

```python
def recall_at_k(train_mat, test_tuples, Krec=50, like_threshold=4.0):
    # 组织测试集：user → [(movie, rating)]
    test_map = {}
    for u, m, r in test_tuples:
        test_map.setdefault(u, []).append((m, r))

    sum_recall = 0
    cnt_user = 0

    for u, pairs in test_map.items():
        # 用户在测试集中真正喜欢的电影
        liked = [m for m, r in pairs if r >= like_threshold]
        if not liked:
            continue  # 没有喜欢电影就跳过，不影响指标

        rec_list = recommend_topk(train_mat, user=u, Krec=Krec)

        hit = sum(1 for m in liked if m in rec_list)
        recall = hit / len(liked)

        sum_recall += recall
        cnt_user += 1

    return sum_recall / cnt_user

topk_recall = recall_at_k(train_sparse, test_tuples, Krec=50, like_threshold=4.0
print("Top-K Recall@50:", topk_recall)
```

```
Top-K Recall@50: 0.4352164995607174
```