1m

```
epoch: 1 loss: 0.347933828830719
epoch: 2 loss: 0.2344128042459488
epoch: 3 loss: 0.24997790157794952
epoch: 4 loss: 0.24614262580871582
epoch: 5 loss: 0.25117945671081543
epoch: 6 loss: 0.24648547172546387
epoch: 7 loss: 0.24806226789951324
epoch: 8 loss: 0.24742282927036285
epoch: 9 loss: 0.24958494305610657
epoch: 10 loss: 0.245456114411354 06
epoch: 11 loss: 0.24767443537712097
epoch: 12 loss: 0.24517996609210968
epoch: 13 loss: 0.24452728033065796
epoch: 14 loss: 0.2457459270954132
epoch: 15 loss: 0.2470356673002243
epoch: 16 loss: 0.2442876249551773
epoch: 17 loss: 0.24664442241191864
epoch: 18 loss: 0.246909037232399
epoch: 19 loss: 0.24370677769184113
epoch: 20 loss: 0.24720165133476257
epoch: 21 loss: 0.24365456402301788
epoch: 22 loss: 0.2476692795753479
epoch: 23 loss: 0.2434154599905014
epoch: 24 loss: 0.24623309075832367
epoch: 25 loss: 0.24343140423297882
epoch: 26 loss: 0.24440884590148926
epoch: 27 loss: 0.24714750051498413
epoch: 28 loss: 0.24368412792682648
epoch: 29 loss: 0.24663424491882324
epoch: 30 loss: 0.24467331171035767
epoch: 31 loss: 0.2471647858619 69
epoch: 32 loss: 0.2426561713218689
epoch: 33 loss: 0.24798832833766937
epoch: 34 loss: 0.2440093308687 21
epoch: 35 loss: 0.24540740251541138
```

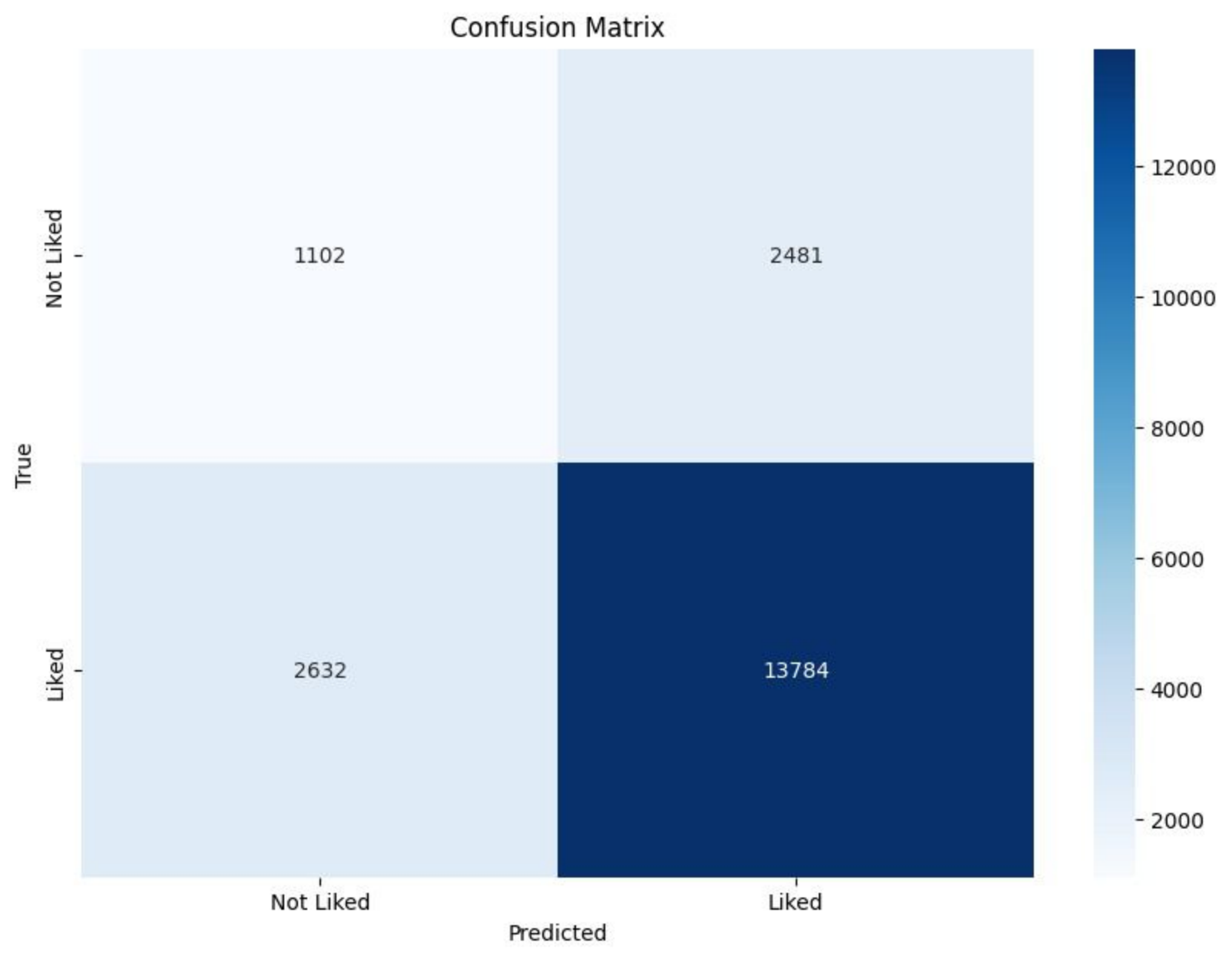
```
epoch: 32 loss: 0.24265617132186890
epoch: 33 loss: 0.24798832833766937
epoch: 34 loss: 0.244009330868721
epoch: 35 loss: 0.24540740251541138
epoch: 36 loss: 0.24706085026264190
epoch: 37 loss: 0.24297849833965302
epoch: 38 loss: 0.2457677274942398
epoch: 39 loss: 0.24576398730278015
epoch: 40 loss: 0.247294083237648
epoch: 41 loss: 0.24493278563022614
epoch: 42 loss: 0.24509644508361816
epoch: 43 loss: 0.24698859453201294
epoch: 44 loss: 0.24548377096652985
epoch: 45 loss: 0.24659368395805360
epoch: 46 loss: 0.24288439750671387
epoch: 47 loss: 0.2455139458179474
epoch: 48 loss: 0.24518926441669464
epoch: 49 loss: 0.24685978889465332
epoch: 50 loss: 0.24402110278606415
test loss: 0.2432047575712204
tensor([[1., 1., 1.,  ..., 0., 1., 0.]])
         UserId  MovieId
0             1        1
1             1        8
2             1       10
3             1       12
4             1       14
...         ...      ...
1006990     943     1674
1006991     943     1675
1006992     943     1677
1006993     943     1679
1006994     943     1681

[1006995 rows x 2 columns]
```

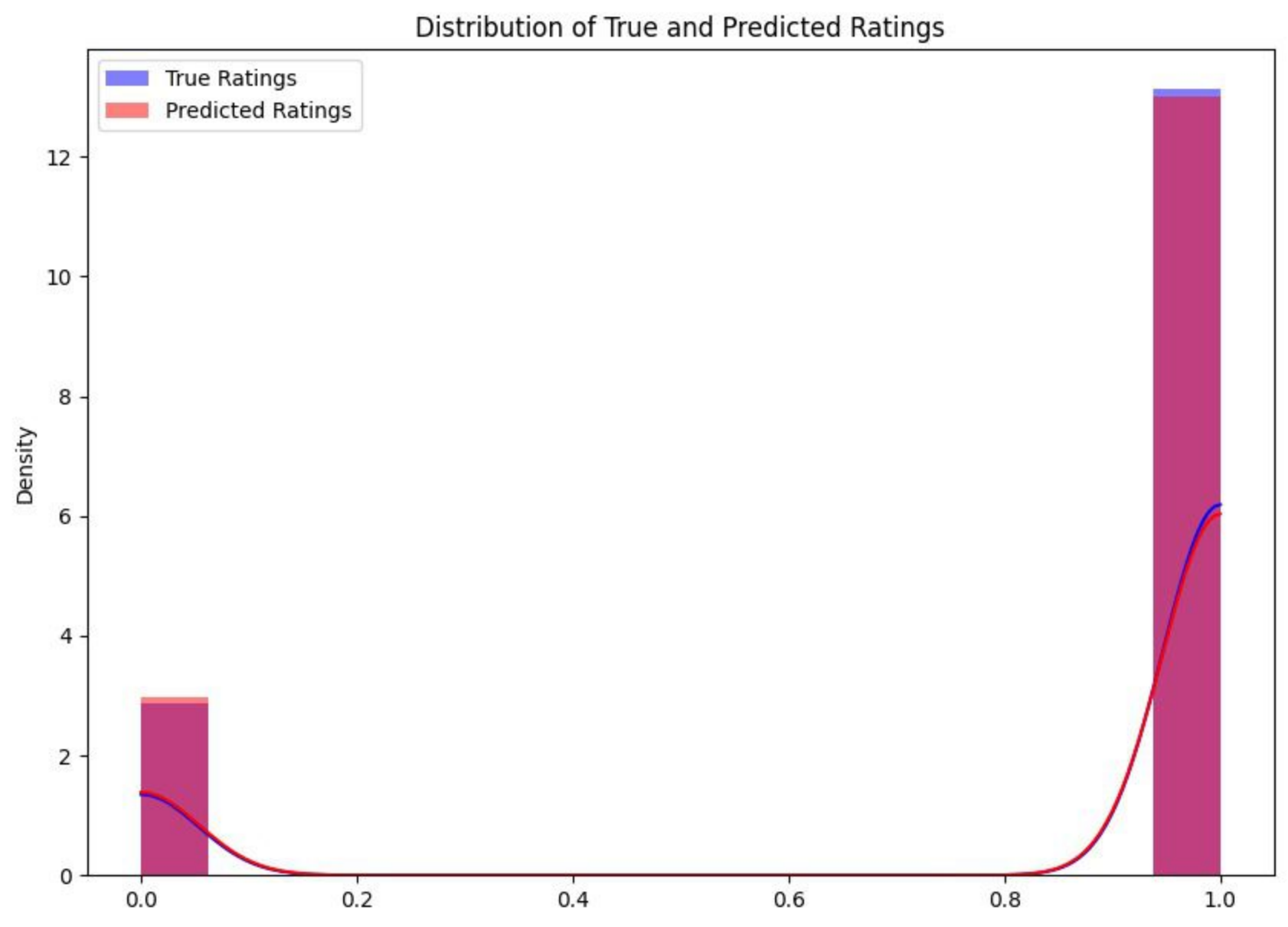
```python
import pandas as pd
import torch
from torch.autograd import Variable

# Load movie names
movies_df = pd.read_csv('/u.item', delimiter='|', encoding='latin-1', usecols=[0, 1], names=['MovieID', 'Title'])

# Function to get movie names
def get_movie_names(movie_ids, movies_df):
    return movies_df[movies_df['MovieID'].isin(movie_ids)]['Title'].tolist()

# Function to recommend movies for a user
def recommend_movies(user_id, test_set, rbm, movies_df, top_n=3):
    user_input = Variable(test_set[user_id - 1]).unsqueeze(0)
    output = rbm.predict(user_input)
    output_numpy = output.detach().numpy()[0]
    # Get movie ratings and IDs
    movie_ratings = [(movie_id + 1, rating) for movie_id, rating in enumerate(output_numpy) if rating >= 1]
    # Sort movies by rating in descending order
    movie_ratings = sorted(movie_ratings, key=lambda x: x[1], reverse=True)
    # Get the top N recommended movie IDs
    recommended_movie_ids = [movie_id for movie_id, rating in movie_ratings[:top_n]]
    recommended_movies = get_movie_names(recommended_movie_ids, movies_df)
    return recommended_movies

# Get top 3 recommendations for a specific user
user_id = 31
recommended_movies = recommend_movies(user_id, test_set, rbm, movies_df, top_n=3)
print(f"Top 3 recommended movies for user {user_id}:")
for movie in recommended_movies:
    print(movie)
```

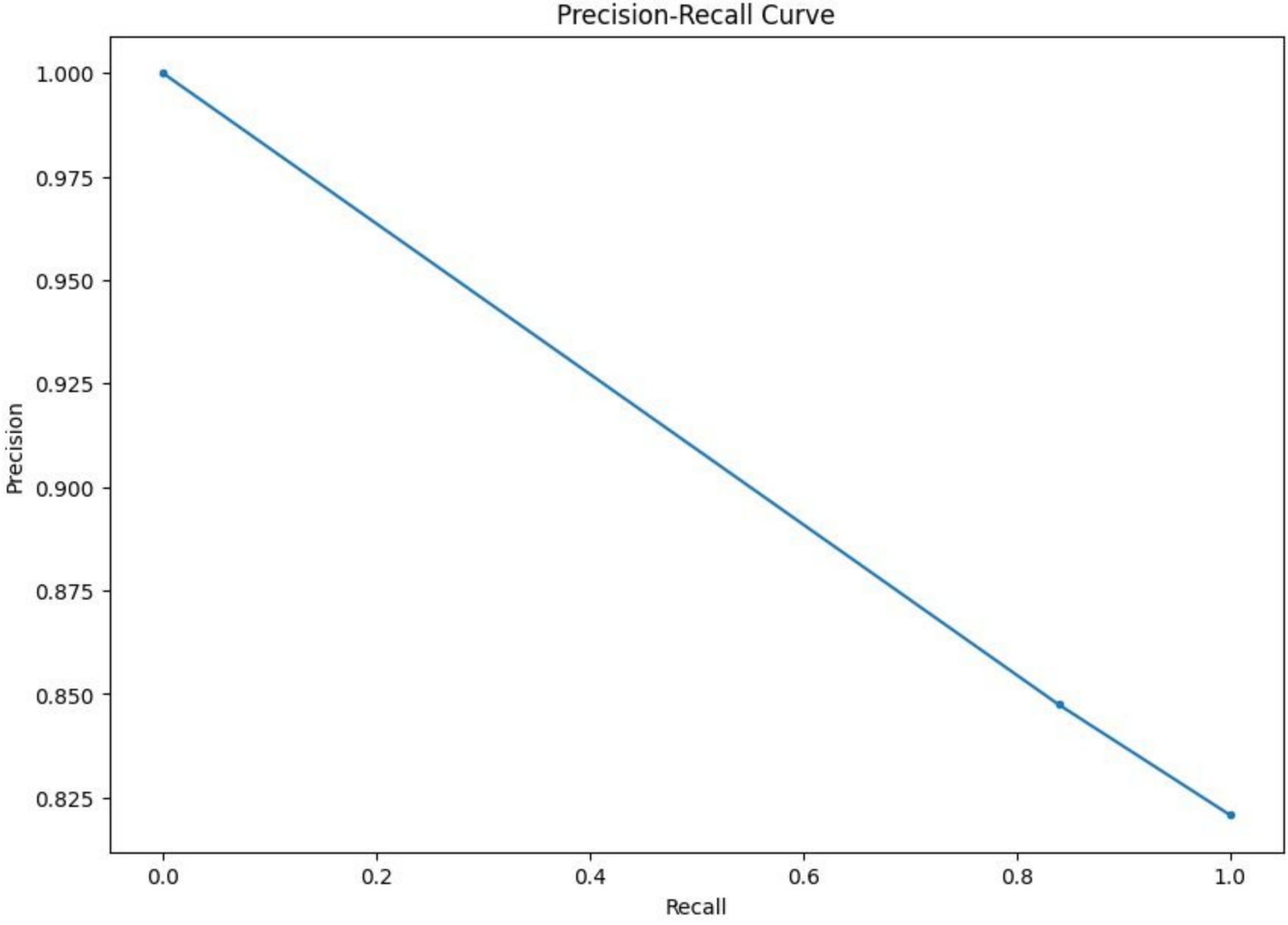
```
Top 3 recommended movies for user 31:
Toy Story (1995)
GoldenEye (1995)
Four Rooms (1995)
```

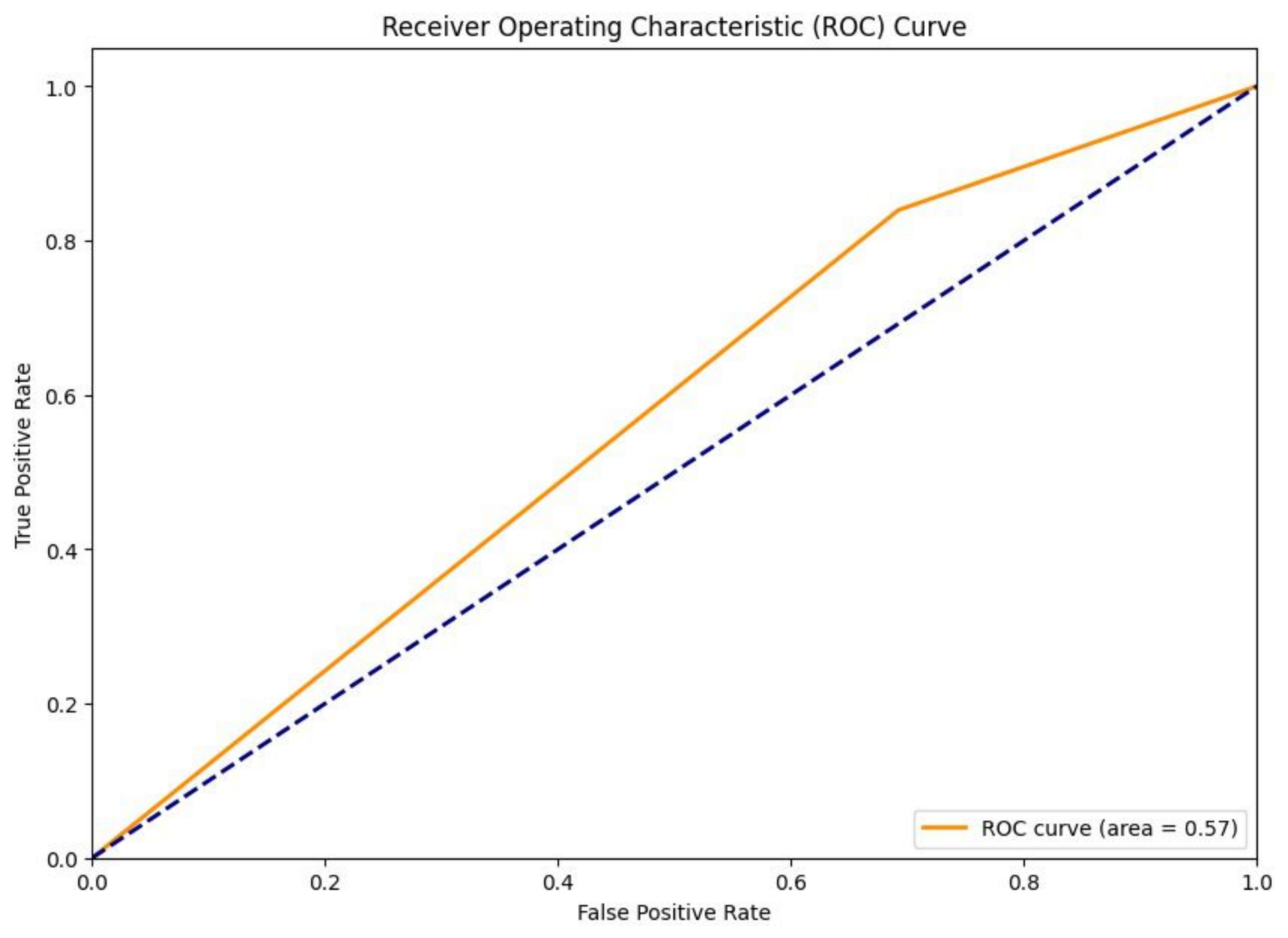
```
Test loss: tensor(0.2395)
Accuracy: 74.6037
Confusion Matrix:
[[ 1117  2466]
 [ 2613 13803]]
Classification Report:
              precision    recall  f1-score   support

   Not Liked       0.30      0.31      0.31      3583
       Liked       0.85      0.84      0.84     16416

    accuracy                           0.75     19999
   macro avg       0.57      0.58      0.58     19999
weighted avg       0.75      0.75      0.75     19999

Analytical Results:
Predicted Rating     0      1
True Rating
0                 1117   2466
1                 2613  13803
```

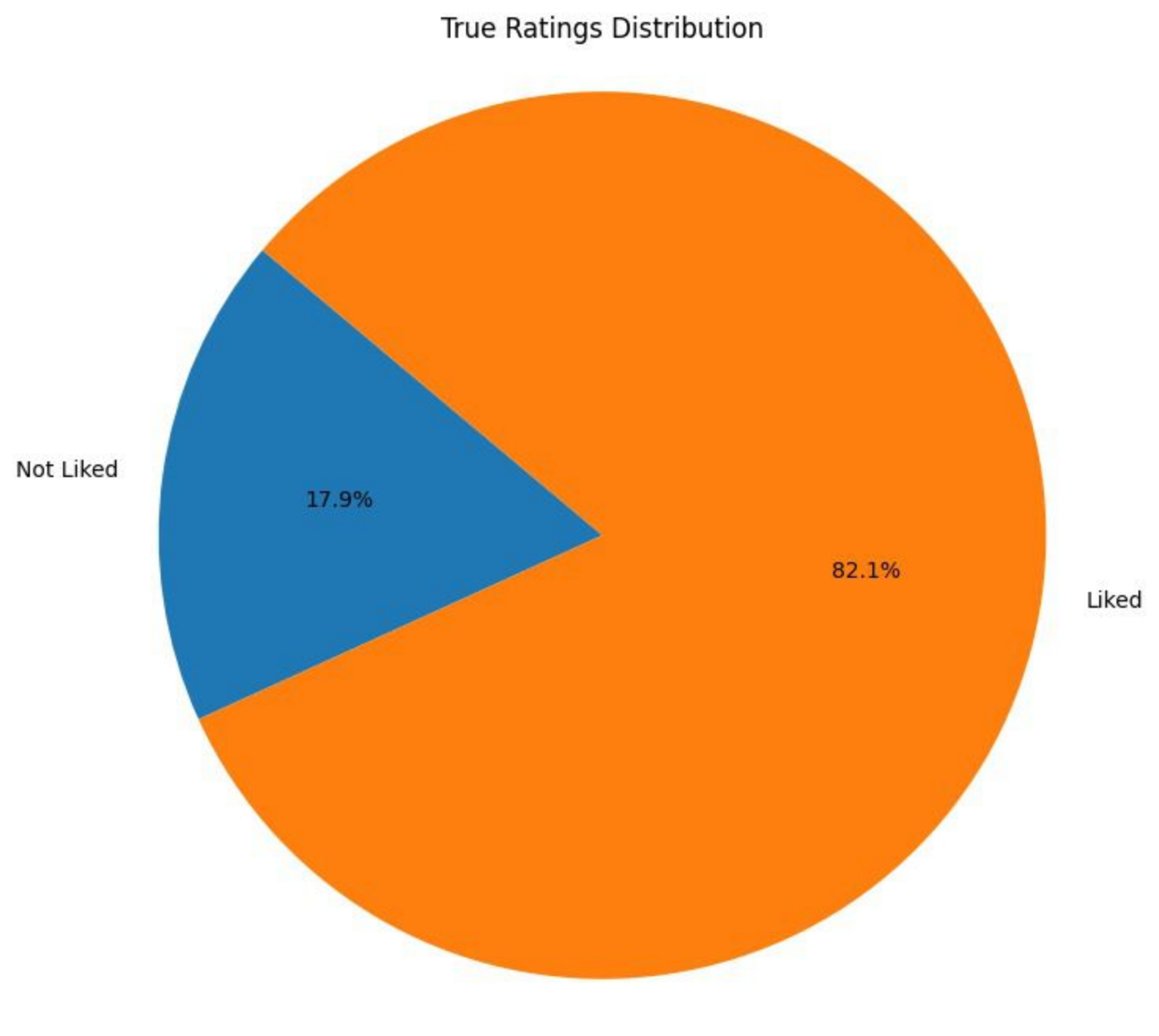
```
Test loss: tensor(0.2450)
Accuracy: 74.4337
Confusion Matrix:
[[ 1102  2481]
 [ 2632 13784]]
Classification Report:
              precision    recall  f1-score   support

   Not Liked       0.30      0.31      0.30      3583
       Liked       0.85      0.84      0.84     16416

    accuracy                           0.74     19999
   macro avg       0.57      0.57      0.57     19999
weighted avg       0.75      0.74      0.75     19999

Analytical Results:
Predicted Rating     0      1
True Rating
0                 1102   2481
1                 2632  13784
```

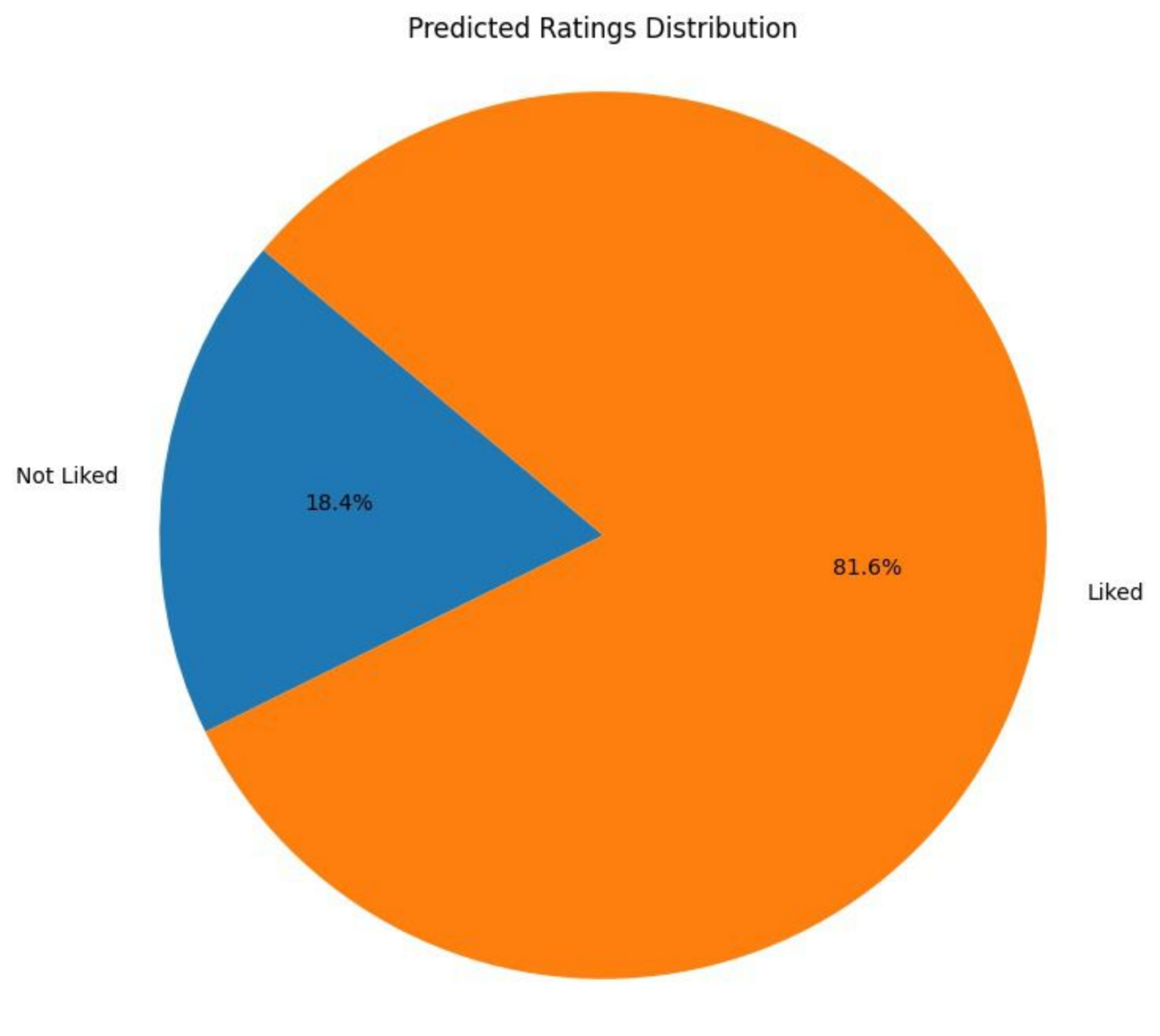

Confusion Matrix
Not Liked
Liked
True
1102
2481
2632
13784
Not Liked
Liked
Predicted
12000
10000
8000
6000
4000
2000

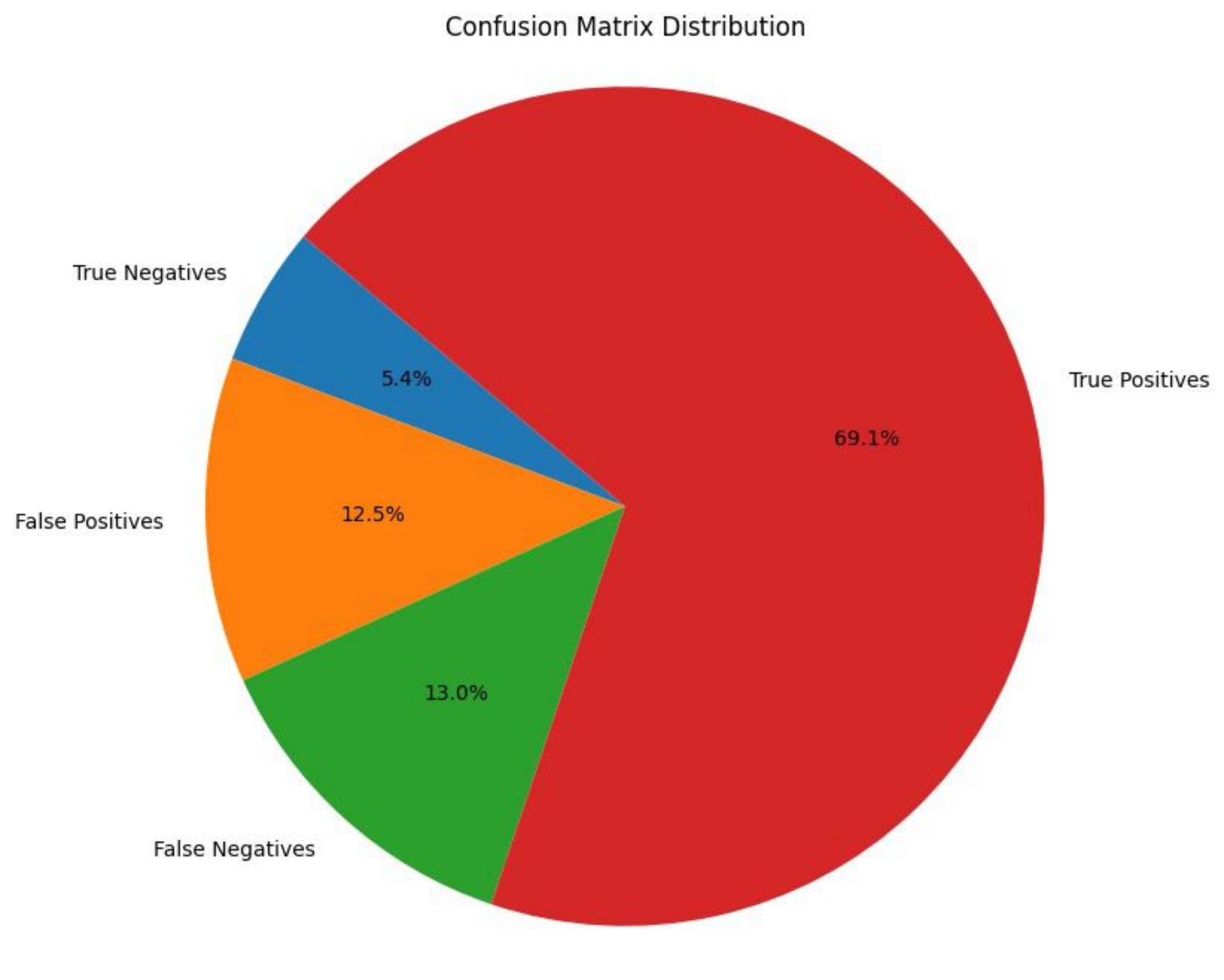

Distribution of True and Predicted Ratings
True Ratings
Predicted Ratings
Density
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0

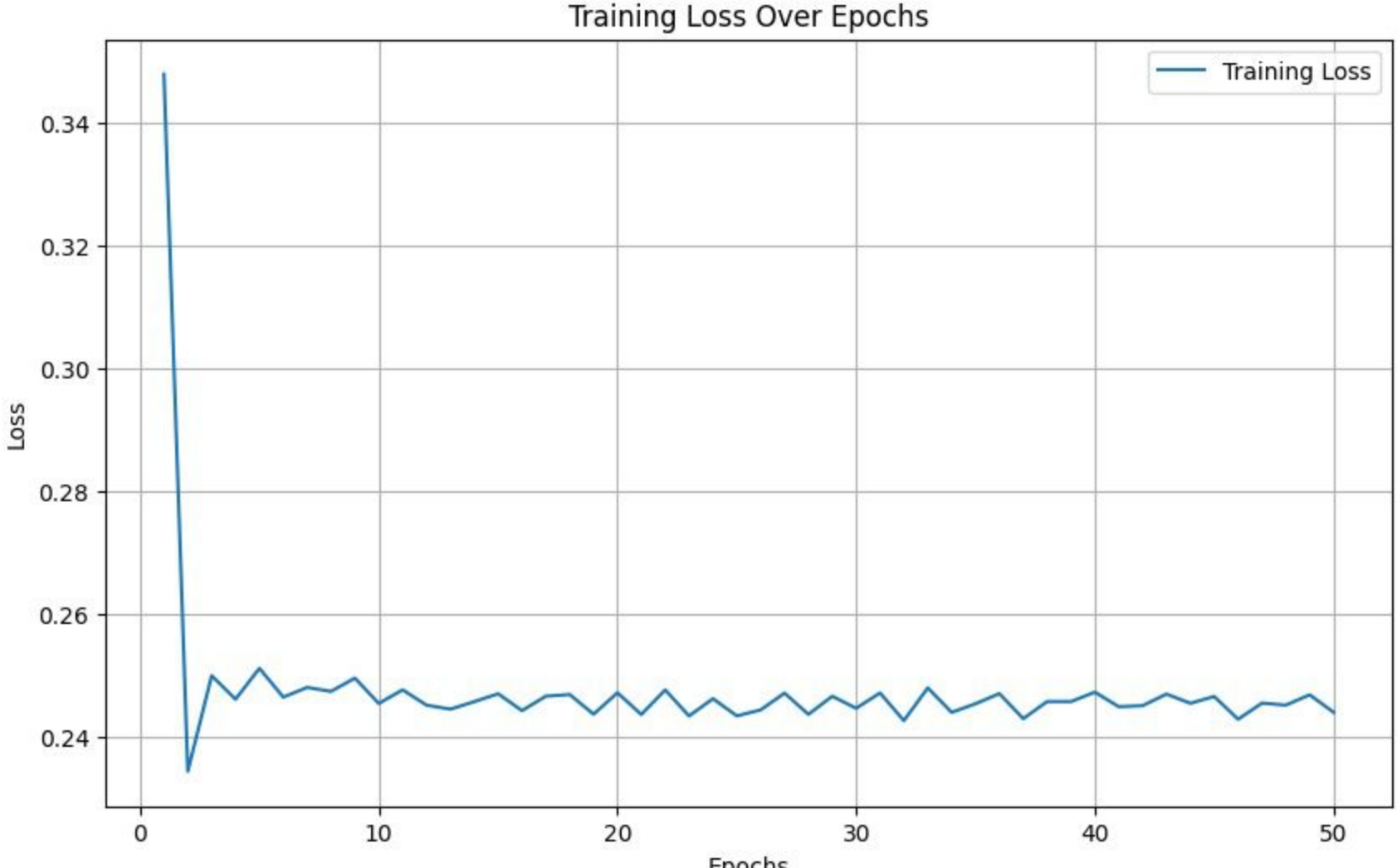

Precision-Recall Curve
Precision
Recall
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.0
0.2
0.4
0.6
0.8
1.0

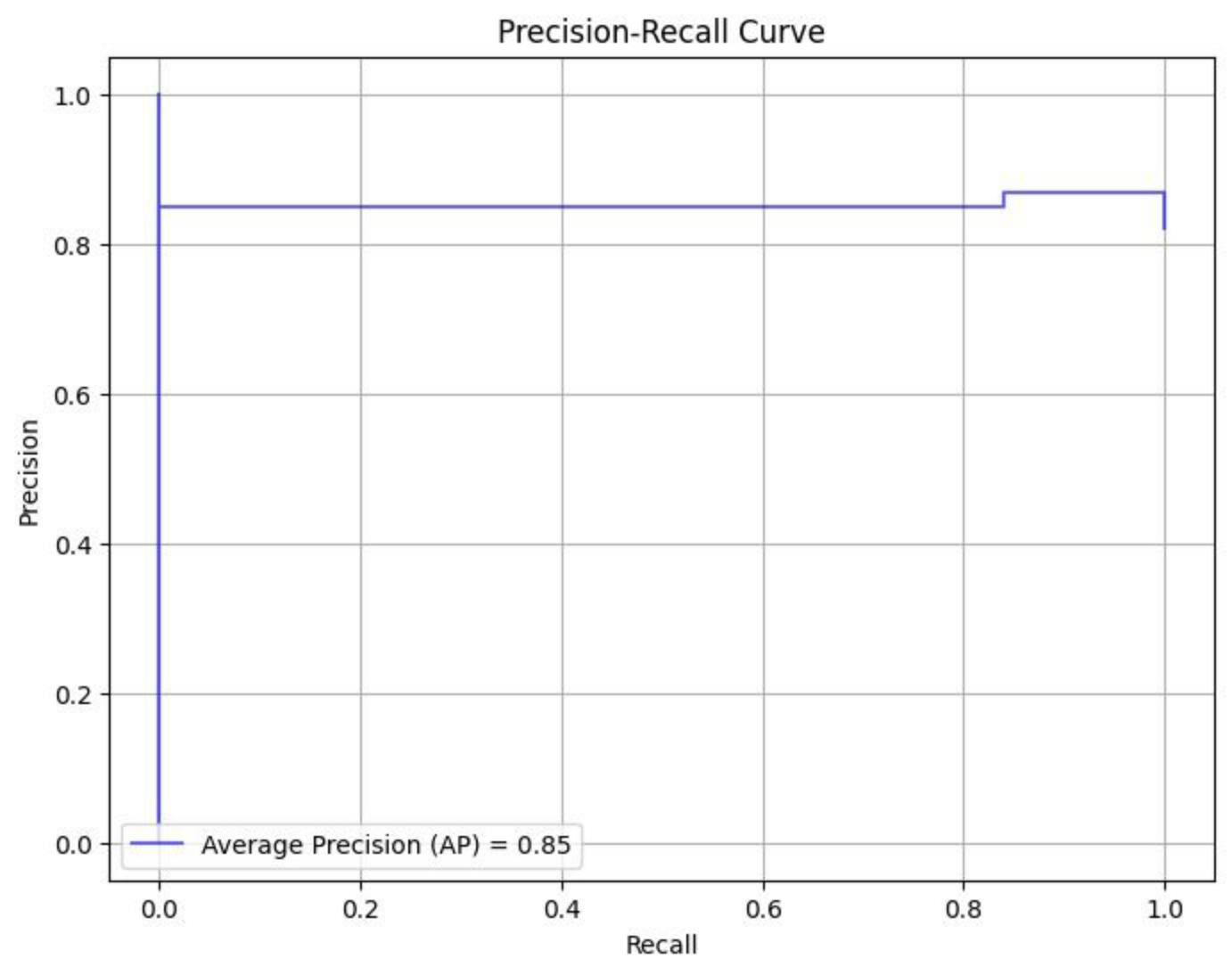

Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.57)


True Ratings Distribution
Not Liked
17.9%
82.1%
Liked


Predicted Ratings Distribution
Not Liked
18.4%
81.6%
Liked


Confusion Matrix Distribution
True Negatives
5.4%
True Positives
69.1%
False Positives
12.5%
13.0%
False Negatives


Training Loss Over Epochs
Training Loss
0.34
0.32
0.30
0.28
0.26
0.24
Loss
0
10
20
30
40
50
Epochs


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Average Precision (AP) = 0.85