```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Concatenate, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split

# Simulated dataset generation (user_id, item_id, rating)
def generate_synthetic_data(num_users, num_items, num_samples):
    users = np.random.randint(0, num_users, num_samples)
    items = np.random.randint(0, num_items, num_samples)
    ratings = np.random.randint(1, 6, num_samples)  # Ratings between 1 and 5
    return users, items, ratings

# Parameters
num_users = 1000
num_items = 500
embedding_dim = 50
num_samples = 100000

# Generate synthetic data
users, items, ratings = generate_synthetic_data(num_users, num_items, num_samples)

# Train-test split
X = np.stack([users, items], axis=1)
y = ratings
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model definition
user_input = Input(shape=(1,), name="user_input")
item_input = Input(shape=(1,), name="item_input")

# User embedding
user_embedding = Embedding(input_dim=num_users, output_dim=embedding_dim, name="user_embedding")(user_input)
user_vector = Flatten()(user_embedding)

# Item embedding
item_embedding = Embedding(input_dim=num_items, output_dim=embedding_dim, name="item_embedding")(item_input)
item_vector = Flatten()(item_embedding)

# Concatenate embeddings
concatenated = Concatenate()([user_vector, item_vector])

# Dense layers
x = Dense(128, activation="relu")(concatenated)
x = Dropout(0.3)(x)
x = Dense(64, activation="relu")(x)
x = Dropout(0.3)(x)
output = Dense(1, activation="linear", name="rating_output")(x)

# Build model
model = Model(inputs=[user_input, item_input], outputs=output)
model.compile(optimizer=Adam(learning_rate=0.001), loss="mse", metrics=["mae"])

# Train the model
history = model.fit(
    [X_train[:, 0], X_train[:, 1]],
    y_train,
    batch_size=128,
    epochs=10,
    validation_data=([X_test[:, 0], X_test[:, 1]], y_test),
    verbose=1
)

# Evaluate the model
test_loss, test_mae = model.evaluate([X_test[:, 0], X_test[:, 1]], y_test, verbose=0)
print(f"Test Loss: {test_loss:.4f}, Test MAE: {test_mae:.4f}")

# Make predictions
user_id = 42  # Example user
item_ids = np.arange(num_items)  # All items
predicted_ratings = model.predict([np.full_like(item_ids, user_id), item_ids])
recommended_items = np.argsort(predicted_ratings[:, 0])[-10:][::-1]  # Top 10 items

print(f"Recommended items for user {user_id}: {recommended_items}")
```

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 3.9417 - mae: 1.6179 - val_loss: 2.0372 - val_mae: 1.2295
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 2.1507 - mae: 1.2609 - val_loss: 2.0482 - val_mae: 1.2364
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - loss: 2.1180 - mae: 1.2515 - val_loss: 2.0411 - val_mae: 1.2339
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 2.1074 - mae: 1.2509 - val_loss: 2.0551 - val_mae: 1.2408
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 2.0720 - mae: 1.2397 - val_loss: 2.0665 - val_mae: 1.2474
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 2.0289 - mae: 1.2247 - val_loss: 2.0708 - val_mae: 1.2473
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 1.9815 - mae: 1.2114 - val_loss: 2.0930 - val_mae: 1.2546
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 1.9385 - mae: 1.1975 - val_loss: 2.0984 - val_mae: 1.2557
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 1.8894 - mae: 1.1803 - val_loss: 2.1347 - val_mae: 1.2659
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - loss: 1.8384 - mae: 1.1598 - val_loss: 2.1415 - val_mae: 1.2666
Test Loss: 2.1415, Test MAE: 1.2666
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Recommended items for user 42: [408 233 499 306 108 440  67  81 415 420]
```