Step 1

pip install transformers torch scikit-learn pandas numpy

Step 2

import pandas as pd

import numpy as np

from transformers import RobertaTokenizer, RobertaModel

import torch

import torch.nn as nn

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

# Load sample data

# Assuming data has columns: 'service\_name', 'review', 'cost', 'latency', 'availability', 'scalability'

df = pd.read\_csv("cloud\_services\_reviews.csv")

# Initialize RoBERTa tokenizer and model

tokenizer = RobertaTokenizer.from\_pretrained('roberta-base')

roberta\_model = RobertaModel.from\_pretrained('roberta-base')

# Function to extract embeddings

def get\_roberta\_embedding(text):

inputs = tokenizer(text, return\_tensors='pt', truncation=True, padding=True, max\_length=64)

with torch.no\_grad():

outputs = roberta\_model(\*\*inputs)

return outputs.last\_hidden\_state[:, 0, :].squeeze().numpy() # [CLS] token

# Generate embeddings for reviews

print("Extracting RoBERTa embeddings...")

df['embedding'] = df['review'].apply(get\_roberta\_embedding)

# Normalize numerical criteria

criteria\_cols = ['cost', 'latency', 'availability', 'scalability']

scaler = MinMaxScaler()

df[criteria\_cols] = scaler.fit\_transform(df[criteria\_cols])

# Prepare features and labels for DL model

X\_text = np.stack(df['embedding'].values)

X\_numeric = df[criteria\_cols].values

X\_combined = np.hstack([X\_text, X\_numeric])

y = df[['availability', 'scalability']] # Target could be composite or multi-output

# Split dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_combined, y.values, test\_size=0.2, random\_state=42)

# Define simple Deep Neural Network

class CloudRankNN(nn.Module):

def \_\_init\_\_(self, input\_size, output\_size):

super(CloudRankNN, self).\_\_init\_\_()

self.fc1 = nn.Linear(input\_size, 256)

self.relu1 = nn.ReLU()

self.fc2 = nn.Linear(256, 128)

self.relu2 = nn.ReLU()

self.fc3 = nn.Linear(128, output\_size)

def forward(self, x):

out = self.relu1(self.fc1(x))

out = self.relu2(self.fc2(out))

return self.fc3(out)

# Model training

model = CloudRankNN(input\_size=X\_combined.shape[1], output\_size=y.shape[1])

criterion = nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Convert data to tensors

X\_train\_tensor = torch.tensor(X\_train, dtype=torch.float32)

y\_train\_tensor = torch.tensor(y\_train, dtype=torch.float32)

# Train loop

print("Training deep learning model...")

for epoch in range(30):

outputs = model(X\_train\_tensor)

loss = criterion(outputs, y\_train\_tensor)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if epoch % 5 == 0:

print(f"Epoch [{epoch}/30], Loss: {loss.item():.4f}")

# Get predictions for MCDA

X\_all\_tensor = torch.tensor(X\_combined, dtype=torch.float32)

predicted\_scores = model(X\_all\_tensor).detach().numpy()

# Add prediction as new criteria for MCDA

df['predicted\_score'] = predicted\_scores.mean(axis=1)

# ------- MCDA using TOPSIS ----------

def topsis(matrix, weights, impacts):

norm\_matrix = matrix / np.sqrt((matrix \*\* 2).sum(axis=0))

weighted\_matrix = norm\_matrix \* weights

ideal\_best = np.max(weighted\_matrix, axis=0) if impacts == '+' else np.min(weighted\_matrix, axis=0)

ideal\_worst = np.min(weighted\_matrix, axis=0) if impacts == '+' else np.max(weighted\_matrix, axis=0)

dist\_best = np.sqrt(((weighted\_matrix - ideal\_best) \*\* 2).sum(axis=1))

dist\_worst = np.sqrt(((weighted\_matrix - ideal\_worst) \*\* 2).sum(axis=1))

score = dist\_worst / (dist\_best + dist\_worst)

return score

# Apply TOPSIS with weights and impact directions

criteria\_matrix = df[['cost', 'latency', 'availability', 'scalability', 'predicted\_score']].values

weights = np.array([0.2, 0.2, 0.2, 0.2, 0.2]) # Equal weights (customizable)

impacts = np.array(['-', '-', '+', '+', '+']) # cost and latency are negatives

# Convert impact to numerical mask

impacts\_mask = np.array([1 if i == '+' else -1 for i in impacts])

topsis\_scores = topsis(criteria\_matrix, weights, impacts\_mask)

# Add ranking to dataframe

df['rank\_score'] = topsis\_scores

df = df.sort\_values(by='rank\_score', ascending=False)

# Final output

print("\nTop Cloud Services based on Hybrid RoBERTa + MCDA Ranking:")

print(df[['service\_name', 'rank\_score']].head(10))