In [ ]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import torch
from transformers import BertTokenizer, BertModel
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error

# Load datasets
def load_data(file_path):
    return pd.read_csv(file_path)

# Preprocess data
def preprocess_data(data):
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    tokens = tokenizer(data['text'].tolist(), padding=True, truncation=True, return_tensors='pt')
    return tokens
# Define model
class MultitaskBERT(torch.nn.Module):
    def __init__(self):
        super(MultitaskBERT, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.dropout = torch.nn.Dropout(0.1)
        self.classifier = torch.nn.Linear(self.bert.config.hidden_size, 1)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pooled_output = outputs[1]
        pooled_output = self.dropout(pooled_output)
        return self.classifier(pooled_output)

# Train model
def train_model(model, train_data,
 val_data, epochs=3):
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)
    loss_fn = torch.nn.BCEWithLogitsLoss()
    for epoch in range(epochs):
        model.train()
        for batch in train_data:
            optimizer.zero_grad()
            outputs = model(batch['input_ids'], batch['attention_mask'])
            loss = loss_fn(outputs, batch['labels'])
            loss.backward ()
            optimizer.step()
        model.eval()
        val_loss = 0
        for batch in val_data:
            with torch.no_grad():
                outputs = model(batch['input_ids'], batch['attention_mask'])
                loss = loss_fn(outputs, batch['labels'])
                val_loss += loss.item()
        print(f'Epoch {epoch+1}, Validation Loss: {val_loss/len(val_data)}')

# Evaluate model
def evaluate_model(model, test_data):
    model.eval()
    predictions = []
    for batch in test_data:
        with torch.no_grad():
            outputs = model(batch['input_ids'], batch['attention_mask'])
            predictions.append(outputs)
    return predictions

# Main function
def main():
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js