```
!pip install -U transformers datasets accelerate -q
!pip install -U tensorboard -q

from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import torch
from sklearn.preprocessing import LabelEncoder
from transformers import DistilBertTokenizer, BertTokenizer, BertForSequenceClas
from sklearn.metrics import classification_report
import numpy as np

# ---------------------------
# 1. Load and Preprocess Data
# ---------------------------

# Paths to your dataset files
train_path = "/content/drive/MyDrive/SentimentAnalysisData/SentimentAnalysisTrai
test_path = "/content/drive/MyDrive/SentimentAnalysisData/SentimentAnalysisTest.

# Load CSVs (note: test file has no header)

# Load the dataset
train_df = pd.read_csv(train_path, encoding='latin-1')
test_df = pd.read_csv(test_path, encoding='latin-1', header=None)

# Define column names based on the dataset description
columns = ['polarity', 'id', 'date', 'query', 'user', 'text']
train_df.columns = columns
test_df.columns = columns

# Keep only the polarity and text columns
train_df = train_df[['text', 'polarity']]
test_df = test_df[['text', 'polarity']]

# Filter valid polarity values (0, 2, 4)
train_df = train_df[train_df['polarity'].isin([0, 2, 4])]
test_df = test_df[test_df['polarity'].isin([0, 2, 4])]

# Map polarity to sentiment labels
sentiment_map = {0: 'Negative', 2: 'Neutral', 4: 'Positive'}
train_df['sentiment'] = train_df['polarity'].map(sentiment_map)
test_df['sentiment'] = test_df['polarity'].map(sentiment_map)

# Encode sentiment labels using LabelEncoder
label_encoder = LabelEncoder()
```

```
label_encoder.fit(['Negative', 'Neutral', 'Positive'])
train_df['label'] = label_encoder.transform(train_df['sentiment'])
test_df['label'] = label_encoder.transform(test_df['sentiment'])

# Prepare lists of texts and labels
train_texts = train_df['text'].tolist()
train_labels = train_df['label'].tolist()
test_texts = test_df['text'].tolist()
test_labels = test_df['label'].tolist()

from concurrent.futures import ThreadPoolExecutor
from tqdm import tqdm

# ---------------------------
# 2. Tokenize the Data (Optimized with Parallel Processing)
# ---------------------------
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")


def tokenize_batch(texts):
    return tokenizer.batch_encode_plus(
        texts,
        truncation=True,
        padding='max_length',  # Pad to max length
        max_length=128,
        return_tensors="pt"
    )



# Using ThreadPoolExecutor to process tokenization in parallel
def parallel_tokenize(texts, batch_size=64, num_workers=16):
    with ThreadPoolExecutor(max_workers=num_workers) as executor:
        batches = [texts[i:i + batch_size] for i in range(0, len(texts), batch_s
        results = list(tqdm(executor.map(tokenize_batch, batches), total=len(bat
    # Combine the results from all batches
    input_ids = torch.cat([result['input_ids'] for result in results], dim=0)
    attention_mask = torch.cat([result['attention_mask'] for result in results],
    return {'input_ids': input_ids, 'attention_mask': attention_mask}

# Tokenize train and test datasets in parallel
train_encodings = parallel_tokenize(train_texts, batch_size=128, num_workers=64)
test_encodings = parallel_tokenize(test_texts, batch_size=128, num_workers=64)




# ---------------------------
# 3. Create a Custom Dataset
# ---------------------------
```

```
class SentimentDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        # Return a dictionary with input_ids, attention_mask, and labels
        item = {key: val[idx].clone().detach() for key, val in self.encodings.it
        item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = SentimentDataset(train_encodings, train_labels)
test_dataset = SentimentDataset(test_encodings, test_labels)

# ---------------------------
# 4. Load the Pretrained Model
# ---------------------------

from transformers import DistilBertForSequenceClassification

model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-unc

# Move model to GPU if available
if torch.cuda.is_available():
    model.to('cuda')

# ---------------------------
# 5. Set Up Training Arguments with GPU Support
# ---------------------------
from datasets import Dataset
from transformers import Trainer, TrainingArguments
import torch
from torch.utils.data import DataLoader, Subset
import numpy as np

# Reduce batch size and number of workers (adjust based on GPU memory)
train_loader = DataLoader(train_dataset, batch_size=32, num_workers=16, pin_memo
test_loader = DataLoader(test_dataset, batch_size=32, num_workers=16, pin_memory

# Training arguments for GPU
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=16,
    eval_strategy="epoch",
```

```
    eval_strategy="epoch",
    logging_strategy="epoch",
    save_strategy="epoch",
    gradient_accumulation_steps=4,
    report_to="none",
    disable_tqdm=False,
    dataloader_num_workers=8,
    logging_dir='./logs',
    fp16=torch.cuda.is_available(),

)

# ---------------------------
# Define the Trainer and Train the Model with 10% data
# ---------------------------
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)

# Start training
trainer.train()

# ---------------------------
# 7. Evaluate the Model and Output a Classification Report
# ---------------------------
predictions = trainer.predict(test_dataset)
y_pred = np.argmax(predictions.predictions, axis=1)

print(classification_report(test_labels, y_pred, target_names=label_encoder.clas
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
  warnings.warn(
100%|██████████| 8192/8192 [07:04<00:00, 19.29it/s]
100%|██████████| 5/5 [00:00<00:00, 17.67it/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not insta
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo,
```

model.safetensors: 100% 268M/268M [00:00<00:00, 328MB/s]

```
Some weights of DistilBertForSequenceClassification were not initialized fr
You should probably TRAIN this model on a down-stream task to be able to us
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624:
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624:
  warnings.warn(
```

[24576/24576 2:18:07, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.280500	3.790577
2	0.216700	5.249511
3	0.162800	6.652027

```
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624:
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624:
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624:
  warnings.warn(
              precision    recall  f1-score   support

    Negative       0.63      0.87      0.73       178
     Neutral       0.00      0.00      0.00       140
    Positive       0.59      0.81      0.69       198

    accuracy                           0.61       516
   macro avg       0.41      0.56      0.47       516
weighted avg       0.45      0.61      0.52       516

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```