In [ ]:
```
pip install notebook
```

In [ ]:
```
!pip install torch transformers
```

In [ ]:
```
import pandas as pd
from transformers import BertTokenizer

# Load datasets
sst_train = pd.read_csv('data/ids-sst-train.csv')
sst_dev = pd.read_csv('data/ids-sst-dev.csv')
sst_test = pd.read_csv('data/ids-sst-test-student.csv')
```

In [ ]:
```
import torch
from transformers import BertModel

class minBERTModel(torch.nn.Module):
    def __init__(self):
        super(minBERTModel, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.dropout = torch.nn.Dropout(0.1)
        self.classifier = torch.nn.Linear(768, 5)  # Assuming 5 sentiment classes

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

In [ ]:
```
from torch.utils.data import DataLoader, Dataset
from transformers import AdamW

class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer

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

    def __getitem__(self, idx):
        encoding = self.tokenizer.encode_plus(
            self.texts[idx],
            add_special_tokens=True,
            max_length=512,
            return_token_type_ids=False,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='pt',
        )
        return {
            'text': self.texts[idx],
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'label': torch.tensor(self.labels[idx], dtype=torch.long)
        }

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_dataset = SentimentDataset(sst_train['text'], sst_train['label'], tokenizer)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)

model = minBERTModel()
optimizer = AdamW(model.parameters(), lr=2e-5)

for epoch in range(3):  # Number of epochs
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids']
        attention_mask = batch['attention_mask']
        labels = batch['label']
        outputs = model(input_ids, attention_mask)
        loss = torch.nn.CrossEntropyLoss()(outputs, labels)
        loss.backward()
        optimizer.step()
```

In [ ]:
```
def evaluate(model, data_loader):
    model.eval()
    correct_predictions = 0
    with torch.no_grad():
        for batch in data_loader:
```

```
            input_ids = batch['input_ids']
            attention_mask = batch['attention_mask']
            labels = batch['label']
            outputs = model(input_ids, attention_mask)
            _, preds = torch.max(outputs, dim=1)
            correct_predictions += torch.sum(preds == labels)
    return correct_predictions.double() / len(data_loader.dataset)

dev_dataset = SentimentDataset(sst_dev['text'], sst_dev['label'], tokenizer)
dev_loader = DataLoader(dev_dataset, batch_size=16)

test_dataset = SentimentDataset(sst_test['text'], sst_test['label'], tokenizer)
test_loader = DataLoader(test_dataset, batch_size=16)

dev_accuracy = evaluate(model, dev_loader)
test_accuracy = evaluate(model, test_loader)
print(f'Dev Accuracy: {dev_accuracy.item()}')
print(f'Test Accuracy: {test_accuracy.item()}')
```