In [ ]:
```
cd minbert-final-project
 source setup.sh
 ```
```

In [ ]:
```
conda activate nlp_fp
 ```
```

In [ ]:
```
def attention(Q, K, V):
     d_k = Q.size(-1)
     scores = torch.matmul(Q, K.transpose(-2, -1)) / math.sqrt(d_k)
     weights = F.softmax(scores, dim=-1)
     output = torch.matmul(weights, V)
     return output
 ```
```

In [ ]:
```
def multi_head_attention(Q, K, V, num_heads):
     d_model = Q.size(-1)
     head_dim = d_model // num_heads
     Q = Q.view(batch_size, -1, num_heads, head_dim).transpose(1, 2)
     K = K.view(batch_size, -1, num_heads, head_dim).transpose(1, 2)
     V = V.view(batch_size, -1, num_heads, head_dim).transpose(1, 2)

     attention_output = attention(Q, K, V)
     attention_output = attention_output.transpose(1, 2).contiguous().view(batch_size, -1, d_model)
     return attention_output
 ```
```

In [ ]:
```
def add_norm(x, sublayer):
     return LayerNorm(x + sublayer(x))
 ```
```

In [ ]:
```
def forward(x):
     x = self.embed(x)
     for layer in self.layers:
         x = layer(x)
     return x
 ```
```

In [ ]:
```
class BertSentimentClassifier(nn.Module):
     def __init__(self, bert_model):
         super(BertSentimentClassifier, self).__init__()
         self.bert = bert_model
         self.dropout = nn.Dropout(0.1)
         self.classifier = nn.Linear(bert_model.config.hidden_size, num_labels)

     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)
         logits = self.classifier(pooled_output)
         return logits
 ```
```

In [ ]:
```
def train_model(model, train_dataloader, val_dataloader, epochs):
     optimizer = Adam(model.parameters(), lr=2e-5)
     for epoch in range(epochs):
         model.train()
         for batch in train_dataloader:
             optimizer.zero_grad()
             input_ids, attention_mask, labels = batch
             outputs = model(input_ids, attention_mask)
             loss = loss_fn(outputs, labels)
             loss.backward()
             optimizer.step()
         evaluate_model(model, val_dataloader)
 ```
```