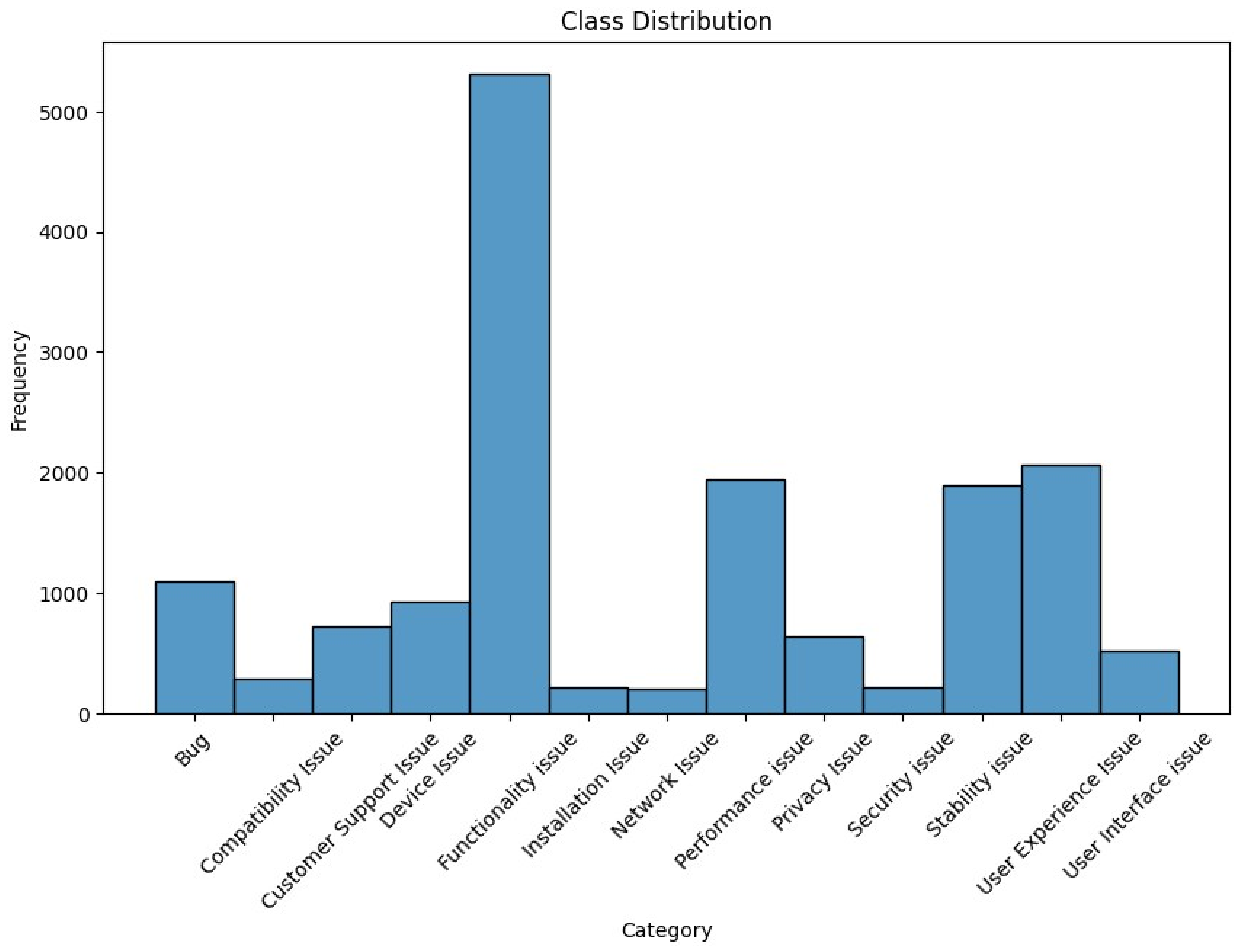
```python
#from google.colab import drive
#drive.mount('/content/drive')
import os

#!pip install transformers
#!pip install torch
#!pip install absl-py
#!pip install ml-dtypes
#!pip install gast
#!pip install astunparse
#!pip install termcolor
#!pip install opt_einsum   flatbuffers
#!pip install flatbuffers

import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import DistilBertTokenizerFast,
DistilBertForSequenceClassification, BertTokenizerFast,
BertForSequenceClassification, RobertaTokenizerFast,
RobertaForSequenceClassification, AdamW
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.utils import resample

# Load your dataset
#df =
pd.read_csv('/root/workspace/aka_project/Naikdil/annotatted_dataset.cs
v')
df = pd.read_csv('ChatGPT_Human_annotated_dataset.csv')
# Compute class distribution
class_counts = df['category'].value_counts()
print(class_counts)
```

```
category
Functionality issue       5310
User Experience Issue     2068
Performance issue         1938
Stability issue           1894
Bug                       1096
Device Issue               935
Customer Support Issue     719
Privacy Issue              641
User Interface issue       513
Compatibility Issue        295
```

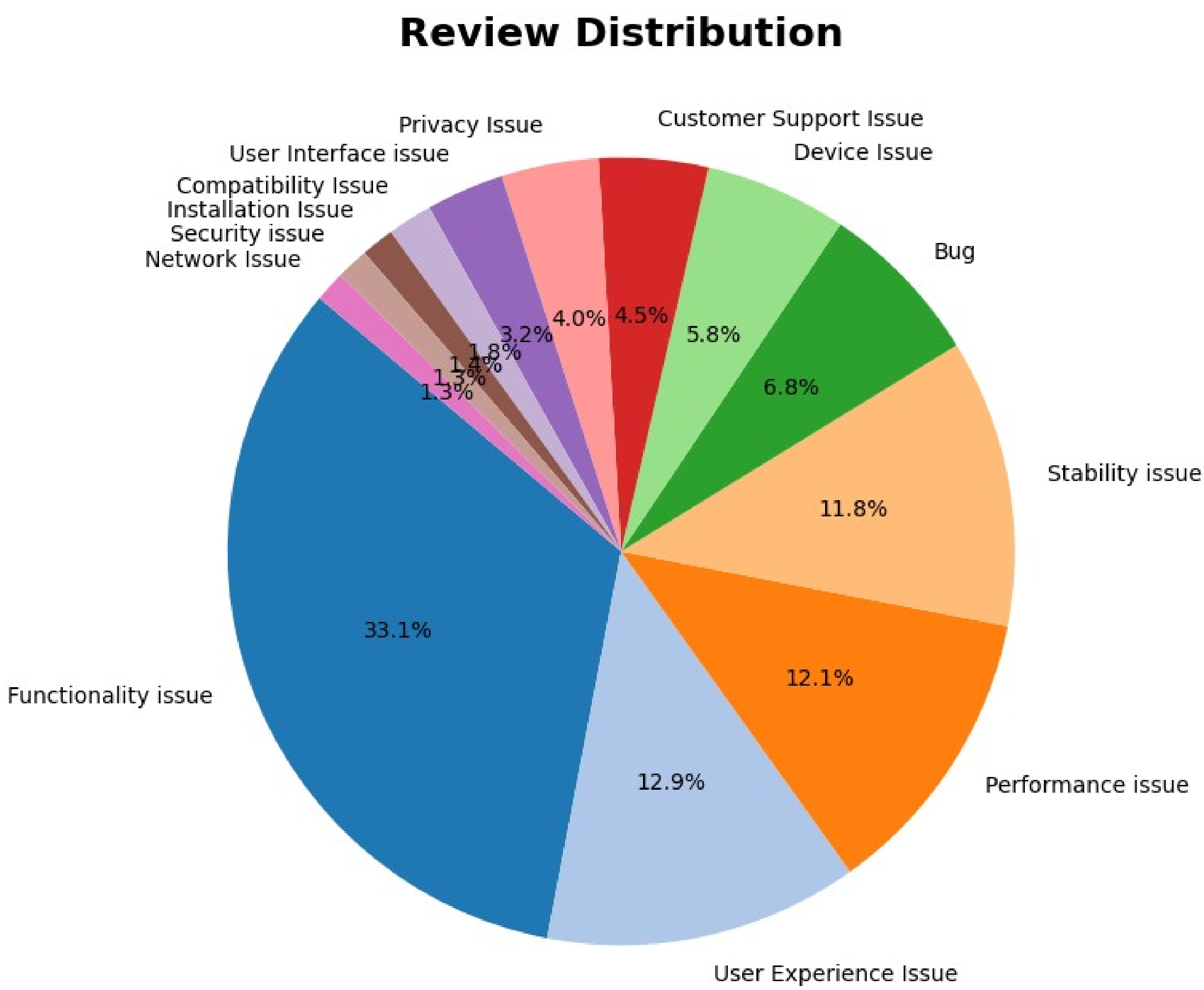
```
Installation Issue          218
Security issue              213
Network Issue               202
Name: count, dtype: int64
```

```python
# Convert categorical class labels to numeric codes
df['category_code'] = df['category'].astype('category').cat.codes

# Create a histogram of class distribution
plt.figure(figsize=(10, 6))
sns.histplot(df, x='category_code', discrete=True, palette='Set2')
plt.title('Class Distribution')
plt.xlabel('Category')
plt.ylabel('Frequency')
plt.xticks(ticks=range(len(df['category'].astype('category').cat.categ
ories)),
           labels=df['category'].astype('category').cat.categories,
           rotation=45)
plt.show()
```

```
/tmp/ipykernel_1097670/3197733234.py:6: UserWarning: Ignoring
`palette` because no `hue` variable has been assigned.
  sns.histplot(df, x='category_code', discrete=True, palette='Set2')
```

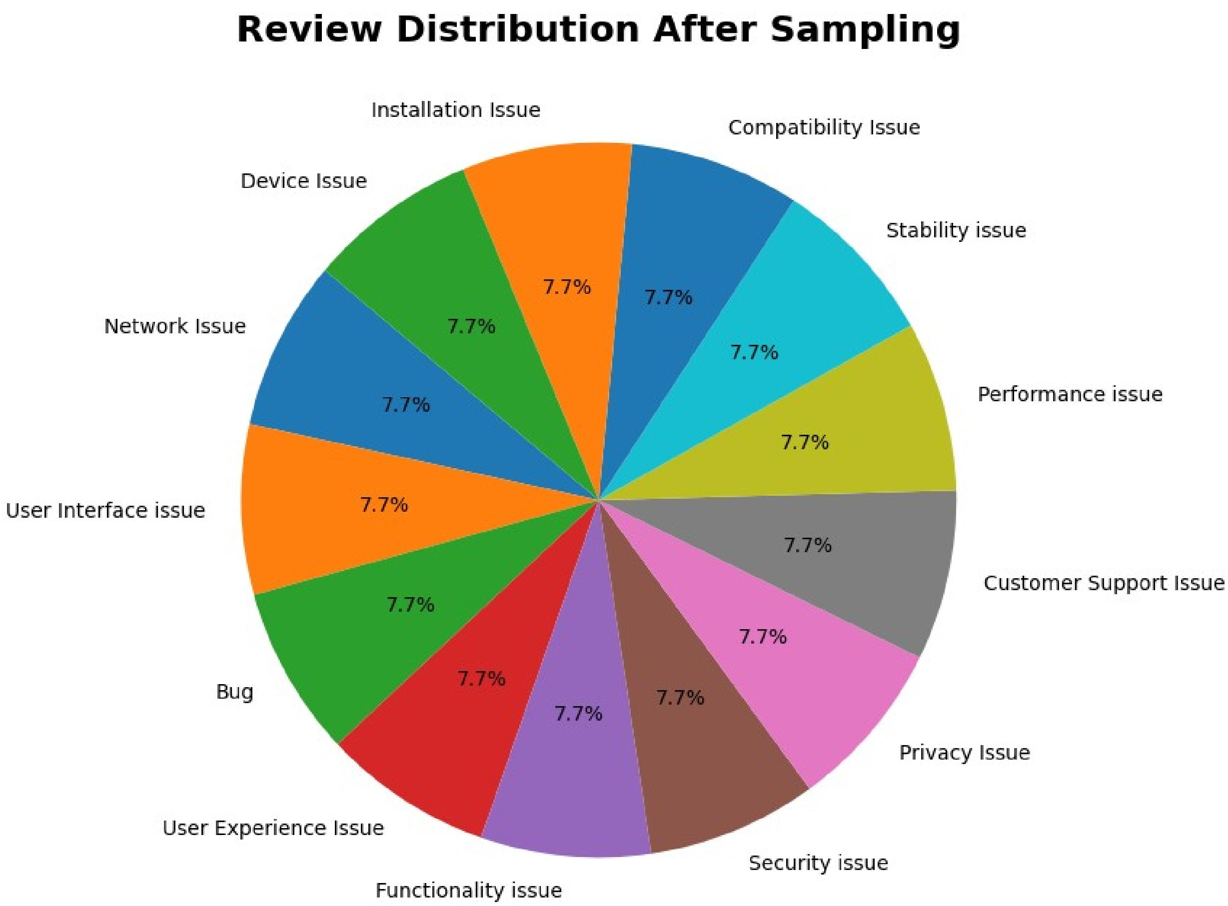



```
# Plot the class distribution as a pie chart
plt.figure(figsize=(8, 8))
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%',
startangle=140, colors=plt.get_cmap('tab20').colors)
plt.title('Review Distribution', fontsize=18, fontweight='bold')
plt.show()
```



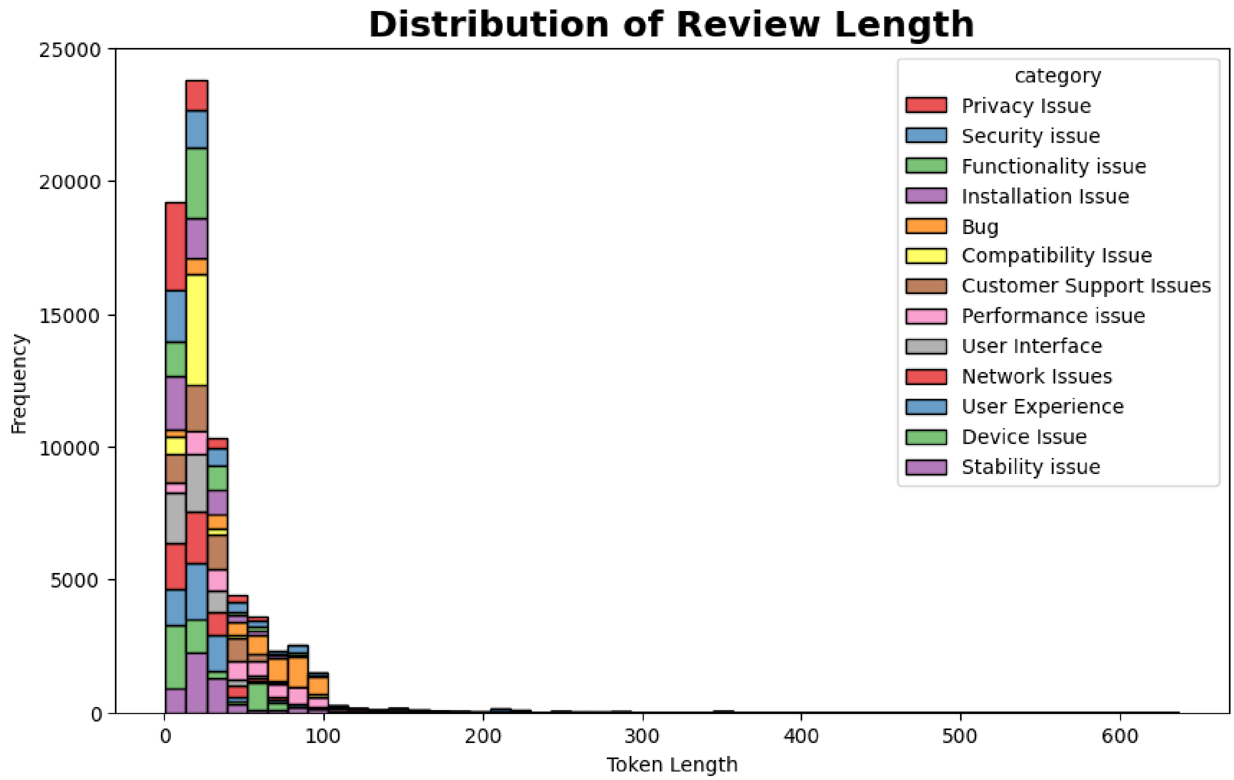

```python
# Balance the dataset
df_majority = df[df['category'] ==
df['category'].value_counts().idxmax()]
df_minority = [df[df['category'] == cls] for cls in
df['category'].unique() if cls !=
df['category'].value_counts().idxmax()]

# Upsample minority classes
df_minority_upsampled = [resample(minority, replace=True,
n_samples=len(df_majority), random_state=42) for minority in
df_minority]

# Combine majority class with upsampled minority classes
df_balanced = pd.concat([df_majority] + df_minority_upsampled)

# Shuffle the dataset
df = df_balanced.sample(frac=1, random_state=42)
```

```
# Display the class distribution
class_counts = df['category'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%',
startangle=140)
plt.title('Review Distribution After Sampling', fontsize=18,
fontweight='bold')
plt.show()
```




```
# Convert categorical labels to numeric
df['category'] = df['category'].astype('category').cat.codes

if df['category'].dtype.name == 'category':
    df['category'] = df['category'].cat.codes

# Mapping from numeric to original labels
category_mapping = {
    0: 'Functionality issue',
    1: 'User Experience',
```

```
    2: 'Performance issue',
    3: 'Stability issue',
    4: 'Bug',
    5: 'Device Issue',
    6: 'Privacy Issue',
    7: 'User Interface',
    8: 'Customer Support Issues',
    9: 'Compatibility Issue',
    10: 'Network Issues',
    11: 'Installation Issue',
    12: 'Security issue',
}

# Apply the mapping
df['category'] = df['category'].map(category_mapping)

# Tokenizing using a simple approach (e.g., splitting the text into
words)
df['token_length'] = df['Base_Reviews'].apply(lambda x:
len(x.split()))

# Plot the distribution of token lengths with actual class names
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='token_length', hue='category',
multiple='stack', palette='Set1', bins=50)
plt.title('Distribution of Review Length', fontsize=18,
fontweight='bold')
plt.xlabel('Token Length')
plt.ylabel('Frequency')
plt.show()
```




```
# Convert categorical class labels to numeric codes
df['category_code'] = df['category'].astype('category').cat.codes
df['category'] = df['category_code']

# Ensure the required package is installed
#%pip install transformers

# Initialize DistilBERT tokenizer
from transformers import DistilBertTokenizerFast
# Ensure the tokenizer is downloaded from the Hugging Face model hub
distilbert_tokenizer =
DistilBertTokenizerFast.from_pretrained('/root/workspace/aka_project/
Naikdil/distilbert', local_files_only=True)

# Tokenize data for DistilBERT
def tokenize_data_distilbert(data):
    return distilbert_tokenizer(data['Base_Reviews'].tolist(),
padding='max_length', truncation=True, max_length=128,
return_tensors='pt', return_attention_mask=True)
# Apply the tokenization function
tokenized_data_distilbert = tokenize_data_distilbert(df)

class TextDatasetDistilBERT(Dataset):
    def __init__(self, inputs, labels, attention_masks):
        self.inputs = inputs
        self.labels = labels
```

```
        self.attention_masks = attention_masks

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

    def __getitem__(self, idx):
        return {
            'input_ids':
self.inputs[idx].clone().detach().to(torch.long),
            'attention_mask':
self.attention_masks[idx].clone().detach().to(torch.long),
            'labels': torch.tensor(self.labels[idx], dtype=torch.long)
# Ensure labels are long
        }


from transformers import get_linear_schedule_with_warmup

model_distilbert =
DistilBertForSequenceClassification.from_pretrained('/root/workspace/
aka_project/Naikdil/distilbert',
num_labels=len(df['category'].unique()))
optimizer_distilbert = AdamW(model_distilbert.parameters(), lr=2e-5)
scheduler = None

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_distilbert.to(device)
# Train model for DistilBert
def train_model_distilbert(train_loader, val_loader, epochs):
    # Initialize the learning rate scheduler
    total_steps = len(train_loader) * epochs
    scheduler = get_linear_schedule_with_warmup(optimizer_distilbert,
num_warmup_steps=0, num_training_steps=total_steps)

    history = {'train_loss': [], 'val_loss': [], 'train_accuracy': [],
'val_accuracy': []}
    for epoch in range(epochs):
        model_distilbert.train()
        total_loss = 0
        correct_predictions = 0
        total_samples = 0

        for batch in train_loader:
            optimizer_distilbert.zero_grad()
            inputs = batch['input_ids'].to(device)
            attention_masks = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model_distilbert(inputs,
attention_mask=attention_masks, labels=labels)
            loss = outputs.loss
```

```
            loss.backward()
            optimizer_distilbert.step()
            scheduler.step()  # Update learning rate

            total_loss += loss.item()
            preds = torch.argmax(outputs.logits, dim=1)
            correct_predictions += torch.sum(preds == labels).item()
            total_samples += labels.size(0)

        avg_train_loss = total_loss / len(train_loader)
        avg_train_accuracy = correct_predictions / total_samples
        history['train_loss'].append(avg_train_loss)
        history['train_accuracy'].append(avg_train_accuracy)

        # Validation
        model_distilbert.eval()
        val_loss = 0
        correct_predictions = 0
        total_samples = 0
        with torch.no_grad():
            for batch in val_loader:
                inputs = batch['input_ids'].to(device)
                attention_masks = batch['attention_mask'].to(device)
                labels = batch['labels'].to(device)
                outputs = model_distilbert(inputs,
attention_mask=attention_masks, labels=labels)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                correct_predictions += torch.sum(preds ==
labels).item()
                total_samples += labels.size(0)

        avg_val_loss = val_loss / len(val_loader)
        avg_val_accuracy = correct_predictions / total_samples
        history['val_loss'].append(avg_val_loss)
        history['val_accuracy'].append(avg_val_accuracy)

        print(f'Epoch {epoch+1}, Train Loss: {avg_train_loss:.4f},
Train Accuracy: {avg_train_accuracy:.4f}, Val Loss:
{avg_val_loss:.4f}, Val Accuracy: {avg_val_accuracy:.4f}')

    return model_distilbert, history
```

```
Some weights of DistilBertForSequenceClassification were not
initialized from the model checkpoint at
/root/workspace/aka_project/Naikdil/distilbert and are newly
initialized: ['classifier.bias', 'classifier.weight',
'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```
/root/miniconda3/lib/python3.11/site-packages/transformers/
optimization.py:591: FutureWarning: This implementation of AdamW is
deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set
`no_deprecation_warning=True` to disable this warning
  warnings.warn(
```

```
kf = KFold(n_splits=5)
accuracies_distilbert = []
train_losses_distilbert = []
val_losses_distilbert = []
train_accuracies_distilbert = []
val_accuracies_distilbert = []

for train_idx, val_idx in
kf.split(tokenized_data_distilbert['input_ids']):
    # Split tokenized data
    train_inputs_distilbert = tokenized_data_distilbert['input_ids']
[train_idx]
    val_inputs_distilbert = tokenized_data_distilbert['input_ids']
[val_idx]
    train_attention_masks_distilbert =
tokenized_data_distilbert['attention_mask'][train_idx]
    val_attention_masks_distilbert =
tokenized_data_distilbert['attention_mask'][val_idx]

    # Labels
    train_labels_distilbert = df['category'].iloc[train_idx].values
    val_labels_distilbert = df['category'].iloc[val_idx].values

    # Create datasets
    train_dataset_distilbert =
TextDatasetDistilBERT(train_inputs_distilbert,
train_labels_distilbert, train_attention_masks_distilbert)
    val_dataset_distilbert =
TextDatasetDistilBERT(val_inputs_distilbert, val_labels_distilbert,
val_attention_masks_distilbert)

    # Create dataloaders
    train_loader_distilbert = DataLoader(train_dataset_distilbert,
batch_size=16, shuffle=True)
    val_loader_distilbert = DataLoader(val_dataset_distilbert,
batch_size=16)

    # Train model
    model_distilbert, history_distilbert =
train_model_distilbert(train_loader_distilbert, val_loader_distilbert,
epochs=6)

    # Collect history
```

```
    train_losses_distilbert.extend(history_distilbert['train_loss'])
    val_losses_distilbert.extend(history_distilbert['val_loss'])

train_accuracies_distilbert.extend(history_distilbert['train_accuracy'
])

val_accuracies_distilbert.extend(history_distilbert['val_accuracy'])
```

```
Epoch 1, Train Loss: 0.5153, Train Accuracy: 0.8427, Val Loss: 0.1188,
Val Accuracy: 0.9630
Epoch 2, Train Loss: 0.0977, Train Accuracy: 0.9684, Val Loss: 0.0773,
Val Accuracy: 0.9723
Epoch 3, Train Loss: 0.0675, Train Accuracy: 0.9750, Val Loss: 0.0675,
Val Accuracy: 0.9768
Epoch 4, Train Loss: 0.0548, Train Accuracy: 0.9781, Val Loss: 0.0565,
Val Accuracy: 0.9790
Epoch 5, Train Loss: 0.0477, Train Accuracy: 0.9793, Val Loss: 0.0585,
Val Accuracy: 0.9788
Epoch 6, Train Loss: 0.0445, Train Accuracy: 0.9801, Val Loss: 0.0557,
Val Accuracy: 0.9793
Epoch 1, Train Loss: 0.0729, Train Accuracy: 0.9722, Val Loss: 0.0485,
Val Accuracy: 0.9801
Epoch 2, Train Loss: 0.0538, Train Accuracy: 0.9780, Val Loss: 0.0483,
Val Accuracy: 0.9802
Epoch 3, Train Loss: 0.0491, Train Accuracy: 0.9788, Val Loss: 0.0470,
Val Accuracy: 0.9797
Epoch 4, Train Loss: 0.0452, Train Accuracy: 0.9798, Val Loss: 0.0461,
Val Accuracy: 0.9802
Epoch 5, Train Loss: 0.0429, Train Accuracy: 0.9805, Val Loss: 0.0444,
Val Accuracy: 0.9808
Epoch 6, Train Loss: 0.0410, Train Accuracy: 0.9813, Val Loss: 0.0449,
Val Accuracy: 0.9807
Epoch 1, Train Loss: 0.0617, Train Accuracy: 0.9749, Val Loss: 0.0450,
Val Accuracy: 0.9812
Epoch 2, Train Loss: 0.0483, Train Accuracy: 0.9790, Val Loss: 0.0443,
Val Accuracy: 0.9815
Epoch 3, Train Loss: 0.0454, Train Accuracy: 0.9797, Val Loss: 0.0425,
Val Accuracy: 0.9820
Epoch 4, Train Loss: 0.0430, Train Accuracy: 0.9801, Val Loss: 0.0436,
Val Accuracy: 0.9815
Epoch 5, Train Loss: 0.0413, Train Accuracy: 0.9802, Val Loss: 0.0420,
Val Accuracy: 0.9826
Epoch 6, Train Loss: 0.0405, Train Accuracy: 0.9808, Val Loss: 0.0424,
Val Accuracy: 0.9826
Epoch 1, Train Loss: 0.0548, Train Accuracy: 0.9779, Val Loss: 0.0441,
Val Accuracy: 0.9792
Epoch 2, Train Loss: 0.0456, Train Accuracy: 0.9802, Val Loss: 0.0472,
Val Accuracy: 0.9785
Epoch 3, Train Loss: 0.0426, Train Accuracy: 0.9807, Val Loss: 0.0456,
```

```
Val Accuracy: 0.9796
Epoch 4, Train Loss: 0.0416, Train Accuracy: 0.9811, Val Loss: 0.0462,
Val Accuracy: 0.9799
Epoch 5, Train Loss: 0.0400, Train Accuracy: 0.9818, Val Loss: 0.0448,
Val Accuracy: 0.9797
Epoch 6, Train Loss: 0.0391, Train Accuracy: 0.9816, Val Loss: 0.0449,
Val Accuracy: 0.9797
Epoch 1, Train Loss: 0.0501, Train Accuracy: 0.9790, Val Loss: 0.0448,
Val Accuracy: 0.9797
Epoch 2, Train Loss: 0.0463, Train Accuracy: 0.9797, Val Loss: 0.0431,
Val Accuracy: 0.9799
Epoch 3, Train Loss: 0.0422, Train Accuracy: 0.9808, Val Loss: 0.0420,
Val Accuracy: 0.9810
Epoch 4, Train Loss: 0.0417, Train Accuracy: 0.9810, Val Loss: 0.0425,
Val Accuracy: 0.9802
Epoch 5, Train Loss: 0.0401, Train Accuracy: 0.9812, Val Loss: 0.0431,
Val Accuracy: 0.9805
Epoch 6, Train Loss: 0.0392, Train Accuracy: 0.9816, Val Loss: 0.0432,
Val Accuracy: 0.9807

# Accuracy calculation
model_distilbert.eval()
preds_distilbert, pred_probs_distilbert, true_labels_distilbert = [],
[], []

with torch.no_grad():
    for batch in val_loader_distilbert:
        inputs = batch['input_ids'].to(device)
        attention_masks = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        outputs = model_distilbert(inputs,
attention_mask=attention_masks)

        # Get class probabilities
        probs = torch.nn.functional.softmax(outputs.logits, dim=1)
        pred_probs_distilbert.extend(probs.cpu().numpy())

        # Get predicted classes
        predictions = torch.argmax(probs, dim=1)
        preds_distilbert.extend(predictions.cpu().numpy())
        true_labels_distilbert.extend(labels.cpu().numpy())

# Calculate accuracy
accuracy_distilbert = sum([pred == label for pred, label in
zip(preds_distilbert, true_labels_distilbert)]) /
len(true_labels_distilbert)
accuracies_distilbert.append(accuracy_distilbert)

print(f"DistilBERT Validation Accuracy: {accuracy_distilbert:.4f}")
```

```
DistilBERT Validation Accuracy: 0.9807
```

```python
import os

# Save the model and tokenizer locally
distilbert_save_directory = './distilbert_model'  # current working directory
if not os.path.exists(distilbert_save_directory):
    os.makedirs(distilbert_save_directory)

model_distilbert.save_pretrained(distilbert_save_directory)
distilbert_tokenizer.save_pretrained(distilbert_save_directory)

print(f"DistilBert Model and Tokenizer saved to {distilbert_save_directory}")
```

```
DistilBert Model and Tokenizer saved to ./distilbert_model
```

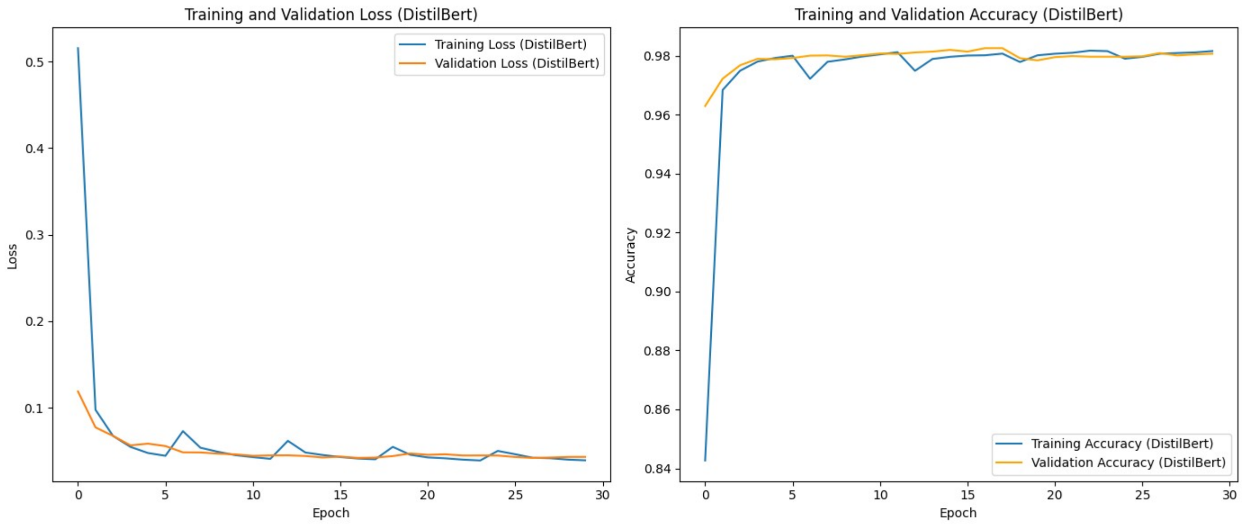
```python
from transformers import DistilBertTokenizerFast, DistilBertForSequenceClassification
import torch

# Load the model and tokenizer from local directory
distilbert_save_directory = './distilbert_model'
model_distilbert = DistilBertForSequenceClassification.from_pretrained(distilbert_save_directory)
distilbert_tokenizer = DistilBertTokenizerFast.from_pretrained(distilbert_save_directory)

# Move the model to the appropriate device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_distilbert.to(device)

# Define the class labels
# class_labels = ['Functionality and Features', 'Performance and Stability',
#                  'User Interface and UX', 'Compatibility and Device Issues',
#                  'Bug', 'Customer Support and Responsiveness', 'Security and Privacy Concerns','Network Issues','Installation Problem']
class_labels = ['Functionality issue', 'User Experience', 'Performance issue', 'Stability issue',
                'Bug', 'Device Issue', 'Privacy Issue', 'User Interface', 'Customer Support Issues',
                'Compatibility Issue', 'Network Issues', 'Installation Issue', 'Security issue']
def classify_review_distilbert(review_text, tokenizer, model, device, class_labels):
    inputs = tokenizer(review_text, padding='max_length',
```

```
truncation=True, max_length=128, return_tensors='pt')
    input_ids = inputs['input_ids'].to(device)
    attention_mask = inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        predicted_class_idx = torch.argmax(logits, dim=1).item()

    predicted_label = class_labels[predicted_class_idx]

    return predicted_label

# Example review
review_text = "I had a terrible time getting it installed on my
tablet. I had to download it from three different sites. Then I had to
install another program on my laptop only to be told my printer wasn't
supported."

# Classify the review
predicted_label_distilbert = classify_review_distilbert(review_text,
distilbert_tokenizer, model_distilbert, device, class_labels)
print(f"Predicted Label for DistilBert: {predicted_label_distilbert}")
```

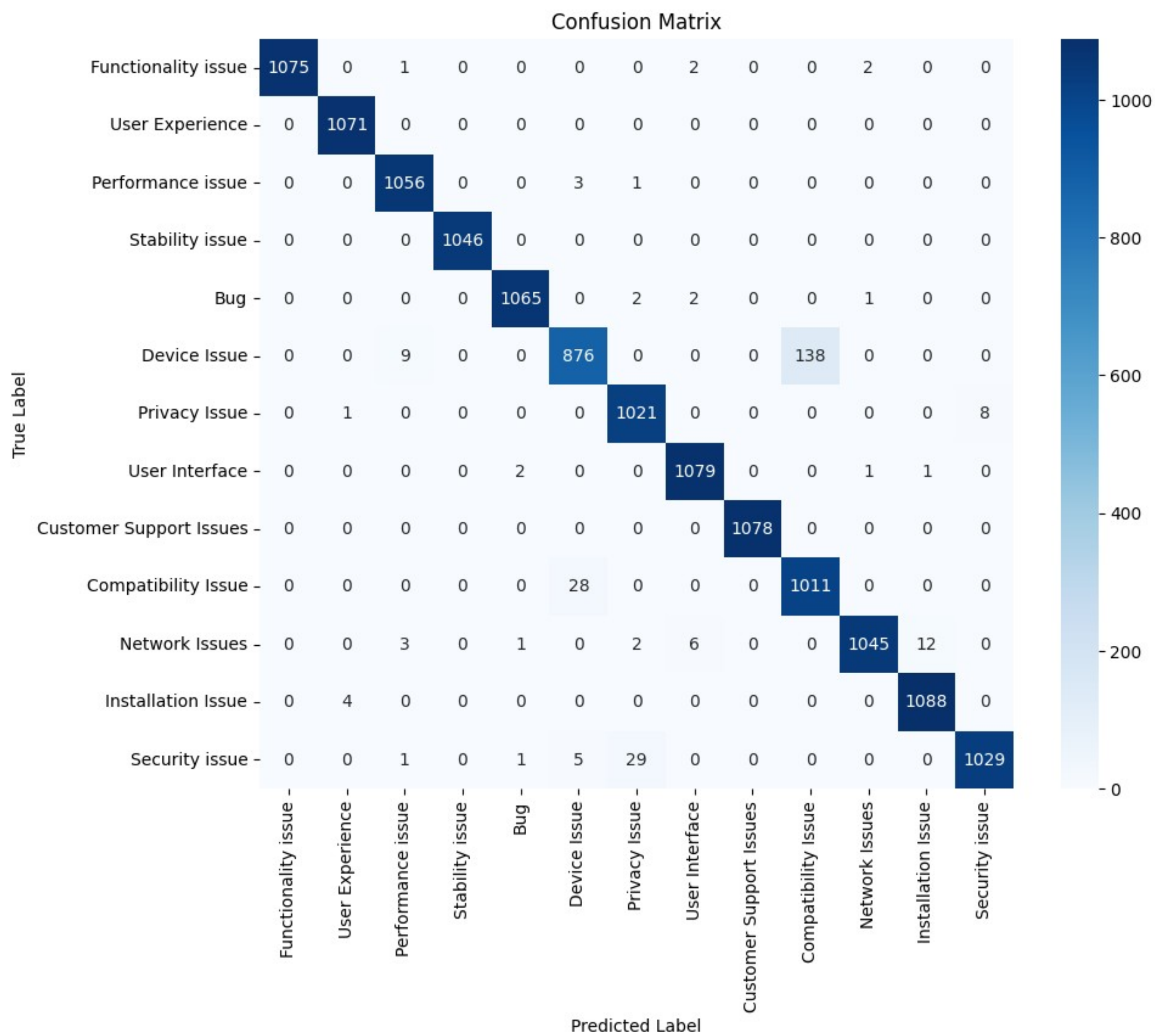
```
Predicted Label for DistilBert: User Interface
```

```
# Plot training and validation loss/accuracy
plt.figure(figsize=(14, 6))

# Training and Validation Loss
plt.subplot(1, 2, 1)
plt.plot(train_losses_distilbert, label='Training Loss (DistilBert)')
plt.plot(val_losses_distilbert, label='Validation Loss (DistilBert)')
plt.title('Training and Validation Loss (DistilBert)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Training and Validation Accuracy
plt.subplot(1, 2, 2)
plt.plot(train_accuracies_distilbert, label='Training Accuracy
(DistilBert)')
plt.plot(val_accuracies_distilbert, label='Validation Accuracy
(DistilBert)', color='orange')
plt.title('Training and Validation Accuracy (DistilBert)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.tight_layout()
plt.show()
```

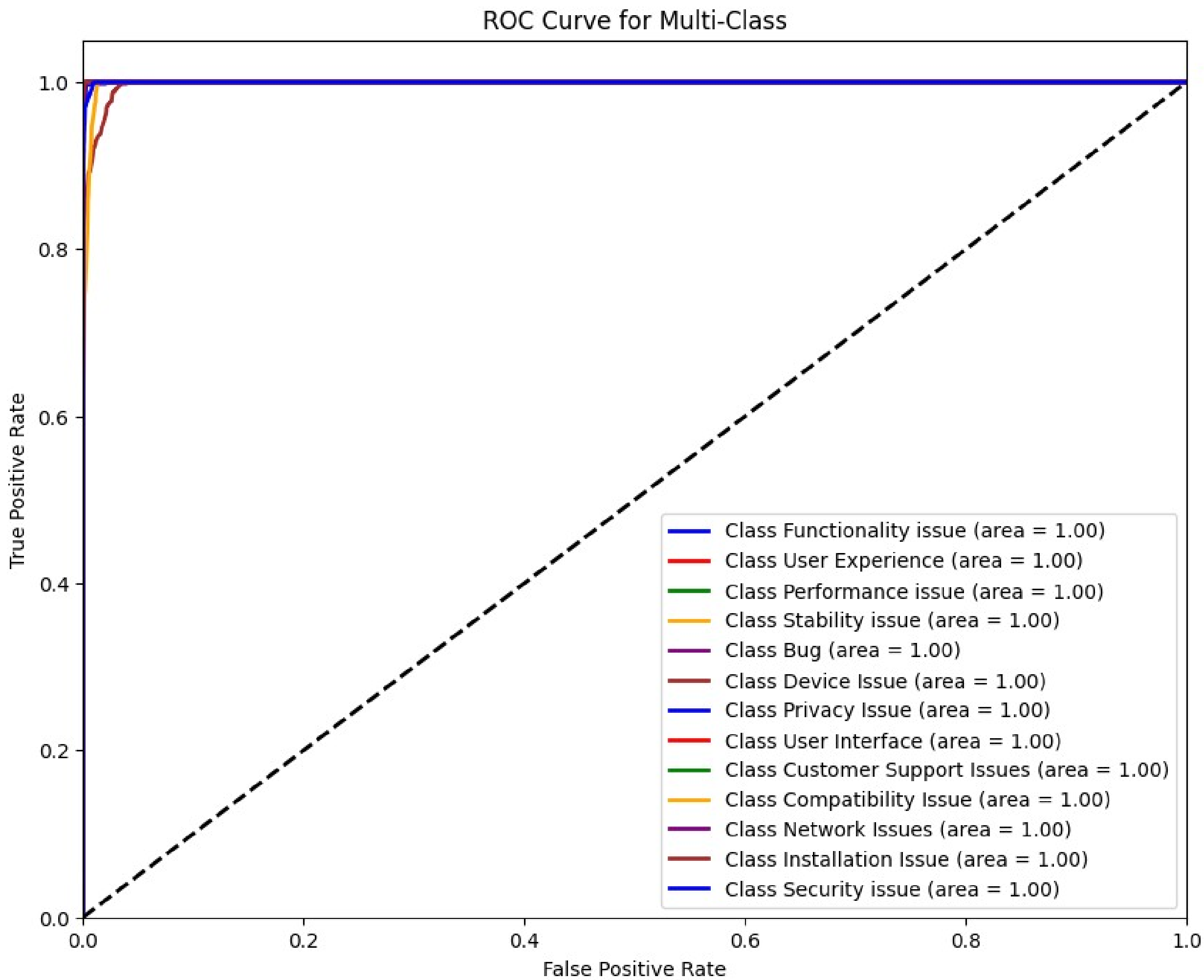



```
from sklearn.metrics import confusion_matrix
import seaborn as sns

def plot_confusion_matrix(true_labels, predictions, class_labels):
    cm = confusion_matrix(true_labels, predictions)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_labels, yticklabels=class_labels)
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.title('Confusion Matrix')
    plt.show()

# Example usage after validation
plot_confusion_matrix(true_labels_distilbert, preds_distilbert,
class_labels)
```




```python
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from itertools import cycle
import matplotlib.pyplot as plt
import numpy as np

# Convert list to numpy arrays
pred_probs_distilbert = np.array(pred_probs_distilbert)
true_labels_distilbert = np.array(true_labels_distilbert)

# Binarize labels for ROC curve
n_classes = len(class_labels)
y_true_bin = label_binarize(true_labels_distilbert,
classes=list(range(n_classes)))

# Generate ROC curve
```

```
fpr = {}
tpr = {}
roc_auc = {}

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i],
pred_probs_distilbert[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves for each class
plt.figure(figsize=(10, 8))
colors = cycle(['blue', 'red', 'green', 'orange', 'purple', 'brown'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'Class
{class_labels[i]} (area = {roc_auc[i]:0.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multi-Class')
plt.legend(loc="lower right")
plt.show()
```




```
from sklearn.metrics import classification_report
# Calculate metrics for DistilBERT
# Ensure the target names match the number of classes in the model
output
if len(class_labels) != model_distilbert.num_labels:
      raise ValueError(f"Mismatch between number of classes in model
output ({model_distilbert.num_labels}) and target names
({len(class_labels)}). Please update `class_labels` to match the
number of classes.")

# Update class_labels to match the number of classes in the model
output
class_labels = [str(i) for i in range(model_distilbert.num_labels)]

# Generate classification report
report_distilbert = classification_report(true_labels_distilbert,
preds_distilbert, target_names=class_labels)
print("DistilBERT Classification Report:")
print(report_distilbert)
```

```
# Initialize the BERT tokenizer
bert_tokenizer = BertTokenizerFast.from_pretrained('bert-base-
uncased')
# Tokenize data for BERT
def tokenize_data_bert(data):
    return bert_tokenizer(data['Base_Reviews'].tolist(),
padding='max_length', truncation=True, max_length=128,
return_tensors='pt', return_attention_mask=True)

tokenized_data_bert = tokenize_data_bert(df)

class TextDatasetBERT(Dataset):
    def __init__(self, inputs, labels, attention_masks):
        self.inputs = inputs
        self.labels = labels
        self.attention_masks = attention_masks

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

    def __getitem__(self, idx):
        return {
            'input_ids':
self.inputs[idx].clone().detach().to(torch.long),
            'attention_mask':
self.attention_masks[idx].clone().detach().to(torch.long),
            'labels': torch.tensor(self.labels[idx], dtype=torch.long)
        }
```

```python
# Initialize BERT model
model_bert = BertForSequenceClassification.from_pretrained('bert-base-
uncased', num_labels=len(df['category'].unique()))
optimizer_bert = AdamW(model_bert.parameters(), lr=2e-5)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_bert.to(device)

# Train model for BERT
def train_model_bert(train_loader, val_loader, epochs):
    # Initialize the learning rate scheduler
    total_steps = len(train_loader) * epochs
    scheduler = get_linear_schedule_with_warmup(optimizer_bert,
num_warmup_steps=0, num_training_steps=total_steps)

    history = {'train_loss': [], 'val_loss': [], 'train_accuracy': [],
'val_accuracy': []}
    for epoch in range(epochs):
        model_bert.train()
        total_loss = 0
        correct_predictions = 0
        total_samples = 0

        for batch in train_loader:
            optimizer_bert.zero_grad()
            inputs = batch['input_ids'].to(device)
            attention_masks = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model_bert(inputs,
attention_mask=attention_masks, labels=labels)
            loss = outputs.loss
            loss.backward()
            optimizer_bert.step()
            scheduler.step()  # Update learning rate

            total_loss += loss.item()
            preds = torch.argmax(outputs.logits, dim=1)
            correct_predictions += torch.sum(preds == labels).item()
            total_samples += labels.size(0)

        avg_train_loss = total_loss / len(train_loader)
        avg_train_accuracy = correct_predictions / total_samples
        history['train_loss'].append(avg_train_loss)
        history['train_accuracy'].append(avg_train_accuracy)

        # Validation
        model_bert.eval()
        val_loss = 0
        correct_predictions = 0
        total_samples = 0
```

```
        with torch.no_grad():
            for batch in val_loader:
                inputs = batch['input_ids'].to(device)
                attention_masks = batch['attention_mask'].to(device)
                labels = batch['labels'].to(device)
                outputs = model_bert(inputs,
attention_mask=attention_masks, labels=labels)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                correct_predictions += torch.sum(preds ==
labels).item()
                total_samples += labels.size(0)

        avg_val_loss = val_loss / len(val_loader)
        avg_val_accuracy = correct_predictions / total_samples
        history['val_loss'].append(avg_val_loss)
        history['val_accuracy'].append(avg_val_accuracy)

        print(f'Epoch {epoch+1}, Train Loss: {avg_train_loss:.4f},
Train Accuracy: {avg_train_accuracy:.4f}, Val Loss:
{avg_val_loss:.4f}, Val Accuracy: {avg_val_accuracy:.4f}')

    return model_bert, history
```

```
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at bert-base-uncased and are newly
initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
/root/miniconda3/lib/python3.11/site-packages/transformers/optimizatio
n.py:591: FutureWarning: This implementation of AdamW is deprecated
and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set
`no_deprecation_warning=True` to disable this warning
  warnings.warn(
```

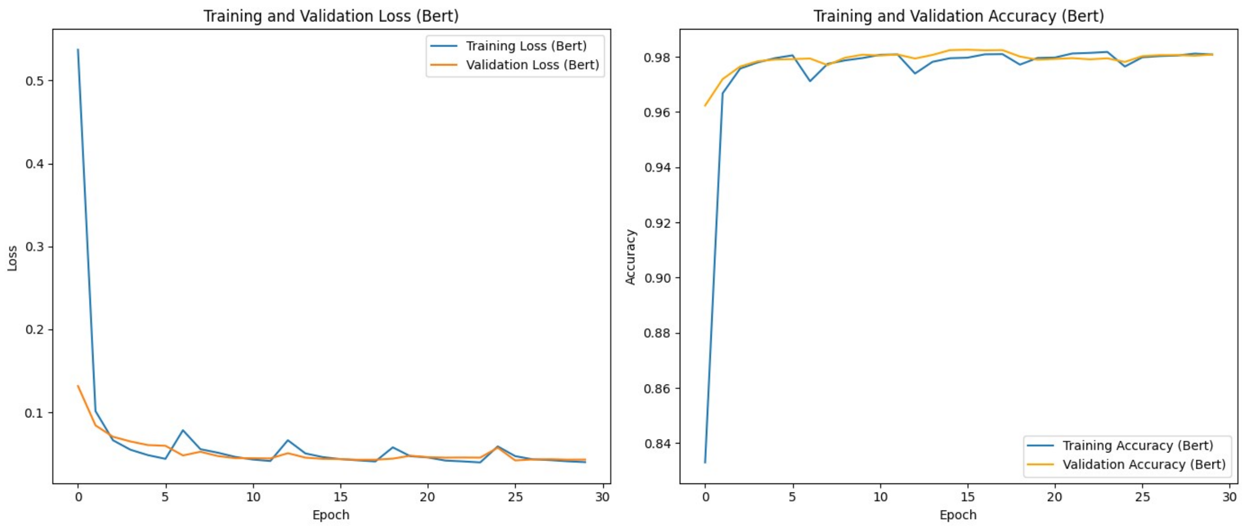
```
# K-Fold Cross Validation for BERT
kf = KFold(n_splits=5)
accuracies_bert = []
train_losses_bert = []
val_losses_bert = []
train_accuracies_bert = []
val_accuracies_bert = []

for train_idx, val_idx in kf.split(tokenized_data_bert['input_ids']):
    # Split tokenized data
    train_inputs_bert = tokenized_data_bert['input_ids'][train_idx]
    val_inputs_bert = tokenized_data_bert['input_ids'][val_idx]
    train_attention_masks_bert = tokenized_data_bert['attention_mask']
[train_idx]
```

```
    val_attention_masks_bert = tokenized_data_bert['attention_mask']
[val_idx]

    # Labels
    train_labels_bert = df['category'].iloc[train_idx].values
    val_labels_bert = df['category'].iloc[val_idx].values

    # Create datasets
    train_dataset_bert = TextDatasetBERT(train_inputs_bert,
train_labels_bert, train_attention_masks_bert)
    val_dataset_bert = TextDatasetBERT(val_inputs_bert,
val_labels_bert, val_attention_masks_bert)

    # Create dataloaders
    train_loader_bert = DataLoader(train_dataset_bert, batch_size=16,
shuffle=True)
    val_loader_bert = DataLoader(val_dataset_bert, batch_size=16)

    # Train model
    model_bert, history_bert = train_model_bert(train_loader_bert,
val_loader_bert, epochs=6)

    # Collect history
    train_losses_bert.extend(history_bert['train_loss'])
    val_losses_bert.extend(history_bert['val_loss'])
    train_accuracies_bert.extend(history_bert['train_accuracy'])
    val_accuracies_bert.extend(history_bert['val_accuracy'])
```

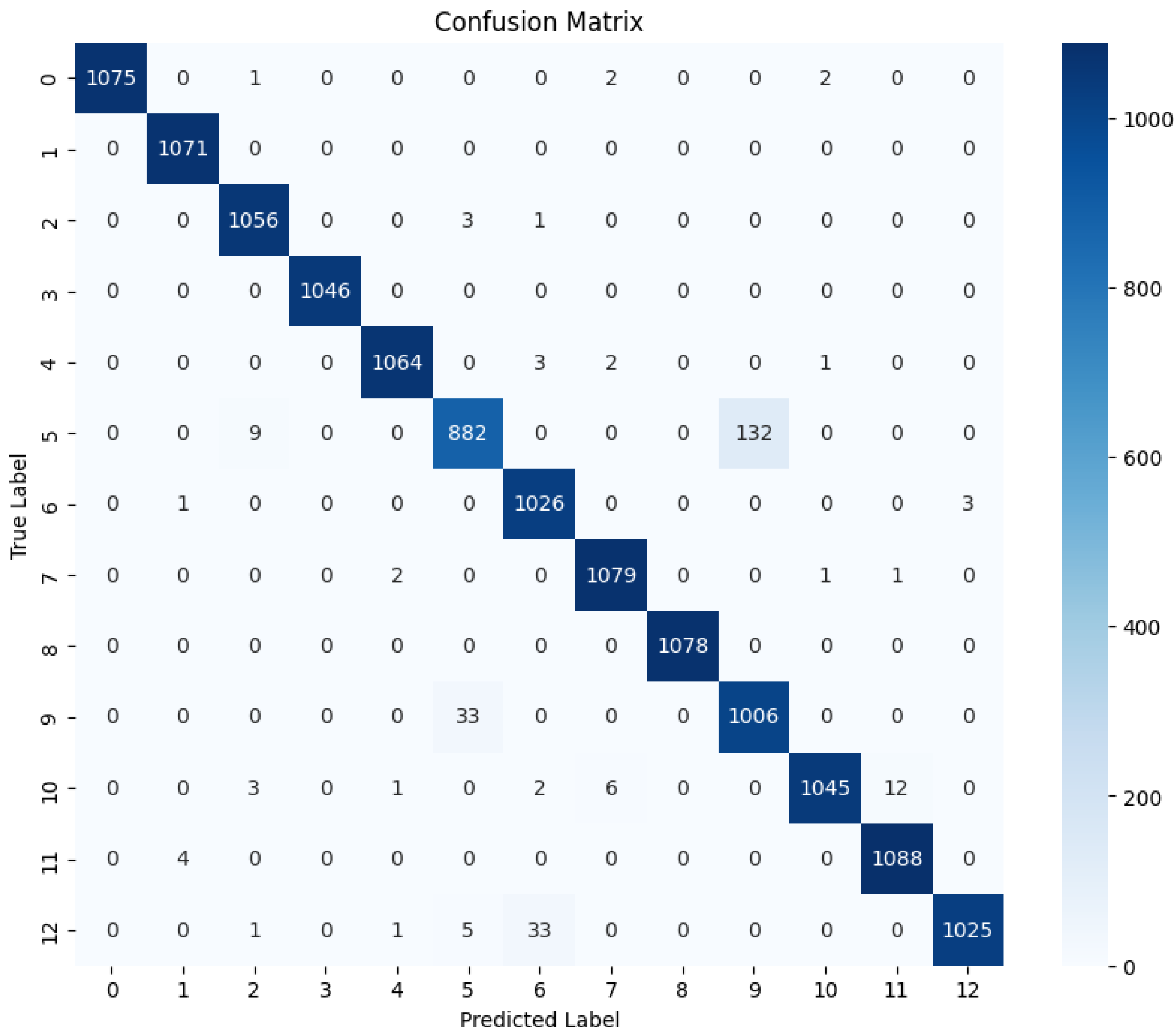
```
Epoch 1, Train Loss: 0.5370, Train Accuracy: 0.8330, Val Loss: 0.1317,
Val Accuracy: 0.9623
Epoch 2, Train Loss: 0.1015, Train Accuracy: 0.9668, Val Loss: 0.0843,
Val Accuracy: 0.9719
Epoch 3, Train Loss: 0.0663, Train Accuracy: 0.9757, Val Loss: 0.0705,
Val Accuracy: 0.9765
Epoch 4, Train Loss: 0.0549, Train Accuracy: 0.9779, Val Loss: 0.0648,
Val Accuracy: 0.9783
Epoch 5, Train Loss: 0.0484, Train Accuracy: 0.9795, Val Loss: 0.0605,
Val Accuracy: 0.9790
Epoch 6, Train Loss: 0.0439, Train Accuracy: 0.9805, Val Loss: 0.0596,
Val Accuracy: 0.9791
Epoch 1, Train Loss: 0.0784, Train Accuracy: 0.9711, Val Loss: 0.0480,
Val Accuracy: 0.9794
Epoch 2, Train Loss: 0.0555, Train Accuracy: 0.9774, Val Loss: 0.0525,
Val Accuracy: 0.9770
Epoch 3, Train Loss: 0.0513, Train Accuracy: 0.9787, Val Loss: 0.0473,
Val Accuracy: 0.9796
Epoch 4, Train Loss: 0.0462, Train Accuracy: 0.9795, Val Loss: 0.0447,
Val Accuracy: 0.9807
Epoch 5, Train Loss: 0.0431, Train Accuracy: 0.9807, Val Loss: 0.0447,
Val Accuracy: 0.9804
```

```
Epoch 6, Train Loss: 0.0413, Train Accuracy: 0.9808, Val Loss: 0.0444,
Val Accuracy: 0.9808
Epoch 1, Train Loss: 0.0663, Train Accuracy: 0.9739, Val Loss: 0.0507,
Val Accuracy: 0.9794
Epoch 2, Train Loss: 0.0505, Train Accuracy: 0.9782, Val Loss: 0.0454,
Val Accuracy: 0.9807
Epoch 3, Train Loss: 0.0460, Train Accuracy: 0.9795, Val Loss: 0.0439,
Val Accuracy: 0.9824
Epoch 4, Train Loss: 0.0436, Train Accuracy: 0.9797, Val Loss: 0.0436,
Val Accuracy: 0.9825
Epoch 5, Train Loss: 0.0421, Train Accuracy: 0.9809, Val Loss: 0.0428,
Val Accuracy: 0.9823
Epoch 6, Train Loss: 0.0408, Train Accuracy: 0.9810, Val Loss: 0.0429,
Val Accuracy: 0.9824
Epoch 1, Train Loss: 0.0577, Train Accuracy: 0.9771, Val Loss: 0.0441,
Val Accuracy: 0.9801
Epoch 2, Train Loss: 0.0471, Train Accuracy: 0.9796, Val Loss: 0.0477,
Val Accuracy: 0.9789
Epoch 3, Train Loss: 0.0456, Train Accuracy: 0.9797, Val Loss: 0.0459,
Val Accuracy: 0.9792
Epoch 4, Train Loss: 0.0418, Train Accuracy: 0.9812, Val Loss: 0.0454,
Val Accuracy: 0.9795
Epoch 5, Train Loss: 0.0408, Train Accuracy: 0.9814, Val Loss: 0.0455,
Val Accuracy: 0.9791
Epoch 6, Train Loss: 0.0396, Train Accuracy: 0.9817, Val Loss: 0.0454,
Val Accuracy: 0.9794
Epoch 1, Train Loss: 0.0589, Train Accuracy: 0.9765, Val Loss: 0.0574,
Val Accuracy: 0.9781
Epoch 2, Train Loss: 0.0472, Train Accuracy: 0.9798, Val Loss: 0.0420,
Val Accuracy: 0.9802
Epoch 3, Train Loss: 0.0434, Train Accuracy: 0.9802, Val Loss: 0.0432,
Val Accuracy: 0.9806
Epoch 4, Train Loss: 0.0424, Train Accuracy: 0.9804, Val Loss: 0.0435,
Val Accuracy: 0.9806
Epoch 5, Train Loss: 0.0409, Train Accuracy: 0.9811, Val Loss: 0.0429,
Val Accuracy: 0.9804
Epoch 6, Train Loss: 0.0400, Train Accuracy: 0.9808, Val Loss: 0.0431,
Val Accuracy: 0.9808

# Accuracy calculation
model_bert.eval()
pred_probs_bert, true_labels_bert = [], []
with torch.no_grad():
    for batch in val_loader_bert:
        inputs = batch['input_ids'].to(device)
        attention_masks = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model_bert(inputs, attention_mask=attention_masks)

        # Collect class probabilities
```

```
        probs = torch.nn.functional.softmax(outputs.logits, dim=1)
        pred_probs_bert.extend(probs.cpu().numpy())
        true_labels_bert.extend(labels.cpu().numpy())

# Convert probabilities to predictions
preds_bert = [np.argmax(prob) for prob in pred_probs_bert]

# Calculate accuracy
accuracy_bert = sum([pred == label for pred, label in zip(preds_bert,
true_labels_bert)]) / len(true_labels_bert)
accuracies_bert.append(accuracy_bert)

# Plot training and validation loss/accuracy
plt.figure(figsize=(14, 6))

# Training and Validation Loss
plt.subplot(1, 2, 1)
plt.plot(train_losses_bert, label='Training Loss (Bert)')
plt.plot(val_losses_bert, label='Validation Loss (Bert)')
plt.title('Training and Validation Loss (Bert)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Training and Validation Accuracy
plt.subplot(1, 2, 2)
plt.plot(train_accuracies_bert, label='Training Accuracy (Bert)')
plt.plot(val_accuracies_bert, label='Validation Accuracy (Bert)',
color='orange')
plt.title('Training and Validation Accuracy (Bert)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

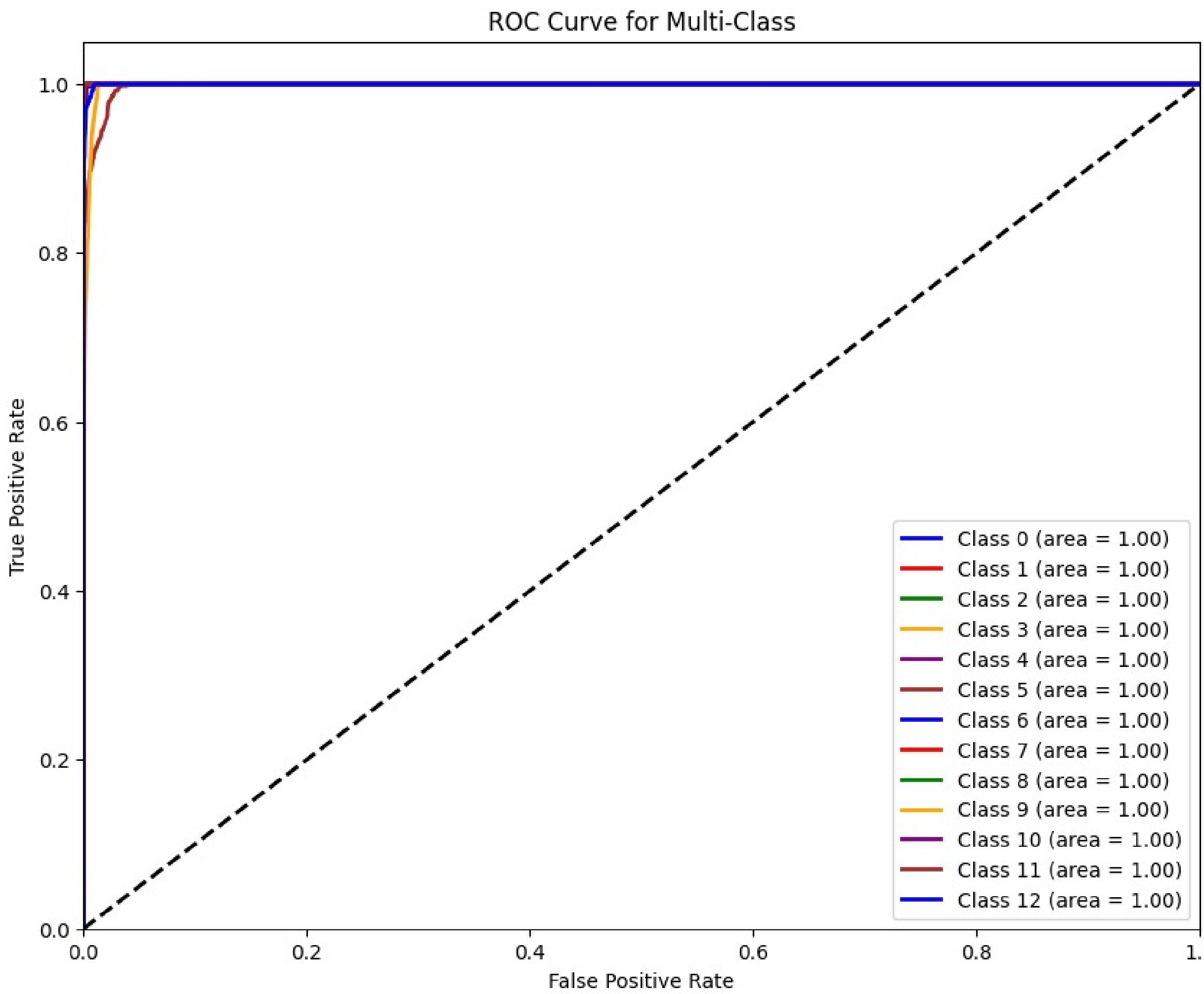



```
from sklearn.metrics import confusion_matrix
import seaborn as sns

def plot_confusion_matrix(true_labels, predictions, class_labels):
    cm = confusion_matrix(true_labels, predictions)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_labels, yticklabels=class_labels)
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.title('Confusion Matrix')
    plt.show()

# Example usage after validation
plot_confusion_matrix(true_labels_bert, preds_bert, class_labels)
```




```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from itertools import cycle
import matplotlib.pyplot as plt
import numpy as np

# Convert list to numpy arrays
pred_probs_bert = np.array(pred_probs_bert)
true_labels_bert = np.array(true_labels_bert)

# Binarize labels for ROC curve
n_classes = len(class_labels)
y_true_bin = label_binarize(true_labels_bert,
classes=list(range(n_classes)))

# Generate ROC curve
fpr = {}
```

```
tpr = {}
roc_auc = {}

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], pred_probs_bert[:,
i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves for each class
plt.figure(figsize=(10, 8))
colors = cycle(['blue', 'red', 'green', 'orange', 'purple', 'brown'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'Class
{class_labels[i]} (area = {roc_auc[i]:0.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multi-Class')
plt.legend(loc="lower right")
plt.show()
```




```
from sklearn.metrics import classification_report
# Calculate metrics for BERT
report_bert = classification_report(true_labels_bert, preds_bert,
target_names=class_labels)
print("BERT Classification Report:")
print(report_bert)
```

```python
# Initialize the RoBERTa tokenizer
roberta_tokenizer = RobertaTokenizerFast.from_pretrained('roberta-
base')
# Tokenize data for RoBERTa
def tokenize_data_roberta(data):
    return roberta_tokenizer(data['Base_Reviews'].tolist(),
padding='max_length', truncation=True, max_length=128,
return_tensors='pt', return_attention_mask=True)

tokenized_data_roberta = tokenize_data_roberta(df)

class TextDatasetRoBERTa(Dataset):
    def __init__(self, inputs, labels, attention_masks):
        self.inputs = inputs
        self.labels = labels
        self.attention_masks = attention_masks

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

    def __getitem__(self, idx):
        return {
            'input_ids':
self.inputs[idx].clone().detach().to(torch.long),
            'attention_mask':
self.attention_masks[idx].clone().detach().to(torch.long),
            'labels': torch.tensor(self.labels[idx], dtype=torch.long)
        }

from transformers import get_linear_schedule_with_warmup
# Initialize RoBERTa model
model_roberta =
RobertaForSequenceClassification.from_pretrained('roberta-base',
num_labels=len(df['category'].unique()))
optimizer_roberta = AdamW(model_roberta.parameters(), lr=2e-5)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_roberta.to(device)

# Train model for RoBERTa
def train_model_roberta(train_loader, val_loader, epochs):
```

```
    # Initialize the learning rate scheduler
    total_steps = len(train_loader) * epochs
    scheduler = get_linear_schedule_with_warmup(optimizer_roberta,
num_warmup_steps=0, num_training_steps=total_steps)

    history = {'train_loss': [], 'val_loss': [], 'train_accuracy': [],
'val_accuracy': []}
    for epoch in range(epochs):
        model_roberta.train()
        total_loss = 0
        correct_predictions = 0
        total_samples = 0

        for batch in train_loader:
            optimizer_roberta.zero_grad()
            inputs = batch['input_ids'].to(device)
            attention_masks = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model_roberta(inputs,
attention_mask=attention_masks, labels=labels)
            loss = outputs.loss
            loss.backward()
            optimizer_roberta.step()
            scheduler.step()  # Update learning rate

            total_loss += loss.item()
            preds = torch.argmax(outputs.logits, dim=1)
            correct_predictions += torch.sum(preds == labels).item()
            total_samples += labels.size(0)

        avg_train_loss = total_loss / len(train_loader)
        avg_train_accuracy = correct_predictions / total_samples
        history['train_loss'].append(avg_train_loss)
        history['train_accuracy'].append(avg_train_accuracy)

        # Validation
        model_roberta.eval()
        val_loss = 0
        correct_predictions = 0
        total_samples = 0
        with torch.no_grad():
            for batch in val_loader:
                inputs = batch['input_ids'].to(device)
                attention_masks = batch['attention_mask'].to(device)
                labels = batch['labels'].to(device)
                outputs = model_roberta(inputs,
attention_mask=attention_masks, labels=labels)
                val_loss += outputs.loss.item()
                preds = torch.argmax(outputs.logits, dim=1)
                correct_predictions += torch.sum(preds ==
```

```
labels).item()
                total_samples += labels.size(0)

        avg_val_loss = val_loss / len(val_loader)
        avg_val_accuracy = correct_predictions / total_samples
        history['val_loss'].append(avg_val_loss)
        history['val_accuracy'].append(avg_val_accuracy)

        print(f'Epoch {epoch+1}, Train Loss: {avg_train_loss:.4f}, 
Train Accuracy: {avg_train_accuracy:.4f}, Val Loss: 
{avg_val_loss:.4f}, Val Accuracy: {avg_val_accuracy:.4f}')

    return model_roberta, history
```

```
Some weights of RobertaForSequenceClassification were not initialized
from the model checkpoint at roberta-base and are newly initialized:
['classifier.dense.bias', 'classifier.dense.weight',
'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
/root/miniconda3/lib/python3.11/site-packages/transformers/optimizatio
n.py:591: FutureWarning: This implementation of AdamW is deprecated
and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set
`no_deprecation_warning=True` to disable this warning
  warnings.warn(
```

```
# K-Fold Cross Validation for RoBERTa
kf = KFold(n_splits=5)
accuracies_roberta = []
train_losses_roberta = []
val_losses_roberta = []
train_accuracies_roberta = []
val_accuracies_roberta = []

for train_idx, val_idx in 
kf.split(tokenized_data_roberta['input_ids']):
    # Split tokenized data
    train_inputs_roberta = tokenized_data_roberta['input_ids']
[train_idx]
    val_inputs_roberta = tokenized_data_roberta['input_ids'][val_idx]
    train_attention_masks_roberta = 
tokenized_data_roberta['attention_mask'][train_idx]
    val_attention_masks_roberta = 
tokenized_data_roberta['attention_mask'][val_idx]

    # Labels
    train_labels_roberta = df['category'].iloc[train_idx].values
    val_labels_roberta = df['category'].iloc[val_idx].values
```

```
    # Create datasets
    train_dataset_roberta = TextDatasetRoBERTa(train_inputs_roberta,
train_labels_roberta, train_attention_masks_roberta)
    val_dataset_roberta = TextDatasetRoBERTa(val_inputs_roberta,
val_labels_roberta, val_attention_masks_roberta)

    # Create dataloaders
    train_loader_roberta = DataLoader(train_dataset_roberta,
batch_size=16, shuffle=True)
    val_loader_roberta = DataLoader(val_dataset_roberta,
batch_size=16)

    # Train model
    model_roberta, history_roberta =
train_model_roberta(train_loader_roberta, val_loader_roberta,
epochs=6)

    # Collect history
    train_losses_roberta.extend(history_roberta['train_loss'])
    val_losses_roberta.extend(history_roberta['val_loss'])
    train_accuracies_roberta.extend(history_roberta['train_accuracy'])
    val_accuracies_roberta.extend(history_roberta['val_accuracy'])
```

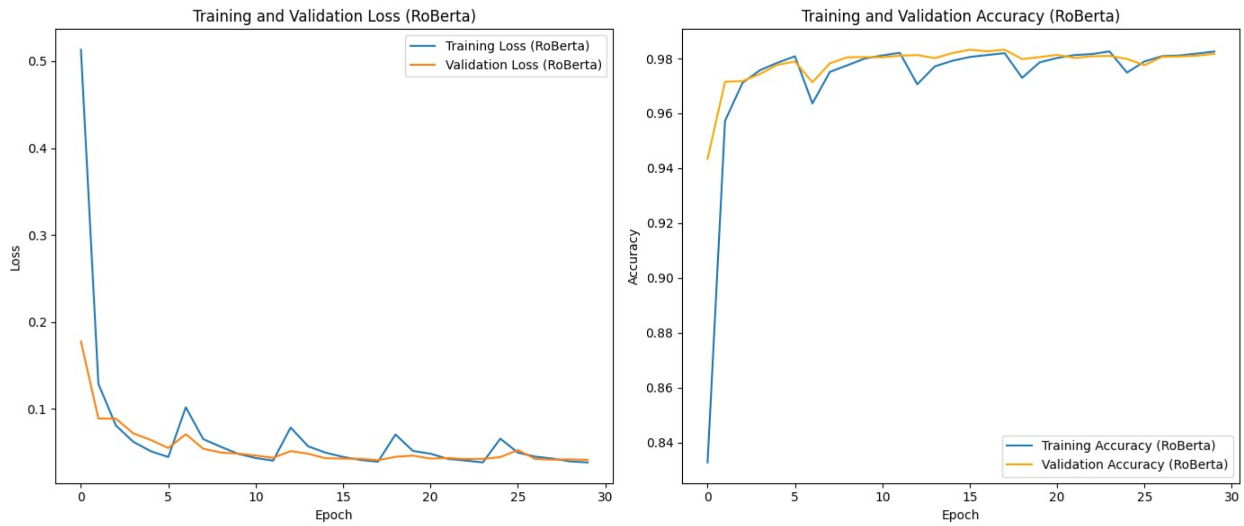
```
Epoch 1, Train Loss: 0.5131, Train Accuracy: 0.8328, Val Loss: 0.1777,
Val Accuracy: 0.9434
Epoch 2, Train Loss: 0.1288, Train Accuracy: 0.9573, Val Loss: 0.0889,
Val Accuracy: 0.9715
Epoch 3, Train Loss: 0.0809, Train Accuracy: 0.9711, Val Loss: 0.0887,
Val Accuracy: 0.9718
Epoch 4, Train Loss: 0.0621, Train Accuracy: 0.9757, Val Loss: 0.0718,
Val Accuracy: 0.9744
Epoch 5, Train Loss: 0.0513, Train Accuracy: 0.9785, Val Loss: 0.0642,
Val Accuracy: 0.9777
Epoch 6, Train Loss: 0.0446, Train Accuracy: 0.9808, Val Loss: 0.0550,
Val Accuracy: 0.9789
Epoch 1, Train Loss: 0.1018, Train Accuracy: 0.9636, Val Loss: 0.0709,
Val Accuracy: 0.9713
Epoch 2, Train Loss: 0.0652, Train Accuracy: 0.9751, Val Loss: 0.0542,
Val Accuracy: 0.9783
Epoch 3, Train Loss: 0.0564, Train Accuracy: 0.9775, Val Loss: 0.0498,
Val Accuracy: 0.9804
Epoch 4, Train Loss: 0.0483, Train Accuracy: 0.9800, Val Loss: 0.0486,
Val Accuracy: 0.9805
Epoch 5, Train Loss: 0.0435, Train Accuracy: 0.9811, Val Loss: 0.0463,
Val Accuracy: 0.9804
Epoch 6, Train Loss: 0.0405, Train Accuracy: 0.9821, Val Loss: 0.0438,
Val Accuracy: 0.9810
Epoch 1, Train Loss: 0.0785, Train Accuracy: 0.9706, Val Loss: 0.0514,
Val Accuracy: 0.9812
Epoch 2, Train Loss: 0.0569, Train Accuracy: 0.9771, Val Loss: 0.0484,
```

```
Val Accuracy: 0.9802
Epoch 3, Train Loss: 0.0496, Train Accuracy: 0.9792, Val Loss: 0.0432,
Val Accuracy: 0.9820
Epoch 4, Train Loss: 0.0448, Train Accuracy: 0.9806, Val Loss: 0.0428,
Val Accuracy: 0.9832
Epoch 5, Train Loss: 0.0412, Train Accuracy: 0.9813, Val Loss: 0.0426,
Val Accuracy: 0.9826
Epoch 6, Train Loss: 0.0392, Train Accuracy: 0.9819, Val Loss: 0.0410,
Val Accuracy: 0.9832
Epoch 1, Train Loss: 0.0706, Train Accuracy: 0.9730, Val Loss: 0.0449,
Val Accuracy: 0.9798
Epoch 2, Train Loss: 0.0516, Train Accuracy: 0.9786, Val Loss: 0.0462,
Val Accuracy: 0.9805
Epoch 3, Train Loss: 0.0484, Train Accuracy: 0.9802, Val Loss: 0.0429,
Val Accuracy: 0.9813
Epoch 4, Train Loss: 0.0425, Train Accuracy: 0.9812, Val Loss: 0.0434,
Val Accuracy: 0.9802
Epoch 5, Train Loss: 0.0406, Train Accuracy: 0.9816, Val Loss: 0.0423,
Val Accuracy: 0.9808
Epoch 6, Train Loss: 0.0384, Train Accuracy: 0.9826, Val Loss: 0.0425,
Val Accuracy: 0.9810
Epoch 1, Train Loss: 0.0657, Train Accuracy: 0.9749, Val Loss: 0.0445,
Val Accuracy: 0.9799
Epoch 2, Train Loss: 0.0497, Train Accuracy: 0.9789, Val Loss: 0.0526,
Val Accuracy: 0.9775
Epoch 3, Train Loss: 0.0450, Train Accuracy: 0.9808, Val Loss: 0.0424,
Val Accuracy: 0.9806
Epoch 4, Train Loss: 0.0428, Train Accuracy: 0.9811, Val Loss: 0.0416,
Val Accuracy: 0.9807
Epoch 5, Train Loss: 0.0396, Train Accuracy: 0.9818, Val Loss: 0.0419,
Val Accuracy: 0.9810
Epoch 6, Train Loss: 0.0384, Train Accuracy: 0.9825, Val Loss: 0.0414,
Val Accuracy: 0.9817

# Initialize lists to store predicted probabilities and true labels
pred_probs_roberta, true_labels_roberta = [], []

# Switch to evaluation mode
model_roberta.eval()

# Collect probabilities and true labels from the validation set
with torch.no_grad():
    for batch in val_loader_roberta:
        inputs = batch['input_ids'].to(device)
        attention_masks = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        # Forward pass
        outputs = model_roberta(inputs,
attention_mask=attention_masks)
```

```
        # Collect class probabilities
        probs = torch.nn.functional.softmax(outputs.logits, dim=1)
        pred_probs_roberta.extend(probs.cpu().numpy())

        # Store true labels
        true_labels_roberta.extend(labels.cpu().numpy())

# Convert the probabilities to predictions (class with the highest
probability)
preds_roberta = np.argmax(pred_probs_roberta, axis=1)

# Accuracy calculation
accuracy_roberta = sum([pred == label for pred, label in
zip(preds_roberta, true_labels_roberta)]) / len(true_labels_roberta)
accuracies_roberta.append(accuracy_roberta)

print(f"Validation Accuracy for Roberta: {accuracy_roberta:.4f}")
```

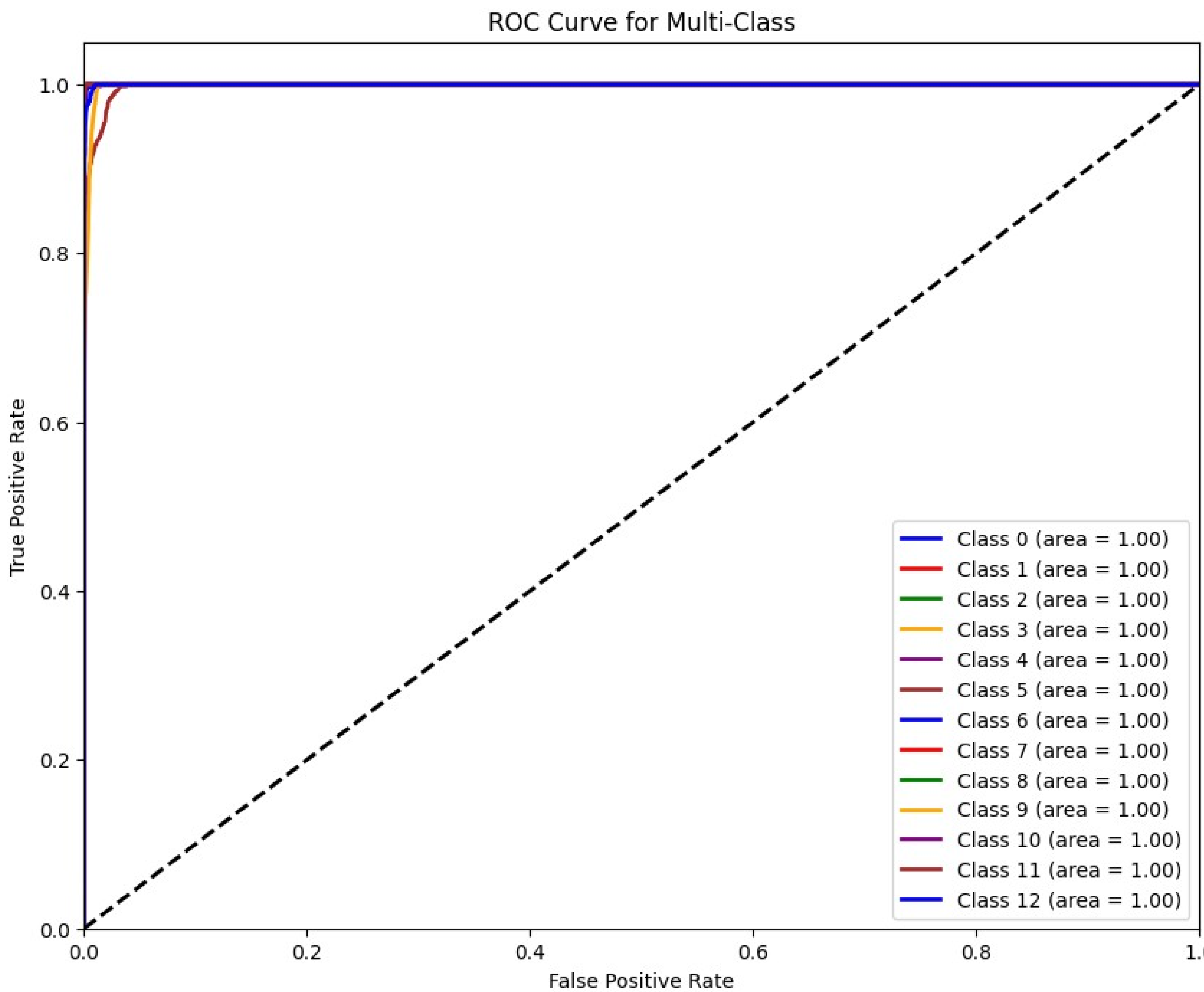
```
Validation Accuracy for Roberta: 0.9817
```

```
# Plot training and validation loss/accuracy
plt.figure(figsize=(14, 6))

# Training and Validation Loss
plt.subplot(1, 2, 1)
plt.plot(train_losses_roberta, label='Training Loss (RoBerta)')
plt.plot(val_losses_roberta, label='Validation Loss (RoBerta)')
plt.title('Training and Validation Loss (RoBerta)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Training and Validation Accuracy
plt.subplot(1, 2, 2)
plt.plot(train_accuracies_roberta, label='Training Accuracy
(RoBerta)')
plt.plot(val_accuracies_roberta, label='Validation Accuracy
(RoBerta)', color='orange')
plt.title('Training and Validation Accuracy (RoBerta)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



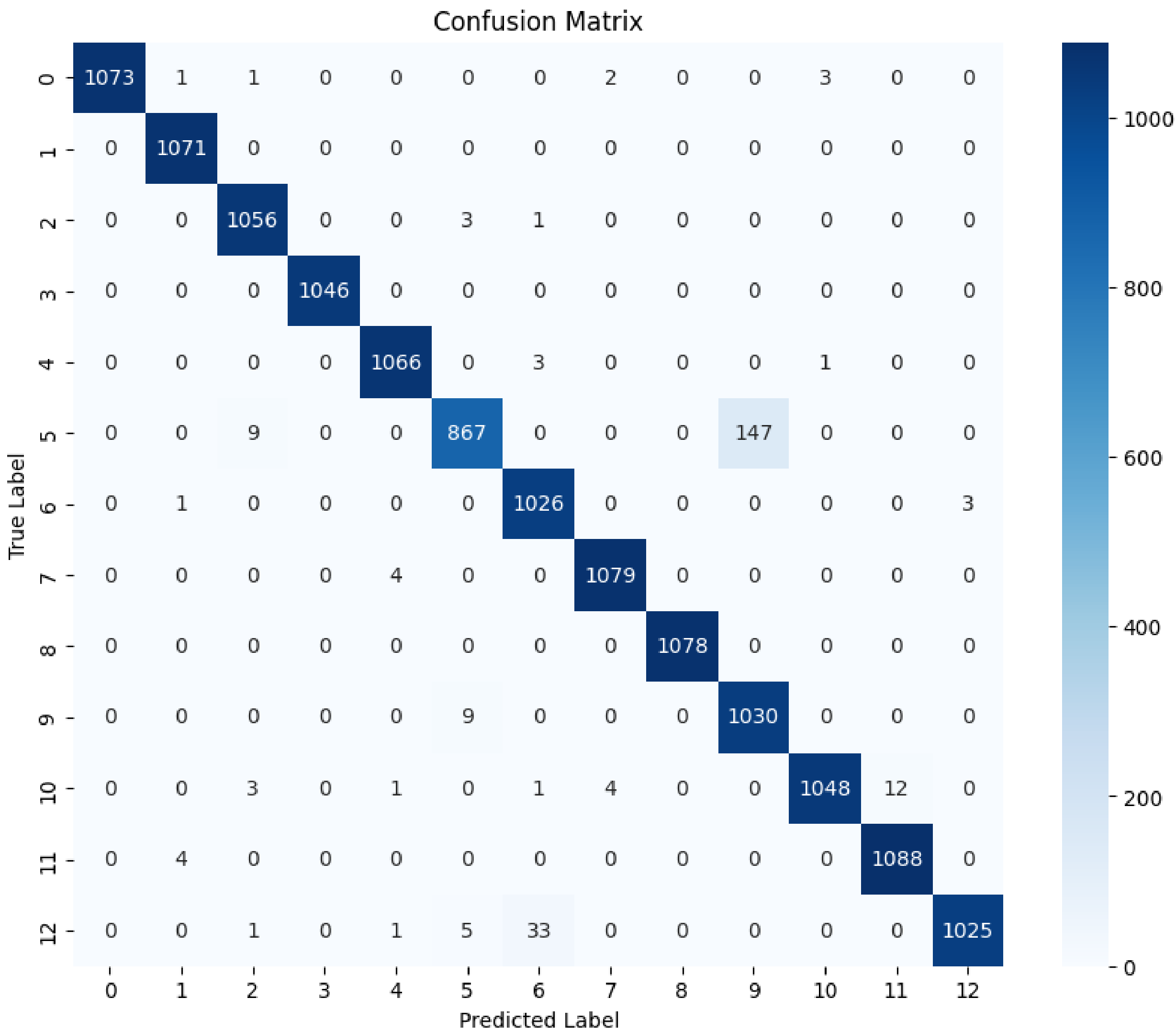

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from itertools import cycle
import matplotlib.pyplot as plt
import numpy as np

# Convert list to numpy arrays
pred_probs_roberta = np.array(pred_probs_roberta)
true_labels_roberta = np.array(true_labels_roberta)

# Binarize labels for ROC curve
n_classes = len(class_labels)
y_true_bin = label_binarize(true_labels_roberta,
classes=list(range(n_classes)))

# Generate ROC curve
fpr = {}
tpr = {}
roc_auc = {}

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i],
pred_probs_roberta[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves for each class
plt.figure(figsize=(10, 8))
colors = cycle(['blue', 'red', 'green', 'orange', 'purple', 'brown'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'Class
{class_labels[i]} (area = {roc_auc[i]:0.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
```

```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multi-Class')
plt.legend(loc="lower right")
plt.show()
```




```
from sklearn.metrics import confusion_matrix
import seaborn as sns

def plot_confusion_matrix(true_labels, predictions, class_labels):
    cm = confusion_matrix(true_labels, predictions)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_labels, yticklabels=class_labels)
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
```

```
    plt.title('Confusion Matrix')
    plt.show()

# Example usage after validation
plot_confusion_matrix(true_labels_roberta, preds_roberta,
class_labels)
```




```
from sklearn.metrics import classification_report
# Calculate metrics for ROBERTA
report_roberta = classification_report(true_labels_roberta,
preds_roberta, target_names=class_labels)
print("ROBERTA Classification Report:")
print(report_roberta)
```

```python
# Print accuracies
print(f"BERT Accuracy: {accuracy_bert:.4f}")
print(f"DistilBERT Accuracy: {accuracy_distilbert:.4f}")
print(f"RoBERTa Accuracy: {accuracy_roberta:.4f}")
```

```
BERT Accuracy: 0.9808
DistilBERT Accuracy: 0.9807
RoBERTa Accuracy: 0.9817
```

```python
# Plot accuracies
import seaborn as sns
import matplotlib.pyplot as plt

# Set the style for the plot
sns.set(style="whitegrid")


# Data for plotting
model_names = ['BERT', 'DistilBERT', 'RoBERTa']
accuracies = [accuracy_bert, accuracy_distilbert, accuracy_roberta]

# Create a DataFrame
data = pd.DataFrame({'Model': model_names, 'Accuracy': accuracies})

# Plot using seaborn
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Accuracy', data=data, palette='Set2')

# Customize the plot
plt.title('Model Accuracy Comparison')
```

```python
plt.ylim(0, 1)  # Accuracy range
plt.show()
```

```
/tmp/ipykernel_1097670/2852100279.py:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='Model', y='Accuracy', data=data, palette='Set2')
```