```
print("Hello World!")
print("This is a test script.")
print("It is running successfully.")
print("Goodbye!")
print("End of script.")
```

```
Hello World!
This is a test script.
It is running successfully.
Goodbye!
End of script.
```

```
## Load all libraries
import pandas as pd
import torch
from torch import nn
from torch.utils.data import TensorDataset, DataLoader, Dataset
from transformers import BertTokenizer, BertModel, AdamW,
get_linear_schedule_with_warmup
from tqdm import tqdm
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_curve, auc
import re
from bs4 import BeautifulSoup
# Install contractions if not already installed
%pip install contractions

import contractions
import csv
from sklearn.utils.class_weight import compute_class_weight
from imblearn.over_sampling import SMOTE
from matplotlib import pyplot as plt
import seaborn as sns
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
from sklearn.metrics import precision_score, recall_score, f1_score

## Load the dataset
/root/workspace/aka_project/Naikdil/ChatGPT_Human_annotated_dataset_fo
r_binary.csv
file_path =
"/root/workspace/aka_project/Naikdil/ChatGPT_Human_annotated_dataset_f
or_binary.csv"  # Update path as needed
df = pd.read_csv(file_path)

# Check and rename columns
print("Original columns:", df.columns)
df = df.rename(columns={'Base_Reviews': 'Review', 'Have_issue':
```

```python
'Issue'})

# Convert issues to binary (1 for any issue, 0 for no issue)
def convert_issue(issue):
    issue = str(issue).lower().strip()
    if issue == 'no' or issue == '0':
        return 0
    return 1  # Treat everything else as an issue

df['Issue'] = df['Issue'].apply(convert_issue)

# Check class distribution
class_counts = df['Issue'].value_counts()
print("\nClass Distribution:")
print(class_counts)

# Plot distribution of issues
plt.figure(figsize=(8, 4))
sns.countplot(data=df, x='Issue')
plt.title('Distribution of Issues (1=Issue, 0=No Issue)')
plt.xticks([0, 1], ['No Issue', 'Issue'])
plt.show()

### Functions for Preprocessing and Model Definition
def preprocess_review(text):
    # Ensure the text is a string before processing
    if not isinstance(text, str):
        text = str(text)  # Convert to string if it's not
    text = contractions.fix(text)
    text = re.sub(r'[^\w\s]', '', text)
    text = BeautifulSoup(text, "html.parser").get_text()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    text = ' '.join(text.split())
    return text

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

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

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = int(self.labels[idx])
        encoding = self.tokenizer(text, return_tensors='pt',
max_length=self.max_length, padding='max_length', truncation=True)
```

```python
        return {'input_ids': encoding['input_ids'].flatten(),
'attention_mask': encoding['attention_mask'].flatten(), 'label':
torch.tensor(label)}

class BERTClassifier(nn.Module):
    def __init__(self, bert_model_name, num_classes):
        super(BERTClassifier, self).__init__()
        self.bert = BertModel.from_pretrained(bert_model_name)
        self.dropout = nn.Dropout(0.1)
        self.fc = nn.Linear(self.bert.config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids,
attention_mask=attention_mask)
        pooled_output = outputs.pooler_output
        x = self.dropout(pooled_output)
        logits = self.fc(x)
        return logits

def train(model, data_loader, optimizer, scheduler, device,
class_weights=None):
    model.train()
    criterion = nn.CrossEntropyLoss(weight=torch.tensor(class_weights,
dtype=torch.float).to(device))
    total_loss = 0.0
    total_correct = 0
    total_samples = 0

    for batch in data_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)
        outputs = model(input_ids=input_ids,
attention_mask=attention_mask)
        loss = criterion(outputs, labels)
        total_loss += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

        _, predicted = torch.max(outputs, 1)
        total_correct += (predicted == labels).sum().item()
        total_samples += labels.size(0)

    avg_loss = total_loss / len(data_loader)
    avg_accuracy = total_correct / total_samples
    return avg_loss, avg_accuracy
```

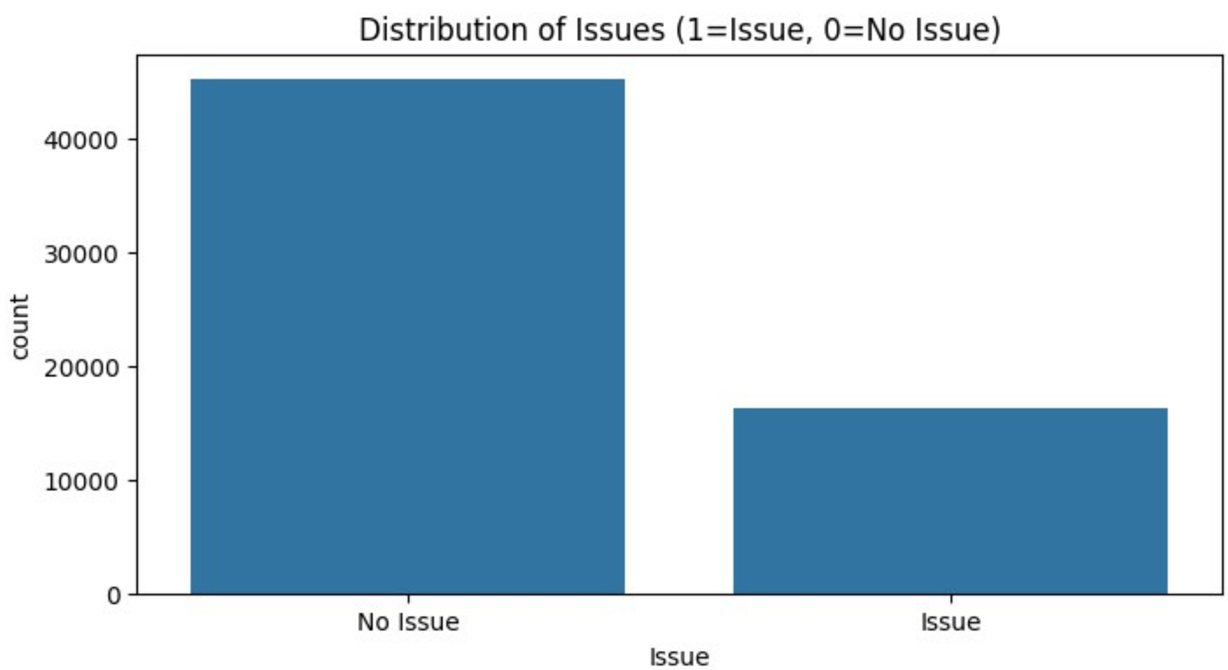
```
def evaluate(model, data_loader, device):
    model.eval()
    predictions = []
    actual_labels = []
    total_loss = 0.0

    with torch.no_grad():
        for batch in data_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)
            outputs = model(input_ids=input_ids,
attention_mask=attention_mask)
            _, preds = torch.max(outputs, dim=1)
            predictions.extend(preds.cpu().tolist())
            actual_labels.extend(labels.cpu().tolist())
            loss = nn.CrossEntropyLoss()(outputs, labels)
            total_loss += loss.item()

    accuracy = accuracy_score(actual_labels, predictions)
    report = classification_report(actual_labels, predictions,
output_dict=True)
    avg_loss = total_loss / len(data_loader)
    return accuracy, report, avg_loss, predictions, actual_labels

def predict_reviewtype(text, model, tokenizer, device,
max_length=128):
    model.eval()
    encoding = tokenizer(text, return_tensors='pt',
max_length=max_length, padding='max_length', truncation=True)
    input_ids = encoding['input_ids'].to(device)
    attention_mask = encoding['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids,
attention_mask=attention_mask)
        _, preds = torch.max(outputs, dim=1)

    return "Issue" if preds.item() == 1 else "No Issue"

# Handle missing values
df['Review'] = df['Review'].fillna('')

# Generate sample from data
sampled_data = df.copy()
class_counts = sampled_data['Issue'].value_counts()
print("The distribution of sampled_data is:", class_counts)

# Apply preprocessing to the 'Review' column
sampled_data['ReviewP'] =
```

```
sampled_data['Review'].apply(preprocess_review)

# Calculate review length
sampled_data['Review_Length'] = sampled_data['ReviewP'].apply(lambda
x: len(x.split()))
median_length = sampled_data['Review_Length'].median()
print('Median Review Length:', median_length)

# Extract texts and labels
texts = sampled_data['ReviewP'].tolist()
labels = sampled_data['Issue'].tolist()

# Compute class weights
class_weights = compute_class_weight('balanced',
classes=np.unique(labels), y=labels)
class_weights_dict = {i: weight for i, weight in
enumerate(class_weights)}

# Set up parameters
bert_model_name = 'bert-base-uncased'
num_classes = 2
max_length = 80
batch_size = 16
num_epochs = 6
learning_rate = 1e-5
tokenizer = BertTokenizer.from_pretrained(bert_model_name)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BERTClassifier(bert_model_name, num_classes).to(device)
optimizer = AdamW(model.parameters(), lr=learning_rate)

# Prepare KFold cross-validation (5 splits)
kf = KFold(n_splits=5, random_state=42, shuffle=True)

train_losses = []
val_accuracies = []
val_reports = []
train_accuracies = []
val_losses = []

all_predictions = []
all_actual_labels = []

for train_index, val_index in kf.split(texts, labels):
    train_texts = np.array(texts)[train_index]
    train_labels = np.array(labels)[train_index]
    val_texts = np.array(texts)[val_index]
    val_labels = np.array(labels)[val_index]

    # Random oversampling
    ros = RandomOverSampler(random_state=42)
```

```
    train_texts_resampled, train_labels_resampled =
ros.fit_resample(train_texts.reshape(-1, 1), train_labels)
    train_texts_resampled = train_texts_resampled.flatten()

    train_dataset = TextClassificationDataset(train_texts_resampled,
train_labels_resampled, tokenizer, max_length)
    val_dataset = TextClassificationDataset(val_texts, val_labels,
tokenizer, max_length)
    train_dataloader = DataLoader(train_dataset,
batch_size=batch_size, shuffle=True)
    val_dataloader = DataLoader(val_dataset, batch_size=batch_size)

    total_steps = len(train_dataloader) * num_epochs
    scheduler = get_linear_schedule_with_warmup(optimizer,
num_warmup_steps=0, num_training_steps=total_steps)

    for epoch in range(num_epochs):
        print(f"Epoch {epoch + 1}/{num_epochs}")
        train_loss, train_accuracy = train(model, train_dataloader,
optimizer, scheduler, device, class_weights)
        print(f"Training Loss: {train_loss:.4f}, Training Accuracy:
{train_accuracy:.4f}")
        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)

        val_accuracy, val_report, val_loss, predictions, actual_labels
= evaluate(model, val_dataloader, device)
        print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy:
{val_accuracy:.4f}")
        print(val_report)
        val_losses.append(val_loss)
        val_accuracies.append(val_accuracy)
        val_reports.append(val_report)

        all_predictions.extend(predictions)
        all_actual_labels.extend(actual_labels)
```

```
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl.metadata (1.2
kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl.metadata (1.5 kB)
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.1.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13 kB)
Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
```

```
Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 472.6 kB/s eta
0:00:0000:0100:01
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (118 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 118.3/118.3 kB 521.7 kB/s eta
0:00:00a 0:00:01
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
Original columns: Index(['Profile_Name', 'Rating_Star', 'Headings',
'Issue_D', 'Base_Reviews',
       'Chagpt annoations', 'category', 'Final annoations', 'Issue
Details',
       'Have_issue'],
      dtype='object')

Class Distribution:
Issue
0    45096
1    16198
Name: count, dtype: int64
```




```
The distribution of sampled_data is: Issue
0    45096
1    16198
```

```
Name: count, dtype: int64
Median Review Length: 16.0

/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(

Epoch 1/6
Training Loss: 0.1092, Training Accuracy: 0.9566
Validation Loss: 0.1111, Validation Accuracy: 0.9725
{'0': {'precision': 0.9904149751917005, 'recall': 0.9721084670724959,
'f1-score': 0.9811763391610344, 'support': 9035.0}, '1': {'precision':
0.9256856384547331, 'recall': 0.9736352357320099, 'f1-score':
0.9490551776266062, 'support': 3224.0}, 'accuracy':
0.9725099926584551, 'macro avg': {'precision': 0.9580503068232168,
'recall': 0.972871851402253, 'f1-score': 0.9651157583938204,
'support': 12259.0}, 'weighted avg': {'precision': 0.973391777407217,
'recall': 0.9725099926584551, 'f1-score': 0.9727287802421181,
'support': 12259.0}}
Epoch 2/6
Training Loss: 0.0354, Training Accuracy: 0.9902
Validation Loss: 0.1132, Validation Accuracy: 0.9791
{'0': {'precision': 0.9902825868423992, 'recall': 0.981294964028777,
'f1-score': 0.9857682899710919, 'support': 9035.0}, '1': {'precision':
0.9488808227465215, 'recall': 0.973014888337469, 'f1-score':
0.9607963246554364, 'support': 3224.0}, 'accuracy':
0.9791173831470756, 'macro avg': {'precision': 0.9695817047944604,
'recall': 0.9771549261831229, 'f1-score': 0.9732823073132642,
'support': 12259.0}, 'weighted avg': {'precision': 0.9793943180239711,
'recall': 0.9791173831470756, 'f1-score': 0.9792009014257232,
'support': 12259.0}}
Epoch 3/6
Training Loss: 0.0156, Training Accuracy: 0.9958
Validation Loss: 0.1231, Validation Accuracy: 0.9812
{'0': {'precision': 0.9909658710684809, 'recall': 0.9833978970669618,
'f1-score': 0.9871673795900228, 'support': 9035.0}, '1': {'precision':
0.9544488308533252, 'recall': 0.9748759305210918, 'f1-score':
0.9645542427497314, 'support': 3224.0}, 'accuracy': 0.981156701199119,
'macro avg': {'precision': 0.972707350960903, 'recall':
0.9791369137940268, 'f1-score': 0.9758608111698771, 'support':
12259.0}, 'weighted avg': {'precision': 0.9813622380108366, 'recall':
0.981156701199119, 'f1-score': 0.981220340420996, 'support': 12259.0}}
Epoch 4/6
Training Loss: 0.0085, Training Accuracy: 0.9979
Validation Loss: 0.1313, Validation Accuracy: 0.9813
{'0': {'precision': 0.9923954372623575, 'recall': 0.982180409518539,
'f1-score': 0.9872615008065861, 'support': 9035.0}, '1': {'precision':
```

```
0.9514621646065722, 'recall': 0.9789081885856079, 'f1-score':
0.9649900626815472, 'support': 3224.0}, 'accuracy':
0.9813198466432825, 'macro avg': {'precision': 0.9719288009344649,
'recall': 0.9805442990520734, 'f1-score': 0.9761257817440667,
'support': 12259.0}, 'weighted avg': {'precision': 0.9816303772213874,
'recall': 0.9813198466432825, 'f1-score': 0.9814043251384954,
'support': 12259.0}}
Epoch 5/6
Training Loss: 0.0043, Training Accuracy: 0.9987
Validation Loss: 0.1427, Validation Accuracy: 0.9819
{'0': {'precision': 0.9911938468398171, 'recall': 0.9841726618705036,
'f1-score': 0.987670776407864, 'support': 9035.0}, '1': {'precision':
0.9565085158150851, 'recall': 0.9754962779156328, 'f1-score':
0.9659090909090909, 'support': 3224.0}, 'accuracy':
0.9818908556978546, 'macro avg': {'precision': 0.9738511813274511,
'recall': 0.9798344698930681, 'f1-score': 0.9767899336584775,
'support': 12259.0}, 'weighted avg': {'precision': 0.9820719358174061,
'recall': 0.9818908556978546, 'f1-score': 0.9819476608153976,
'support': 12259.0}}
Epoch 6/6
Training Loss: 0.0025, Training Accuracy: 0.9993
Validation Loss: 0.1438, Validation Accuracy: 0.9830
{'0': {'precision': 0.990770599355054, 'recall': 0.9861649142224682,
'f1-score': 0.9884623918349235, 'support': 9035.0}, '1': {'precision':
0.9617268830373545, 'recall': 0.9742555831265509, 'f1-score':
0.9679506933744222, 'support': 3224.0}, 'accuracy':
0.9830328738069989, 'macro avg': {'precision': 0.9762487411962042,
'recall': 0.9802102486745096, 'f1-score': 0.9782065426046729,
'support': 12259.0}, 'weighted avg': {'precision': 0.9831323791569739,
'recall': 0.9830328738069989, 'f1-score': 0.9830680109036357,
'support': 12259.0}}
Epoch 1/6
Training Loss: 0.0267, Training Accuracy: 0.9934
Validation Loss: 0.0101, Validation Accuracy: 0.9980
{'0': {'precision': 0.9998880304557161, 'recall': 0.9974310287054619,
'f1-score': 0.998658018340416, 'support': 8953.0}, '1': {'precision':
0.9930889423076923, 'recall': 0.999697519661222, 'f1-score':
0.996382273138378, 'support': 3306.0}, 'accuracy': 0.9980422546700384,
'macro avg': {'precision': 0.9964884863817042, 'recall':
0.9985642741833419, 'f1-score': 0.997520145739397, 'support':
12259.0}, 'weighted avg': {'precision': 0.9980544563128523, 'recall':
0.9980422546700384, 'f1-score': 0.9980442966960783, 'support':
12259.0}}
Epoch 2/6
Training Loss: 0.0099, Training Accuracy: 0.9977
Validation Loss: 0.0109, Validation Accuracy: 0.9978
{'0': {'precision': 1.0, 'recall': 0.9969842510890204, 'f1-score':
0.9984898484255271, 'support': 8953.0}, '1': {'precision':
0.991899189918992, 'recall': 1.0, 'f1-score': 0.9959331224582015,
```

```
'support': 3306.0}, 'accuracy': 0.9977975365037931, 'macro avg':
{'precision': 0.995949594959496, 'recall': 0.9984921255445103, 'f1-
score': 0.9972114854418643, 'support': 12259.0}, 'weighted avg':
{'precision': 0.9978153782422864, 'recall': 0.9977975365037931, 'f1-
score': 0.9978003520515996, 'support': 12259.0}}
Epoch 3/6
Training Loss: 0.0056, Training Accuracy: 0.9986
Validation Loss: 0.0119, Validation Accuracy: 0.9978
{'0': {'precision': 0.9997760358342666, 'recall': 0.9972076398972411,
'f1-score': 0.9984901862103673, 'support': 8953.0}, '1': {'precision':
0.992490237308501, 'recall': 0.999395039322444, 'f1-score':
0.9959306706857574, 'support': 3306.0}, 'accuracy':
0.9977975365037931, 'macro avg': {'precision': 0.9961331365713838,
'recall': 0.9983013396098426, 'f1-score': 0.9972104284480623,
'support': 12259.0}, 'weighted avg': {'precision': 0.9978112059194136,
'recall': 0.9977975365037931, 'f1-score': 0.9977999375502514,
'support': 12259.0}}
Epoch 4/6
Training Loss: 0.0030, Training Accuracy: 0.9993
Validation Loss: 0.0107, Validation Accuracy: 0.9979
{'0': {'precision': 0.9996641665733796, 'recall': 0.9974310287054619,
'f1-score': 0.9985463490998546, 'support': 8953.0}, '1': {'precision':
0.9930847865303668, 'recall': 0.9990925589836661, 'f1-score':
0.9960796139927623, 'support': 3306.0}, 'accuracy':
0.9978791092258749, 'macro avg': {'precision': 0.9963744765518732,
'recall': 0.9982617938445639, 'f1-score': 0.9973129815463084,
'support': 12259.0}, 'weighted avg': {'precision': 0.9978898431846693,
'recall': 0.9978791092258749, 'f1-score': 0.9978811214088482,
'support': 12259.0}}
Epoch 5/6
Training Loss: 0.0020, Training Accuracy: 0.9995
Validation Loss: 0.0114, Validation Accuracy: 0.9979
{'0': {'precision': 0.9996641665733796, 'recall': 0.9974310287054619,
'f1-score': 0.9985463490998546, 'support': 8953.0}, '1': {'precision':
0.9930847865303668, 'recall': 0.9990925589836661, 'f1-score':
0.9960796139927623, 'support': 3306.0}, 'accuracy':
0.9978791092258749, 'macro avg': {'precision': 0.9963744765518732,
'recall': 0.9982617938445639, 'f1-score': 0.9973129815463084,
'support': 12259.0}, 'weighted avg': {'precision': 0.9978898431846693,
'recall': 0.9978791092258749, 'f1-score': 0.9978811214088482,
'support': 12259.0}}
Epoch 6/6
Training Loss: 0.0013, Training Accuracy: 0.9996
Validation Loss: 0.0109, Validation Accuracy: 0.9980
{'0': {'precision': 0.9997760859829825, 'recall': 0.9974310287054619,
'f1-score': 0.9986021805982667, 'support': 8953.0}, '1': {'precision':
0.9930868650435828, 'recall': 0.999395039322444, 'f1-score':
0.9962309663802201, 'support': 3306.0}, 'accuracy':
0.9979606819479566, 'macro avg': {'precision': 0.9964314755132826,
```

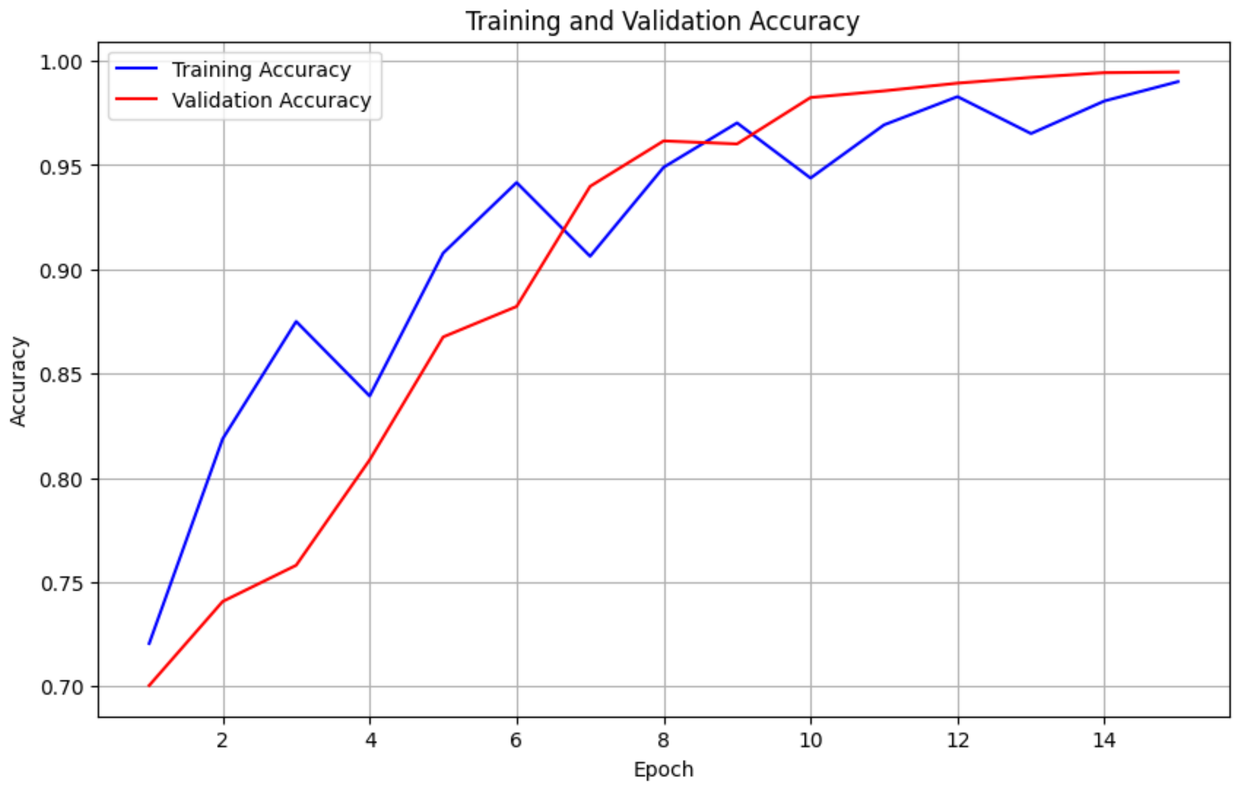
```
'recall': 0.998413034013953, 'f1-score': 0.9974165734892434,
'support': 12259.0}, 'weighted avg': {'precision': 0.9979721407651299,
'recall': 0.9979606819479566, 'f1-score': 0.9979627129251399,
'support': 12259.0}}
Epoch 1/6
Training Loss: 0.0118, Training Accuracy: 0.9972
Validation Loss: 0.0030, Validation Accuracy: 0.9992
{'0': {'precision': 0.9996690203000883, 'recall': 0.9992280546978386,
'f1-score': 0.999448488859475, 'support': 9068.0}, '1': {'precision':
0.9978090766823161, 'recall': 0.9990598558445628, 'f1-score':
0.998434074538052, 'support': 3191.0}, 'accuracy': 0.9991842727791826,
'macro avg': {'precision': 0.9987390484912022, 'recall':
0.9991439552712007, 'f1-score': 0.9989412816987635, 'support':
12259.0}, 'weighted avg': {'precision': 0.999184879661838, 'recall':
0.9991842727791826, 'f1-score': 0.9991844382762577, 'support':
12259.0}}
Epoch 2/6
Training Loss: 0.0056, Training Accuracy: 0.9987
Validation Loss: 0.0039, Validation Accuracy: 0.9990
{'0': {'precision': 0.9997792494481236, 'recall': 0.9988972209969122,
'f1-score': 0.9993380406001765, 'support': 9068.0}, '1': {'precision':
0.9968740231322288, 'recall': 0.9993732372297085, 'f1-score':
0.9981220657276996, 'support': 3191.0}, 'accuracy':
0.9990211273350191, 'macro avg': {'precision': 0.9983266362901762,
'recall': 0.9991352291133104, 'f1-score': 0.998730053163938,
'support': 12259.0}, 'weighted avg': {'precision': 0.9990230232327699,
'recall': 0.9990211273350191, 'f1-score': 0.9990215240965404,
'support': 12259.0}}
Epoch 3/6
Training Loss: 0.0039, Training Accuracy: 0.9990
Validation Loss: 0.0055, Validation Accuracy: 0.9990
{'0': {'precision': 0.9997792494481236, 'recall': 0.9988972209969122,
'f1-score': 0.9993380406001765, 'support': 9068.0}, '1': {'precision':
0.9968740231322288, 'recall': 0.9993732372297085, 'f1-score':
0.9981220657276996, 'support': 3191.0}, 'accuracy':
0.9990211273350191, 'macro avg': {'precision': 0.9983266362901762,
'recall': 0.9991352291133104, 'f1-score': 0.998730053163938,
'support': 12259.0}, 'weighted avg': {'precision': 0.9990230232327699,
'recall': 0.9990211273350191, 'f1-score': 0.9990215240965404,
'support': 12259.0}}
Epoch 4/6
Training Loss: 0.0023, Training Accuracy: 0.9993
Validation Loss: 0.0041, Validation Accuracy: 0.9992
{'0': {'precision': 0.9997792981681748, 'recall': 0.9991177767975298,
'f1-score': 0.9994484280198566, 'support': 9068.0}, '1': {'precision':
0.9974976540506725, 'recall': 0.9993732372297085, 'f1-score':
0.9984345648090169, 'support': 3191.0}, 'accuracy':
0.9991842727791826, 'macro avg': {'precision': 0.9986384761094236,
'recall': 0.9992455070136191, 'f1-score': 0.9989414964144367,
```

```
'support': 12259.0}, 'weighted avg': {'precision': 0.9991853894987116,
'recall': 0.9991842727791826, 'f1-score': 0.9991845208899285,
'support': 12259.0}}
Epoch 5/6
Training Loss: 0.0017, Training Accuracy: 0.9994
Validation Loss: 0.0040, Validation Accuracy: 0.9993
{'0': {'precision': 0.9997793225201368, 'recall': 0.9992280546978386,
'f1-score': 0.9995036125972092, 'support': 9068.0}, '1': {'precision':
0.9978097622027534, 'recall': 0.9993732372297085, 'f1-score':
0.9985908877407234, 'support': 3191.0}, 'accuracy':
0.9992658455012644, 'macro avg': {'precision': 0.998794542361445,
'recall': 0.9993006459637735, 'f1-score': 0.9990472501689662,
'support': 12259.0}, 'weighted avg': {'precision': 0.9992666488132462,
'recall': 0.9992658455012644, 'f1-score': 0.9992660316348919,
'support': 12259.0}}
Epoch 6/6
Training Loss: 0.0011, Training Accuracy: 0.9996
Validation Loss: 0.0044, Validation Accuracy: 0.9993
{'0': {'precision': 0.9997793225201368, 'recall': 0.9992280546978386,
'f1-score': 0.9995036125972092, 'support': 9068.0}, '1': {'precision':
0.9978097622027534, 'recall': 0.9993732372297085, 'f1-score':
0.9985908877407234, 'support': 3191.0}, 'accuracy':
0.9992658455012644, 'macro avg': {'precision': 0.998794542361445,
'recall': 0.9993006459637735, 'f1-score': 0.9990472501689662,
'support': 12259.0}, 'weighted avg': {'precision': 0.9992666488132462,
'recall': 0.9992658455012644, 'f1-score': 0.9992660316348919,
'support': 12259.0}}
Epoch 1/6
Training Loss: 0.0077, Training Accuracy: 0.9983
Validation Loss: 0.0052, Validation Accuracy: 0.9991
{'0': {'precision': 0.9996681048788583, 'recall': 0.9991154356479434,
'f1-score': 0.9993916938561079, 'support': 9044.0}, '1': {'precision':
0.9975155279503105, 'recall': 0.9990668740279938, 'f1-score':
0.9982905982905983, 'support': 3215.0}, 'accuracy':
0.9991027000571009, 'macro avg': {'precision': 0.9985918164145844,
'recall': 0.9990911548379686, 'f1-score': 0.9988411460733531,
'support': 12259.0}, 'weighted avg': {'precision': 0.9991035780148987,
'recall': 0.9991027000571009, 'f1-score': 0.9991029246055073,
'support': 12259.0}}
Epoch 2/6
Training Loss: 0.0050, Training Accuracy: 0.9987
Validation Loss: 0.0072, Validation Accuracy: 0.9985
{'0': {'precision': 0.99966784765279, 'recall': 0.9983414418398938,
'f1-score': 0.9990042044700155, 'support': 9044.0}, '1': {'precision':
0.9953517198636505, 'recall': 0.9990668740279938, 'f1-score':
0.9972058366966781, 'support': 3215.0}, 'accuracy':
0.9985316910025288, 'macro avg': {'precision': 0.9975097837582203,
'recall': 0.9987041579339437, 'f1-score': 0.9981050205833468,
'support': 12259.0}, 'weighted avg': {'precision': 0.9985359159420399,
```

```
'recall': 0.9985316910025288, 'f1-score': 0.9985325711890563,
'support': 12259.0}}
Epoch 3/6
Training Loss: 0.0028, Training Accuracy: 0.9991
Validation Loss: 0.0046, Validation Accuracy: 0.9990
{'0': {'precision': 0.9997786631252766, 'recall': 0.9988942945599293,
'f1-score': 0.9993362831858407, 'support': 9044.0}, '1': {'precision':
0.9968973006515669, 'recall': 0.9993779160186625, 'f1-score':
0.9981360671015843, 'support': 3215.0}, 'accuracy':
0.9990211273350191, 'macro avg': {'precision': 0.9983379818884217,
'recall': 0.999136105289296, 'f1-score': 0.9987361751437125,
'support': 12259.0}, 'weighted avg': {'precision': 0.9990230076596613,
'recall': 0.9990211273350191, 'f1-score': 0.9990215189545913,
'support': 12259.0}}
Epoch 4/6
Training Loss: 0.0020, Training Accuracy: 0.9994
Validation Loss: 0.0046, Validation Accuracy: 0.9993
{'0': {'precision': 0.9997787610619469, 'recall': 0.9993365767359575,
'f1-score': 0.9995576199955762, 'support': 9044.0}, '1': {'precision':
0.9981360671015843, 'recall': 0.9993779160186625, 'f1-score':
0.9987566055331054, 'support': 3215.0}, 'accuracy':
0.9993474182233462, 'macro avg': {'precision': 0.9989574140817656,
'recall': 0.99935724637731, 'f1-score': 0.9991571127643408, 'support':
12259.0}, 'weighted avg': {'precision': 0.9993479542194176, 'recall':
0.9993474182233462, 'f1-score': 0.9993475489052063, 'support':
12259.0}}
Epoch 5/6
Training Loss: 0.0012, Training Accuracy: 0.9995
Validation Loss: 0.0047, Validation Accuracy: 0.9993
{'0': {'precision': 0.9997787610619469, 'recall': 0.9993365767359575,
'f1-score': 0.9995576199955762, 'support': 9044.0}, '1': {'precision':
0.9981360671015843, 'recall': 0.9993779160186625, 'f1-score':
0.9987566055331054, 'support': 3215.0}, 'accuracy':
0.9993474182233462, 'macro avg': {'precision': 0.9989574140817656,
'recall': 0.99935724637731, 'f1-score': 0.9991571127643408, 'support':
12259.0}, 'weighted avg': {'precision': 0.9993479542194176, 'recall':
0.9993474182233462, 'f1-score': 0.9993475489052063, 'support':
12259.0}}
Epoch 6/6
Training Loss: 0.0012, Training Accuracy: 0.9995
Validation Loss: 0.0049, Validation Accuracy: 0.9993
{'0': {'precision': 0.9997787610619469, 'recall': 0.9993365767359575,
'f1-score': 0.9995576199955762, 'support': 9044.0}, '1': {'precision':
0.9981360671015843, 'recall': 0.9993779160186625, 'f1-score':
0.9987566055331054, 'support': 3215.0}, 'accuracy':
0.9993474182233462, 'macro avg': {'precision': 0.9989574140817656,
'recall': 0.99935724637731, 'f1-score': 0.9991571127643408, 'support':
12259.0}, 'weighted avg': {'precision': 0.9993479542194176, 'recall':
0.9993474182233462, 'f1-score': 0.9993475489052063, 'support':
```

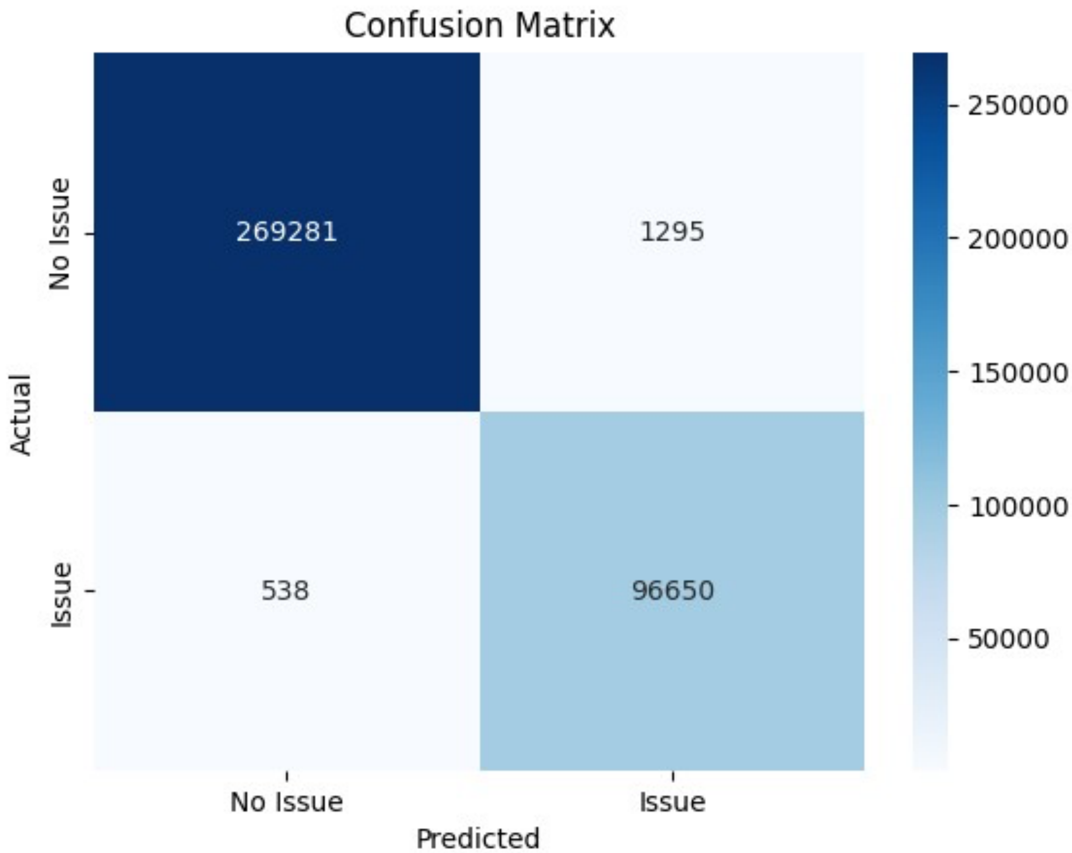
```
12259.0}}
Epoch 1/6
Training Loss: 0.0065, Training Accuracy: 0.9985
Validation Loss: 0.0035, Validation Accuracy: 0.9992
{'0': {'precision': 0.9998887405429462, 'recall': 0.9989995553579368,
'f1-score': 0.999443950177936, 'support': 8996.0}, '1': {'precision':
0.9972477064220183, 'recall': 0.9996934396076027, 'f1-score':
0.9984690753214942, 'support': 3262.0}, 'accuracy':
0.9991842062326644, 'macro avg': {'precision': 0.9985682234824822,
'recall': 0.9993464974827697, 'f1-score': 0.9989565127497151,
'support': 12258.0}, 'weighted avg': {'precision': 0.9991859298640045,
'recall': 0.9991842062326644, 'f1-score': 0.999184524351397,
'support': 12258.0}}
Epoch 2/6
Training Loss: 0.0035, Training Accuracy: 0.9991
Validation Loss: 0.0073, Validation Accuracy: 0.9988
{'0': {'precision': 0.9995550116809434, 'recall': 0.9987772343263672,
'f1-score': 0.9991659716430359, 'support': 8996.0}, '1': {'precision':
0.9966350565922301, 'recall': 0.9987737584304108, 'f1-score':
0.9977032613688562, 'support': 3262.0}, 'accuracy':
0.9987763093489965, 'macro avg': {'precision': 0.9980950341365867,
'recall': 0.998775496378389, 'f1-score': 0.9984346165059461,
'support': 12258.0}, 'weighted avg': {'precision': 0.9987779768058103,
'recall': 0.9987763093489965, 'f1-score': 0.9987767269934703,
'support': 12258.0}}
Epoch 3/6
Training Loss: 0.0021, Training Accuracy: 0.9994
Validation Loss: 0.0068, Validation Accuracy: 0.9991
{'0': {'precision': 0.9996663329996663, 'recall': 0.9991107158737217,
'f1-score': 0.9993884472118753, 'support': 8996.0}, '1': {'precision':
0.997551270278543, 'recall': 0.9990803188228081, 'f1-score':
0.9983152090672385, 'support': 3262.0}, 'accuracy':
0.9991026268559309, 'macro avg': {'precision': 0.9986088016391046,
'recall': 0.999095517348265, 'f1-score': 0.9988518281395569,
'support': 12258.0}, 'weighted avg': {'precision': 0.9991034895834234,
'recall': 0.9991026268559309, 'f1-score': 0.9991028457411781,
'support': 12258.0}}
Epoch 4/6
Training Loss: 0.0020, Training Accuracy: 0.9995
Validation Loss: 0.0057, Validation Accuracy: 0.9990
{'0': {'precision': 0.9996662958843159, 'recall': 0.9989995553579368,
'f1-score': 0.9993328144112087, 'support': 8996.0}, '1': {'precision':
0.9972460220318238, 'recall': 0.9990803188228081, 'f1-score':
0.9981623277182236, 'support': 3262.0}, 'accuracy':
0.9990210474791973, 'macro avg': {'precision': 0.9984561589580698,
'recall': 0.9990399370903724, 'f1-score': 0.9987475710647161,
'support': 12258.0}, 'weighted avg': {'precision': 0.9990222321457918,
'recall': 0.9990210474791973, 'f1-score': 0.9990213339419219,
'support': 12258.0}}
```

```
Epoch 5/6
Training Loss: 0.0015, Training Accuracy: 0.9995
Validation Loss: 0.0059, Validation Accuracy: 0.9992
{'0': {'precision': 0.9996663701067615, 'recall': 0.9992218763895064,
'f1-score': 0.9994440738269957, 'support': 8996.0}, '1': {'precision':
0.9978567054500919, 'recall': 0.9990803188228081, 'f1-score':
0.9984681372549019, 'support': 3262.0}, 'accuracy':
0.9991842062326644, 'macro avg': {'precision': 0.9987615377784267,
'recall': 0.9991510976061573, 'f1-score': 0.9989561055409488,
'support': 12258.0}, 'weighted avg': {'precision': 0.9991847967579236,
'recall': 0.9991842062326644, 'f1-score': 0.9991843654652588,
'support': 12258.0}}
Epoch 6/6
Training Loss: 0.0014, Training Accuracy: 0.9995
Validation Loss: 0.0061, Validation Accuracy: 0.9992
{'0': {'precision': 0.9996663701067615, 'recall': 0.9992218763895064,
'f1-score': 0.9994440738269957, 'support': 8996.0}, '1': {'precision':
0.9978567054500919, 'recall': 0.9990803188228081, 'f1-score':
0.9984681372549019, 'support': 3262.0}, 'accuracy':
0.9991842062326644, 'macro avg': {'precision': 0.9987615377784267,
'recall': 0.9991510976061573, 'f1-score': 0.9989561055409488,
'support': 12258.0}, 'weighted avg': {'precision': 0.9991847967579236,
'recall': 0.9991842062326644, 'f1-score': 0.9991843654652588,
'support': 12258.0}}

# Plot accuracy vs epoch
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(train_accuracies) + 1), train_accuracies,
label='Training Accuracy', color='blue')
plt.plot(range(1, len(val_accuracies) + 1), val_accuracies,
label='Validation Accuracy', color='red')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# Plot loss vs epoch
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(train_losses) + 1), train_losses,
label='Training Loss', color='blue')
plt.plot(range(1, len(val_losses) + 1), val_losses, label='Validation
Loss', color='orange')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid(True)
```

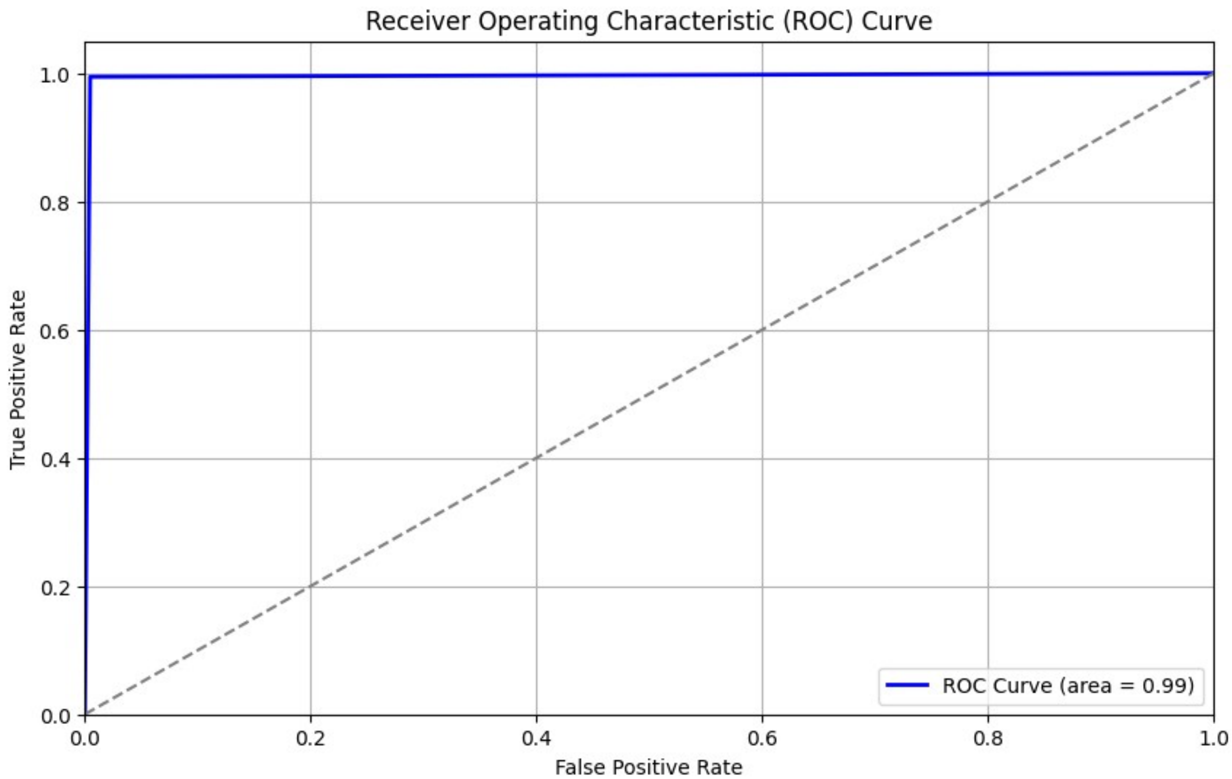

Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
1.00
0.95
0.90
0.85
0.80
0.75
0.70
2
4
6
8
10
12
14


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2
4
6
8
10
12
14

```
# Confusion Matrix
cm = confusion_matrix(all_actual_labels, all_predictions)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["No
Issue", "Issue"], yticklabels=["No Issue", "Issue"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
# ROC Curve
fpr, tpr, _ = roc_curve(all_actual_labels, all_predictions)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (area =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




```python
# Calculate and print average metrics
avg_accuracy = np.mean([report['accuracy'] for report in val_reports])
macro_precision = np.mean([report['macro avg']['precision'] for report
in val_reports])
macro_recall = np.mean([report['macro avg']['recall'] for report in
val_reports])
macro_f1 = np.mean([report['macro avg']['f1-score'] for report in
val_reports])
weighted_precision = np.mean([report['weighted avg']['precision'] for
report in val_reports])
weighted_recall = np.mean([report['weighted avg']['recall'] for report
in val_reports])
weighted_f1 = np.mean([report['weighted avg']['f1-score'] for report
in val_reports])

print(f"Average Metrics:")
print(f"Average Accuracy: {avg_accuracy:.4f}")
print(f"Macro-Precision: {macro_precision:.4f}")
print(f"Macro-Recall: {macro_recall:.4f}")
print(f"Macro-F1 Score: {macro_f1:.4f}")
print(f"Weighted-Precision: {weighted_precision:.4f}")
print(f"Weighted-Recall: {weighted_recall:.4f}")
print(f"Weighted-F1 Score: {weighted_f1:.4f}")
```

```
Average Metrics:
Average Accuracy: 0.9950
```

```
Macro-Precision: 0.9925
Macro-Recall: 0.9948
Macro-F1 Score: 0.9936
Weighted-Precision: 0.9951
Weighted-Recall: 0.9950
Weighted-F1 Score: 0.9950
```