```
print("Hello World")
```

```
Hello World
```

```
## Load all libraries
import pandas as pd
import torch
from torch import nn
from torch.utils.data import TensorDataset, DataLoader, Dataset
from transformers import DistilBertTokenizer, DistilBertModel, AdamW,
get_linear_schedule_with_warmup
from tqdm import tqdm
import numpy as np
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_curve, auc
import re
from bs4 import BeautifulSoup
import contractions
import csv
from sklearn.metrics import precision_score, recall_score, f1_score
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

# Load the dataset
file_path =
"/root/workspace/aka_project/Naikdil/Datset_for_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':
'Issue'})

# Convert 'Issue' to binary (1 for 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()
```

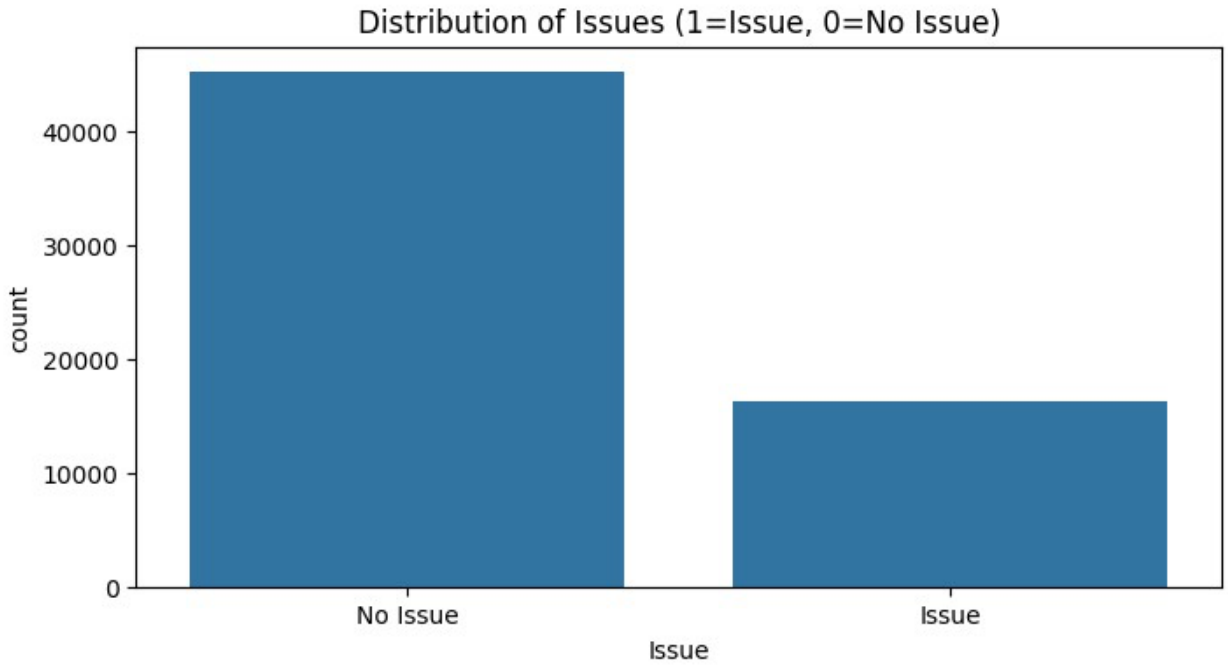
```
print("\nClass Distribution:")
print(class_counts)

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

class DistilBERTClassifier(nn.Module):
    def __init__(self, bert_model_name, num_classes):
        super(DistilBERTClassifier, self).__init__()
        self.bert = DistilBertModel.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.last_hidden_state[:, 0]  # Using the
[CLS] token representation
        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)

# 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
distbert_model_name =
'/root/workspace/aka_project/Naikdil/distilbert_model'
num_classes = 2
max_length = 128
batch_size = 16
num_epochs = 6
learning_rate = 1e-5
tokenizer =
DistilBertTokenizer.from_pretrained("/root/workspace/aka_project/Naikd
il/distilbert_model")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = DistilBERTClassifier(distbert_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)



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

/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: 0.1125, Training Accuracy: 0.9541
Validation Loss: 0.1098, Validation Accuracy: 0.9719
{'0': {'precision': 0.9936378095887298, 'recall': 0.9680132816823465,
'f1-score': 0.9806581824297808, 'support': 9035.0}, '1': {'precision':
0.9164015041943882, 'recall': 0.9826302729528535, 'f1-score':
0.9483610237988325, 'support': 3224.0}, 'accuracy':
0.9718574108818011, 'macro avg': {'precision': 0.955019656891559,
'recall': 0.9753217773176, 'f1-score': 0.9645096031143066, 'support':
12259.0}, 'weighted avg': {'precision': 0.9733253984139719, 'recall':
0.9718574108818011, 'f1-score': 0.9721643379541973, 'support':
12259.0}}
Epoch 2/6
Training Loss: 0.0316, Training Accuracy: 0.9899
Validation Loss: 0.1026, Validation Accuracy: 0.9800
{'0': {'precision': 0.99205105239588, 'recall': 0.9807415605976757,
'f1-score': 0.9863638893527021, 'support': 9035.0}, '1': {'precision':
```

```
0.9477006311992786, 'recall': 0.9779776674937966, 'f1-score':
0.9626011295985346, 'support': 3224.0}, 'accuracy':
0.9800146830899747, 'macro avg': {'precision': 0.9698758417975792,
'recall': 0.9793596140457361, 'f1-score': 0.9744825094756184,
'support': 12259.0}, 'weighted avg': {'precision': 0.9803873149019698,
'recall': 0.9800146830899747, 'f1-score': 0.9801145103293366,
'support': 12259.0}}
Epoch 3/6
Training Loss: 0.0135, Training Accuracy: 0.9960
Validation Loss: 0.1036, Validation Accuracy: 0.9821
{'0': {'precision': 0.9917447568049977, 'recall': 0.9839513004980631,
'f1-score': 0.9878326573698539, 'support': 9035.0}, '1': {'precision':
0.9559939301972686, 'recall': 0.9770471464019851, 'f1-score':
0.966405890473999, 'support': 3224.0}, 'accuracy': 0.9821355738640999,
'macro avg': {'precision': 0.9738693435011332, 'recall':
0.980499223450024, 'f1-score': 0.9771192739219265, 'support':
12259.0}, 'weighted avg': {'precision': 0.9823426306133575, 'recall':
0.9821355738640999, 'f1-score': 0.9821976221734892, 'support':
12259.0}}
Epoch 4/6
Training Loss: 0.0067, Training Accuracy: 0.9982
Validation Loss: 0.1240, Validation Accuracy: 0.9799
{'0': {'precision': 0.9933752526386705, 'recall': 0.9791920309905922,
'f1-score': 0.986232651468703, 'support': 9035.0}, '1': {'precision':
0.9439308082314345, 'recall': 0.9816997518610422, 'f1-score':
0.9624448836855709, 'support': 3224.0}, 'accuracy':
0.9798515376458112, 'macro avg': {'precision': 0.9686530304350525,
'recall': 0.9804458914258172, 'f1-score': 0.9743387675771369,
'support': 12259.0}, 'weighted avg': {'precision': 0.9803718356577642,
'recall': 0.9798515376458112, 'f1-score': 0.9799766955723969,
'support': 12259.0}}
Epoch 5/6
Training Loss: 0.0040, Training Accuracy: 0.9987
Validation Loss: 0.1224, Validation Accuracy: 0.9828
{'0': {'precision': 0.9925206519312346, 'recall': 0.9840619811842833,
'f1-score': 0.9882732173623076, 'support': 9035.0}, '1': {'precision':
0.9563768554983338, 'recall': 0.9792183622828784, 'f1-score':
0.9676628352490422, 'support': 3224.0}, 'accuracy':
0.9827881556407537, 'macro avg': {'precision': 0.9744487537147842,
'recall': 0.9816401717335809, 'f1-score': 0.9779680263056749,
'support': 12259.0}, 'weighted avg': {'precision': 0.9830151784260815,
'recall': 0.9827881556407537, 'f1-score': 0.9828528835721805,
'support': 12259.0}}
Epoch 6/6
Training Loss: 0.0022, Training Accuracy: 0.9992
Validation Loss: 0.1229, Validation Accuracy: 0.9833
{'0': {'precision': 0.9914292074799644, 'recall': 0.9858328721638074,
'f1-score': 0.9886231200399578, 'support': 9035.0}, '1': {'precision':
0.9609160305343512, 'recall': 0.9761166253101737, 'f1-score':
```

```
0.9684566856439453, 'support': 3224.0}, 'accuracy':
0.9832775919732442, 'macro avg': {'precision': 0.9761726190071578,
'recall': 0.9809747487369906, 'f1-score': 0.9785399028419515,
'support': 12259.0}, 'weighted avg': {'precision': 0.9834045331612878,
'recall': 0.9832775919732442, 'f1-score': 0.9833195402624274,
'support': 12259.0}}
Epoch 1/6
Training Loss: 0.0231, Training Accuracy: 0.9946
Validation Loss: 0.0054, Validation Accuracy: 0.9988
{'0': {'precision': 1.0, 'recall': 0.9983245839383447, 'f1-score':
0.9991615896260689, 'support': 8953.0}, '1': {'precision':
0.995483288166215, 'recall': 1.0, 'f1-score': 0.9977365323675872,
'support': 3306.0}, 'accuracy': 0.998776409168774, 'macro avg':
{'precision': 0.9977416440831075, 'recall': 0.9991622919691723, 'f1-
score': 0.9984490609968281, 'support': 12259.0}, 'weighted avg':
{'precision': 0.9987819357759611, 'recall': 0.998776409168774, 'f1-
score': 0.9987772810122718, 'support': 12259.0}}
Epoch 2/6
Training Loss: 0.0060, Training Accuracy: 0.9982
Validation Loss: 0.0070, Validation Accuracy: 0.9985
{'0': {'precision': 0.9998880931065354, 'recall': 0.9979895007260137,
'f1-score': 0.9989378947956845, 'support': 8953.0}, '1': {'precision':
0.9945832079446284, 'recall': 0.999697519661222, 'f1-score':
0.9971338060039222, 'support': 3306.0}, 'accuracy': 0.998450118280447,
'macro avg': {'precision': 0.9972356505255819, 'recall':
0.9988435101936178, 'f1-score': 0.9980358503998034, 'support':
12259.0}, 'weighted avg': {'precision': 0.9984574747571378, 'recall':
0.998450118280447, 'f1-score': 0.9984513691781327, 'support':
12259.0}}
Epoch 3/6
Training Loss: 0.0046, Training Accuracy: 0.9988
Validation Loss: 0.0073, Validation Accuracy: 0.9985
{'0': {'precision': 0.9998880931065354, 'recall': 0.9979895007260137,
'f1-score': 0.9989378947956845, 'support': 8953.0}, '1': {'precision':
0.9945832079446284, 'recall': 0.999697519661222, 'f1-score':
0.9971338060039222, 'support': 3306.0}, 'accuracy': 0.998450118280447,
'macro avg': {'precision': 0.9972356505255819, 'recall':
0.9988435101936178, 'f1-score': 0.9980358503998034, 'support':
12259.0}, 'weighted avg': {'precision': 0.9984574747571378, 'recall':
0.998450118280447, 'f1-score': 0.9984513691781327, 'support':
12259.0}}
Epoch 4/6
Training Loss: 0.0029, Training Accuracy: 0.9992
Validation Loss: 0.0072, Validation Accuracy: 0.9985
{'0': {'precision': 1.0, 'recall': 0.9979895007260137, 'f1-score':
0.9989937388193202, 'support': 8953.0}, '1': {'precision':
0.9945848375451264, 'recall': 1.0, 'f1-score': 0.9972850678733032,
'support': 3306.0}, 'accuracy': 0.9985316910025288, 'macro avg':
{'precision': 0.9972924187725631, 'recall': 0.9989947503630068, 'f1-
```

score': 0.9981394033463117, 'support': 12259.0}, 'weighted avg': {'precision': 0.998539642134284, 'recall': 0.9985316910025288, 'f1-score': 0.9985329454309907, 'support': 12259.0}}
Epoch 5/6
Training Loss: 0.0019, Training Accuracy: 0.9994
Validation Loss: 0.0073, Validation Accuracy: 0.9986
{'0': {'precision': 1.0, 'recall': 0.998101195130124, 'f1-score': 0.9990496953435072, 'support': 8953.0}, '1': {'precision': 0.9948841408365935, 'recall': 1.0, 'f1-score': 0.9974355106350883, 'support': 3306.0}, 'accuracy': 0.9986132637246105, 'macro avg': {'precision': 0.9974420704182967, 'recall': 0.999050597565062, 'f1-score': 0.9982426029892977, 'support': 12259.0}, 'weighted avg': {'precision': 0.9986203580720922, 'recall': 0.9986132637246105, 'f1-score': 0.9986143829488556, 'support': 12259.0}}
Epoch 6/6
Training Loss: 0.0012, Training Accuracy: 0.9995
Validation Loss: 0.0074, Validation Accuracy: 0.9985
{'0': {'precision': 1.0, 'recall': 0.9978778063219033, 'f1-score': 0.9989377760384637, 'support': 8953.0}, '1': {'precision': 0.9942857142857143, 'recall': 1.0, 'f1-score': 0.997134670487106, 'support': 3306.0}, 'accuracy': 0.998450118280447, 'macro avg': {'precision': 0.9971428571428571, 'recall': 0.9989389031609517, 'f1-score': 0.9980362232627848, 'support': 12259.0}, 'weighted avg': {'precision': 0.9984589747474159, 'recall': 0.998450118280447, 'f1-score': 0.9984515155806132, 'support': 12259.0}}
Epoch 1/6
Training Loss: 0.0099, Training Accuracy: 0.9977
Validation Loss: 0.0025, Validation Accuracy: 0.9993
{'0': {'precision': 0.9998896369054189, 'recall': 0.9991177767975298, 'f1-score': 0.9995035578355121, 'support': 9068.0}, '1': {'precision': 0.9974984365228268, 'recall': 0.9996866186148543, 'f1-score': 0.9985913288464549, 'support': 3191.0}, 'accuracy': 0.9992658455012644, 'macro avg': {'precision': 0.9986940367141228, 'recall': 0.999402197706192, 'f1-score': 0.9990474433409835, 'support': 12259.0}, 'weighted avg': {'precision': 0.9992672108983343, 'recall': 0.9992658455012644, 'f1-score': 0.9992661059467707, 'support': 12259.0}}
Epoch 2/6
Training Loss: 0.0048, Training Accuracy: 0.9986
Validation Loss: 0.0048, 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 3/6
Training Loss: 0.0034, Training Accuracy: 0.9991
Validation Loss: 0.0056, 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: 0.0028, Training Accuracy: 0.9993
Validation Loss: 0.0047, 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 5/6
Training Loss: 0.0015, Training Accuracy: 0.9995
Validation Loss: 0.0047, 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 6/6
Training Loss: 0.0015, Training Accuracy: 0.9995
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
```

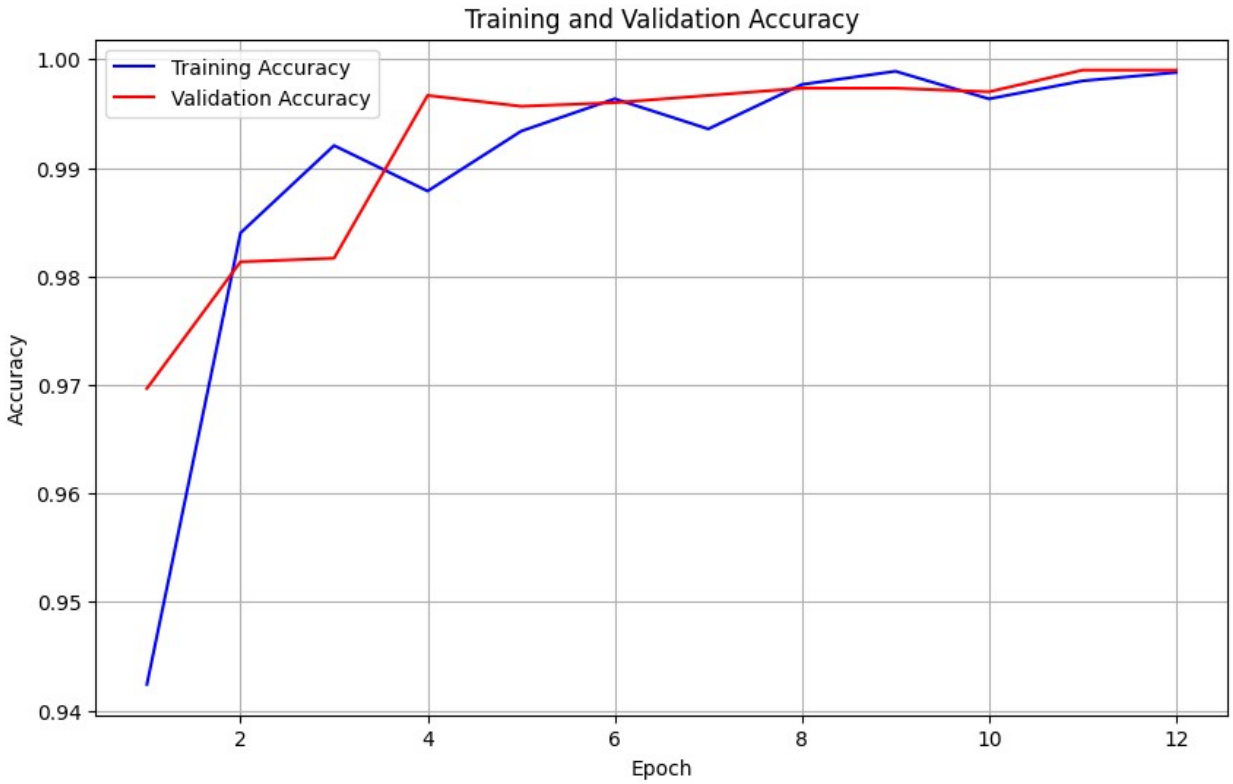
```
Training Loss: 0.0080, Training Accuracy: 0.9982
Validation Loss: 0.0035, Validation Accuracy: 0.9993
{'0': {'precision': 0.999889356052224, 'recall': 0.9992260061919505,
'f1-score': 0.9995575710651476, 'support': 9044.0}, '1': {'precision':
0.997826761875194, 'recall': 0.9996889580093312, 'f1-score':
0.9987569919204475, 'support': 3215.0}, 'accuracy':
0.9993474182233462, 'macro avg': {'precision': 0.998858058963709,
'recall': 0.9994574821006408, 'f1-score': 0.9991572814927976,
'support': 12259.0}, 'weighted avg': {'precision': 0.9993484277318757,
'recall': 0.9993474182233462, 'f1-score': 0.9993476141396064,
'support': 12259.0}}
Epoch 2/6
Training Loss: 0.0036, Training Accuracy: 0.9990
Validation Loss: 0.0029, Validation Accuracy: 0.9994
{'0': {'precision': 1.0, 'recall': 0.9992260061919505, 'f1-score':
0.9996128532713898, 'support': 9044.0}, '1': {'precision':
0.9978274363749224, 'recall': 1.0, 'f1-score': 0.9989125368960696,
'support': 3215.0}, 'accuracy': 0.9994289909454278, 'macro avg':
{'precision': 0.9989137181874612, 'recall': 0.9996130030959752, 'f1-
score': 0.9992626950837298, 'support': 12259.0}, 'weighted avg':
{'precision': 0.9994302314989295, 'recall': 0.9994289909454278, 'f1-
score': 0.9994291908889236, 'support': 12259.0}}
Epoch 3/6
Training Loss: 0.0023, Training Accuracy: 0.9993
Validation Loss: 0.0030, Validation Accuracy: 0.9993
{'0': {'precision': 0.999889356052224, 'recall': 0.9992260061919505,
'f1-score': 0.9995575710651476, 'support': 9044.0}, '1': {'precision':
0.997826761875194, 'recall': 0.9996889580093312, 'f1-score':
0.9987569919204475, 'support': 3215.0}, 'accuracy':
0.9993474182233462, 'macro avg': {'precision': 0.998858058963709,
'recall': 0.9994574821006408, 'f1-score': 0.9991572814927976,
'support': 12259.0}, 'weighted avg': {'precision': 0.9993484277318757,
'recall': 0.9993474182233462, 'f1-score': 0.9993476141396064,
'support': 12259.0}}
Epoch 4/6
Training Loss: 0.0026, Training Accuracy: 0.9992
Validation Loss: 0.0028, Validation Accuracy: 0.9994
{'0': {'precision': 1.0, 'recall': 0.9992260061919505, 'f1-score':
0.9996128532713898, 'support': 9044.0}, '1': {'precision':
0.9978274363749224, 'recall': 1.0, 'f1-score': 0.9989125368960696,
'support': 3215.0}, 'accuracy': 0.9994289909454278, 'macro avg':
{'precision': 0.9989137181874612, 'recall': 0.9996130030959752, 'f1-
score': 0.9992626950837298, 'support': 12259.0}, 'weighted avg':
{'precision': 0.9994302314989295, 'recall': 0.9994289909454278, 'f1-
score': 0.9994291908889236, 'support': 12259.0}}
Epoch 5/6
Training Loss: 0.0016, Training Accuracy: 0.9994
Validation Loss: 0.0028, Validation Accuracy: 0.9995
{'0': {'precision': 1.0, 'recall': 0.9993365767359575, 'f1-score':
```

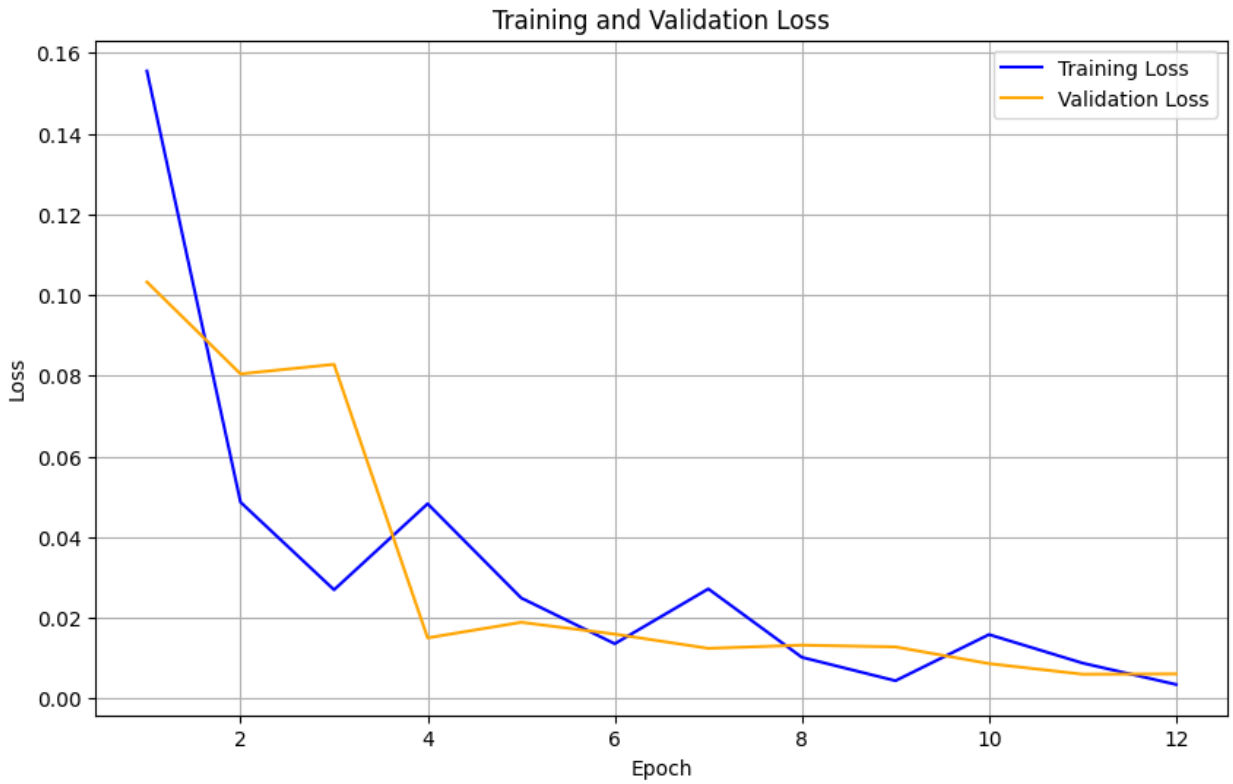
```
0.9996681782988608, 'support': 9044.0}, '1': {'precision':
0.9981372244644521, 'recall': 1.0, 'f1-score': 0.9990677439403356,
'support': 3215.0}, 'accuracy': 0.9995105636675096, 'macro avg':
{'precision': 0.9990686122322261, 'recall': 0.9996682883679788, 'f1-
score': 0.9993679611195982, 'support': 12259.0}, 'weighted avg':
{'precision': 0.9995114753775359, 'recall': 0.9995105636675096, 'f1-
score': 0.9995107106047046, 'support': 12259.0}}
Epoch 6/6
Training Loss: 0.0013, Training Accuracy: 0.9995
Validation Loss: 0.0029, Validation Accuracy: 0.9995
{'0': {'precision': 1.0, 'recall': 0.9993365767359575, 'f1-score':
0.9996681782988608, 'support': 9044.0}, '1': {'precision':
0.9981372244644521, 'recall': 1.0, 'f1-score': 0.9990677439403356,
'support': 3215.0}, 'accuracy': 0.9995105636675096, 'macro avg':
{'precision': 0.9990686122322261, 'recall': 0.9996682883679788, 'f1-
score': 0.9993679611195982, 'support': 12259.0}, 'weighted avg':
{'precision': 0.9995114753775359, 'recall': 0.9995105636675096, 'f1-
score': 0.9995107106047046, 'support': 12259.0}}
Epoch 1/6
Training Loss: 0.0058, Training Accuracy: 0.9986
Validation Loss: 0.0036, 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: 0.0034, Training Accuracy: 0.9990
Validation Loss: 0.0032, Validation Accuracy: 0.9993
{'0': {'precision': 1.0, 'recall': 0.9989995553579368, 'f1-score':
0.9994995273313685, 'support': 8996.0}, '1': {'precision':
0.9972485478446959, 'recall': 1.0, 'f1-score': 0.9986223786927905,
'support': 3262.0}, 'accuracy': 0.9992657856093979, 'macro avg':
{'precision': 0.9986242739223479, 'recall': 0.9994997776789685, 'f1-
score': 0.9990609530120795, 'support': 12258.0}, 'weighted avg':
{'precision': 0.9992678057651654, 'recall': 0.9992657856093979, 'f1-
score': 0.9992661076169744, 'support': 12258.0}}
Epoch 3/6
Training Loss: 0.0028, Training Accuracy: 0.9992
Validation Loss: 0.0027, Validation Accuracy: 0.9994
{'0': {'precision': 1.0, 'recall': 0.9992218763895064, 'f1-score':
0.9996107867667501, 'support': 8996.0}, '1': {'precision':
0.9978586723768736, 'recall': 1.0, 'f1-score': 0.9989281886387996,
'support': 3262.0}, 'accuracy': 0.9994289443628651, 'macro avg':
{'precision': 0.9989293361884368, 'recall': 0.9996109381947532, 'f1-
```

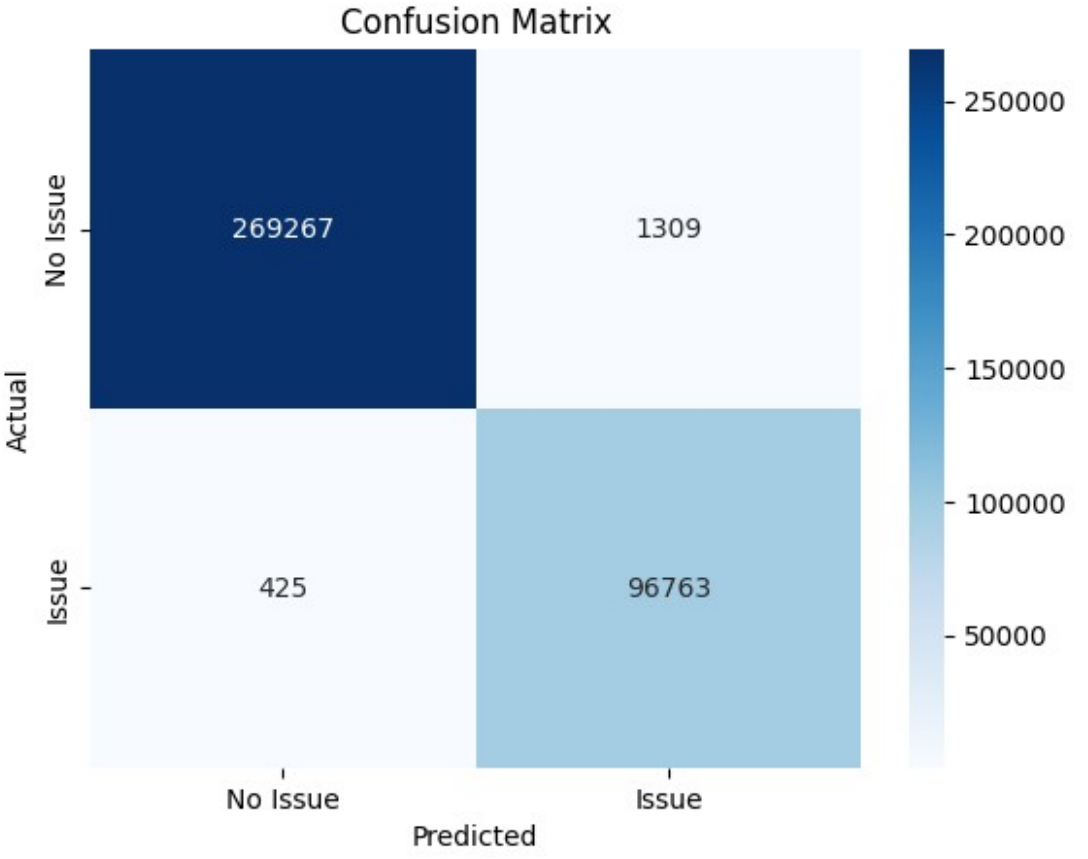
```
score': 0.9992694877027748, 'support': 12258.0}, 'weighted avg':
{'precision': 0.9994301671800753, 'recall': 0.9994289443628651, 'f1-
score': 0.9994291392636195, 'support': 12258.0}}
Epoch 4/6
Training Loss: 0.0015, Training Accuracy: 0.9994
Validation Loss: 0.0045, Validation Accuracy: 0.9993
{'0': {'precision': 0.9998887529202358, 'recall': 0.9991107158737217,
'f1-score': 0.9994995829858215, 'support': 8996.0}, '1': {'precision':
0.9975527684307127, 'recall': 0.9996934396076027, 'f1-score':
0.9986219568213137, 'support': 3262.0}, 'accuracy':
0.9992657856093979, 'macro avg': {'precision': 0.9987207606754742,
'recall': 0.9994020777406623, 'f1-score': 0.9990607699035676,
'support': 12258.0}, 'weighted avg': {'precision': 0.9992671195865089,
'recall': 0.9992657856093979, 'f1-score': 0.9992660361960822,
'support': 12258.0}}
Epoch 5/6
Training Loss: 0.0012, Training Accuracy: 0.9995
Validation Loss: 0.0044, Validation Accuracy: 0.9993
{'0': {'precision': 0.9998887529202358, 'recall': 0.9991107158737217,
'f1-score': 0.9994995829858215, 'support': 8996.0}, '1': {'precision':
0.9975527684307127, 'recall': 0.9996934396076027, 'f1-score':
0.9986219568213137, 'support': 3262.0}, 'accuracy':
0.9992657856093979, 'macro avg': {'precision': 0.9987207606754742,
'recall': 0.9994020777406623, 'f1-score': 0.9990607699035676,
'support': 12258.0}, 'weighted avg': {'precision': 0.9992671195865089,
'recall': 0.9992657856093979, 'f1-score': 0.9992660361960822,
'support': 12258.0}}
Epoch 6/6
Training Loss: 0.0010, Training Accuracy: 0.9995
Validation Loss: 0.0042, 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()
```

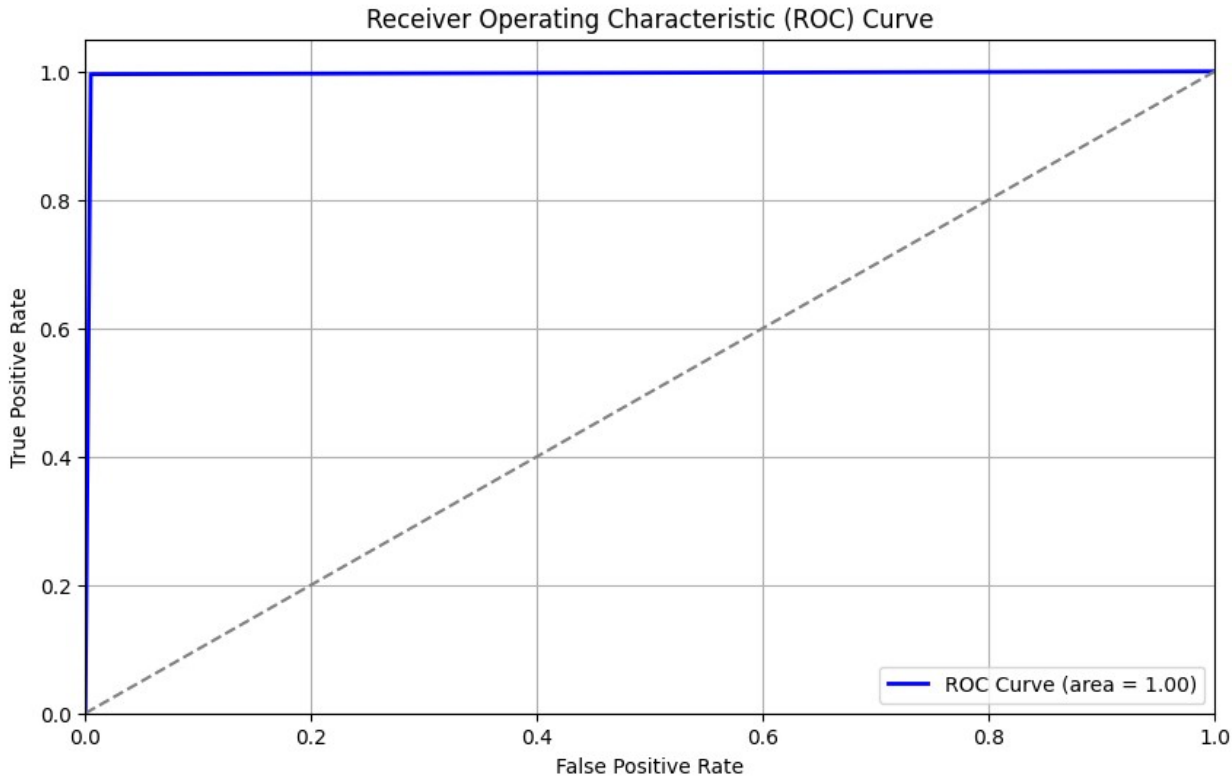
```
plt.grid(True)
plt.show()

# 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)
plt.show()

# 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()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
1.00
0.99
0.98
0.97
0.96
0.95
0.94
2
4
6
8
10
12


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2
4
6
8
10
12


Confusion Matrix
269267
1309
425
96763
No Issue
Issue
Actual
No Issue
Issue
Predicted
250000
200000
150000
100000
50000

```
# 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()
```




```
# 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: 0.9923
Macro-Precision: 0.9923
Macro-Recall: 0.9923
Macro-F1 Score: 0.9923
Weighted-Precision: 0.9923
Weighted-Recall: 0.9923
Weighted-F1 Score: 0.9923
```