```
!pip install contractions

print("Hello World")

Hello World

## Load all libraries
import pandas as pd
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
from transformers import RobertaTokenizer, RobertaModel, AdamW,
get_linear_schedule_with_warmup
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
from sklearn.utils.class_weight import compute_class_weight
from imblearn.over_sampling import RandomOverSampler
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

# Suppress warnings
warnings.filterwarnings("ignore")

# 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 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()
```

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

# Plot distribution
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()

### Text Preprocessing
def preprocess_review(text):
    text = str(text)
    # Fix contractions
    text = contractions.fix(text)
    # Remove HTML
    text = BeautifulSoup(text, "html.parser").get_text()
    # Clean special characters
    text = re.sub(r'[^a-zA-Z0-9\s]', ' ', text)
    # Normalize whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    return text

df['Clean_Review'] = df['Review'].apply(preprocess_review)

### Dataset Class
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 = str(self.texts[idx])
        label = int(self.labels[idx])
        encoding = self.tokenizer(
            text,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )
        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'label': torch.tensor(label, dtype=torch.long)
```

```
        }

### Model Architecture
class RoBERTaClassifier(nn.Module):
    def __init__(self, roberta_model_name, num_classes):
        super(RoBERTaClassifier, self).__init__()
        self.roberta =
RobertaModel.from_pretrained(roberta_model_name)
        self.dropout = nn.Dropout(0.1)
        self.fc = nn.Linear(self.roberta.config.hidden_size,
num_classes)

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

### Training Function
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)
        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
```

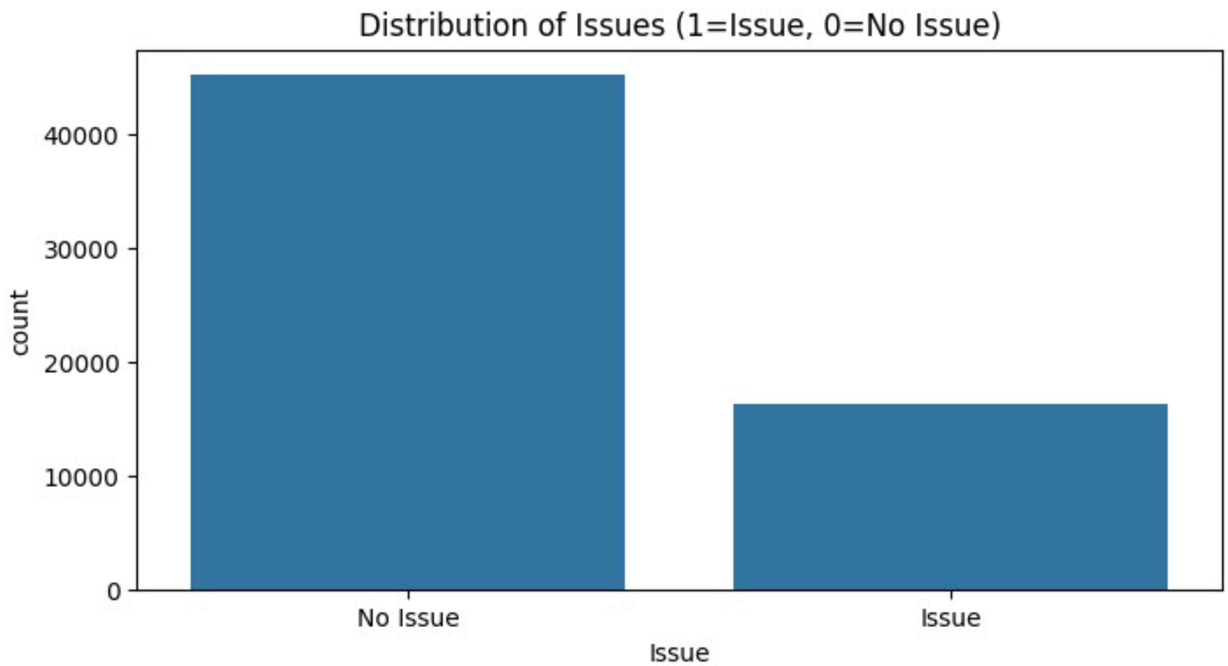
```
    return avg_loss, avg_accuracy

### Evaluation Function
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,
target_names=['No Issue', 'Issue'], output_dict=True)
    avg_loss = total_loss / len(data_loader)
    return accuracy, report, avg_loss, predictions, actual_labels

### Main Training Setup
# Parameters
roberta_model_name = "roberta-base"
num_classes = 2
max_length = 128
batch_size = 16
num_epochs = 6
learning_rate = 1e-5

# Initialize
tokenizer = RobertaTokenizer.from_pretrained(roberta_model_name)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = RoBERTaClassifier(roberta_model_name, num_classes).to(device)
optimizer = AdamW(model.parameters(), lr=learning_rate)

# Prepare data
texts = df['Clean_Review'].values
labels = df['Issue'].values

# Compute class weights
class_weights = compute_class_weight('balanced',
```

```
classes=np.unique(labels), y=labels)
print("\nClass Weights:", class_weights)

# Cross-validation
n_splits = 5
kf = KFold(n_splits=n_splits, random_state=42, shuffle=True)

# Track metrics
all_predictions = []
all_actual_labels = []

for fold, (train_idx, val_idx) in enumerate(kf.split(texts)):
    print(f"\n=== Fold {fold + 1}/{n_splits} ===")

    # Split data
    train_texts, val_texts = texts[train_idx], texts[val_idx]
    train_labels, val_labels = labels[train_idx], labels[val_idx]

    # Handle class imbalance
    ros = RandomOverSampler(random_state=42)
    train_texts_reshaped = train_texts.reshape(-1, 1)
    train_texts_resampled, train_labels_resampled =
ros.fit_resample(train_texts_reshaped, train_labels)
    train_texts = train_texts_resampled[:, 0]
    train_labels = train_labels_resampled

    # Create datasets
    train_dataset = TextClassificationDataset(train_texts,
train_labels, tokenizer, max_length)
    val_dataset = TextClassificationDataset(val_texts, val_labels,
tokenizer, max_length)

    # Create dataloaders
    train_dataloader = DataLoader(train_dataset,
batch_size=batch_size, shuffle=True)
    val_dataloader = DataLoader(val_dataset, batch_size=batch_size)

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

    # Training loop
    for epoch in range(num_epochs):
        print(f"\nEpoch {epoch + 1}/{num_epochs}")
        train_loss, train_accuracy = train(model, train_dataloader,
optimizer, scheduler, device, class_weights)
        print(f"Train Loss: {train_loss:.4f}, Accuracy:
{train_accuracy:.4f}")
```

```
        val_accuracy, val_report, val_loss, val_preds, val_true =
evaluate(model, val_dataloader, device)
        print(f"Val Loss: {val_loss:.4f}, Accuracy:
{val_accuracy:.4f}")
        print(classification_report(val_true, val_preds,
target_names=['No Issue', 'Issue']))

        all_predictions.extend(val_preds)
        all_actual_labels.extend(val_true)

### Final Evaluation
# Confusion Matrix
cm = confusion_matrix(all_actual_labels, all_predictions)
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Issue', 'Issue'],
            yticklabels=['No Issue', 'Issue'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

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

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

# Classification Report
print("\n=== Final Classification Report ===")
print(classification_report(all_actual_labels, all_predictions,
target_names=['No Issue', 'Issue']))
```

```
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
```

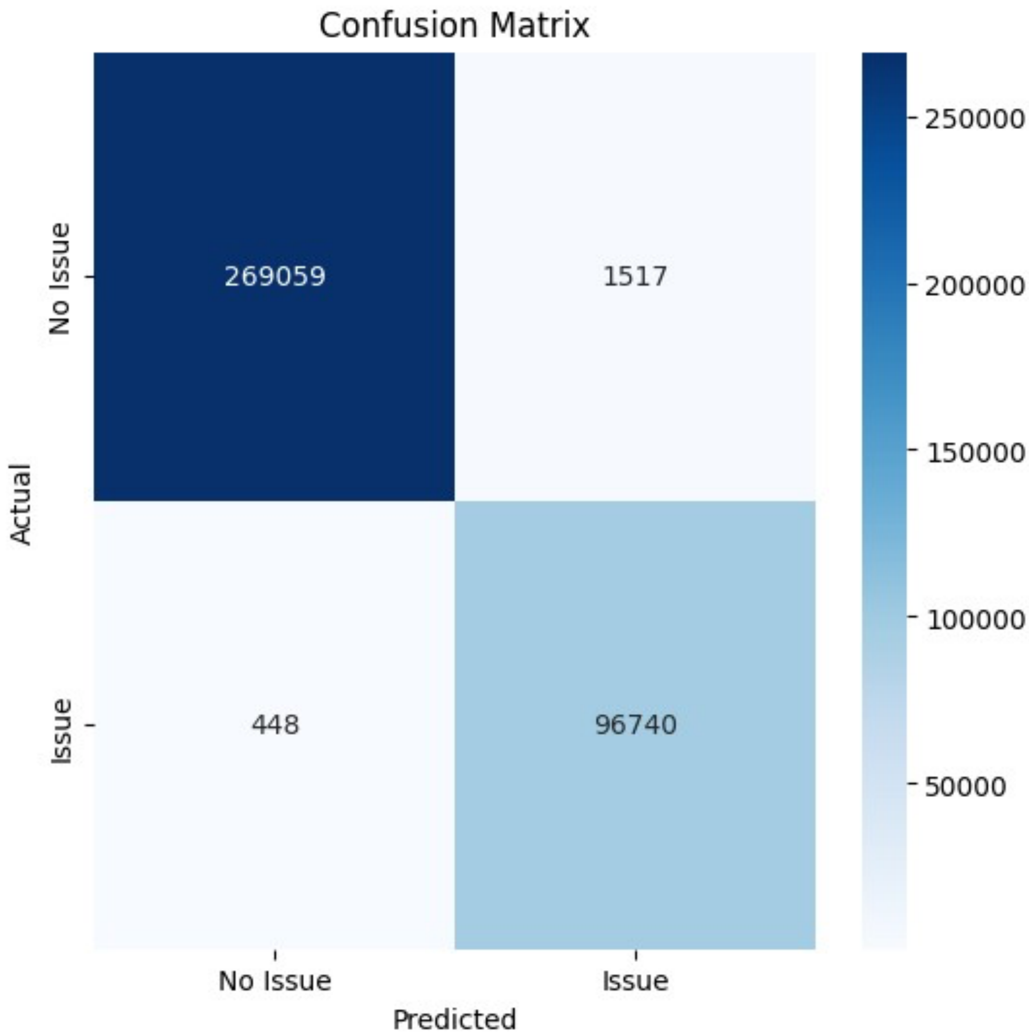
```
0    45096
1    16198
Name: count, dtype: int64
```

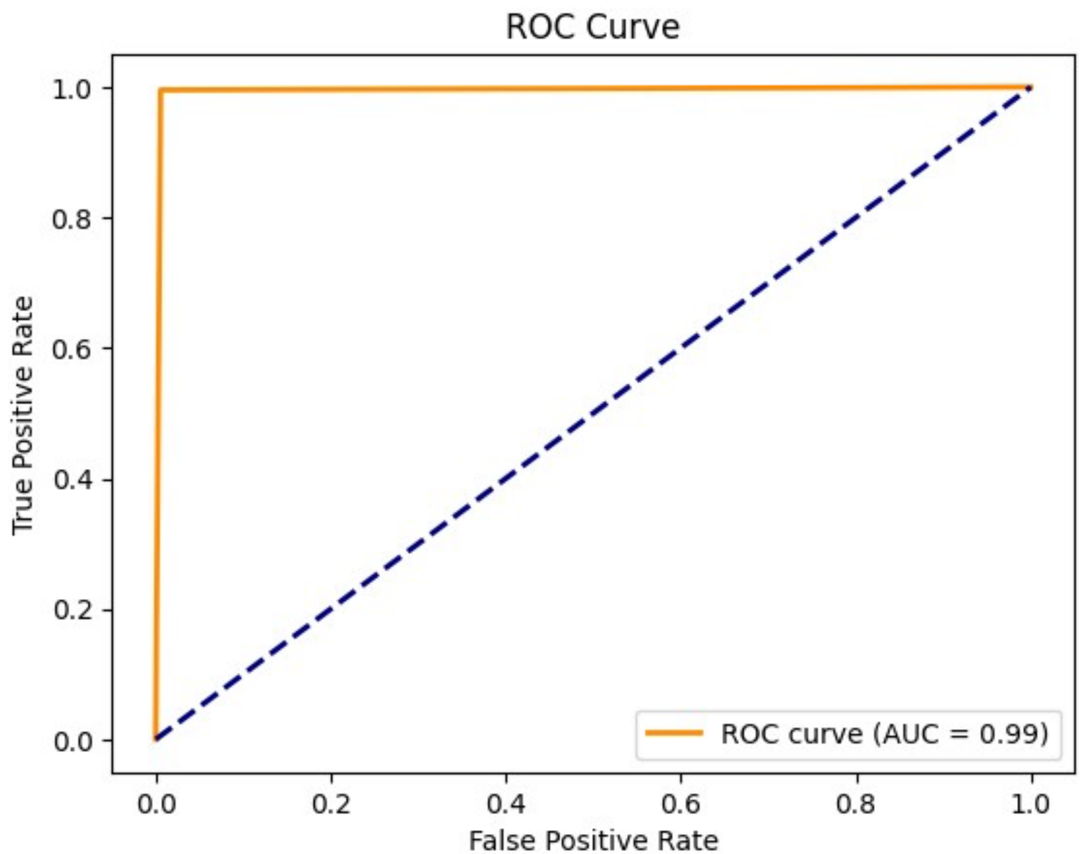


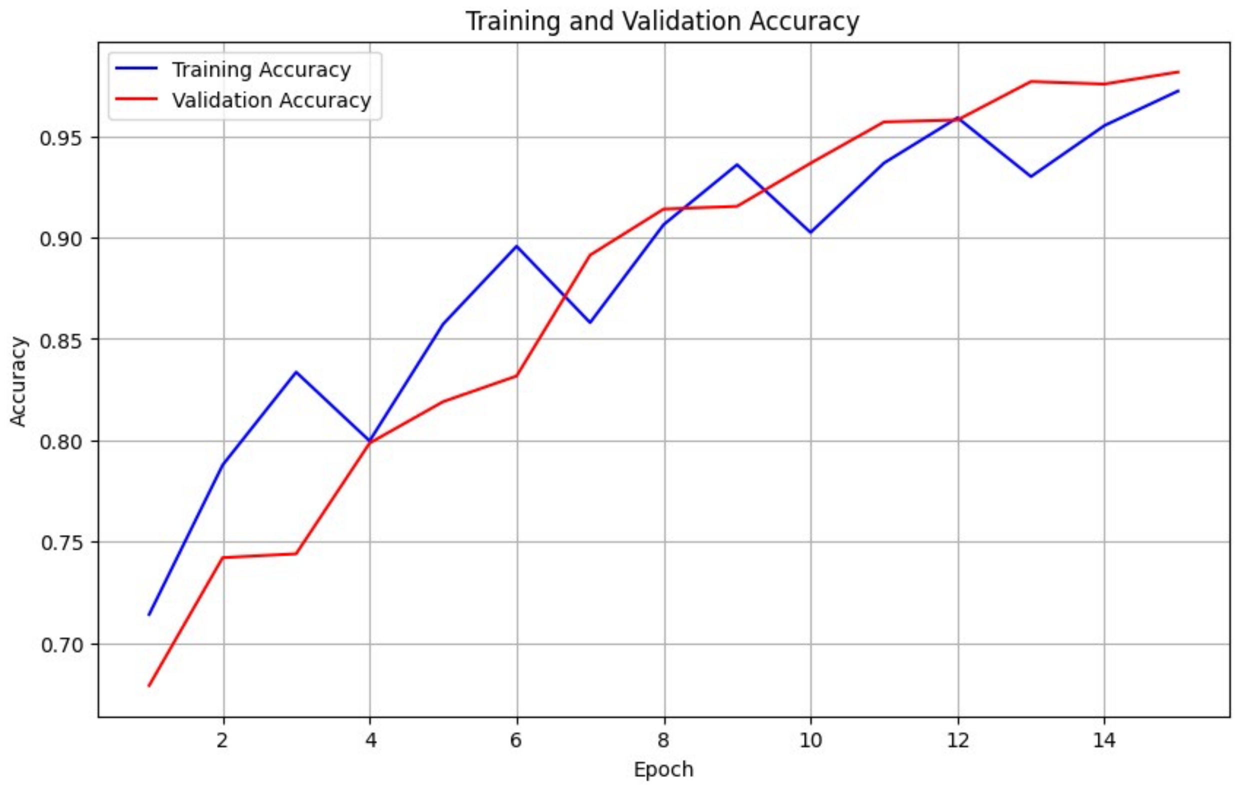

```
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-base and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.


Class Weights: [0.67959464 1.89202371]

=== Fold 1/5 ===

Epoch 1/6
Train Loss: 0.0000, Accuracy: 0.9561
Val Loss: 0.1254, Accuracy: 0.9713
              precision    recall  f1-score   support

    No Issue       0.99      0.97      0.98      9035
       Issue       0.92      0.97      0.95      3224

    accuracy                           0.97     12259
   macro avg       0.96      0.97      0.96     12259
weighted avg       0.97      0.97      0.97     12259


Epoch 2/6
```

```
Train Loss: 0.0000, Accuracy: 0.9866
Val Loss: 0.1236, Accuracy: 0.9786
              precision    recall  f1-score   support

    No Issue       0.99      0.98      0.99      9035
       Issue       0.94      0.98      0.96      3224

    accuracy                           0.98     12259
   macro avg       0.97      0.98      0.97     12259
weighted avg       0.98      0.98      0.98     12259


Epoch 3/6
Train Loss: 0.0000, Accuracy: 0.9927
Val Loss: 0.1362, Accuracy: 0.9784
              precision    recall  f1-score   support

    No Issue       0.99      0.98      0.99      9035
       Issue       0.94      0.98      0.96      3224

    accuracy                           0.98     12259
   macro avg       0.97      0.98      0.97     12259
weighted avg       0.98      0.98      0.98     12259


Epoch 4/6
Train Loss: 0.0000, Accuracy: 0.9962
Val Loss: 0.1397, Accuracy: 0.9806
              precision    recall  f1-score   support

    No Issue       0.99      0.98      0.99      9035
       Issue       0.94      0.98      0.96      3224

    accuracy                           0.98     12259
   macro avg       0.97      0.98      0.98     12259
weighted avg       0.98      0.98      0.98     12259


Epoch 5/6
Train Loss: 0.0000, Accuracy: 0.9977
Val Loss: 0.1393, Accuracy: 0.9835
              precision    recall  f1-score   support

    No Issue       0.99      0.99      0.99      9035
       Issue       0.96      0.98      0.97      3224

    accuracy                           0.98     12259
   macro avg       0.98      0.98      0.98     12259
weighted avg       0.98      0.98      0.98     12259
```

```
Epoch 6/6
Train Loss: 0.0000, Accuracy: 0.9986
Val Loss: 0.1439, Accuracy: 0.9834
              precision    recall  f1-score   support

    No Issue       0.99      0.98      0.99      9035
       Issue       0.96      0.98      0.97      3224

    accuracy                           0.98     12259
   macro avg       0.98      0.98      0.98     12259
weighted avg       0.98      0.98      0.98     12259


=== Fold 2/5 ===

Epoch 1/6
Train Loss: 0.0000, Accuracy: 0.9916
Val Loss: 0.0134, Accuracy: 0.9971
              precision    recall  f1-score   support

    No Issue       1.00      1.00      1.00      8953
       Issue       0.99      1.00      0.99      3306

    accuracy                           1.00     12259
   macro avg       0.99      1.00      1.00     12259
weighted avg       1.00      1.00      1.00     12259


Epoch 2/6
Train Loss: 0.0000, Accuracy: 0.9961
Val Loss: 0.0250, Accuracy: 0.9957
              precision    recall  f1-score   support

    No Issue       1.00      0.99      1.00      8953
       Issue       0.99      1.00      0.99      3306

    accuracy                           1.00     12259
   macro avg       0.99      1.00      0.99     12259
weighted avg       1.00      1.00      1.00     12259


Epoch 3/6
Train Loss: 0.0000, Accuracy: 0.9976
Val Loss: 0.0221, Accuracy: 0.9967
              precision    recall  f1-score   support

    No Issue       1.00      1.00      1.00      8953
       Issue       0.99      1.00      0.99      3306

    accuracy                           1.00     12259
   macro avg       0.99      1.00      1.00     12259
```


Confusion Matrix
269059
1517
448
96740
No Issue
Issue
Actual
No Issue
Issue
Predicted
250000
200000
150000
100000
50000


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.99)



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

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

```
Average Metrics:
Average Accuracy: 0.9849
Macro-Precision: 0.9850
Macro-Recall: 0.9849
Macro-F1 Score: 0.9849
Weighted-Precision: 0.9850
Weighted-Recall: 0.9849
Weighted-F1 Score: 0.9849
```