In [ ]:
```python
from transformers import TFBertForSequenceClassification, BertTokenizer

# Load the model
model_path = 'D:\Mini Project\minipj\models\model_final'
model = TFBertForSequenceClassification.from_pretrained(model_path)

# Load the tokenizer
tokenizer_path = 'D:\Mini Project\minipj\models\\final_tokenizer'
tokenizer = BertTokenizer.from_pretrained(tokenizer_path)
```

In [ ]:
```python
from transformers import BertTokenizer, TFBertForSequenceClassification, InputEx
import tensorflow as tf
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Load the dataset
train = pd.read_csv("D:\Mini Project\minipj\data\Suicide_Detection.csv")

# Preprocess the labels
train['class'] = train['class'].map({'suicide': 1, 'non-suicide': 0})
data = train['text'].tolist()
labels = train['class'].tolist()

# Split the dataset
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.10
x_train, x_val, y_train, y_val= train_test_split(x_train, y_train, test_size=0.2

test_encodings = tokenizer(x_test, truncation=True, padding=True, max_length=128

# Convert to TensorFlow dataset
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings), y_test)
test_dataset = test_dataset.batch(32)  # Use the same batch size as used in trai
```

In [ ]:
```python
from sklearn.metrics import accuracy_score
import numpy as np

# Make predictions
preds = model.predict(test_dataset)
predicted_labels = np.argmax(preds.logits, axis=1)

# Calculate accuracy
accuracy = accuracy_score(y_test, predicted_labels)
```

Performance Metrics

In [ ]:
```python
print(f'Test Accuracy: {accuracy*100:.2f}%')
```

```
Test Accuracy: 96.99%
```

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_curve, auc
import seaborn as sns
```

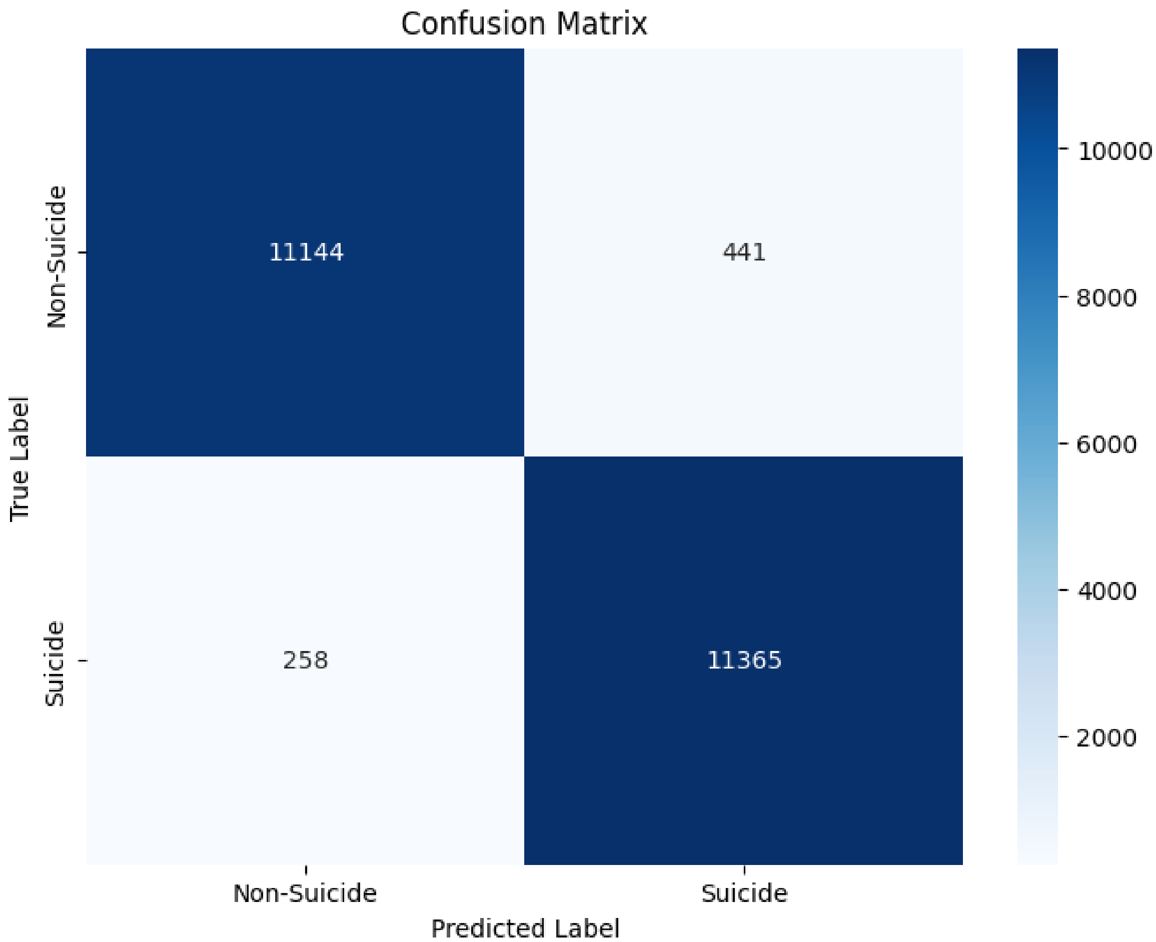
In [ ]:
```
# Assuming 'y_test' is your true labels and 'predicted_labels' are the labels pr
print(classification_report(y_test, predicted_labels, target_names=['Non-Suicide
```

```
              precision    recall  f1-score   support

 Non-Suicide       0.98      0.96      0.97     11585
     Suicide       0.96      0.98      0.97     11623

    accuracy                           0.97     23208
   macro avg       0.97      0.97      0.97     23208
weighted avg       0.97      0.97      0.97     23208
```

In [ ]:
```
cm = confusion_matrix(y_test, predicted_labels)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Suicide', '
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

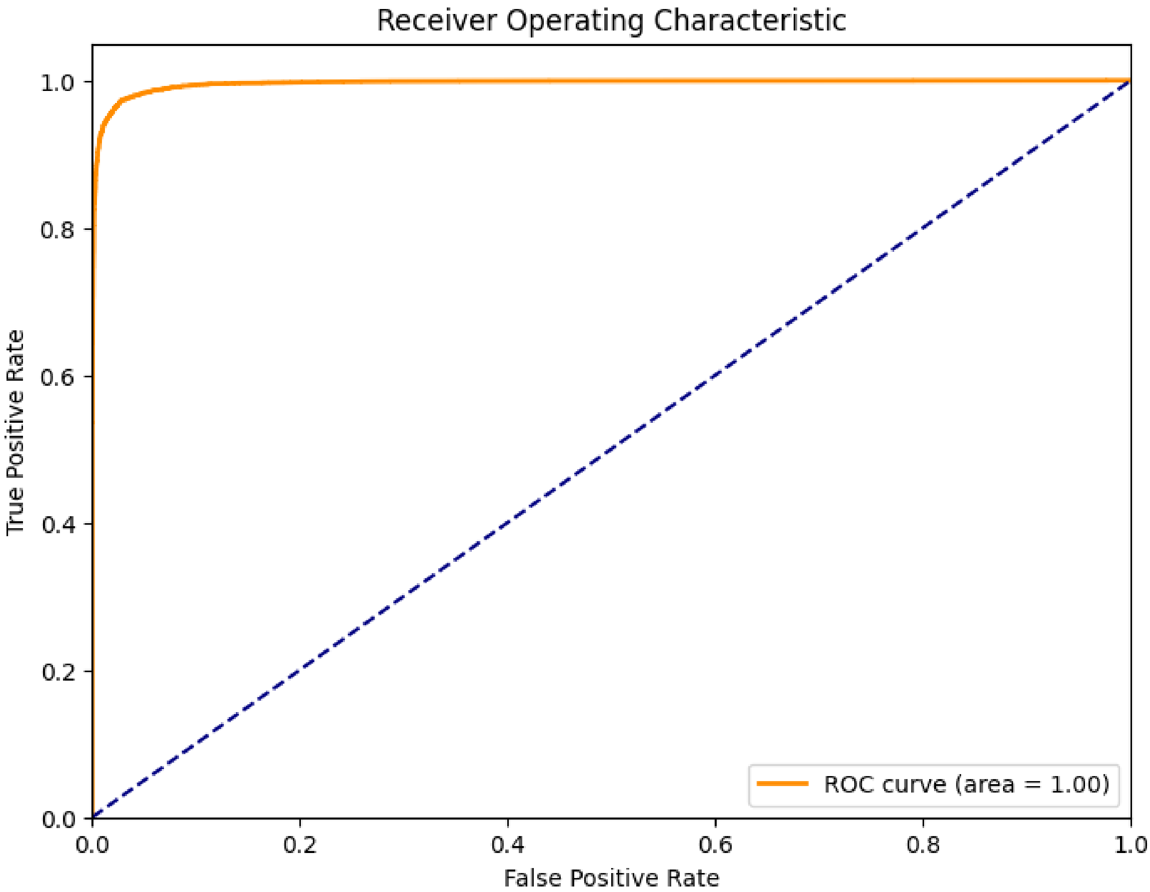


In [ ]:
```
# Compute probabilities using softmax
probs = tf.nn.softmax(preds.logits, axis=1).numpy()[:, 1]

# Compute ROC curve and ROC area
fpr, tpr, thresholds = roc_curve(y_test, probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', 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')
plt.legend(loc="lower right")
plt.show()
```



In [ ]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

accuracy = accuracy_score(y_test, predicted_labels)
precision = precision_score(y_test, predicted_labels)
recall = recall_score(y_test, predicted_labels)
f1 = f1_score(y_test, predicted_labels)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
Accuracy: 0.9699
Precision: 0.9626
Recall: 0.9778
F1-Score: 0.9702
```

In [ ]: