In [1]:
```python
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import roc_auc_score, classification_report, confusion_matrix, roc_curve, auc
import torch
from torch.utils.data import DataLoader, Dataset
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification
from transformers import Trainer, TrainingArguments, EarlyStoppingCallback
```

In [2]:
```python
# Load datasets
def load_np_array(file_name):
    X_array = np.load('data/X_' + file_name + '_array.npy')
    y_array = np.load('data/y_' + file_name + '_array.npy')
    return X_array, y_array

# Convert numerical data to strings
def convert_to_string(data):
    return [" ".join(map(str, seq.flatten())) for seq in data]

# Define the compute_metrics function to calculate accuracy
def compute_metrics(p):
    pred, labels = p
    pred = np.argmax(pred, axis=1)
    accuracy = accuracy_score(labels, pred)
    return {"eval_accuracy": accuracy}

def set_seeds(seed):
    np.random.seed(seed)
    random.seed(seed)
    torch.manual_seed(seed)

# Call this function before creating and training your model
set_seeds(42)
```

In [3]:
```python
# CONSTANT
LEARNING_RATE_VAL = 0.0001
BATCH_SIZE_VAL = 32
```

```python
VALIDATION_SPLIT_VAL = 0.2
EPOCH_NUMBER = 50
```

In [4]:
```python
X_train_fall, y_train_fall = load_np_array("train_fall")
X_train_notfall, y_train_notfall = load_np_array("train_notfall")
X_test_fall, y_test_fall = load_np_array("test_fall")
X_test_notfall, y_test_notfall = load_np_array("test_notfall")

# Combine fall and notfall datasets
X_train = np.concatenate((X_train_fall, X_train_notfall), axis=0)
y_train = np.concatenate((y_train_fall, y_train_notfall), axis=0)
X_test = np.concatenate((X_test_fall, X_test_notfall), axis=0)
y_test = np.concatenate((y_test_fall, y_test_notfall), axis=0)

print("Combined training data shape:", X_train.shape, y_train.shape)
print("Combined testing data shape:", X_test.shape, y_test.shape)
```

```
Combined training data shape: (5824, 40, 3) (5824,)
Combined testing data shape: (2912, 40, 3) (2912,)
```

In [5]:
```python
# Split train data into train and validation sets
X_train_split, X_val, y_train_split, y_val = train_test_split(
    X_train, y_train, test_size=VALIDATION_SPLIT_VAL, random_state=42
)

# Convert numerical data to strings
X_train_text_split = convert_to_string(X_train_split)
X_val_text = convert_to_string(X_val)
X_train_text = convert_to_string(X_train)
X_test_text = convert_to_string(X_test)

# Tokenizer and Model Initialization
model_name = 'distilbert-base-uncased'
tokenizer = DistilBertTokenizer.from_pretrained(model_name)
model = DistilBertForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at disti
lbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bia
s', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inferen
ce.
```

In [6]:
```python
# Create Dataset class
class FallDetectionDataset(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 = self.labels[idx]
        encoding = self.tokenizer.encode_plus(
            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(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# Create Dataset and DataLoader instances
max_length = 128  # Adjust max_length as needed

train_dataset_split = FallDetectionDataset(X_train_text_split, y_train_split, tokenizer, max_length)
val_dataset = FallDetectionDataset(X_val_text, y_val, tokenizer, max_length)
test_dataset = FallDetectionDataset(X_test_text, y_test, tokenizer, max_length)

train_loader_split = DataLoader(train_dataset_split, batch_size=BATCH_SIZE_VAL, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE_VAL)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE_VAL)
```

In [7]:
```python
# Training Arguments
training_args = TrainingArguments(
    output_dir='./results',
```

```python
    num_train_epochs=50,  # Number of epochs
    per_device_train_batch_size=BATCH_SIZE_VAL,  # Batch size
    per_device_eval_batch_size=BATCH_SIZE_VAL,
    warmup_steps=100,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy='epoch',
    save_strategy='epoch',
    load_best_model_at_end=True,  # Load best model at the end
    gradient_accumulation_steps=2,  # Simulate larger batch size
    learning_rate=LEARNING_RATE_VAL,  # Learning rate
)

# Trainer with EarlyStoppingCallback
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset_split,  # Use split data
    eval_dataset=val_dataset,  # Use validation data
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=1)]  # Early stopping
)
```

```
/opt/miniconda3/envs/torch/lib/python3.11/site-packages/transformers/training_args.py:1525: FutureWarning:
`evaluation_strategy` is deprecated and will be removed in version 4.46 of 🤗 Transformers. Use `eval_strat
egy` instead
  warnings.warn(
```

In [8]:

```python
# Train the model
trainer.train()

# Evaluate the model and print results
results = trainer.evaluate()

# Inspect the tokenization process
sample_text = X_train_text[0]
encoding = tokenizer.encode_plus(
    sample_text,
    max_length=max_length,
    padding='max_length',
```

```python
    truncation=True,
    return_tensors='pt'
)

print("Sample text:", sample_text)
print("Tokenized input_ids:", encoding['input_ids'])
print("Tokenized attention_mask:", encoding['attention_mask'])

# Inspect a few examples from the dataset
for i in range(2):
    print(f"\nExample {i+1}")
    print("Text:", X_train_text[i])
    print("Label:", y_train[i])
    encoding = tokenizer.encode_plus(
        X_train_text[i],
        max_length=max_length,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )
    print("Tokenized input_ids:", encoding['input_ids'].flatten().tolist())
    print("Tokenized attention_mask:", encoding['attention_mask'].flatten().tolist())
```

[292/3650 08:06 < 1:33:58, 0.60 it/s, Epoch 4/50]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.567200	0.585737	0.715021
2	0.575100	0.540995	0.734764
3	0.470100	0.473432	0.777682
4	0.417000	0.487888	0.789700

```
Sample text: 0.7175293 0.17138672 0.18847658 0.35571292 0.108154304 0.33398438 0.35571292 0.108154304 0.333
98438 0.35571292 0.108154304 0.33398438 0.23779297 0.0703125 0.29296875 0.23779297 0.0703125 0.29296875 0.2
644043 0.16650392 0.2529297 0.29858398 0.12670898 0.15185548 0.29858398 0.12670898 0.15185548 0.29858398 0.
12670898 0.15185548 0.2915039 0.004394531 0.0703125 0.2915039 0.004394531 0.0703125 0.2890625 0.11694337 0.
072753906 0.2890625 0.11694337 0.072753906 0.2890625 0.11694337 0.072753906 0.58032227 0.20654297 0.1684570
3 0.58032227 0.20654297 0.16845703 0.58032227 0.20654297 0.16845703 0.8317871 0.20800781 0.18896484 1.27783
22 0.30639648 0.5510254 0.7670899 0.34057617 0.8496094 0.16235352 -0.4787598 0.26538086 0.44921875 -0.71655
28 -0.53881836 0.689209 -1.6711427 -0.91284186 0.8618164 0.095947266 -2.5136719 0.8618164 0.095947266 -2.51
36719 1.9758302 0.5515137 -2.2919922 4.548828 0.8195801 1.6223145 4.928711 0.5324707 1.5439453 1.4716797 0.
45605472 0.47949222 -0.59277344 0.22460938 0.09814453 0.9501954 0.22021484 0.8041992 0.9501954 0.22021484
0.8041992 0.9050293 0.009277344 0.72143555 0.6845703 0.39501953 0.1381836 1.2570801 0.114990234 0.6828614
1.3525392 0.54296875 0.72998047 0.86572266 -0.061523438 0.54418945 0.58154297 0.005371094 0.7531739 0.58154
297 -0.13647461 0.5866699
Tokenized input_ids: tensor([[  101,  1014,  1012,  6390, 23352, 24594,  2509,  1014,  1012, 27016,
         20842,  2581,  2475,  1014,  1012,  6988,  2581, 26187,  2620,  1014,
          1012, 26271,  2581, 12521,  2683,  2475,  1014,  1012, 10715, 16068,
         23777,  2692,  2549,  1014,  1012, 21211,  2683,  2620, 23777,  2620,
          1014,  1012, 26271,  2581, 12521,  2683,  2475,  1014,  1012, 10715,
         16068, 23777,  2692,  2549,  1014,  1012, 21211,  2683,  2620, 23777,
          2620,  1014,  1012, 26271,  2581, 12521,  2683,  2475,  1014,  1012,
         10715, 16068, 23777,  2692,  2549,  1014,  1012, 21211,  2683,  2620,
         23777,  2620,  1014,  1012, 23297,  2581,  2683, 24594,  2581,  1014,
          1012,  5718,  2692, 21486, 17788,  1014,  1012, 25797,  2683,  2575,
          2620, 23352,  1014,  1012, 23297,  2581,  2683, 24594,  2581,  1014,
          1012,  5718,  2692, 21486, 17788,  1014,  1012, 25797,  2683,  2575,
          2620, 23352,  1014,  1012, 21611, 12740, 23777,   102]])
Tokenized attention_mask: tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1]])

Example 1
Text: 0.7175293 0.17138672 0.18847658 0.35571292 0.108154304 0.33398438 0.35571292 0.108154304 0.33398438
0.35571292 0.108154304 0.33398438 0.23779297 0.0703125 0.29296875 0.23779297 0.0703125 0.29296875 0.2644043
0.16650392 0.2529297 0.29858398 0.12670898 0.15185548 0.29858398 0.12670898 0.15185548 0.29858398 0.1267089
8 0.15185548 0.2915039 0.004394531 0.0703125 0.2915039 0.004394531 0.0703125 0.2890625 0.11694337 0.0727539
06 0.2890625 0.11694337 0.072753906 0.2890625 0.11694337 0.072753906 0.58032227 0.20654297 0.16845703 0.580
32227 0.20654297 0.16845703 0.58032227 0.20654297 0.16845703 0.8317871 0.20800781 0.18896484 1.2778322 0.30
639648 0.5510254 0.7670899 0.34057617 0.8496094 0.16235352 -0.4787598 0.26538086 0.44921875 -0.7165528 -0.5
3881836 0.689209 -1.6711427 -0.91284186 0.8618164 0.095947266 -2.5136719 0.8618164 0.095947266 -2.5136719
```

1.9758302 0.5515137 -2.2919922 4.548828 0.8195801 1.6223145 4.928711 0.5324707 1.5439453 1.4716797 0.456054
72 0.47949222 -0.59277344 0.22460938 0.09814453 0.9501954 0.22021484 0.8041992 0.9501954 0.22021484 0.80419
92 0.9050293 0.009277344 0.72143555 0.6845703 0.39501953 0.1381836 1.2570801 0.114990234 0.6828614 1.352539
2 0.54296875 0.72998047 0.86572266 -0.061523438 0.54418945 0.58154297 0.005371094 0.7531739 0.58154297 -0.1
3647461 0.5866699
Label: 1
Tokenized input_ids: [101, 1014, 1012, 6390, 23352, 24594, 2509, 1014, 1012, 27016, 20842, 2581, 2475, 101
4, 1012, 6988, 2581, 26187, 2620, 1014, 1012, 26271, 2581, 12521, 2683, 2475, 1014, 1012, 10715, 16068, 237
77, 2692, 2549, 1014, 1012, 21211, 2683, 2620, 23777, 2620, 1014, 1012, 26271, 2581, 12521, 2683, 2475, 101
4, 1012, 10715, 16068, 23777, 2692, 2549, 1014, 1012, 21211, 2683, 2620, 23777, 2620, 1014, 1012, 26271, 25
81, 12521, 2683, 2475, 1014, 1012, 10715, 16068, 23777, 2692, 2549, 1014, 1012, 21211, 2683, 2620, 23777, 2
620, 1014, 1012, 23297, 2581, 2683, 24594, 2581, 1014, 1012, 5718, 2692, 21486, 17788, 1014, 1012, 25797, 2
683, 2575, 2620, 23352, 1014, 1012, 23297, 2581, 2683, 24594, 2581, 1014, 1012, 5718, 2692, 21486, 17788, 1
014, 1012, 25797, 2683, 2575, 2620, 23352, 1014, 1012, 21611, 12740, 23777, 102]
Tokenized attention_mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Example 2
Text: 0.35571292 0.108154304 0.33398438 0.35571292 0.108154304 0.33398438 0.35571292 0.108154304 0.33398438
0.23779297 0.0703125 0.29296875 0.23779297 0.0703125 0.29296875 0.2644043 0.16650392 0.2529297 0.29858398
0.12670898 0.15185548 0.29858398 0.12670898 0.15185548 0.29858398 0.12670898 0.15185548 0.2915039 0.0043945
31 0.0703125 0.2915039 0.004394531 0.0703125 0.2890625 0.11694337 0.072753906 0.2890625 0.11694337 0.072753
906 0.2890625 0.11694337 0.072753906 0.58032227 0.20654297 0.16845703 0.58032227 0.20654297 0.16845703 0.58
032227 0.20654297 0.16845703 0.8317871 0.20800781 0.18896484 1.2778322 0.30639648 0.5510254 0.7670899 0.340
57617 0.8496094 0.16235352 -0.4787598 0.26538086 0.44921875 -0.7165528 -0.53881836 0.689209 -1.6711427 -0.9
1284186 0.8618164 0.095947266 -2.5136719 0.8618164 0.095947266 -2.5136719 1.9758302 0.5515137 -2.2919922 4.
548828 0.8195801 1.6223145 4.928711 0.5324707 1.5439453 1.4716797 0.45605472 0.47949222 -0.59277344 0.22460
938 0.09814453 0.9501954 0.22021484 0.8041992 0.9501954 0.22021484 0.8041992 0.9050293 0.009277344 0.721435
55 0.6845703 0.39501953 0.1381836 1.2570801 0.114990234 0.6828614 1.3525392 0.54296875 0.72998047 0.8657226
6 -0.061523438 0.54418945 0.58154297 0.005371094 0.7531739 0.58154297 -0.13647461 0.5866699 0.4157715 0.242
6758 0.6875
Label: 1
Tokenized input_ids: [101, 1014, 1012, 26271, 2581, 12521, 2683, 2475, 1014, 1012, 10715, 16068, 23777, 269
2, 2549, 1014, 1012, 21211, 2683, 2620, 23777, 2620, 1014, 1012, 26271, 2581, 12521, 2683, 2475, 1014, 101
2, 10715, 16068, 23777, 2692, 2549, 1014, 1012, 21211, 2683, 2620, 23777, 2620, 1014, 1012, 26271, 2581, 12
521, 2683, 2475, 1014, 1012, 10715, 16068, 23777, 2692, 2549, 1014, 1012, 21211, 2683, 2620, 23777, 2620, 1
014, 1012, 23297, 2581, 2683, 24594, 2581, 1014, 1012, 5718, 2692, 21486, 17788, 1014, 1012, 25797, 2683, 2
575, 2620, 23352, 1014, 1012, 23297, 2581, 2683, 24594, 2581, 1014, 1012, 5718, 2692, 21486, 17788, 1014, 1
012, 25797, 2683, 2575, 2620, 23352, 1014, 1012, 21611, 12740, 23777, 1014, 1012, 27676, 2692, 23499, 2475,
1014, 1012, 22898, 2683, 24594, 2581, 1014, 1012, 27240, 27814, 23499, 2620, 102]

```
Tokenized attention_mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

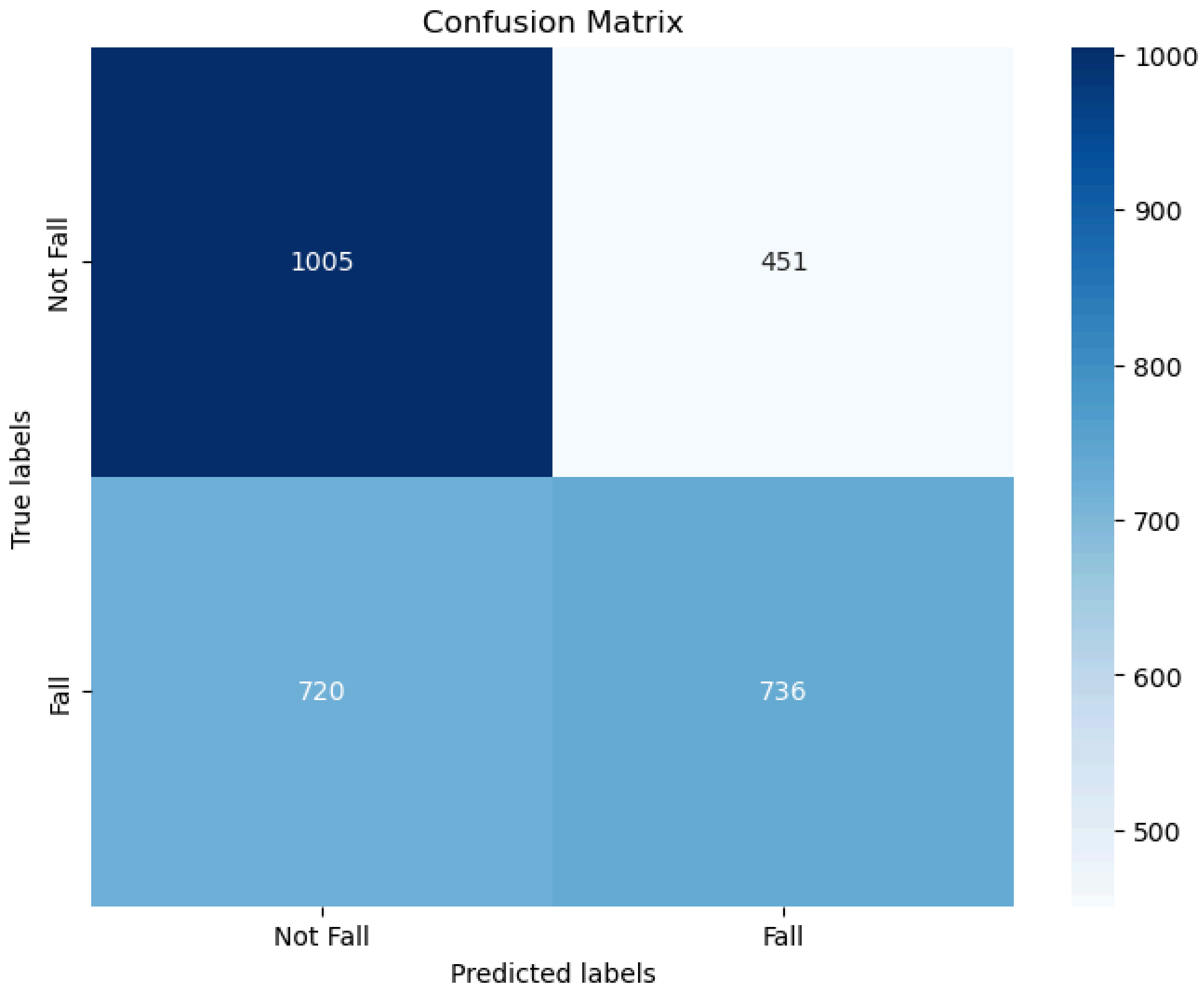
In [9]:

```python
# Predict on the test dataset
test_predictions = trainer.predict(test_dataset)
y_test_pred = np.argmax(test_predictions.predictions, axis=1)

# Ensure y_test and y_test_pred are numpy arrays
y_test = np.array(y_test)
y_test_pred = np.array(y_test_pred)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, average='weighted')
recall = recall_score(y_test, y_test_pred, average='weighted')
f1 = f1_score(y_test, y_test_pred, average='weighted')
roc_auc = roc_auc_score(y_test, test_predictions.predictions[:, 1], average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")

# Print classification report
report = classification_report(y_test, y_test_pred, target_names=['Not Fall', 'Fall'], digits=4)
print("\nClassification Report:\n", report)
```

```
Accuracy: 0.5979
Precision: 0.6013
Recall: 0.5979
F1 Score: 0.5944
ROC AUC: 0.6476

Classification Report:
              precision    recall  f1-score   support

    Not Fall     0.5826    0.6902    0.6319      1456
        Fall     0.6201    0.5055    0.5569      1456

    accuracy                         0.5979      2912
   macro avg     0.6013    0.5979    0.5944      2912
weighted avg     0.6013    0.5979    0.5944      2912
```

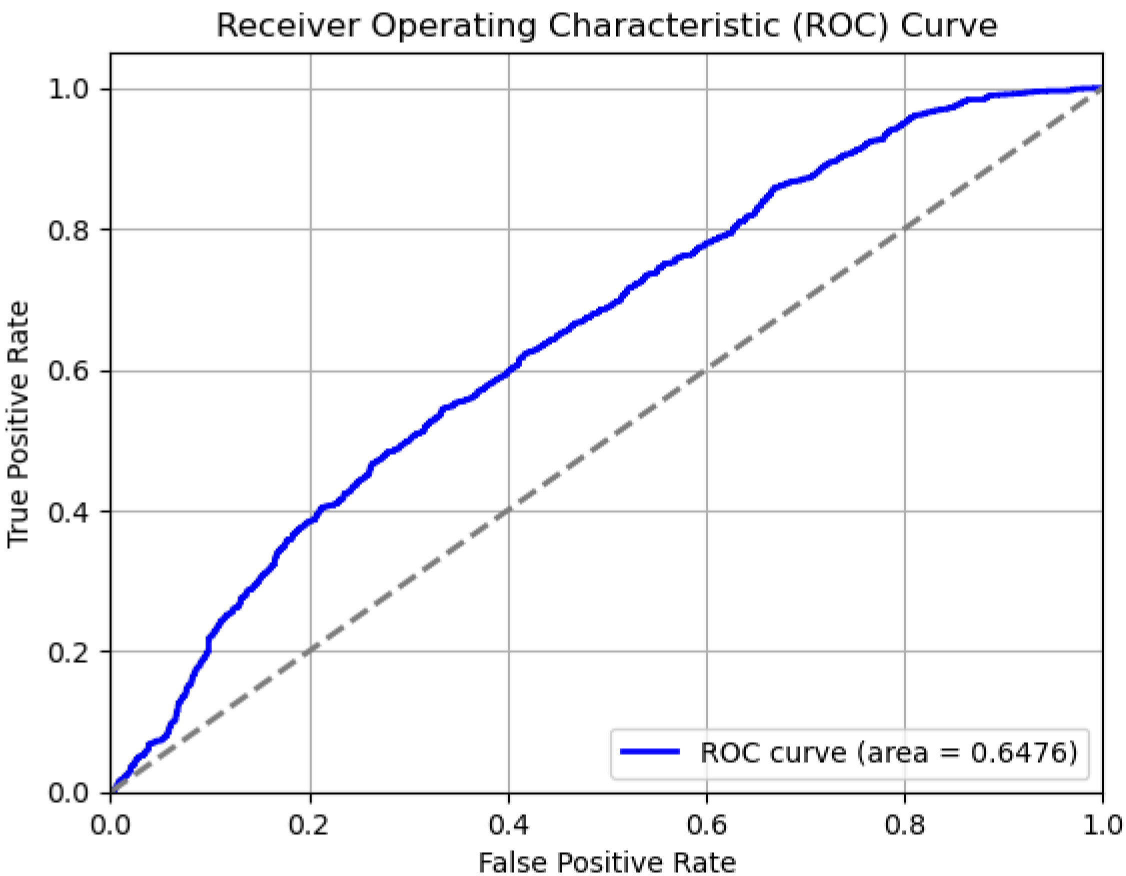
In [10]:

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_test_pred, labels=[0, 1])  # Assuming 0 is 'Not Fall' and 1 is 'Fall'

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Fall', 'Fall'], yticklabels=['Not Fal
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



In [11]:
```
# Plot ROC Curve for Test Set
# Compute ROC curve and ROC area for the test set
fpr, tpr, _ = roc_curve(y_test, test_predictions.predictions[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='grey', lw=2, 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()
```



In [ ]: