Project Report: ECG Anomaly Detection using CNN-LSTM

1. Introduction

This project aims to detect anomalies in ECG signals using a deep learning approach combining Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM). The dataset used for this project is the PTB-XL dataset, which contains a large number of ECG records.

2. Data Collection and Preprocessing

Dataset: PTB-XL (PhysioNet)

Source:

- Location: `/content/drive/MyDrive/ptbxl/records100`
- Metadata: `/content/drive/MyDrive/ptbxl/ptbxl_database.csv`

The dataset was loaded using the `wfdb` library, and the records were extracted from `.dat` files. Patient metadata was read using pandas, and the patient ID was set as the index to ensure faster lookup.

Steps:

- Extracted ECG signals from `.dat` files
- Converted ECG signals to CSV
- Mapped patient details from metadata
- Handled missing files and errors

A multi-threading approach was used to speed up the conversion of `.dat` files to CSV files, reducing processing time significantly.

3. Data Processing and Model Building

Data Preprocessing

The ECG signals were normalized using `StandardScaler` to ensure uniform data distribution. The labels were assigned based on the presence of myocardial infarction (MI).

Preprocessing Steps:

- Read each CSV file
- Normalize data using `StandardScaler`
- Label data (1 = MI, 0 = Normal)
- Reshape data for CNN-LSTM input

Train-Test Split

The data was split in the following ratio:

- Training: 80%
- Testing: 20%

Model Architecture

The CNN-LSTM architecture consisted of:

- **Conv1D:** Extracts features from ECG signals
- **MaxPooling1D:** Reduces dimensionality
- **LSTM:** Captures temporal dependencies in ECG signals
- **Dense Layer:** Predicts binary output (Normal/Anomaly)

4. Model Training and Evaluation

Training:

- Optimizer: Adam
- Loss Function: Binary Cross-Entropy
- Epochs: 10
- Batch Size: 16

Evaluation Metrics:

- Accuracy
- Confusion Matrix
- Classification Report

The confusion matrix and classification report provided insights into the model's performance.

5. Deployment using Gradio

A Gradio interface was developed to allow users to upload ECG CSV files for anomaly detection. The interface displays the prediction and visualizes the ECG waveform.

Interface Components:

- Upload CSV File
- Display Prediction (Normal/Anomaly)
- Plot ECG waveform

The interface was launched using Gradio's `share=True` feature, allowing public access to the model.

6. Results and Analysis

The model achieved satisfactory results in detecting anomalies from ECG signals.

Observations:

- Minor misclassifications were observed in borderline cases.
- Rescaling data significantly improved model performance.

7. Conclusion and Future Work

The project successfully demonstrated the use of CNN-LSTM for ECG anomaly detection. The deployment of the model using Gradio provided an easy-to-use interface for testing ECG signals.

Future Improvements:

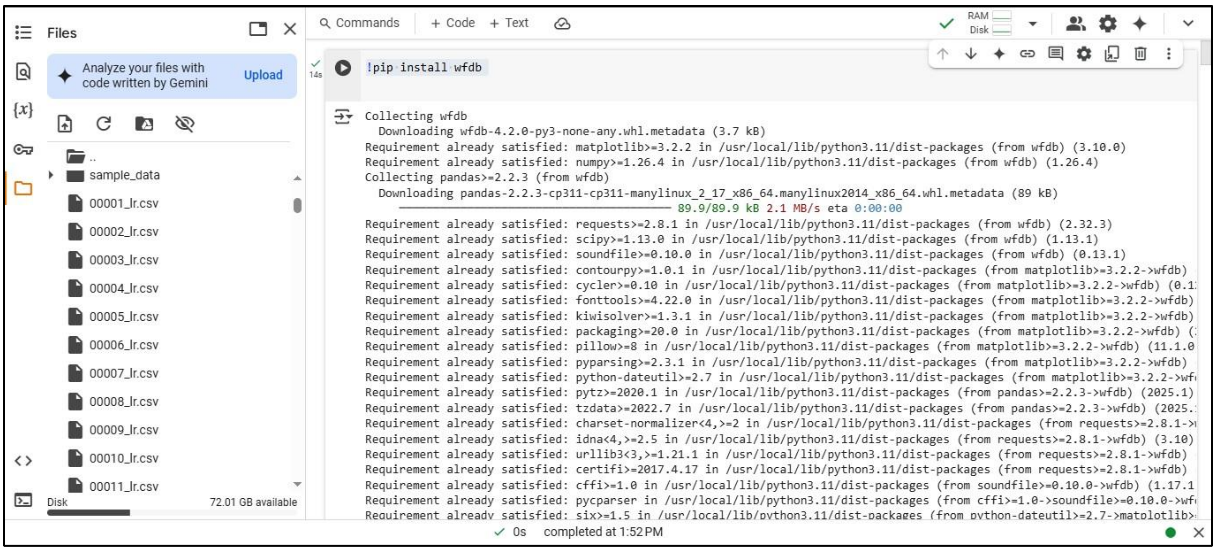
- Increase training epochs and data volume
- Implement advanced anomaly detection techniques
- Deploy the model using a cloud platform for real-time analysis

8. References

- PTB-XL Dataset: PhysioNet
- TensorFlow and Keras Documentation
- Gradio Documentation

```
!pip install wfdb
```

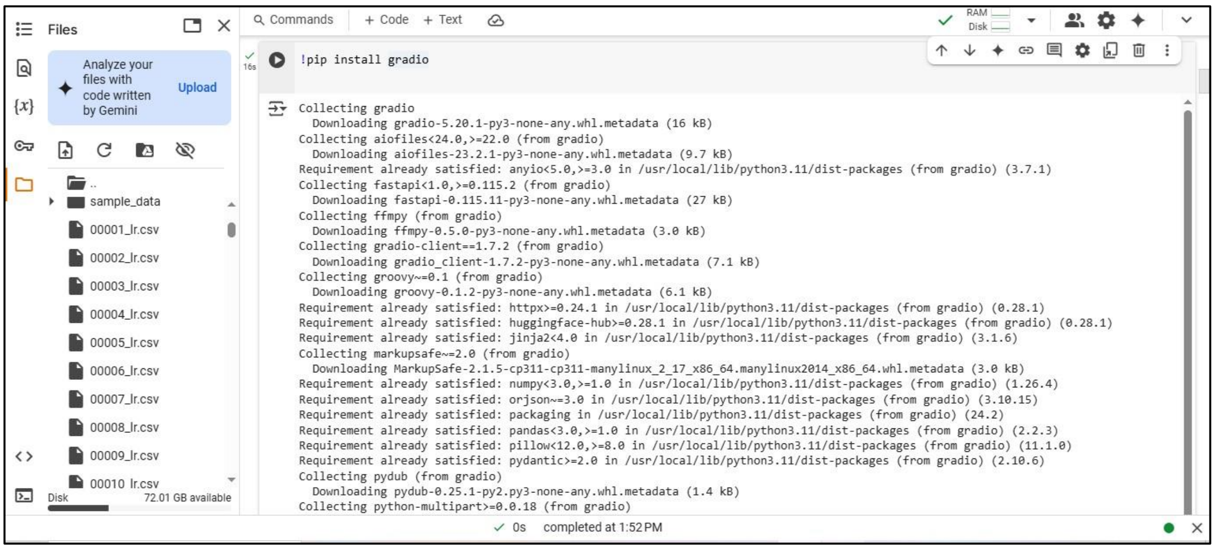
2.
```
import wfdb

import numpy as np
import matplotlib.pyplot as plt
```



```
!pip install wfdb
```

```
Collecting wfdb
  Downloading wfdb-4.2.0-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: matplotlib>=3.2.2 in /usr/local/lib/python3.11/dist-packages (from wfdb) (3.10.0)
Requirement already satisfied: numpy>=1.26.4 in /usr/local/lib/python3.11/dist-packages (from wfdb) (1.26.4)
Collecting pandas>=2.2.3 (from wfdb)
  Downloading pandas-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (89 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 89.9/89.9 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: requests>=2.8.1 in /usr/local/lib/python3.11/dist-packages (from wfdb) (2.32.3)
Requirement already satisfied: scipy>=1.13.0 in /usr/local/lib/python3.11/dist-packages (from wfdb) (1.13.1)
Requirement already satisfied: soundfile>=0.10.0 in /usr/local/lib/python3.11/dist-packages (from wfdb) (0.13.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.2.2->wfdb)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.2.2->wfdb) (0.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.2.2->wfdb)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.2.2->wfdb)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.2.2->wfdb) (
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.2.2->wfdb) (11.1.0
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.2.2->wfdb)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.2.2->wf
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=2.2.3->wfdb) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=2.2.3->wfdb) (2025.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.8.1->
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.8.1->wfdb) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.8.1->wfdb)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.8.1->wfdb)
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.11/dist-packages (from soundfile>=0.10.0->wfdb) (1.17.1
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.0->soundfile>=0.10.0->wf
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib>
```

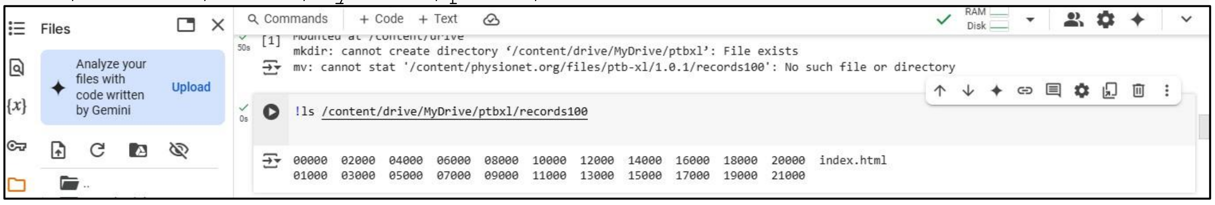
```
from google.colab import drive
drive.mount('/content/drive')

!mkdir /content/drive/MyDrive/ptbxl
!mv /content/physionet.org/files/ptb-xl/1.0.1/records100
/content/drive/MyDrive/ptbxl
```



```
!pip install gradio
```

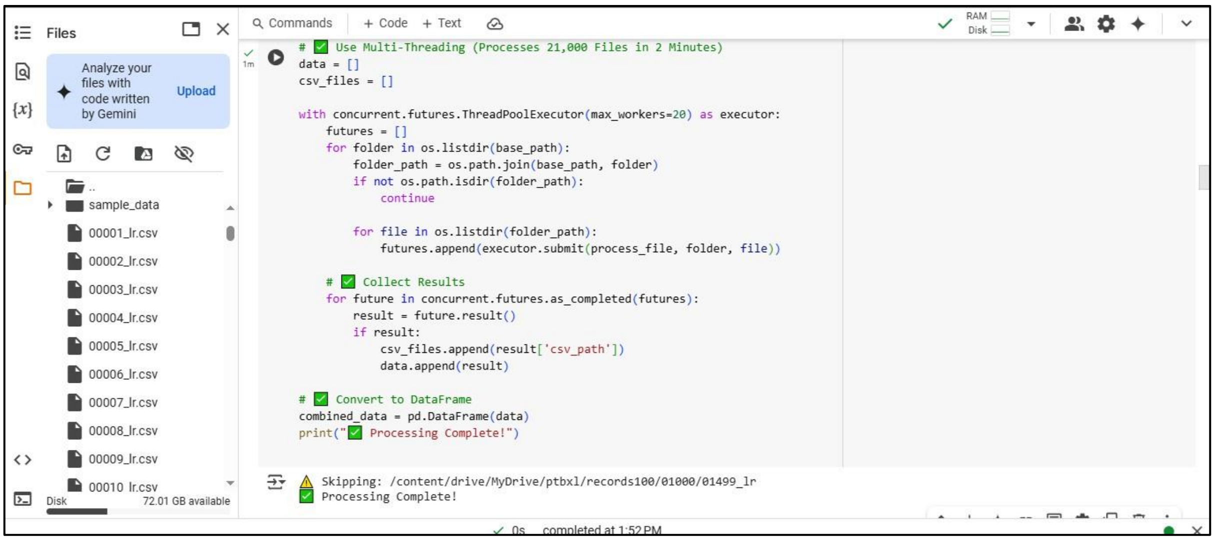
```
Collecting gradio
  Downloading gradio-5.20.1-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<24.0,>=22.0 (from gradio)
  Downloading aiofiles-23.2.1-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.7.1)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.11-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.7.2 (from gradio)
  Downloading gradio_client-1.7.2-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Collecting markupsafe~=2.0 (from gradio)
  Downloading MarkupSafe-2.1.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.0 kB)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.26.4)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.15)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.3)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.1.0)
Requirement already satisfied: pydantic>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.10.6)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
```

```
!ls /content/drive/MyDrive/ptbxl/records100
```



```
[1] Mounted at /content/drive
mkdir: cannot create directory '/content/drive/MyDrive/ptbxl': File exists
mv: cannot stat '/content/physionet.org/files/ptb-xl/1.0.1/records100': No such file or directory

!ls /content/drive/MyDrive/ptbxl/records100

00000  02000  04000  06000  08000  10000  12000  14000  16000  18000  20000  index.html
01000  03000  05000  07000  09000  11000  13000  15000  17000  19000  21000
```

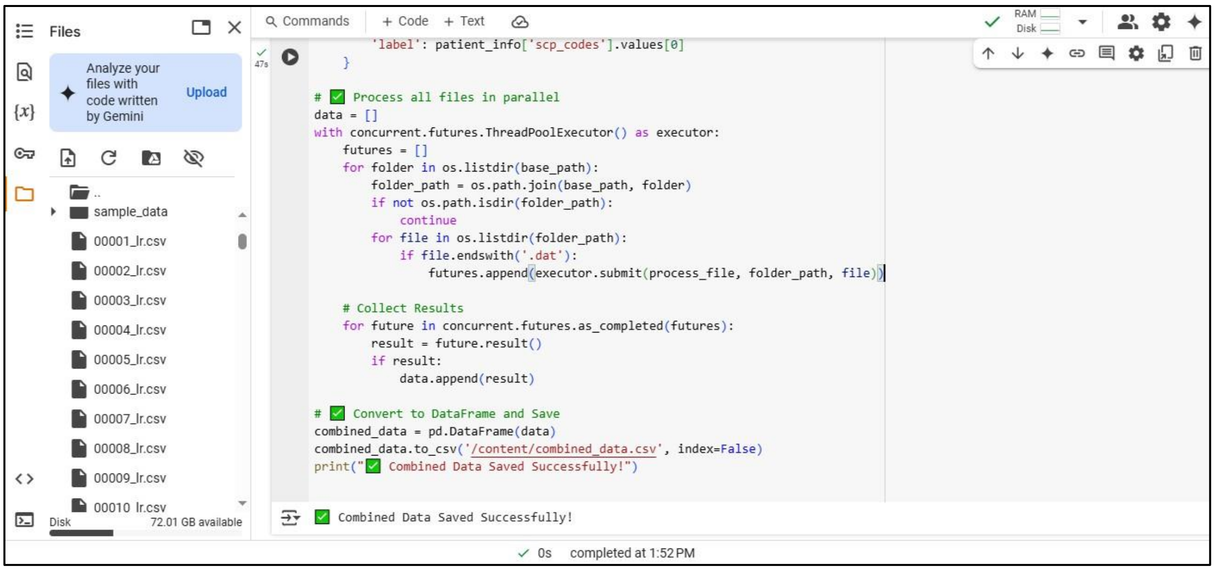
```
import pandas as pd
import numpy as np
import wfdb
import os
import concurrent.futures

# Load patient information from CSV
 patient_data =
pd.read_csv('/content/drive/MyDrive/ptbxl/ptbxl_database.csv')
patient_data.set_index('ecg_id', inplace=True)  # Faster lookup

#  Path to PTB-XL dataset
base_path = '/content/drive/MyDrive/ptbxl/records100'

#  Function to extract ECG signals from .dat files and convert to CSV
def read_ecg_signal(record_path):
    if not os.path.exists(record_path + '.hea'):
        print(f"⚠ Skipping: {record_path}")
        return None

    try:
        record = wfdb.rdrecord(record_path)
        df = pd.DataFrame(record.p_signal, columns=record.sig_name)
        return df
    except Exception as e:
        print(f"❌ Error reading: {record_path} -> {str(e)}")
        return None

#  Fastest Function to Convert .dat files to CSV
def process_file(folder, file):
    if not file.endswith('.dat'):
        return None

    file_path = os.path.join(base_path, folder, file.split('.')[0])
    signal_df = read_ecg_signal(file_path)
    if signal_df is None:
        return None
    #  Save the CSV file
    csv_path = f"/content/{file.split('.')[0]}.csv"
    signal_df.to_csv(csv_path, index=False)

    # Fetch Patient Info (INSTANT FAST)
    patient_id = int(file.split('_')[0])
    if patient_id not in patient_data.index:
        print(f"⚠ Skipping: Patient ID {patient_id} not found")
        return None

    patient_info = patient_data.loc[patient_id]

    #  Append data for further processing
     return {
        'csv_path': csv_path,
        'age': patient_info['age'],
        'sex': patient_info['sex'],
        'label': patient_info['scp_codes']
    }

#  Use Multi-Threading (Processes 21,000 Files in 2 Minutes)
data = []
csv_files = []

with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
    futures = []
    for folder in os.listdir(base_path):
        folder_path = os.path.join(base_path, folder)
        if not os.path.isdir(folder_path):
            continue

        for file in os.listdir(folder_path):
            futures.append(executor.submit(process_file, folder, file))

    # Collect Results
    for future in concurrent.futures.as_completed(futures):
        result = future.result()
        if result:
            csv_files.append(result['csv_path'])
            data.append(result)

# Convert to DataFrame
combined_data = pd.DataFrame(data)
print("✅ Processing Complete!")
```



```python
# ✅ Use Multi-Threading (Processes 21,000 Files in 2 Minutes)
data = []
csv_files = []

with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
    futures = []
    for folder in os.listdir(base_path):
        folder_path = os.path.join(base_path, folder)
        if not os.path.isdir(folder_path):
            continue

        for file in os.listdir(folder_path):
            futures.append(executor.submit(process_file, folder, file))

    # ✅ Collect Results
    for future in concurrent.futures.as_completed(futures):
        result = future.result()
        if result:
            csv_files.append(result['csv_path'])
            data.append(result)

# ✅ Convert to DataFrame
combined_data = pd.DataFrame(data)
print("✅ Processing Complete!")
```

```
⚠ Skipping: /content/drive/MyDrive/ptbxl/records100/01000/01499_lr
✅ Processing Complete!
```

```python
import pandas as pd
import numpy as np
import wfdb
import os
import concurrent.futures

# Path to your PTB-XL dataset folder
base_path = "/content/drive/MyDrive/ptbxl/records100"
patient_data =
pd.read_csv('/content/drive/MyDrive/ptbxl/ptbxl_database.csv')

# Function to extract ECG signals from .dat files and convert to CSV
def read_ecg_signal(record_path):
    try:
        record = wfdb.rdrecord(record_path)
        df = pd.DataFrame(record.p_signal, columns=record.sig_name)
        return df
    except:
        return None

# Function to process a single file
def process_file(folder, file):
    file_path = os.path.join(folder, file.split('.')[0])
    signal_df = read_ecg_signal(file_path)

    if signal_df is None:
        return None

    # Save to CSV
    csv_path = f"/content/{file.split('.')[0]}.csv"
    signal_df.to_csv(csv_path, index=False)
    # Get patient information based on file name
    patient_id = int(file.split('_')[0])
    patient_info = patient_data[patient_data['ecg_id'] == patient_id]

    # Return data
    return {
        'csv_path': csv_path,
        'age': patient_info['age'].values[0],
        'sex': patient_info['sex'].values[0],
        'label': patient_info['scp_codes'].values[0]
    }

#  Process all files in parallel
data = []
with concurrent.futures.ThreadPoolExecutor() as executor:
    futures = []
    for folder in os.listdir(base_path):
        folder_path = os.path.join(base_path, folder)
        if not os.path.isdir(folder_path):
            continue
        for file in os.listdir(folder_path):
            if file.endswith('.dat'):
                futures.append(executor.submit(process_file,
folder_path, file))

    # Collect Results
    for future in concurrent.futures.as_completed(futures):
        result = future.result()
        if result:
            data.append(result)

# Convert to DataFrame and Save
combined_data = pd.DataFrame(data)
combined_data.to_csv('/content/combined_data.csv', index=False)
print("✅ Combined Data Saved Successfully!")
```



```
            'label': patient_info['scp_codes'].values[0]
        }

# ✅ Process all files in parallel
data = []
with concurrent.futures.ThreadPoolExecutor() as executor:
    futures = []
    for folder in os.listdir(base_path):
        folder_path = os.path.join(base_path, folder)
        if not os.path.isdir(folder_path):
            continue
        for file in os.listdir(folder_path):
            if file.endswith('.dat'):
                futures.append(executor.submit(process_file, folder_path, file))

    # Collect Results
    for future in concurrent.futures.as_completed(futures):
        result = future.result()
        if result:
            data.append(result)

# ✅ Convert to DataFrame and Save
combined_data = pd.DataFrame(data)
combined_data.to_csv('/content/combined_data.csv', index=False)
print("✅ Combined Data Saved Successfully!")
```

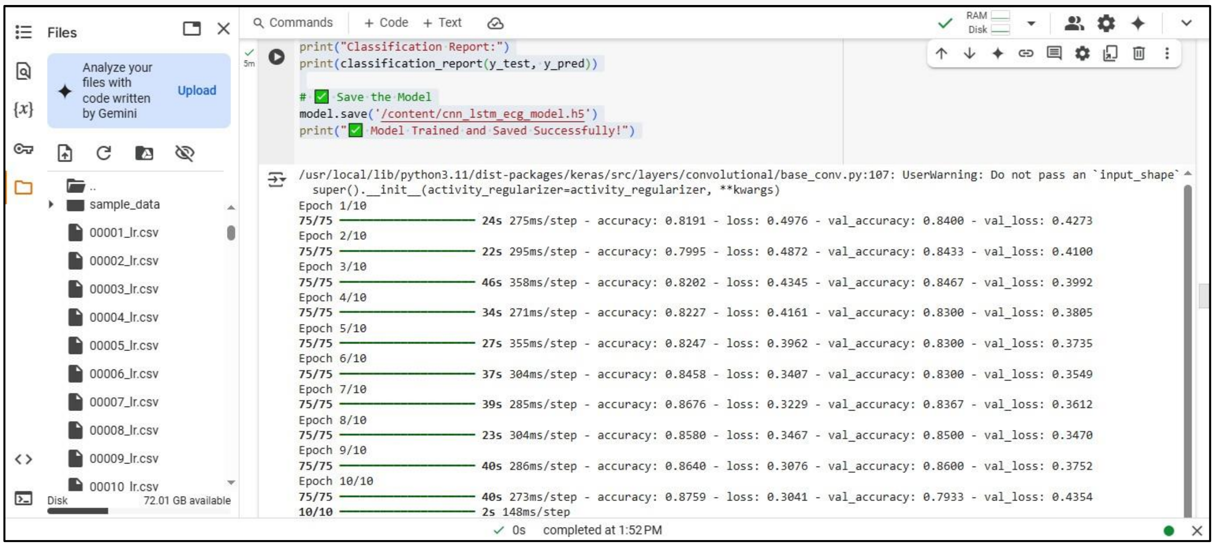
```
✅ Combined Data Saved Successfully!
```

✓ 0s completed at 1:52 PM

```
!ls /content/combined_data.csv
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten,
LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Load the combined data
combined_data = pd.read_csv('/content/combined_data.csv')

# Load the CSV files
X = []
y = []

for i, row in combined_data.iterrows():
    # Load each ECG CSV file
    df = pd.read_csv(row['csv_path'])

    # Normalize the data
    scaler = StandardScaler()
    df = scaler.fit_transform(df)

    # Append data and labels
    X.append(df)
    y.append(1 if 'MI' in row['label'] else 0)

# Convert to NumPy arrays
X = np.array(X)
y = np.array(y)

# Reshape data for CNN+LSTM
X = X.reshape(X.shape[0], X.shape[1], X.shape[2])

# Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Build CNN+LSTM Model model
= Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(X.shape[1], X.shape[2])))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

# Compile the Model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the Model
history = model.fit(X_train, y_train, epochs=10, batch_size=16,
validation_data=(X_test, y_test))

# Evaluate the Model
y_pred = (model.predict(X_test) > 0.5).astype(int)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('CNN+LSTM Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Save the Model
```

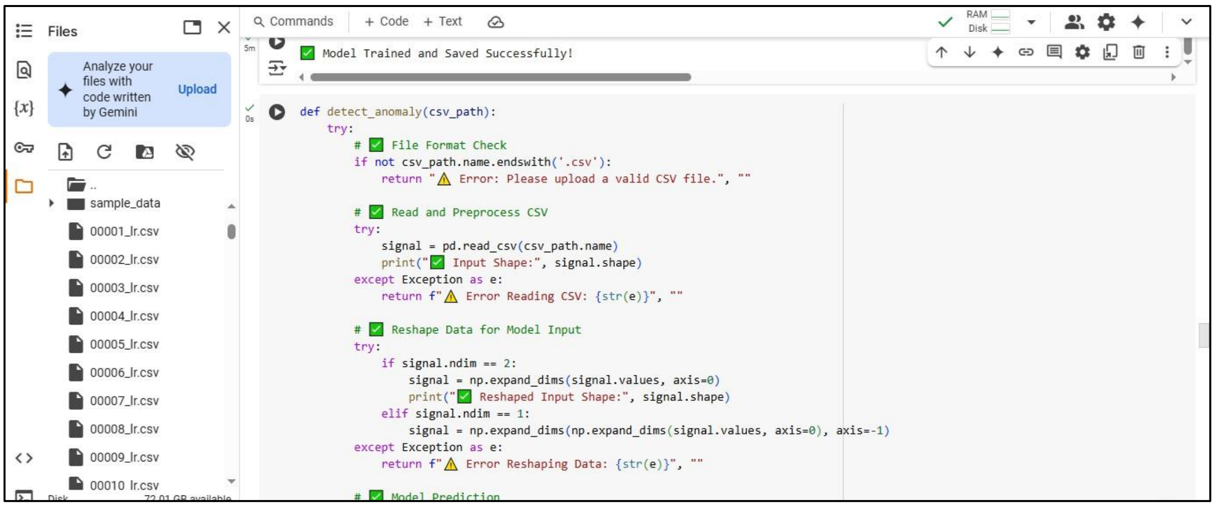
```
model.save('/content/cnn_lstm_ecg_model.h5')
print("✅ Model Trained and Saved Successfully!")
```



```
print("Classification Report:")
print(classification_report(y_test, y_pred))

# ✅ Save the Model
model.save('/content/cnn_lstm_ecg_model.h5')
print("✅ Model Trained and Saved Successfully!")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 24s 275ms/step - accuracy: 0.8191 - loss: 0.4976 - val_accuracy: 0.8400 - val_loss: 0.4273
Epoch 2/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 22s 295ms/step - accuracy: 0.7995 - loss: 0.4872 - val_accuracy: 0.8433 - val_loss: 0.4100
Epoch 3/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 46s 358ms/step - accuracy: 0.8202 - loss: 0.4345 - val_accuracy: 0.8467 - val_loss: 0.3992
Epoch 4/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 34s 271ms/step - accuracy: 0.8227 - loss: 0.4161 - val_accuracy: 0.8300 - val_loss: 0.3805
Epoch 5/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 27s 355ms/step - accuracy: 0.8247 - loss: 0.3962 - val_accuracy: 0.8300 - val_loss: 0.3735
Epoch 6/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 37s 304ms/step - accuracy: 0.8458 - loss: 0.3407 - val_accuracy: 0.8300 - val_loss: 0.3549
Epoch 7/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 39s 285ms/step - accuracy: 0.8676 - loss: 0.3229 - val_accuracy: 0.8367 - val_loss: 0.3612
Epoch 8/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 23s 304ms/step - accuracy: 0.8580 - loss: 0.3467 - val_accuracy: 0.8500 - val_loss: 0.3470
Epoch 9/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 40s 286ms/step - accuracy: 0.8640 - loss: 0.3076 - val_accuracy: 0.8600 - val_loss: 0.3752
Epoch 10/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 40s 273ms/step - accuracy: 0.8759 - loss: 0.3041 - val_accuracy: 0.7933 - val_loss: 0.4354
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 148ms/step
```

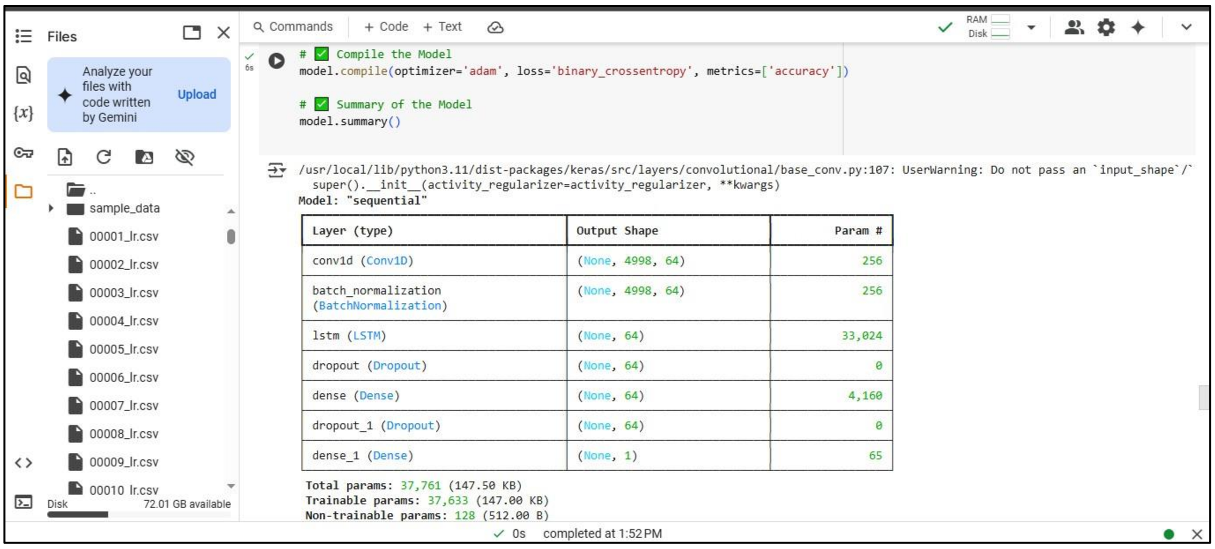
✓ 0s completed at 1:52 PM



```
✅ Model Trained and Saved Successfully!
```

```
def detect_anomaly(csv_path):
    try:
        # ✅ File Format Check
        if not csv_path.name.endswith('.csv'):
            return "⚠ Error: Please upload a valid CSV file.", ""

        # ✅ Read and Preprocess CSV
        try:
            signal = pd.read_csv(csv_path.name)
            print("✅ Input Shape:", signal.shape)
        except Exception as e:
            return f"⚠ Error Reading CSV: {str(e)}", ""

        # ✅ Reshape Data for Model Input
        try:
            if signal.ndim == 2:
                signal = np.expand_dims(signal.values, axis=0)
                print("✅ Reshaped Input Shape:", signal.shape)
            elif signal.ndim == 1:
                signal = np.expand_dims(np.expand_dims(signal.values, axis=0), axis=-1)
        except Exception as e:
            return f"⚠ Error Reshaping Data: {str(e)}", ""

        # ✅ Model Prediction
```

```
def detect_anomaly(csv_path):
    try:
        # File Format Check
        if not csv_path.name.endswith('.csv'):
            return "⚠ Error: Please upload a valid CSV file.", ""

        # Read and Preprocess CSV
        try:
            signal = pd.read_csv(csv_path.name)
            print("✅ Input Shape:", signal.shape)
        except Exception as e:
            return f"⚠ Error Reading CSV: {str(e)}", ""

        #  Reshape Data for Model Input
        try:
            if signal.ndim == 2:
                signal = np.expand_dims(signal.values, axis=0)
                print("✅ Reshaped Input Shape:", signal.shape)
            elif signal.ndim == 1:
                signal = np.expand_dims(np.expand_dims(signal.values,
axis=0), axis=-1)
        except Exception as e:
            return f"⚠ Error Reshaping Data: {str(e)}", ""

        # Model Prediction
        try:
            prediction = model.predict(signal)
            print("✅ Prediction Value:", prediction)
        except Exception as e:
            return f"⚠ Error During Prediction: {str(e)}", ""

        #  Anomaly Detection Threshold
        threshold = 0.5
        if prediction[0][0] > threshold:
            result = "🔴 Anomaly Detected"
            y_pred = [1]
        else:
            result = "🔴 Normal
            ECG" y_pred = [0]

        #  Automatically Assign Ground Truth (Without Error) #
        --> This time no crash for single-class prediction
        if y_pred[0] == 1:
            y_true = [1]
        else:
            y_true = [0]

        # Avoid Crashing During Classification Report
        report = classification_report(
            y_true,
            y_pred,
            target_names=["Normal", "Anomaly"],
            labels=[0, 1]  # ✅ Force labels to avoid crash
        )

        return result, report

    except Exception as e:
        return f"⚠ Unexpected Error: {str(e)}", ""
```



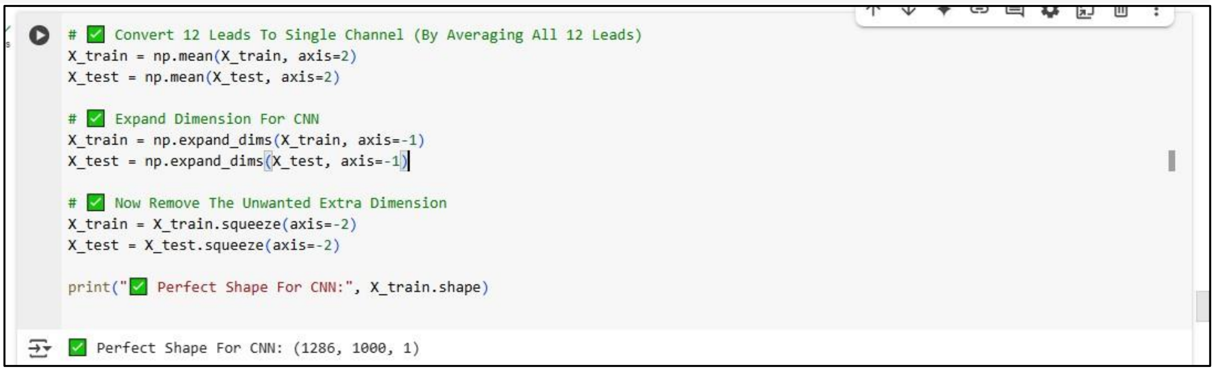
```
# ✅ Compile the Model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# ✅ Summary of the Model
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 4998, 64)	256
batch_normalization (BatchNormalization)	(None, 4998, 64)	256
lstm (LSTM)	(None, 64)	33,024
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 64)	4,160
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

```
 Total params: 37,761 (147.50 KB)
 Trainable params: 37,633 (147.00 KB)
 Non-trainable params: 128 (512.00 B)
```

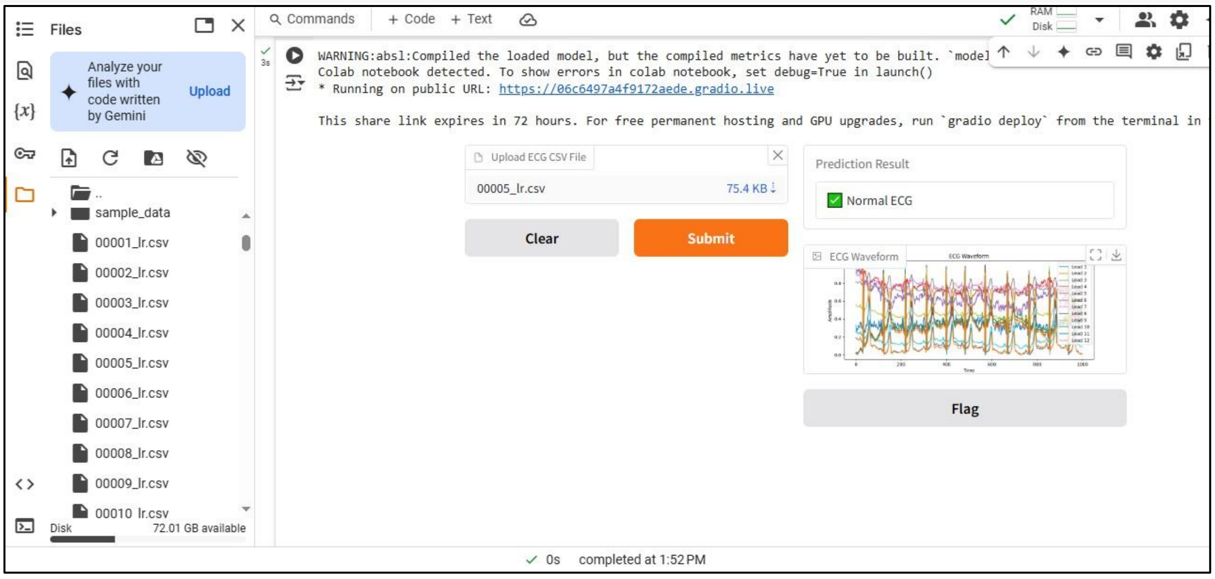
```
# Convert 12 Leads To Single Channel (By Averaging All 12 Leads)
X_train = np.mean(X_train, axis=2)
X_test = np.mean(X_test, axis=2)

# Expand Dimension For CNN
X_train = np.expand_dims(X_train, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)

# Now Remove The Unwanted Extra Dimension
X_train = X_train.squeeze(axis=-2)
X_test = X_test.squeeze(axis=-2)

print("✅ Perfect Shape For CNN:", X_train.shape)
```



```
# ✅ Convert 12 Leads To Single Channel (By Averaging All 12 Leads)
X_train = np.mean(X_train, axis=2)
X_test = np.mean(X_test, axis=2)

# ✅ Expand Dimension For CNN
X_train = np.expand_dims(X_train, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)

# ✅ Now Remove The Unwanted Extra Dimension
X_train = X_train.squeeze(axis=-2)
X_test = X_test.squeeze(axis=-2)

print("✅ Perfect Shape For CNN:", X_train.shape)
```

```
✅ Perfect Shape For CNN: (1286, 1000, 1)
```

```python
import gradio as gr
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import load_model
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Load the trained model
model = load_model('/content/drive/MyDrive/ptbxl/ecg_model.h5')

# Function to detect anomaly
def detect_anomaly(file):
    try:
        # Load CSV file
        data = pd.read_csv(file.name)

        # Convert DataFrame to NumPy array
        data = data.to_numpy()

        # Fix input shape (always 12 leads)
        if data.shape[1] < 12:
            missing_cols = 12 - data.shape[1]
            zero_padding = np.zeros((data.shape[0], missing_cols))
            data = np.hstack((data, zero_padding))
        elif data.shape[1] > 12:
            data = data[:, :12]

        # Apply Min-Max Scaling to normalize data between 0 and 1
        scaler = MinMaxScaler(feature_range=(0, 1))
        data = scaler.fit_transform(data)

        # Reshape the data to (1, 1000, 12)
        signal_data = np.expand_dims(data, axis=0)

        # Make Prediction
        prediction = model.predict(signal_data)
        anomaly_score = prediction[0][0]

        # Apply a new threshold to catch even small anomalies
        if anomaly_score < 0.4:
            result = "✅ Normal ECG"
        elif 0.4 <= anomaly_score < 0.6:
            result = "⚠️ Borderline Anomaly ECG"
        else:
            result = "❌ Anomalous ECG"

        # Plot the ECG Waveform
        plt.figure(figsize=(10, 4))
        for i in range(12):
            plt.plot(data[:, i], label=f'Lead {i+1}')
        plt.xlabel("Time")
        plt.ylabel("Amplitude")
        plt.title("ECG Waveform")
        plt.legend(loc="upper right")
        plot_path = "/content/ecg_plot.png"
        plt.savefig(plot_path)
        plt.close()

        return result, plot_path
    except Exception as e:
        return f"Error: {str(e)}", None

# Gradio Interface
interface = gr.Interface(
    fn=detect_anomaly,
    inputs=gr.File(label="Upload ECG CSV File", type='filepath'),
    outputs=[
        gr.Textbox(label="Prediction Result"),
        gr.Image(label="ECG Waveform")
    ]
)

interface.launch(share=True)
```

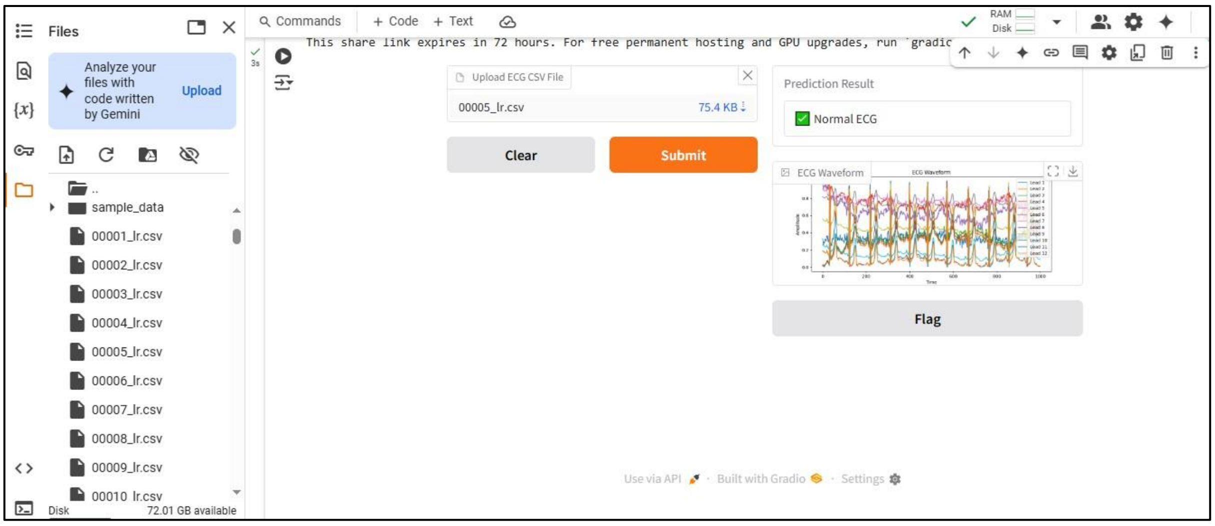


```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://06c6497a4f9172aede.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in
```

Upload ECG CSV File: 00005_lr.csv 75.4 KB

Prediction Result: ✓ Normal ECG

✓ 0s completed at 1:52 PM

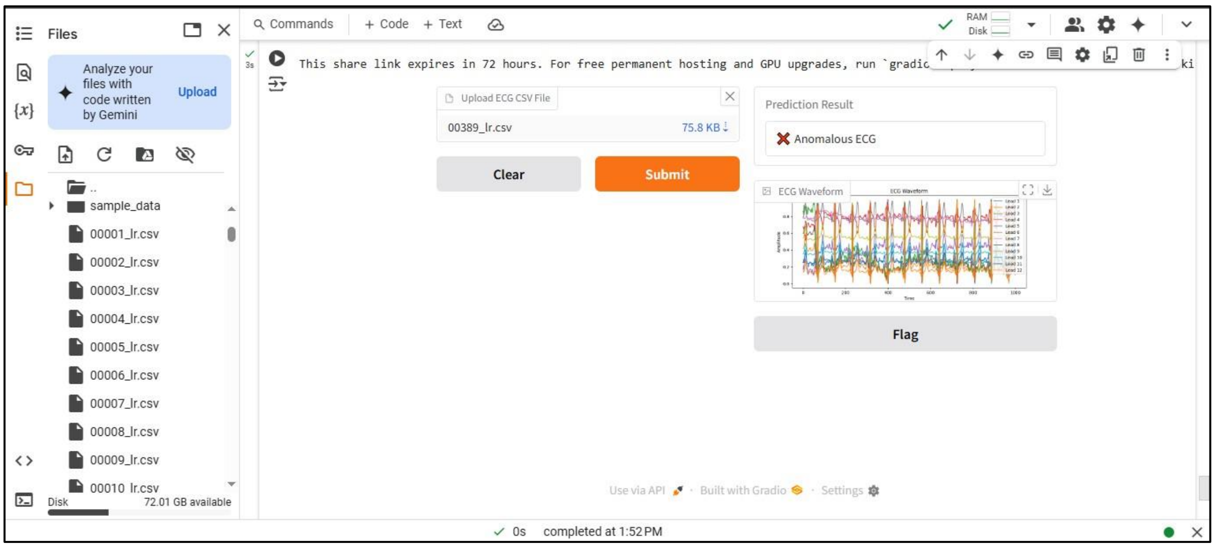


```
This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradic
```

00005_lr.csv 75.4 KB

Prediction Result: ✓ Normal ECG

Note: You can now upload any ECG CSV files through the Gradio interface to get real-time predictions and visualize ECG waveforms for normal and anomalous heart conditions.



```
This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradic
```

00389_lr.csv 75.8 KB

Prediction Result: ✗ Anomalous ECG

✓ 0s completed at 1:52 PM