```
!pip install pycbc
!pip install h5py
!pip install gwpy
!pip install tensorflow
!pip install sklearn
!pip install matplotlib
!pip install scipy
```

```
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.3
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0.
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorf
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.1
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,>
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflo
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0-
Collecting sklearn
  Downloading sklearn-0.0.post12.tar.gz (2.6 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (1.14.1)
Requirement already satisfied: numpy<2.3,>=1.23.5 in /usr/local/lib/python3.11/dist-packages (from scipy) (2.0.2)
```

```
!wget https://gwosc.org/eventapi/html/GWTC-1-confident/GW170817/v3/H-H1_GWOSC_4KHZ_R1-1187006835-4096.hdf5
!wget https://gwosc.org/eventapi/html/GWTC-1-confident/GW170817/v3/L-L1_GWOSC_4KHZ_R1-1187006835-4096.hdf5
```

```
--2025-04-13 21:57:11--  https://gwosc.org/eventapi/html/GWTC-1-confident/GW170817/v3/H-H1_GWOSC_4KHZ_R1-1187006835-4096.hdf5
Resolving gwosc.org (gwosc.org)... 131.215.113.72
Connecting to gwosc.org (gwosc.org)|131.215.113.72|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 129346425 (123M) [application/octet-stream]
Saving to: ‘H-H1_GWOSC_4KHZ_R1-1187006835-4096.hdf5’

H-H1_GWOSC_4KHZ_R1- 100%[===================>] 123.35M  57.2MB/s    in 2.2s

2025-04-13 21:57:14 (57.2 MB/s) - ‘H-H1_GWOSC_4KHZ_R1-1187006835-4096.hdf5’ saved [129346425/129346425]
```

```
--2025-04-13 21:57:14--  https://gwosc.org/eventapi/html/GWTC-1-confident/GW170817/v3/L-L1_GWOSC_4KHZ_R1-1187006835-4096.hdf5
Resolving gwosc.org (gwosc.org)... 131.215.113.72
Connecting to gwosc.org (gwosc.org)|131.215.113.72|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 129727892 (124M) [application/octet-stream]
Saving to: ‘L-L1_GWOSC_4KHZ_R1-1187006835-4096.hdf5’

L-L1_GWOSC_4KHZ_R1- 100%[===================>] 123.72M  53.5MB/s    in 2.3s

2025-04-13 21:57:16 (53.5 MB/s) - ‘L-L1_GWOSC_4KHZ_R1-1187006835-4096.hdf5’ saved [129727892/129727892]
```

```
#########FINAAAALLLLLL WORKINGGGG####3
import numpy as np
import pycbc
from pycbc.waveform import get_td_waveform
from pycbc.noise.gaussian import noise_from_psd
from pycbc.psd import aLIGOZeroDetHighPower
from pycbc.filter import highpass
from gwpy.timeseries import TimeSeries
import h5py
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_recall_fscore_support, confusion_matrix
import matplotlib.pyplot as plt
import os
import warnings
warnings.filterwarnings('ignore')
warnings.filterwarnings("ignore", message="Wswiglal-redir-stdio")

# Set random seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# 1. Simulate Gravitational Wave Data with PyCBC
def generate_bbh_data(num_samples, sample_rate=4096, duration=8):  # Updated default sample rate
    signals, labels = [], []
    target_length = int(duration * sample_rate)
    for i in range(num_samples):
        print(f"Generating BBH sample {i+1}/{num_samples}")
        # Binary Black Hole parameters
        mass1, mass2 = np.random.uniform(20, 40), np.random.uniform(20, 40)
        distance = np.random.uniform(10, 50)
        hp, _ = get_td_waveform(approximant="IMRPhenomPv2",
                                mass1=mass1, mass2=mass2,
                                delta_t=1.0/sample_rate,
                                f_lower=20,
                                distance=distance,
                                inclination=np.random.uniform(0, np.pi),
                                spin1z=np.random.uniform(-0.5, 0.5),
                                spin2z=np.random.uniform(-0.5, 0.5))
        hp *= 1e21
        hp.start_time = 0
        ts = pycbc.types.TimeSeries(hp, delta_t=1.0/sample_rate)
        current_length = len(ts)

        # Ensure consistent length
        if current_length < target_length:
            padding = np.zeros(target_length - current_length, dtype=np.float64)
            padded_signal = np.concatenate([ts.numpy(), padding]).astype(np.float64)
            ts = pycbc.types.TimeSeries(padded_signal, delta_t=1.0/sample_rate)
        elif current_length > target_length:
            ts = ts[:target_length]

        signal_with_noise = inject_noise(ts, sample_rate, duration)
        if signal_with_noise is None:
            print(f"Failed to generate signal for BBH sample {i}, retrying...")
            i -= 1  # Retry this sample
            continue

        signals.append(signal_with_noise)
        labels.append('BBH')

    return signals, labels

def generate_bns_data(num_samples, sample_rate=4096, duration=8):  # Updated default sample rate
```

```
    def generate_bns_data(num_samples, sample_rate=4096, duration=8):  # Updated default sample rate
        signals, labels = [], []
        target_length = int(duration * sample_rate)
        for i in range(num_samples):
            print(f"Generating BNS sample {i+1}/{num_samples}")
            # Binary Neutron Star parameters
            mass1, mass2 = np.random.uniform(1.2, 1.6), np.random.uniform(1.2, 1.6)
            distance = np.random.uniform(50, 100)
            hp, _ = get_td_waveform(approximant="TaylorF2",
                                    mass1=mass1, mass2=mass2,
                                    delta_t=1.0/sample_rate,
                                    f_lower=15,
                                    distance=distance,
                                    inclination=np.random.uniform(0, np.pi),
                                    spin1z=np.random.uniform(-0.1, 0.1),
                                    spin2z=np.random.uniform(-0.1, 0.1))
            hp *= 1e21
            hp.start_time = 0
            ts = pycbc.types.TimeSeries(hp, delta_t=1.0/sample_rate)
            current_length = len(ts)

            # Ensure consistent length
            if current_length < target_length:
                padding = np.zeros(target_length - current_length, dtype=np.float64)
                padded_signal = np.concatenate([ts.numpy(), padding]).astype(np.float64)
                ts = pycbc.types.TimeSeries(padded_signal, delta_t=1.0/sample_rate)
            elif current_length > target_length:
                ts = ts[:target_length]

            signal_with_noise = inject_noise(ts, sample_rate, duration)
            if signal_with_noise is None:
                print(f"Failed to generate signal for BNS sample {i}, retrying...")
                i -= 1  # Retry this sample
                continue

            signals.append(signal_with_noise)
            labels.append('BNS')

        return signals, labels

    def generate_nsbh_data(num_samples, sample_rate=4096, duration=8):  # Updated default sample rate
        signals, labels = [], []
        target_length = int(duration * sample_rate)
        for i in range(num_samples):
            print(f"Generating NSBH sample {i+1}/{num_samples}")
            # Neutron Star-Black Hole parameters
            mass1, mass2 = np.random.uniform(20, 40), np.random.uniform(1.2, 1.6)
            distance = np.random.uniform(10, 50)
            hp, _ = get_td_waveform(approximant="IMRPhenomPv2",
                                    mass1=mass1, mass2=mass2,
                                    delta_t=1.0/sample_rate,
                                    f_lower=20,
                                    distance=distance,
                                    inclination=np.random.uniform(0, np.pi),
                                    spin1z=np.random.uniform(-0.5, 0.5),
                                    spin2z=np.random.uniform(-0.5, 0.5))
            hp *= 1e21
            hp.start_time = 0
            ts = pycbc.types.TimeSeries(hp, delta_t=1.0/sample_rate)
            current_length = len(ts)

            # Ensure consistent length
            if current_length < target_length:
                padding = np.zeros(target_length - current_length, dtype=np.float64)
                padded_signal = np.concatenate([ts.numpy(), padding]).astype(np.float64)
                ts = pycbc.types.TimeSeries(padded_signal, delta_t=1.0/sample_rate)
            elif current_length > target_length:
                ts = ts[:target_length]

            signal_with_noise = inject_noise(ts, sample_rate, duration)
            if signal_with_noise is None:
                print(f"Failed to generate signal for NSBH sample {i}, retrying...")
                i -= 1  # Retry this sample
                continue

            signals.append(signal_with_noise)
            labels.append('NSBH')
```

```
        return signals, labels

    def inject_noise(signal, sample_rate=4096, duration=8):  # Updated default sample rate
        try:
            # Create PSD for advanced LIGO
            flen = int(sample_rate * duration // 2) + 1
            delta_f = 1.0 / duration
            psd = aLIGOZeroDetHighPower(flen, delta_f, low_freq_cutoff=20)

            # Generate colored noise
            noise = noise_from_psd(length=int(sample_rate * duration),
                                   delta_t=1.0/sample_rate,
                                   psd=psd)

            # Scale noise (SNR control)
            noise *= 1.5  # Reduced noise amplitude for better signal visibility
            noise = noise[:len(signal)]
            noise.start_time = 0

            # Check for invalid values
            noise_data = noise.numpy()
            if np.any(np.isnan(noise_data)) or np.any(np.isinf(noise_data)):
                print("Generated noise contains NaN or Inf values")
                return None

            # Combine signal and noise
            combined = noise + signal
            combined_data = combined.numpy()
            if np.any(np.isnan(combined_data)) or np.any(np.isinf(combined_data)):
                print("Combined signal contains NaN or Inf values")
                return None

            # Apply high-pass filter
            combined = highpass(combined, 15.0, 8)
            combined_data = combined.numpy()
            if np.any(np.isnan(combined_data)) or np.any(np.isinf(combined_data)):
                print("Signal after highpass contains NaN or Inf values")
                return None

            # Whiten the signal
            ts_gwpy = TimeSeries(combined.numpy(), sample_rate=sample_rate)
            whitened_gwpy = ts_gwpy.whiten(fftlength=4, overlap=2, fduration=4)
            if whitened_gwpy is None:
                print("GWpy whitening returned None")
                return None

            return whitened_gwpy

        except Exception as e:
            print(f"Error in inject_noise: {e}")
            return None

    def generate_spectrogram(data, qrange=(4, 64), frange=(20, 2000)):  # Increased frequency range for 4096 Hz sample rate
        """
        Generate Q-transform spectrogram with improved parameters for better signal visibility
        """
        if data is None:
            return None
        try:
            if np.any(np.isnan(data.value)) or np.any(np.isinf(data.value)):
                print("Input to Q-transform contains NaN or Inf values")
                return None

            # Use higher q-range for better frequency resolution
            qspec = data.q_transform(qrange=qrange, frange=frange, gps=None, outseg=None, whiten=False)
            return qspec
        except Exception as e:
            print(f"Error generating Q-transform: {e}")
            return None

    def preprocess_data(signals, labels, sample_rate=4096, visualize_samples=3):  # Updated default sample rate
        """
        Preprocess data with improved normalization and visualization
        """
        X, y = [], []
        signal_count = {'BBH': 0, 'BNS': 0, 'NSBH': 0}  # For tracking visualized samples by type

        for i, (signal, label) in enumerate(zip(signals, labels)):
```

```
        for i, (signal, label) in enumerate(zip(signals, labels)):
            if signal is None:
                print(f"Signal {label} at index {i} is None, skipping")
                continue

            print(f"Processing signal {label} at index {i}")

            # Generate Q-transform spectrogram
            spec = generate_spectrogram(signal)
            if spec is None:
                print(f"Failed to generate Q-transform for {label} at index {i}")
                continue

            spec_data = spec.value

            # Apply log scaling for better dynamic range
            spec_data = np.maximum(spec_data, 1e-5)  # Avoid log(0)
            spec_data = np.log10(spec_data)

            # Check for invalid values
            if np.any(np.isnan(spec_data)) or np.any(np.isinf(spec_data)):
                print(f"Invalid values in Q-transform for {label} at index {i}")
                continue

            # Visualize a few samples of each class
            if signal_count[label] < visualize_samples:
                plt.figure(figsize=(6, 4))
                plt.imshow(spec_data, aspect='auto', origin='lower',
                           extent=[0, spec.dt.value * spec.shape[1],
                                   spec.f0.value, spec.f0.value + spec.df.value * spec.shape[0]])
                plt.colorbar(label='Log Energy')
                plt.title(f"Q-transform Spectrogram - {label}")
                plt.xlabel("Time (s)")
                plt.ylabel("Frequency (Hz)")
                plt.tight_layout()
                plt.show()
                signal_count[label] += 1

            # Normalize using z-score
            mean, std = np.mean(spec_data), np.std(spec_data)
            if std < 1e-10:  # Avoid division by zero
                print(f"Near-zero standard deviation for {label} at index {i}. Using std=1.")
                std = 1.0

            spec_data = (spec_data - mean) / std

            # Resize to consistent dimensions
            from scipy.ndimage import zoom
            target_shape = (128, 128)
            spec_resized = zoom(spec_data, (target_shape[0]/spec_data.shape[0],
                                            target_shape[1]/spec_data.shape[1]))

            X.append(spec_resized)
            y.append(label)

        if len(X) == 0:
            print("No data successfully preprocessed.")
            return None, None, None

        # Convert to numpy arrays
        X = np.array(X)[..., np.newaxis]

        # Encode labels
        encoder = LabelEncoder()
        y_encoded = encoder.fit_transform(y)

        # Print label encoding mapping for verification
        print("Encoded labels:", np.unique(y_encoded))
        print("Label mapping:", dict(zip(encoder.classes_, range(len(encoder.classes_)))))

        return X, y_encoded, encoder

    def build_cnn_model(input_shape=(128, 128, 1), num_classes=3):
        """
        Build a CNN model with reduced complexity to prevent overfitting
        """
        model = models.Sequential([
            # First convolutional block
```

```
            layers.Conv2D(16, (3, 3), activation='relu', padding='same', input_shape=input_shape,
                          kernel_regularizer=tf.keras.regularizers.l2(0.001)),
            layers.BatchNormalization(),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.2),

            # Second convolutional block
            layers.Conv2D(32, (3, 3), activation='relu', padding='same',
                          kernel_regularizer=tf.keras.regularizers.l2(0.001)),
            layers.BatchNormalization(),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.3),

            # Third convolutional block
            layers.Conv2D(64, (3, 3), activation='relu', padding='same',
                          kernel_regularizer=tf.keras.regularizers.l2(0.001)),
            layers.BatchNormalization(),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.4),

            # Flatten and dense layers
            layers.Flatten(),
            layers.Dense(64, activation='relu',
                         kernel_regularizer=tf.keras.regularizers.l2(0.001)),
            layers.BatchNormalization(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation='softmax')
        ])

        # Use a slower learning rate for better convergence
        optimizer = tf.keras.optimizers.Adam(learning_rate=5e-4)

        model.compile(optimizer=optimizer,
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])

        return model

    def augment_data(X, y, noise_factor=0.1, shift_max=10, scale_factor_range=(0.9, 1.1)):
        """
        Perform data augmentation with controlled parameters
        """
        X_aug, y_aug = [], []

        for x, label in zip(X, y):
            # Original sample
            X_aug.append(x)
            y_aug.append(label)

            # Add noise
            noise = noise_factor * np.random.normal(0, 1, x.shape)
            x_noise = x + noise
            X_aug.append(x_noise)
            y_aug.append(label)

            # Shift in time (horizontal)
            shift = np.random.randint(-shift_max, shift_max)
            x_shift = np.roll(x, shift, axis=1)
            X_aug.append(x_shift)
            y_aug.append(label)

            # Scale amplitude
            scale = np.random.uniform(scale_factor_range[0], scale_factor_range[1])
            x_scaled = x * scale
            X_aug.append(x_scaled)
            y_aug.append(label)

        return np.array(X_aug), np.array(y_aug)

    def train_model(X, y, encoder, epochs=50, batch_size=32, validation_split=0.2, cv_folds=5):
        """
        Train model with cross-validation and better monitoring
        """
        if X is None or y is None or len(X) == 0:
            print("Cannot train model: No data available.")
            return None, None
```

```
# Apply data augmentation
X_aug, y_aug = augment_data(X, y)
print(f"Data shape after augmentation: {X_aug.shape}")

# Calculate class weights to handle imbalance if present
from sklearn.utils.class_weight import compute_class_weight
classes = np.unique(y_aug)
class_weights = compute_class_weight('balanced', classes=classes, y=y_aug)
class_weight_dict = dict(zip(classes, class_weights))
print("Class weights:", class_weight_dict)

# Split data for final validation
X_train, X_val, y_train, y_val = train_test_split(
    X_aug, y_aug, test_size=validation_split, random_state=42, stratify=y_aug
)

print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
print("Training set class distribution:", np.bincount(y_train))
print("Validation set class distribution:", np.bincount(y_val))

# Build model
input_shape = X_train.shape[1:]
num_classes = len(encoder.classes_)
model = build_cnn_model(input_shape=input_shape, num_classes=num_classes)
model.summary()

# Define callbacks
callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        min_delta=0.001,
        restore_best_weights=True
    ),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5,
        min_lr=1e-6
    )
]

# Train model
history = model.fit(
    X_train, y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(X_val, y_val),
    class_weight=class_weight_dict,
    callbacks=callbacks,
    verbose=1
)

# Evaluate on validation set
y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)

# Calculate metrics
precision, recall, f1, _ = precision_recall_fscore_support(
    y_val, y_pred_classes, average='weighted'
)

print(f"Validation Precision: {precision:.3f}")
print(f"Validation Recall: {recall:.3f}")
print(f"Validation F1-Score: {f1:.3f}")

# Confusion matrix
cm = confusion_matrix(y_val, y_pred_classes)
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion matrix')
plt.colorbar()

# Add labels
tick_marks = np.arange(len(encoder.classes_))
plt.xticks(tick_marks, encoder.classes_, rotation=45)
plt.yticks(tick_marks, encoder.classes_)
```

```
        # Add text annotations
        thresh = cm.max() / 2.
        for i in range(cm.shape[0]):
            for j in range(cm.shape[1]):
                plt.text(j, i, format(cm[i, j], 'd'),
                        horizontalalignment="center",
                        color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()

        return model, history

    def process_real_data(file_path, sample_rate=4096, duration=8):  # Updated default sample rate
        """
        Process real LIGO data from HDF5 files
        """
        try:
            with h5py.File(file_path, 'r') as f:
                strain = f['strain']['Strain'][:]

                # Ensure correct duration
                if len(strain) > duration * sample_rate:
                    strain = strain[:duration * sample_rate]
                elif len(strain) < duration * sample_rate:
                    strain = np.pad(strain, (0, duration * sample_rate - len(strain)), 'constant')

                # Create time series
                ts = pycbc.types.TimeSeries(strain, delta_t=1.0/sample_rate)

                # Apply high-pass filter
                ts = highpass(ts, 15.0, 8)

                # Whiten the signal
                ts_gwpy = TimeSeries(ts.numpy(), sample_rate=sample_rate)
                ts_whitened = ts_gwpy.whiten(fftlength=4, overlap=2, fduration=4)

                # Generate spectrogram
                spec = generate_spectrogram(ts_whitened)
                if spec is None:
                    return None

                # Process spectrogram
                spec_data = spec.value
                spec_data = np.maximum(spec_data, 1e-5)
                spec_data = np.log10(spec_data)

                # Visualize
                plt.figure(figsize=(8, 6))
                plt.imshow(spec_data, aspect='auto', origin='lower',
                          extent=[0, spec.dt.value * spec.shape[1],
                                  spec.f0.value, spec.f0.value + spec.df.value * spec.shape[0]])
                plt.colorbar(label='Log Energy')
                plt.title(f"Q-transform Spectrogram - {os.path.basename(file_path)}")
                plt.xlabel("Time (s)")
                plt.ylabel("Frequency (Hz)")
                plt.tight_layout()
                plt.show()

                # Check for invalid values
                if np.any(np.isnan(spec_data)) or np.any(np.isinf(spec_data)):
                    print(f"Invalid values in Q-transform for {file_path}")
                    return None

                # Normalize
                mean, std = np.mean(spec_data), np.std(spec_data)
                if std < 1e-10:
                    print(f"Near-zero standard deviation for {file_path}")
                    std = 1.0
                spec_data = (spec_data - mean) / std

                # Resize
                from scipy.ndimage import zoom
                target_shape = (128, 128)
```

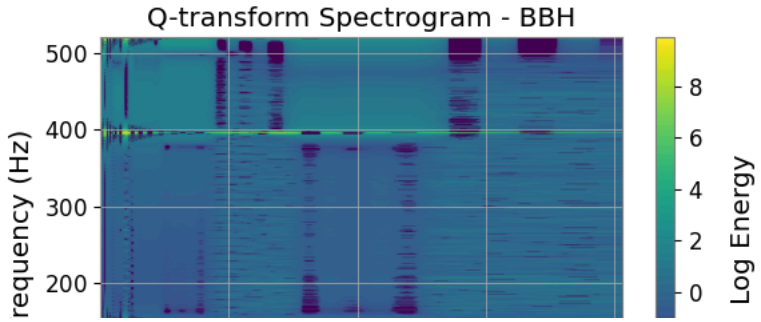
```
            spec_resized = zoom(spec_data, (target_shape[0]/spec_data.shape[0],
                                            target_shape[1]/spec_data.shape[1]))

            return spec_resized[..., np.newaxis]

        except Exception as e:
            print(f"Error processing {file_path}: {e}")
            return None

    def test_model_with_hdf5(model, hdf5_files, encoder, duration=8, sample_rate=4096):  # Updated default sample rate
        """
        Test model on multiple HDF5 files with better visualization
        """
        if model is None or encoder is None:
            print("Model or encoder not available.")
            return None

        results = {}
        print("Label mapping for prediction:", dict(zip(encoder.classes_, range(len(encoder.classes_)))))

        for file_path in hdf5_files:
            detector = os.path.basename(file_path).split('-')[0]

            # Process data
            processed_data = process_real_data(file_path, sample_rate, duration)
            if processed_data is None:
                results[detector] = "Error processing data"
                continue

            # Make prediction
            pred = model.predict(np.array([processed_data]))
            pred_probs = pred[0]
            pred_class = np.argmax(pred_probs)

            # Get label and confidence
            pred_label = encoder.inverse_transform([pred_class])[0]
            confidence = pred_probs[pred_class]

            # Store results
            results[detector] = {
                'prediction': pred_label,
                'confidence': confidence,
                'all_probs': {cls: prob for cls, prob in zip(encoder.classes_, pred_probs)}
            }

            # Visualize prediction probabilities
            plt.figure(figsize=(8, 4))
            plt.bar(encoder.classes_, pred_probs)
            plt.title(f"Prediction Probabilities for {detector}")
            plt.xlabel("Class")
            plt.ylabel("Probability")
            plt.ylim(0, 1)
            plt.tight_layout()
            plt.show()

        return results

    if __name__ == "__main__":
        num_samples_per_class = 50  # Reduced for faster execution
        sample_rate = 4096  # Updated to match input data
        duration = 8

        # Generate and preprocess data
        print("Generating BBH dataset...")
        bbh_signals, bbh_labels = generate_bbh_data(num_samples_per_class, sample_rate, duration)

        print("Generating BNS dataset...")
        bns_signals, bns_labels = generate_bns_data(num_samples_per_class, sample_rate, duration)

        print("Generating NSBH dataset...")
        nsbh_signals, nsbh_labels = generate_nsbh_data(num_samples_per_class, sample_rate, duration)

        # Preprocess each dataset separately first
        print("Preprocessing BBH data...")
        X_bbh, y_bbh, encoder_bbh = preprocess_data(bbh_signals, bbh_labels)

        print("Preprocessing BNS data...")
        X_bns, y_bns, encoder_bns = preprocess_data(bns_signals, bns_labels)
```

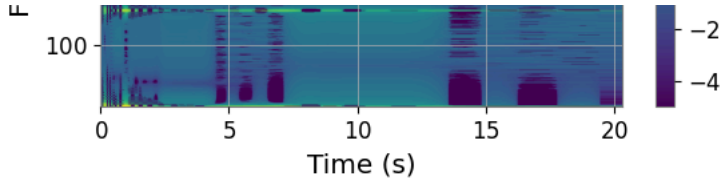
```
X_bns, y_bns, encoder_bns = preprocess_data(bns_signals, bns_labels)

print("Preprocessing NSBH data...")
X_nsbh, y_nsbh, encoder_nsbh = preprocess_data(nsbh_signals, nsbh_labels)

# Combine datasets
X_list, y_list = [], []
if X_bbh is not None and y_bbh is not None:
    X_list.append(X_bbh)
    y_list.append(y_bbh)
if X_bns is not None and y_bns is not None:
    X_list.append(X_bns)
    y_list.append(y_bns)
if X_nsbh is not None and y_nsbh is not None:
    X_list.append(X_nsbh)
    y_list.append(y_nsbh)

if not X_list or not y_list:
    print("No data successfully preprocessed for any merger type. Exiting.")
    exit()

# Combine all data
X = np.concatenate(X_list, axis=0)

# Re-encode combined labels to ensure consistency
all_labels = []
for signals, labels in [(bbh_signals, bbh_labels), (bns_signals, bns_labels), (nsbh_signals, nsbh_labels)]:
    all_labels.extend([l for s, l in zip(signals, labels) if s is not None])

encoder = LabelEncoder()
encoder.fit(all_labels)  # Fit encoder on all class names

# Transform labels for each dataset
all_y_encoded = []
for signals, labels in [(bbh_signals, bbh_labels), (bns_signals, bns_labels), (nsbh_signals, nsbh_labels)]:
    valid_labels = [l for s, l in zip(signals, labels) if s is not None]
    if valid_labels:
        y_encoded = encoder.transform(valid_labels)
        all_y_encoded.append(y_encoded)

y = np.concatenate(all_y_encoded)

print("Final X shape:", X.shape)
print("Final y shape:", y.shape)
print("Final y values distribution:", np.bincount(y))
print("Label mapping:", dict(zip(encoder.classes_, range(len(encoder.classes_)))))

# Train the model
print("Training model...")
model, history = train_model(X, y, encoder)

# Plot training history
if history is not None:
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Test on real data if available
hdf5_files = [
    "/content/H-H1_GWOSC_4KHZ_R1-1187006835-4096.hdf5",
    "/content/L-L1_GWOSC_4KHZ_R1-1187006835-4096.hdf5"
```

```
]

hdf5_files_available = all(os.path.exists(f) for f in hdf5_files)

if hdf5_files_available:
    print("Testing model with HDF5 files...")
    results = test_model_with_hdf5(model, hdf5_files, encoder, duration, sample_rate)

    print("\nPrediction Results:")
    for detector, result in results.items():
            if isinstance(result, dict):
                print(f"{detector}: {result['prediction']} (confidence: {result['confidence']:.2f})")
                print(f"  All probabilities: {result['all_probs']}")
            else:
                print(f"{detector}: {result}")

    # Save the model and encoder for future use
    model.save('gw_classifier_model.h5')
    import joblib
    joblib.dump(encoder, 'gw_classifier_encoder.joblib')

    print("\nModel and encoder saved for future use.")
    print("Model: 'gw_classifier_model.h5'")
    print("Encoder: 'gw_classifier_encoder.joblib'")
else:
      print("HDF5 files not found. Please provide valid file paths.")
```

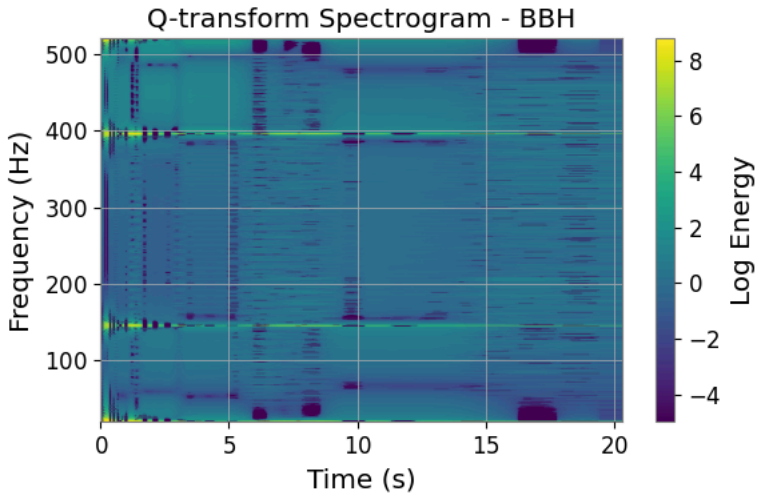
```
Generating BBH dataset...
Generating BBH sample 1/50
Generating BBH sample 2/50
Generating BBH sample 3/50
Generating BBH sample 4/50
Generating BBH sample 5/50
Generating BBH sample 6/50
Generating BBH sample 7/50
Generating BBH sample 8/50
Generating BBH sample 9/50
Generating BBH sample 10/50
Generating BBH sample 11/50
Generating BBH sample 12/50
Generating BBH sample 13/50
Generating BBH sample 14/50
Generating BBH sample 15/50
Generating BBH sample 16/50
Generating BBH sample 17/50
Generating BBH sample 18/50
Generating BBH sample 19/50
Generating BBH sample 20/50
Generating BBH sample 21/50
Generating BBH sample 22/50
Generating BBH sample 23/50
Generating BBH sample 24/50
Generating BBH sample 25/50
Generating BBH sample 26/50
Generating BBH sample 27/50
Generating BBH sample 28/50
Generating BBH sample 29/50
Generating BBH sample 30/50
Generating BBH sample 31/50
Generating BBH sample 32/50
Generating BBH sample 33/50
Generating BBH sample 34/50
Generating BBH sample 35/50
Generating BBH sample 36/50
Generating BBH sample 37/50
Generating BBH sample 38/50
Generating BBH sample 39/50
Generating BBH sample 40/50
Generating BBH sample 41/50
Generating BBH sample 42/50
Generating BBH sample 43/50
Generating BBH sample 44/50
Generating BBH sample 45/50
Generating BBH sample 46/50
Generating BBH sample 47/50
Generating BBH sample 48/50
Generating BBH sample 49/50
Generating BBH sample 50/50
Generating BNS dataset...
Generating BNS sample 1/50
Generating BNS sample 2/50
Generating BNS sample 3/50
Generating BNS sample 4/50
Generating BNS sample 5/50
Generating BNS sample 6/50
Generating BNS sample 7/50
Generating BNS sample 8/50
Generating BNS sample 9/50
Generating BNS sample 10/50
Generating BNS sample 11/50
Generating BNS sample 12/50
Generating BNS sample 13/50
Generating BNS sample 14/50
Generating BNS sample 15/50
Generating BNS sample 16/50
Generating BNS sample 17/50
Generating BNS sample 18/50
Generating BNS sample 19/50
Generating BNS sample 20/50
Generating BNS sample 21/50
Generating BNS sample 22/50
Generating BNS sample 23/50
Generating BNS sample 24/50
Generating BNS sample 25/50
Generating BNS sample 26/50
Generating BNS sample 27/50
Generating BNS sample 28/50
Generating BNS sample 29/50
Generating BNS sample 30/50
Generating BNS sample 31/50
Generating BNS sample 32/50
```

```
Generating BNS sample 33/50
Generating BNS sample 34/50
Generating BNS sample 35/50
Generating BNS sample 36/50
Generating BNS sample 37/50
Generating BNS sample 38/50
Generating BNS sample 39/50
Generating BNS sample 40/50
Generating BNS sample 41/50
Generating BNS sample 42/50
Generating BNS sample 43/50
Generating BNS sample 44/50
Generating BNS sample 45/50
Generating BNS sample 46/50
Generating BNS sample 47/50
Generating BNS sample 48/50
Generating BNS sample 49/50
Generating BNS sample 50/50
Generating NSBH dataset...
Generating NSBH sample 1/50
Generating NSBH sample 2/50
Generating NSBH sample 3/50
Generating NSBH sample 4/50
Generating NSBH sample 5/50
Generating NSBH sample 6/50
Generating NSBH sample 7/50
Generating NSBH sample 8/50
Generating NSBH sample 9/50
Generating NSBH sample 10/50
Generating NSBH sample 11/50
Generating NSBH sample 12/50
Generating NSBH sample 13/50
Generating NSBH sample 14/50
Generating NSBH sample 15/50
Generating NSBH sample 16/50
Generating NSBH sample 17/50
Generating NSBH sample 18/50
Generating NSBH sample 19/50
Generating NSBH sample 20/50
Generating NSBH sample 21/50
Generating NSBH sample 22/50
Generating NSBH sample 23/50
Generating NSBH sample 24/50
Generating NSBH sample 25/50
Generating NSBH sample 26/50
Generating NSBH sample 27/50
Generating NSBH sample 28/50
Generating NSBH sample 29/50
Generating NSBH sample 30/50
Generating NSBH sample 31/50
Generating NSBH sample 32/50
Generating NSBH sample 33/50
Generating NSBH sample 34/50
Generating NSBH sample 35/50
Generating NSBH sample 36/50
Generating NSBH sample 37/50
Generating NSBH sample 38/50
Generating NSBH sample 39/50
Generating NSBH sample 40/50
Generating NSBH sample 41/50
Generating NSBH sample 42/50
Generating NSBH sample 43/50
Generating NSBH sample 44/50
Generating NSBH sample 45/50
Generating NSBH sample 46/50
Generating NSBH sample 47/50
Generating NSBH sample 48/50
Generating NSBH sample 49/50
Generating NSBH sample 50/50
Preprocessing BBH data...
Processing signal BBH at index 0
```

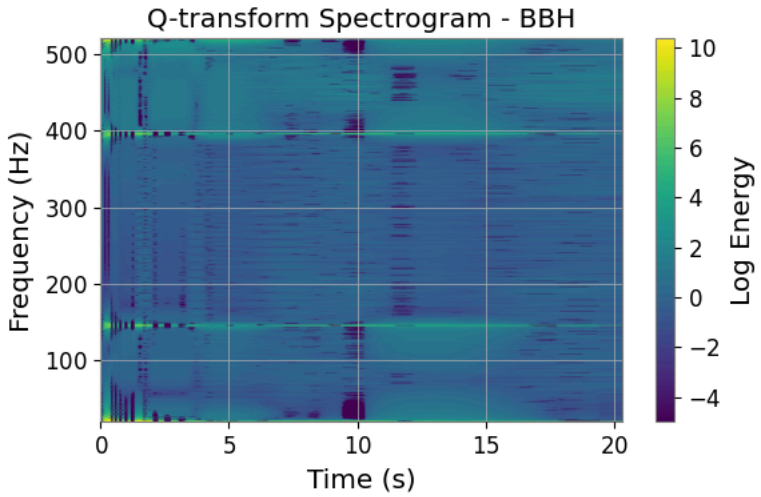





```
Processing signal BBH at index 1
```

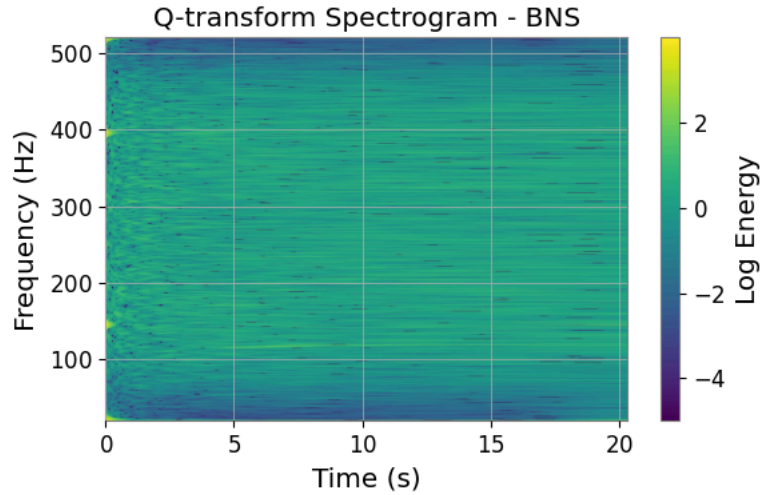



```
Processing signal BBH at index 2
```

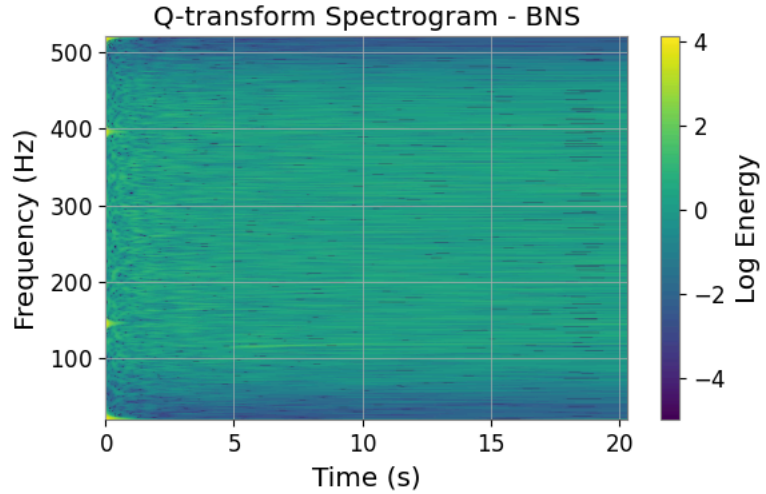



```
Processing signal BBH at index 3
Processing signal BBH at index 4
Processing signal BBH at index 5
Processing signal BBH at index 6
Processing signal BBH at index 7
Processing signal BBH at index 8
Processing signal BBH at index 9
Processing signal BBH at index 10
Processing signal BBH at index 11
Processing signal BBH at index 12
Processing signal BBH at index 13
Processing signal BBH at index 14
Processing signal BBH at index 15
Processing signal BBH at index 16
Processing signal BBH at index 17
Processing signal BBH at index 18
Processing signal BBH at index 19
Processing signal BBH at index 20
Processing signal BBH at index 21
Processing signal BBH at index 22
Processing signal BBH at index 23
Processing signal BBH at index 24
Processing signal BBH at index 25
Processing signal BBH at index 26
Processing signal BBH at index 27
Processing signal BBH at index 28
Processing signal BBH at index 29
Processing signal BBH at index 30
Processing signal BBH at index 31
Processing signal BBH at index 32
```

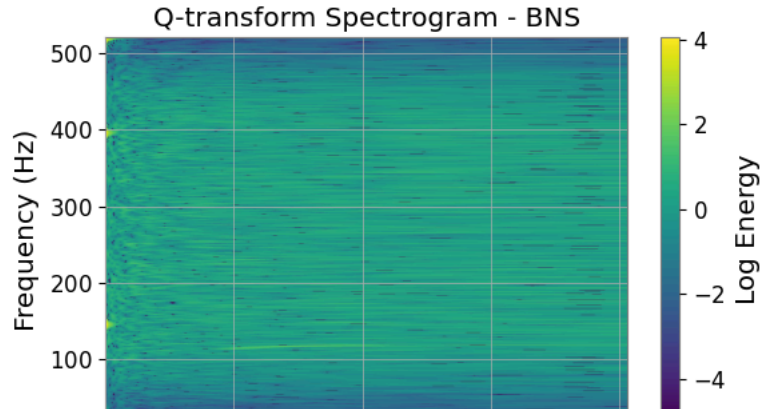
```
Processing signal BBH at index 33
Processing signal BBH at index 34
Processing signal BBH at index 35
Processing signal BBH at index 36
Processing signal BBH at index 37
Processing signal BBH at index 38
Processing signal BBH at index 39
Processing signal BBH at index 40
Processing signal BBH at index 41
Processing signal BBH at index 42
Processing signal BBH at index 43
Processing signal BBH at index 44
Processing signal BBH at index 45
Processing signal BBH at index 46
Processing signal BBH at index 47
Processing signal BBH at index 48
Processing signal BBH at index 49
Encoded labels: [0]
Label mapping: {np.str_('BBH'): 0}
Preprocessing BNS data...
Processing signal BNS at index 0
```

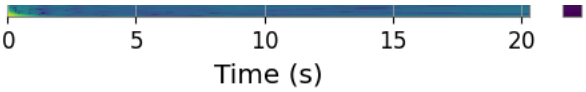



```
Processing signal BNS at index 1
```

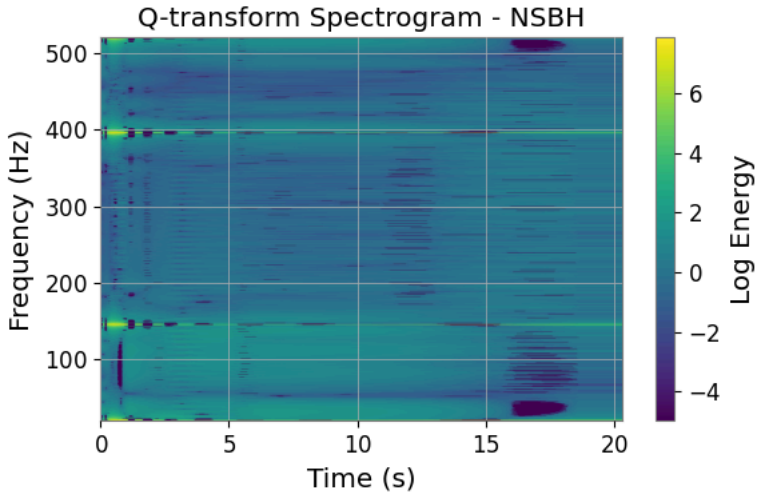



```
Processing signal BNS at index 2
```

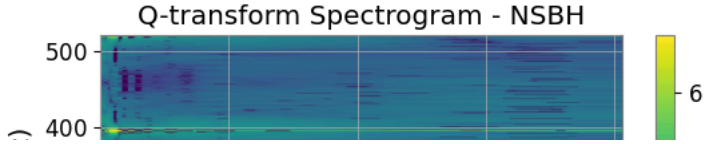


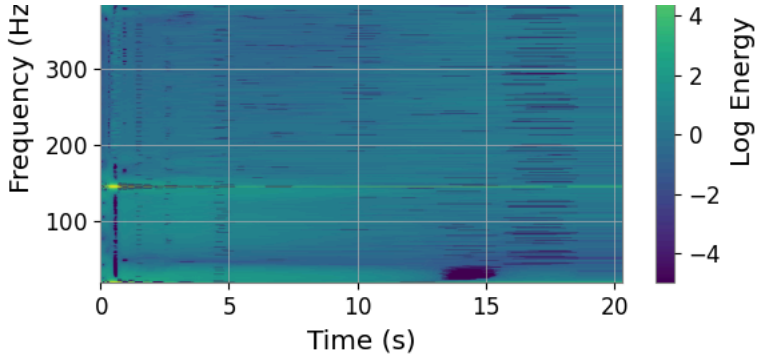



```
Processing signal BNS at index 3
Processing signal BNS at index 4
Processing signal BNS at index 5
Processing signal BNS at index 6
Processing signal BNS at index 7
Processing signal BNS at index 8
Processing signal BNS at index 9
Processing signal BNS at index 10
Processing signal BNS at index 11
Processing signal BNS at index 12
Processing signal BNS at index 13
Processing signal BNS at index 14
Processing signal BNS at index 15
Processing signal BNS at index 16
Processing signal BNS at index 17
Processing signal BNS at index 18
Processing signal BNS at index 19
Processing signal BNS at index 20
Processing signal BNS at index 21
Processing signal BNS at index 22
Processing signal BNS at index 23
Processing signal BNS at index 24
Processing signal BNS at index 25
Processing signal BNS at index 26
Processing signal BNS at index 27
Processing signal BNS at index 28
Processing signal BNS at index 29
Processing signal BNS at index 30
Processing signal BNS at index 31
Processing signal BNS at index 32
Processing signal BNS at index 33
Processing signal BNS at index 34
Processing signal BNS at index 35
Processing signal BNS at index 36
Processing signal BNS at index 37
Processing signal BNS at index 38
Processing signal BNS at index 39
Processing signal BNS at index 40
Processing signal BNS at index 41
Processing signal BNS at index 42
Processing signal BNS at index 43
Processing signal BNS at index 44
Processing signal BNS at index 45
Processing signal BNS at index 46
Processing signal BNS at index 47
Processing signal BNS at index 48
Processing signal BNS at index 49
Encoded labels: [0]
Label mapping: {np.str_('BNS'): 0}
Preprocessing NSBH data...
Processing signal NSBH at index 0
```

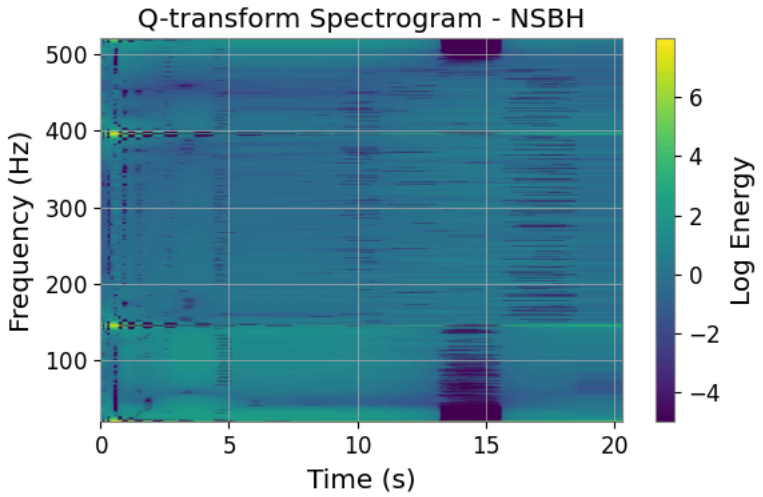



```
Processing signal NSBH at index 1
```






```
Processing signal NSBH at index 2
```




```
Processing signal NSBH at index 3
Processing signal NSBH at index 4
Processing signal NSBH at index 5
Processing signal NSBH at index 6
Processing signal NSBH at index 7
Processing signal NSBH at index 8
Processing signal NSBH at index 9
Processing signal NSBH at index 10
Processing signal NSBH at index 11
Processing signal NSBH at index 12
Processing signal NSBH at index 13
Processing signal NSBH at index 14
Processing signal NSBH at index 15
Processing signal NSBH at index 16
Processing signal NSBH at index 17
Processing signal NSBH at index 18
Processing signal NSBH at index 19
Processing signal NSBH at index 20
Processing signal NSBH at index 21
Processing signal NSBH at index 22
Processing signal NSBH at index 23
Processing signal NSBH at index 24
Processing signal NSBH at index 25
Processing signal NSBH at index 26
Processing signal NSBH at index 27
Processing signal NSBH at index 28
Processing signal NSBH at index 29
Processing signal NSBH at index 30
Processing signal NSBH at index 31
Processing signal NSBH at index 32
Processing signal NSBH at index 33
Processing signal NSBH at index 34
Processing signal NSBH at index 35
Processing signal NSBH at index 36
Processing signal NSBH at index 37
Processing signal NSBH at index 38
Processing signal NSBH at index 39
Processing signal NSBH at index 40
Processing signal NSBH at index 41
Processing signal NSBH at index 42
Processing signal NSBH at index 43
Processing signal NSBH at index 44
Processing signal NSBH at index 45
Processing signal NSBH at index 46
Processing signal NSBH at index 47
Processing signal NSBH at index 48
```