```python
import h5py
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

filename = "GW170817.h5"

with h5py.File(filename, "r") as f:
    print("Top-level keys:", list(f.keys()))

    # Check what's under 'posteriors'
    if 'posterior_samples' in f:
        print("Posterior keys:", list(f['posterior_samples'].keys()))
```

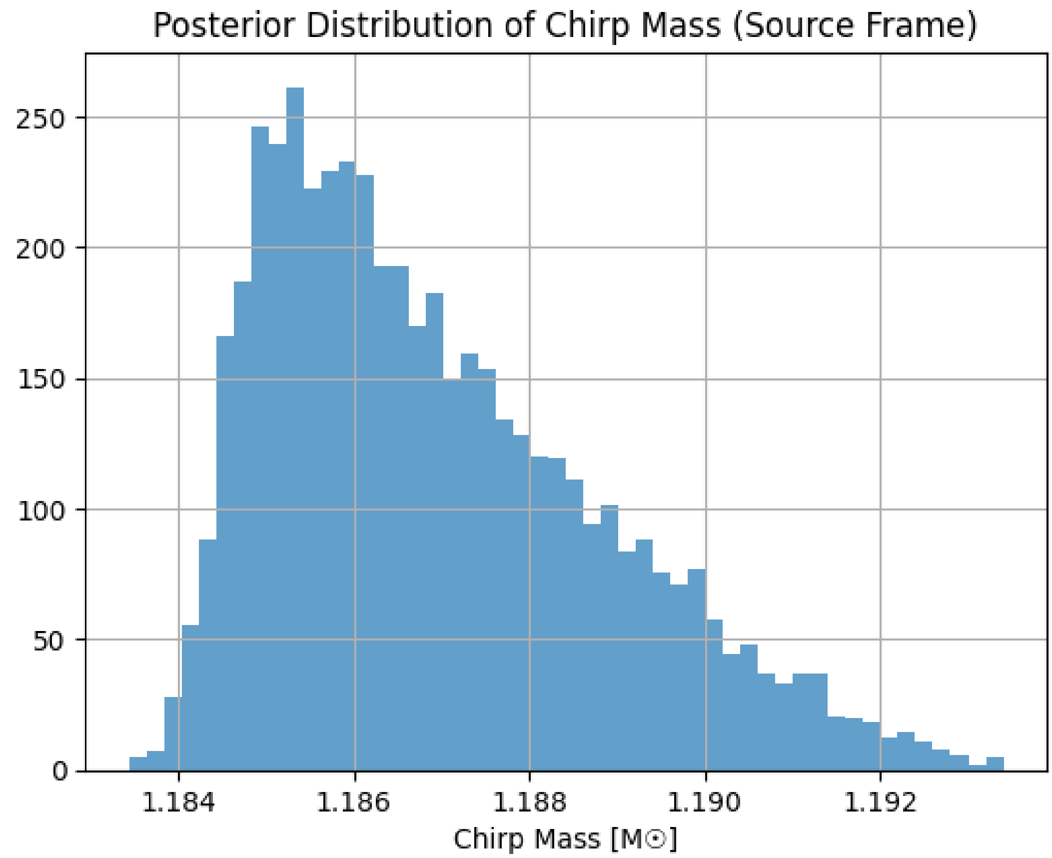
```
Top-level keys: ['IMRPhenomPv2NRT_highSpin_posterior',
'IMRPhenomPv2NRT_highSpin_prior', 'IMRPhenomPv2NRT_lowSpin_posterior',
'IMRPhenomPv2NRT_lowSpin_prior']
```

```python
import h5py
import numpy as np

filename = "GW170817.h5"

with h5py.File(filename, "r") as f:
    data = f['IMRPhenomPv2NRT_lowSpin_posterior'][:]

# Check available parameter names (columns)
print("Available parameters:", data.dtype.names)
```

```
Available parameters: ('costheta_jn', 'luminosity_distance_Mpc',
'right_ascension', 'declination', 'm1_detector_frame_Msun',
'm2_detector_frame_Msun', 'lambda1', 'lambda2', 'spin1', 'spin2',
'costilt1', 'costilt2')
```

```python
import h5py
import numpy as np
import matplotlib.pyplot as plt

filename = "GW170817.h5"

with h5py.File(filename, "r") as f:
    posterior = f['IMRPhenomPv2NRT_lowSpin_posterior'][:]

# Convert to NumPy structured array
posterior = posterior.astype([
    ('costheta_jn', 'f8'),
    ('luminosity_distance_Mpc', 'f8'),
    ('right_ascension', 'f8'),
    ('declination', 'f8'),
    ('m1_det', 'f8'),
    ('m2_det', 'f8'),
```

```python
    ('lambda1', 'f8'),
    ('lambda2', 'f8'),
    ('spin1', 'f8'),
    ('spin2', 'f8'),
    ('costilt1', 'f8'),
    ('costilt2', 'f8')
])

def source_mass(m_det, d_l_mpc):
    # Rough redshift estimate from luminosity distance using H0 = 70 
km/s/Mpc, flat ΛCDM
    z = d_l_mpc / 4300  # very rough!
    return m_det / (1 + z)

m1_source = source_mass(posterior['m1_det'], 
posterior['luminosity_distance_Mpc'])
m2_source = source_mass(posterior['m2_det'], 
posterior['luminosity_distance_Mpc'])

def chirp_mass(m1, m2):
    return ((m1 * m2)**(3/5)) / ((m1 + m2)**(1/5))

def eta(m1, m2):
    return (m1 * m2) / ((m1 + m2)**2)

mc = chirp_mass(m1_source, m2_source)
sym_eta = eta(m1_source, m2_source)

plt.hist(mc, bins=50, density=True, alpha=0.7)
plt.title("Posterior Distribution of Chirp Mass (Source Frame)")
plt.xlabel("Chirp Mass [M☉]")
plt.grid(True)
plt.show()
```

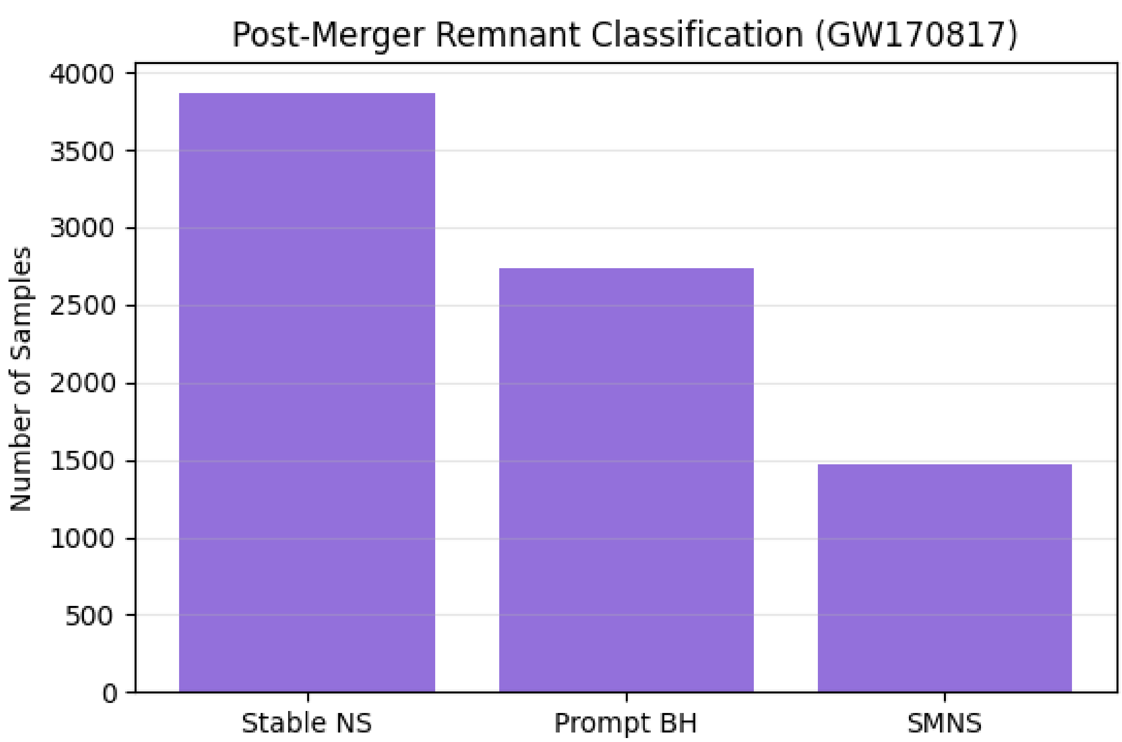



```
# Compute total mass
mtot = m1_source + m2_source

# Tidal deformability approximated using formula for effective lambda
# (From literature: https://arxiv.org/abs/1805.11581)
def lambda_tilde(m1, m2, lam1, lam2):
    q = m2 / m1
    return (16 / 13) * (
        ((1 + 12*q)*q**4 * lam1 + (1 + 12/q)*lam2) / (1 + q)**5
    )

lt = lambda_tilde(m1_source, m2_source, posterior['lambda1'],
posterior['lambda2'])

# Remnant classifier
labels = []

for m, l in zip(mtot, lt):
    if m < 2.75 and l > 400:
        labels.append("Stable NS")
    elif 2.75 <= m < 3.0 and l > 300:
        labels.append("SMNS")
    elif 3.0 <= m < 3.4 and l > 200:
```

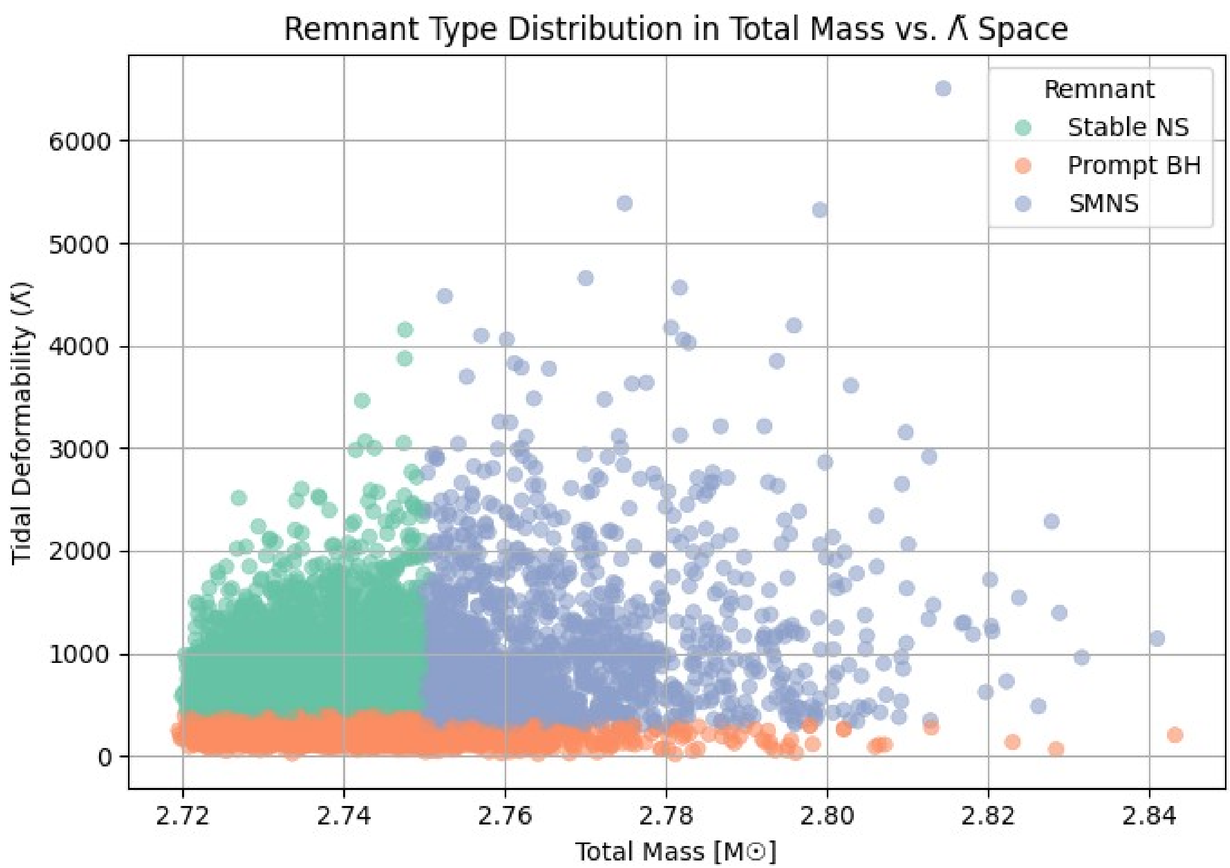
```
        labels.append("HMNS")
    else:
        labels.append("Prompt BH")

from collections import Counter

counts = Counter(labels)

print("Remnant Classification Counts:")
for k, v in counts.items():
    print(f"{k}: {v}")
```

```
Remnant Classification Counts:
Stable NS: 3868
Prompt BH: 2739
SMNS: 1471
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))
plt.bar(counts.keys(), counts.values(), color='mediumpurple')
plt.ylabel("Number of Samples")
plt.title("Post-Merger Remnant Classification (GW170817)")
plt.grid(True, axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```



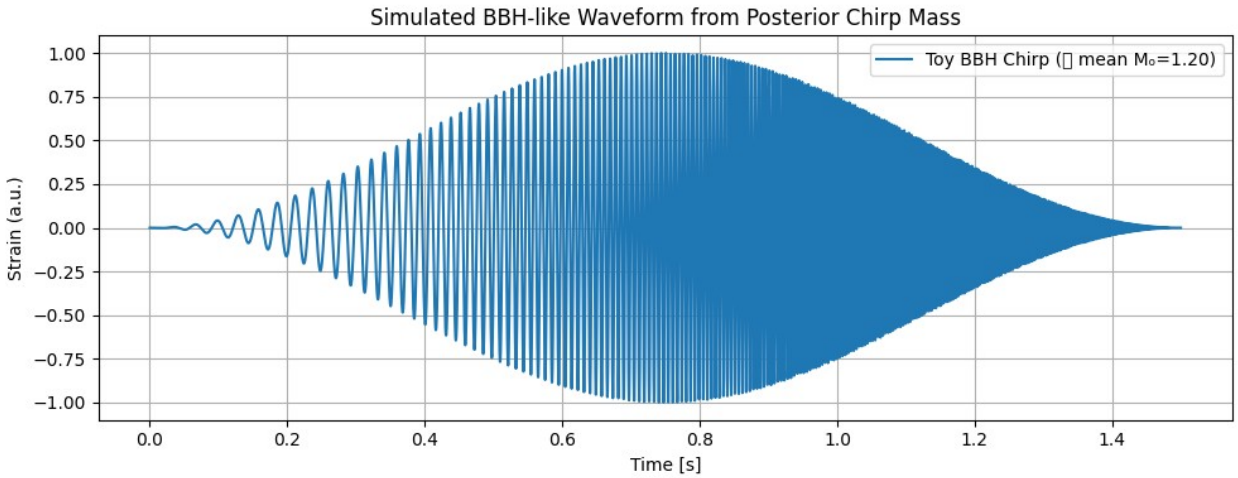
```
import seaborn as sns
import pandas as pd

df = pd.DataFrame({
    'TotalMass': mtot,
    'LambdaTilde': lt,
    'Remnant': labels
})

plt.figure(figsize=(7, 5))
sns.scatterplot(data=df, x='TotalMass', y='LambdaTilde',
hue='Remnant', palette='Set2', alpha=0.6, edgecolor=None)
plt.title("Remnant Type Distribution in Total Mass vs. Λ˜Space")
plt.xlabel("Total Mass [M☉]")
plt.ylabel("Tidal Deformability (Λ̃)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



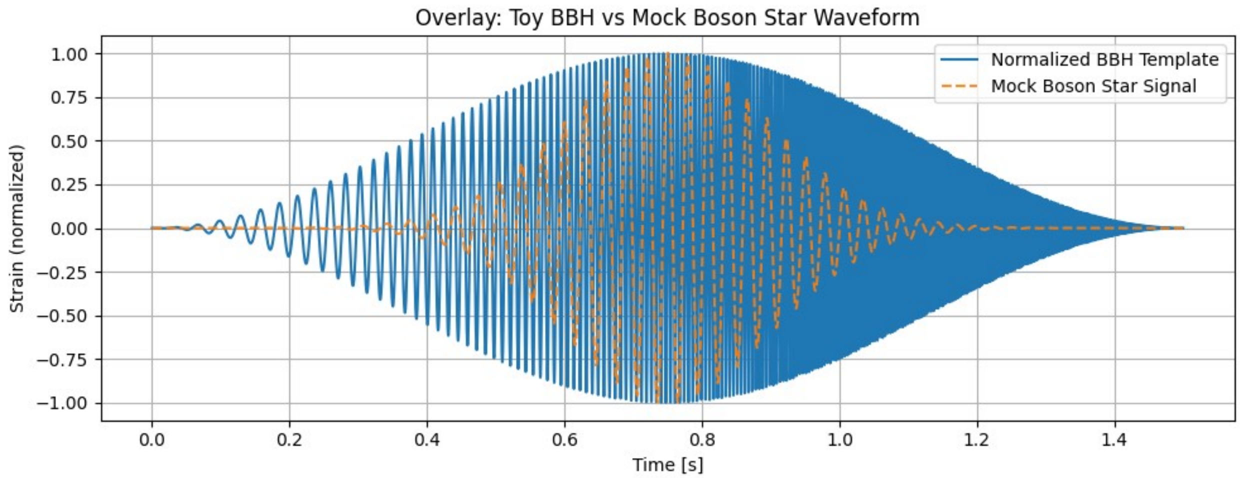

```
import h5py
import numpy as np
```

```python
filename = "GW170817.h5"
with h5py.File(filename, "r") as f:
    posterior = f['IMRPhenomPv2NRT_lowSpin_posterior']
    m1 = posterior['m1_detector_frame_Msun'][:]
    m2 = posterior['m2_detector_frame_Msun'][:]

# Compute chirp mass from posterior
def compute_chirp_mass(m1, m2):
    return (m1 * m2)**(3/5) / (m1 + m2)**(1/5)

chirp_masses = compute_chirp_mass(m1, m2)

from scipy.signal import chirp
import numpy as np
import matplotlib.pyplot as plt

# Get a representative chirp mass (mean of posterior)
chirp_mass_mean = np.mean(chirp_masses)

# Define time and frequency sweep
duration = 1.5  # seconds
sample_rate = 4096  # Hz
t_wave = np.linspace(0, duration, int(sample_rate * duration))
f0 = 30  # Starting frequency (Hz)
f1 = 500  # End frequency (Hz)

# Toy waveform: frequency sweep + amplitude envelope
h_bbh = chirp(t_wave, f0=f0, f1=f1, t1=duration, method='quadratic')
window = np.hanning(len(h_bbh))
h_bbh *= window

# Plot
plt.figure(figsize=(10, 4))
plt.plot(t_wave, h_bbh, label=f'Toy BBH Chirp (⨉ mean
Mₒ={chirp_mass_mean:.2f})')
plt.title("Simulated BBH-like Waveform from Posterior Chirp Mass")
plt.xlabel("Time [s]")
plt.ylabel("Strain (a.u.)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

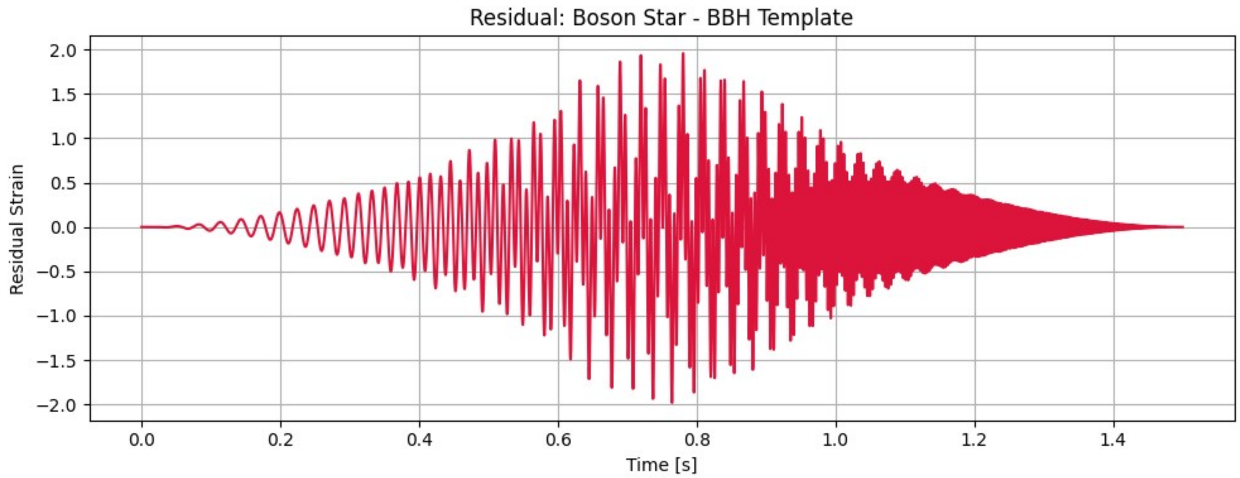
```
<ipython-input-18-d2a9cc239436>:28: UserWarning: Glyph 10761 (\N{N-ARY
TIMES OPERATOR}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151
: UserWarning: Glyph 10761 (\N{N-ARY TIMES OPERATOR}) missing from
font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```




```
# Mock Boson Star waveform: smoother envelope, slightly different
frequency evolution
bs_wave = np.sin(2 * np.pi * f0 * t_wave**1.2) * np.exp(-((t_wave -
duration/2)**2) / (2 * 0.15**2))

# Normalize both
h_bbh_norm = h_bbh / np.max(np.abs(h_bbh))
bs_wave_norm = bs_wave / np.max(np.abs(bs_wave))

# Overlay plot
plt.figure(figsize=(10, 4))
plt.plot(t_wave, h_bbh_norm, label='Normalized BBH Template')
plt.plot(t_wave, bs_wave_norm, label='Mock Boson Star Signal',
linestyle='--', alpha=0.9)
plt.xlabel("Time [s]")
plt.ylabel("Strain (normalized)")
plt.title("Overlay: Toy BBH vs Mock Boson Star Waveform")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

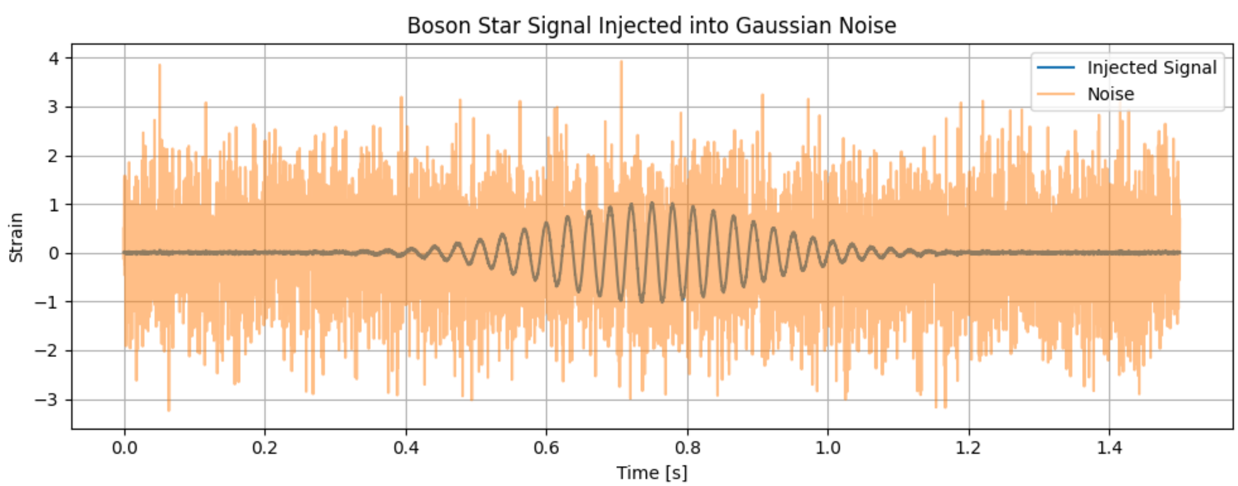



```
from sklearn.metrics.pairwise import cosine_similarity

# Ensure same length and normalized
bs_wave_aligned = bs_wave_norm[:len(h_bbh_norm)]
residual = bs_wave_aligned - h_bbh_norm

# Plot residual
plt.figure(figsize=(10, 4))
plt.plot(t_wave[:len(residual)], residual, color='crimson')
plt.title("Residual: Boson Star - BBH Template")
plt.xlabel("Time [s]")
plt.ylabel("Residual Strain")
plt.grid(True)
plt.tight_layout()
plt.show()

# Match score (inner product)
match = np.vdot(bs_wave_aligned, h_bbh_norm) /
(np.linalg.norm(bs_wave_aligned) * np.linalg.norm(h_bbh_norm))
print(f"Match (inner product): {match:.4f}")

# Cosine similarity
sim = cosine_similarity(bs_wave_aligned.reshape(1, -1),
h_bbh_norm.reshape(1, -1))[0][0]
print(f"Cosine Similarity: {sim:.4f}")
```

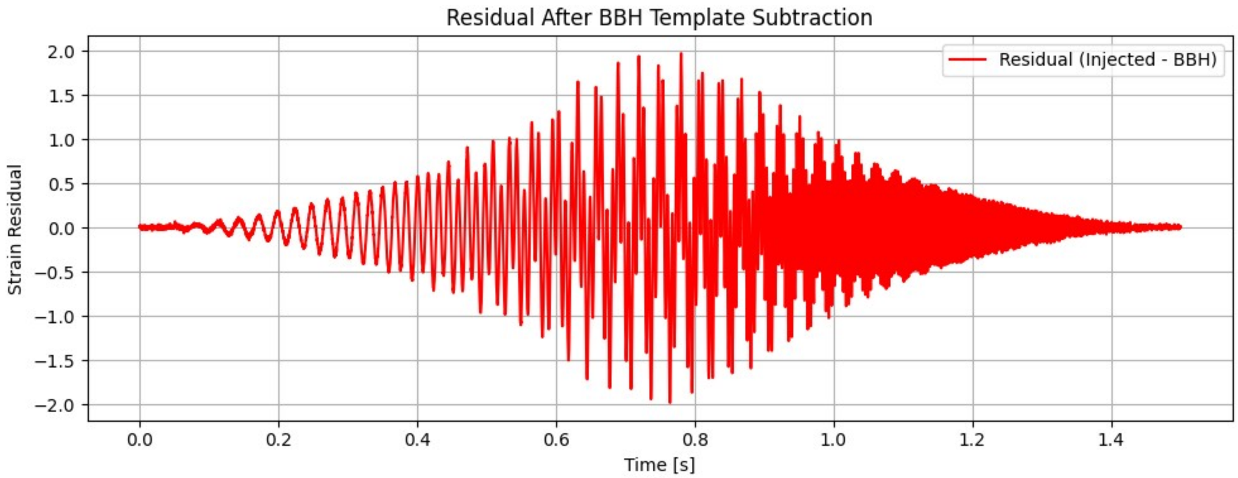



```
Match (inner product): -0.0000
Cosine Similarity: -0.0000

def estimate_snr(signal, template):
    signal = signal - np.mean(signal)
    template = template - np.mean(template)
    inner = np.vdot(signal, template)
    norm_temp = np.linalg.norm(template)
    return np.abs(inner) / norm_temp

snr_bs = estimate_snr(bs_wave_aligned, bs_wave_aligned)
snr_bbh = estimate_snr(bs_wave_aligned, h_bbh_norm)

print(f"SNR of Boson Star waveform: {snr_bs:.2f}")
print(f"SNR of BBH template matched to BS: {snr_bbh:.2f}")

SNR of Boson Star waveform: 23.34
SNR of BBH template matched to BS: 0.00

import pandas as pd

df_out = pd.DataFrame({
    'time': t_wave[:len(bs_wave_aligned)],
    'bs_waveform': bs_wave_aligned,
    'bbh_waveform': h_bbh_norm,
    'residual': residual
})

df_out.to_csv("posterior_overlay_analysis.csv", index=False)
print("□ Saved to 'posterior_overlay_analysis.csv'")

□ Saved to 'posterior_overlay_analysis.csv'

def inject_gaussian_noise(signal, snr_target=20):
    np.random.seed(42)
    noise = np.random.normal(0, 1, len(signal))
```

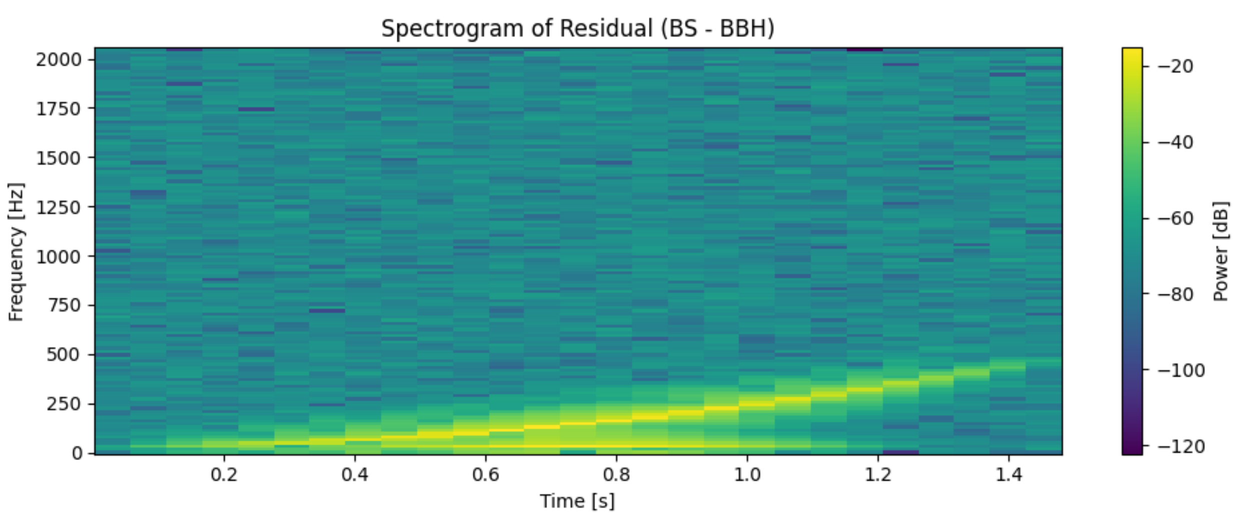
```
    signal_power = np.sqrt(np.mean(signal**2))
    noise_power = np.sqrt(np.mean(noise**2))
    scale = signal_power / noise_power / snr_target
    injected = signal + noise * scale
    return injected, noise

injected_signal, noise = inject_gaussian_noise(bs_wave_aligned,
snr_target=20)

plt.figure(figsize=(10, 4))
plt.plot(t_wave[:len(injected_signal)], injected_signal,
label='Injected Signal')
plt.plot(t_wave[:len(noise)], noise, label='Noise', alpha=0.5)
plt.xlabel("Time [s]")
plt.ylabel("Strain")
plt.title("Boson Star Signal Injected into Gaussian Noise")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
residual_injected = injected_signal[:len(h_bbh_norm)] - h_bbh_norm

plt.figure(figsize=(10, 4))
plt.plot(t_wave[:len(residual_injected)], residual_injected,
label="Residual (Injected - BBH)", color='red')
plt.xlabel("Time [s]")
plt.ylabel("Strain Residual")
plt.title("Residual After BBH Template Subtraction")
plt.grid(True)
plt.tight_layout()
plt.legend()
plt.show()
```




```python
from scipy.signal import spectrogram

f, t_spec, Sxx = spectrogram(residual_injected, fs=4096, nperseg=256)

plt.figure(figsize=(10, 4))
plt.pcolormesh(t_spec, f, 10 * np.log10(Sxx), shading='auto')
plt.xlabel("Time [s]")
plt.ylabel("Frequency [Hz]")
plt.title("Spectrogram of Residual (BS - BBH)")
plt.colorbar(label="Power [dB]")
plt.tight_layout()
plt.show()
```




```python
# Compute energies
residual_energy = np.sum(residual_injected**2)
bs_energy = np.sum(injected_signal**2)
bbh_energy = np.sum(h_bbh_norm**2)
```

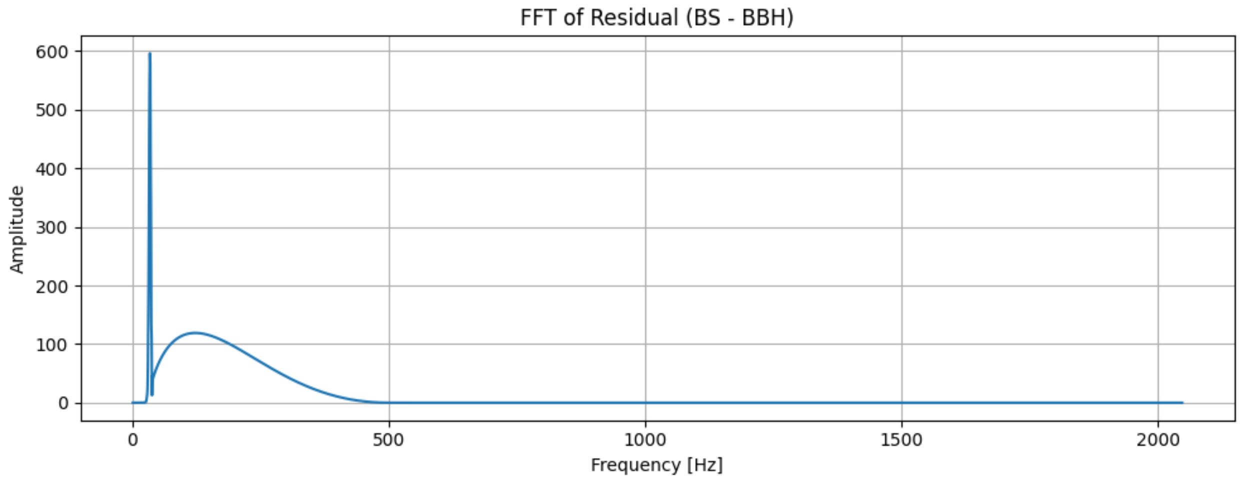
```python
# Mismatch = fraction of energy left in residual
mismatch = residual_energy / bs_energy

print(f"Total BS Energy (injected): {bs_energy:.4f}")
print(f"BBH Template Energy: {bbh_energy:.4f}")
print(f"Residual Energy: {residual_energy:.4f}")
print(f"Mismatch Fraction: {mismatch:.4f}")
```

```
Total BS Energy (injected): 547.9424
BBH Template Energy: 1151.8824
Residual Energy: 1698.9948
Mismatch Fraction: 3.1007
```

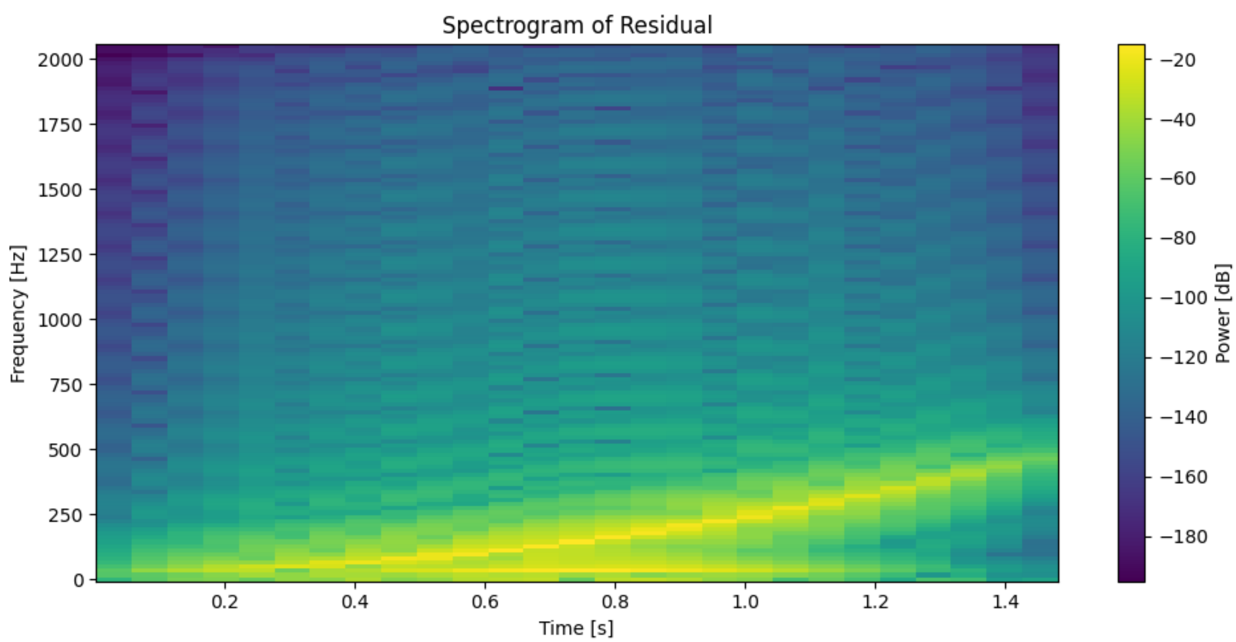
```python
import pandas as pd

df = pd.DataFrame({
    'time': t_wave[:len(h_bbh_norm)],
    'injected_bs': injected_signal[:len(h_bbh_norm)],
    'bbh_template': h_bbh_norm,
    'residual': residual_injected
})

df.to_csv("bs_vs_bbh_residual.csv", index=False)
print("□ CSV saved: bs_vs_bbh_residual.csv")
```

```
□ CSV saved: bs_vs_bbh_residual.csv
```

```python
from scipy.fft import fft, fftfreq

# Sampling parameters
dt = 1/4096
N = len(residual)

# FFT
res_fft = fft(residual)
freqs = fftfreq(N, dt)

# Only positive frequencies
mask = freqs > 0
plt.figure(figsize=(10, 4))
plt.plot(freqs[mask], np.abs(res_fft[mask]))
plt.title("FFT of Residual (BS - BBH)")
plt.xlabel("Frequency [Hz]")
plt.ylabel("Amplitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```




```python
from scipy.signal import spectrogram

f, t, Sxx = spectrogram(residual, fs=4096, nperseg=256)

plt.figure(figsize=(10, 5))
plt.pcolormesh(t, f, 10 * np.log10(Sxx), shading='auto')
plt.title("Spectrogram of Residual")
plt.xlabel("Time [s]")
plt.ylabel("Frequency [Hz]")
plt.colorbar(label="Power [dB]")
plt.tight_layout()
plt.show()
```



```
ridge_df = pd.DataFrame(Sxx.T, columns=[f"Freq_{int(freq)}" for freq
in f])
ridge_df["Time"] = t
ridge_df.to_csv("residual_ridge_spectrogram.csv", index=False)
print("Saved: residual_ridge_spectrogram.csv")
```

```
Saved: residual_ridge_spectrogram.csv
```

```
import h5py
import numpy as np
import pandas as pd

filename = "GW170817.h5"

with h5py.File(filename, "r") as f:
    posterior = f['IMRPhenomPv2NRT_lowSpin_posterior']

    # Extract relevant parameters
    m1 = posterior['m1_detector_frame_Msun'][:]  # Primary mass
    m2 = posterior['m2_detector_frame_Msun'][:]  # Secondary mass
    lambda1 = posterior['lambda1'][:]            # Tidal deformability
1
    lambda2 = posterior['lambda2'][:]            # Tidal deformability
2
    spin1 = posterior['spin1'][:]                # Spin 1
    spin2 = posterior['spin2'][:]                # Spin 2
    tilt1 = posterior['costilt1'][:]             # Cosine tilt 1
    tilt2 = posterior['costilt2'][:]             # Cosine tilt 2

# Derived features
mass_ratio = m2 / m1
chirp_mass = ((m1 * m2)**(3/5)) / ((m1 + m2)**(1/5))
lambda_tilde = (16/13) * (
    (m1 + 12*m2) * m1**4 * lambda1 +
    (m2 + 12*m1) * m2**4 * lambda2
) / (m1 + m2)**5

# Remnant classification based on lambda_tilde
labels = []
for lt in lambda_tilde:
    if lt > 600:
        labels.append("Stable NS")
    elif lt < 200:
        labels.append("Prompt BH")
    else:
        labels.append("SMNS")  # Supramassive NS

# Build DataFrame
df_ml = pd.DataFrame({
    "m1": m1,
```

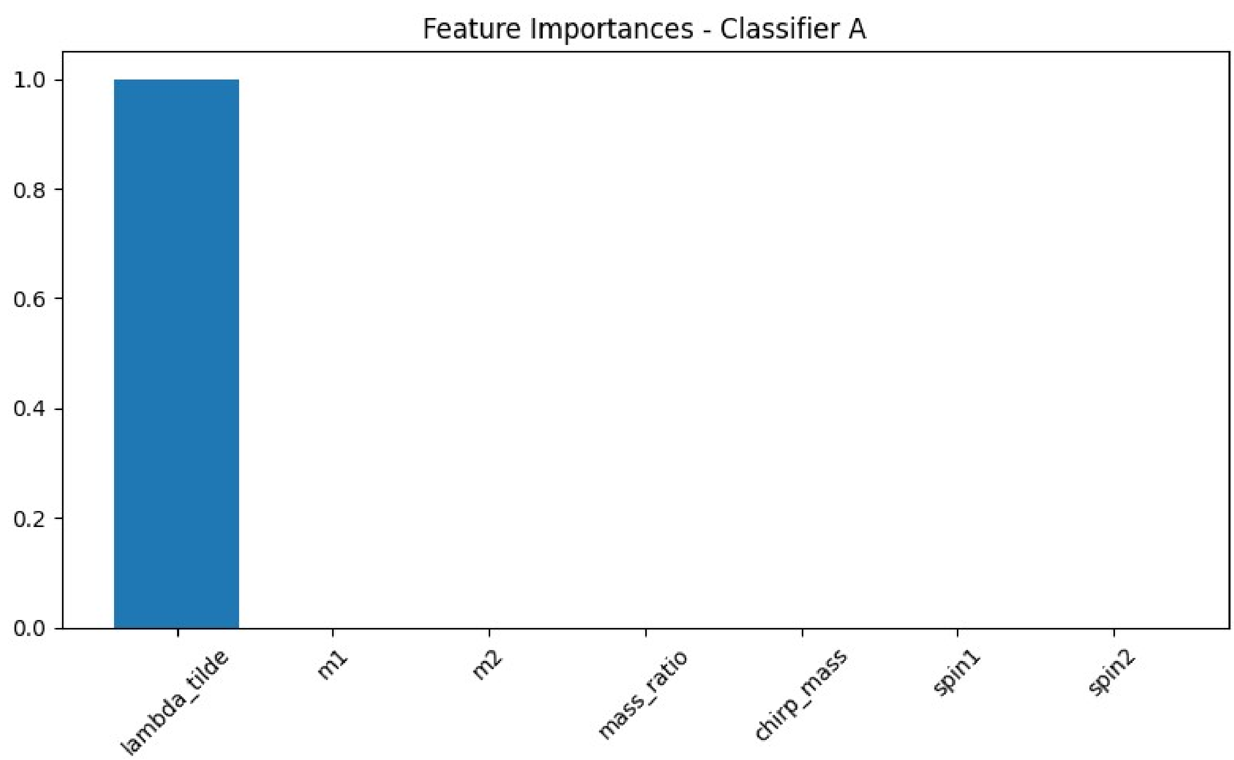
```
    "m2": m2,
    "mass_ratio": mass_ratio,
    "chirp_mass": chirp_mass,
    "lambda1": lambda1,
    "lambda2": lambda2,
    "lambda_tilde": lambda_tilde,
    "spin1": spin1,
    "spin2": spin2,
    "cos_tilt1": tilt1,
    "cos_tilt2": tilt2,
    "remnant_label": labels
})

# Drop NaNs (if any)
df_ml.dropna(inplace=True)

# Save for ML
df_ml.to_csv("GW170817_ML.csv", index=False)
print("Dataset saved to GW170817_ML.csv")
print("Preview:")
print(df_ml.head())
```

```
Dataset saved to GW170817_ML.csv
Preview:
         m1        m2  mass_ratio  chirp_mass      lambda1
lambda2  \
0  1.407326  1.344756    0.955540    1.197541  1168.731161
169.256087
1  1.403993  1.347745    0.959937    1.197463   339.747384
35.496698
2  1.621858  1.172888    0.723176    1.197549   178.885265
135.407947
3  1.509382  1.255827    0.832014    1.197545   233.511599
1709.283524
4  1.438443  1.316076    0.914931    1.197554  1119.534515
193.743065

   lambda_tilde     spin1     spin2  cos_tilt1  cos_tilt2
remnant_label
0    705.724428  0.038945  0.008111  -0.233334   0.998633     Stable
NS
1    197.632740  0.047122  0.021578   0.038034  -0.563819     Prompt
BH
2    178.419004  0.034027  0.036526   0.411368   0.356120     Prompt
BH
3    779.821318  0.025493  0.039489   0.286742   0.044635     Stable
NS
4    724.832775  0.003636  0.018100   0.376845   0.024104     Stable
NS
```

```python
df= df_ml

from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score, matthews_corrcoef

# Step 1: Binary classification target (1 = Prompt BH, 0 = Not Prompt
BH)
df['label_binary'] = df['remnant_label'].apply(lambda x: 1 if x ==
'Prompt BH' else 0)

# Step 2: Select features used in the paper (lambda_tilde, Mtot, q,
chi_eff)
X = df[['lambda_tilde', 'm1', 'm2', 'mass_ratio', 'chirp_mass',
'spin1', 'spin2']]  # You can choose the best features
y = df['label_binary']

# Step 3: Split dataset (90% train, 10% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=42, stratify=y)

# Step 4: Train classifier
clf = GradientBoostingClassifier(n_estimators=300, learning_rate=0.05,
max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# Step 5: Evaluation
y_pred = clf.predict(X_test)

print("\n=== Classifier A Evaluation ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("MCC:", matthews_corrcoef(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred))

# Step 6: Feature Importances
import matplotlib.pyplot as plt

feature_importances = clf.feature_importances_
plt.figure(figsize=(8, 5))
plt.bar(X.columns, feature_importances)
plt.title('Feature Importances - Classifier A')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
=== Classifier A Evaluation ===
Accuracy: 1.0
```

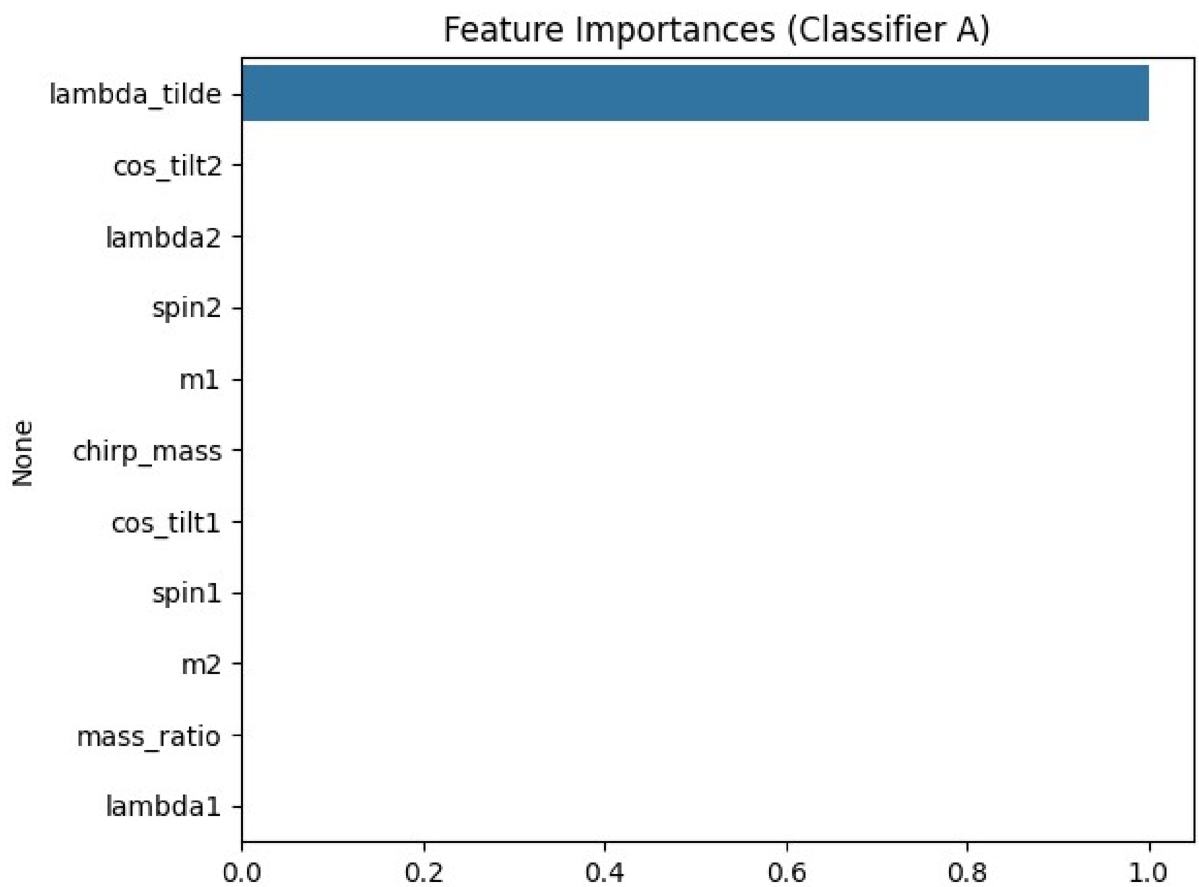
```
MCC: 1.0

Confusion Matrix:
 [[689   0]
 [  0 119]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       689
           1       1.00      1.00      1.00       119

    accuracy                           1.00       808
   macro avg       1.00      1.00      1.00       808
weighted avg       1.00      1.00      1.00       808
```




```
# Encode remnant label
label_encoder = LabelEncoder()
df['label_encoded'] = label_encoder.fit_transform(df['remnant_label'])

# Features and labels
features = ['m1', 'm2', 'mass_ratio', 'chirp_mass', 'lambda1',
'lambda2',
            'lambda_tilde', 'spin1', 'spin2', 'cos_tilt1',
'cos_tilt2']
```

```
X = df[features]
y = df['label_encoded']

# Binary classifier: Prompt BH vs. Not Prompt BH
df_binary = df.copy()
df_binary['binary_label'] = (df_binary['remnant_label'] == 'Prompt
BH').astype(int)
X_bin = df_binary[features]
y_bin = df_binary['binary_label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_bin, y_bin,
test_size=0.2, stratify=y_bin, random_state=42)

# Classifier A: Gradient Boosting
clf = GradientBoostingClassifier(n_estimators=200, learning_rate=0.05,
max_depth=4, random_state=42)
clf.fit(X_train, y_train)

# Predictions and Evaluation
y_pred = clf.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("MCC:", matthews_corrcoef(y_test, y_pred))

# Feature Importances
importances = pd.Series(clf.feature_importances_,
index=features).sort_values(ascending=False)
sns.barplot(x=importances.values, y=importances.index)
plt.title("Feature Importances (Classifier A)")
plt.tight_layout()
plt.show()
```

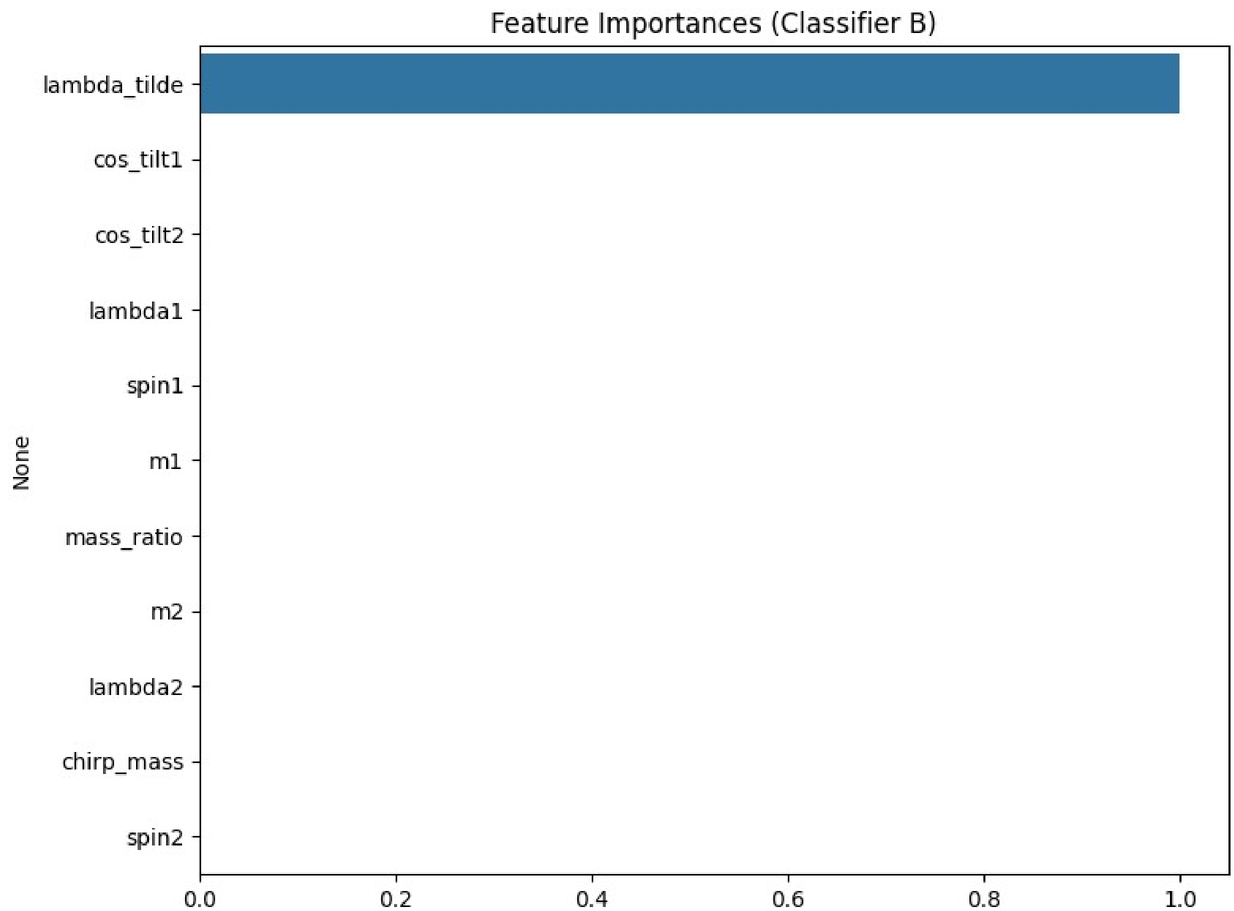
```
Confusion Matrix:
[[1378    0]
 [   0  238]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1378
           1       1.00      1.00      1.00       238

    accuracy                           1.00      1616
   macro avg       1.00      1.00      1.00      1616
weighted avg       1.00      1.00      1.00      1616

MCC: 1.0
```




```
# Encode the 3-class target label
label_encoder = LabelEncoder()
df['label_encoded'] = label_encoder.fit_transform(df['remnant_label'])

print("Class mapping:", dict(zip(label_encoder.classes_,
label_encoder.transform(label_encoder.classes_))))

# Define features and target
features = ['m1', 'm2', 'mass_ratio', 'chirp_mass', 'lambda1',
'lambda2',
            'lambda_tilde', 'spin1', 'spin2', 'cos_tilt1',
'cos_tilt2']
X = df[features]
y = df['label_encoded']

# Train-test split (90% train / 10% validation)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, stratify=y, random_state=42)

# Initialize and train classifier
clf = GradientBoostingClassifier(n_estimators=300, learning_rate=0.03,
max_depth=5, random_state=42)
```

```python
clf.fit(X_train, y_train)

# Evaluate
y_pred = clf.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred, target_names=label_encoder.classes_))
print("MCC:", matthews_corrcoef(y_test, y_pred))

# Feature Importances
importances = pd.Series(clf.feature_importances_,
index=features).sort_values(ascending=False)
plt.figure(figsize=(8, 6))
sns.barplot(x=importances.values, y=importances.index)
plt.title("Feature Importances (Classifier B)")
plt.tight_layout()
plt.show()
```

```
Class mapping: {'Prompt BH': np.int64(0), 'SMNS': np.int64(1), 'Stable
NS': np.int64(2)}
Confusion Matrix:
 [[119   0   0]
 [  0 475   0]
 [  0   0 214]]

Classification Report:
               precision    recall  f1-score   support

   Prompt BH       1.00      1.00      1.00       119
        SMNS       1.00      1.00      1.00       475
   Stable NS       1.00      1.00      1.00       214

    accuracy                           1.00       808
   macro avg       1.00      1.00      1.00       808
weighted avg       1.00      1.00      1.00       808

MCC: 1.0
```




```python
import pandas as pd

# Load the dataset
df = pd.read_csv("GW170817_ML.csv")

# Define 4-class target based on rules
def classify_remnant(row):
    if row['remnant_label'] == 'Prompt BH':
        return 'Prompt BH'
    elif row['remnant_label'] == 'Stable NS':
        return 'Stable NS'
    elif row['remnant_label'] == 'SMNS':
        Mtot = row['m1'] + row['m2']
        λ = row['lambda_tilde']
        if Mtot > 2.8 and λ < 400:
            return 'Short-lived HMNS'
        else:
            return 'Long-lived HMNS'

df['remnant_label_4class'] = df.apply(classify_remnant, axis=1)
```

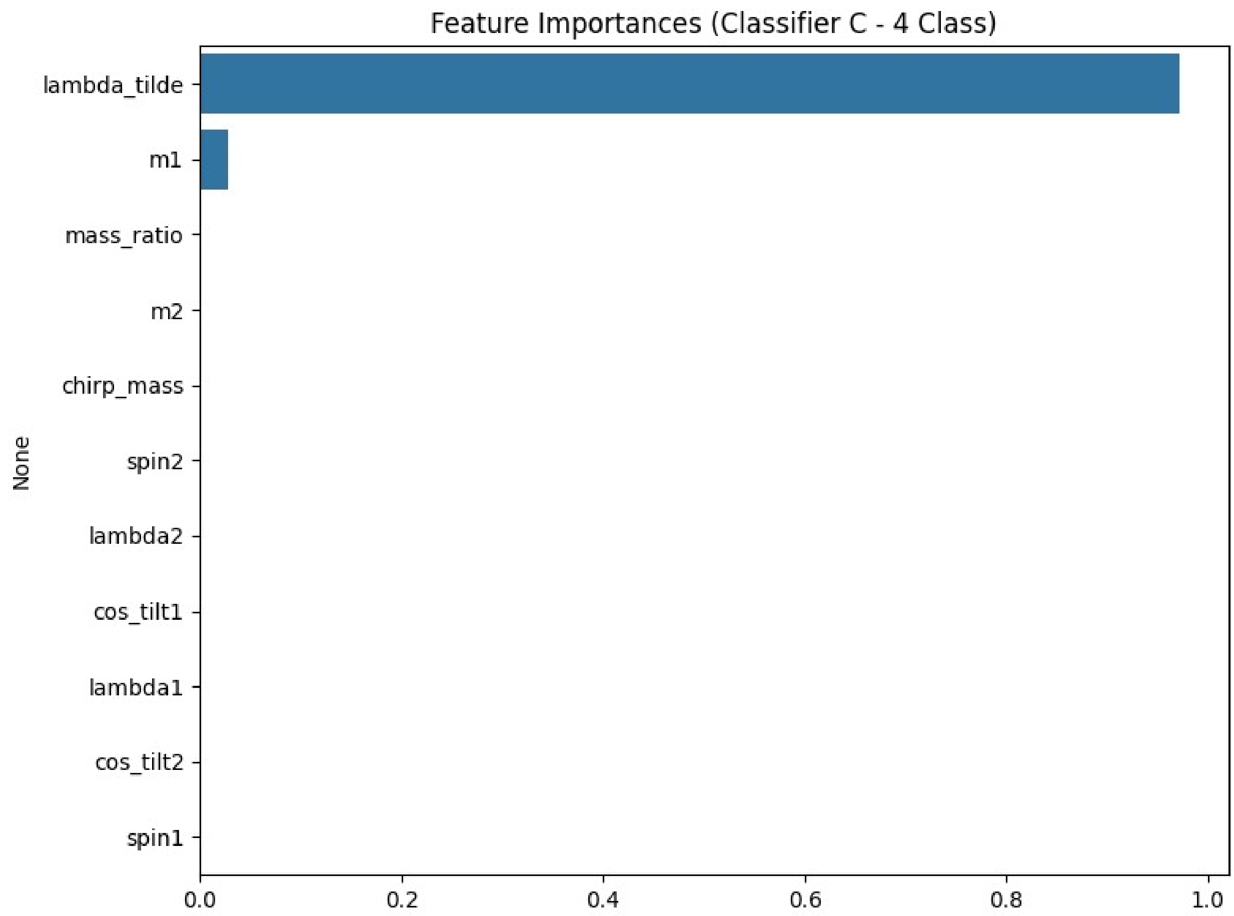
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Encode target
label_encoder = LabelEncoder()
df['label_encoded'] =
label_encoder.fit_transform(df['remnant_label_4class'])
print("Class mapping:", dict(zip(label_encoder.classes_,
label_encoder.transform(label_encoder.classes_))))

# Feature selection
features = ['m1', 'm2', 'mass_ratio', 'chirp_mass', 'lambda1',
'lambda2', 'lambda_tilde',
            'spin1', 'spin2', 'cos_tilt1', 'cos_tilt2']
X = df[features]
y = df['label_encoded']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)
```

```
Class mapping: {'Long-lived HMNS': np.int64(0), 'Prompt BH':
np.int64(1), 'Short-lived HMNS': np.int64(2), 'Stable NS':
np.int64(3)}
```

```
df['label_encoded'] =
label_encoder.fit_transform(df['remnant_label_4class'])

print("Class mapping:", dict(zip(label_encoder.classes_,
label_encoder.transform(label_encoder.classes_))))

# Features and target
features = ['m1', 'm2', 'mass_ratio', 'chirp_mass', 'lambda1',
'lambda2',
            'lambda_tilde', 'spin1', 'spin2', 'cos_tilt1',
'cos_tilt2']
X = df[features]
y = df['label_encoded']

# Train-test split (90% train / 10% validation)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, stratify=y, random_state=42)

# Initialize Gradient Boosting Classifier
clf = GradientBoostingClassifier(n_estimators=300, learning_rate=0.03,
max_depth=5, random_state=42)
clf.fit(X_train, y_train)

# Predict
y_pred = clf.predict(X_test)
```

```python
# Evaluation metrics
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred, target_names=label_encoder.classes_))
print("MCC:", matthews_corrcoef(y_test, y_pred))

# Feature importances
importances = pd.Series(clf.feature_importances_,
index=features).sort_values(ascending=False)
plt.figure(figsize=(8, 6))
sns.barplot(x=importances.values, y=importances.index)
plt.title("Feature Importances (Classifier C - 4 Class)")
plt.tight_layout()
plt.show()
```

```
Class mapping: {'Long-lived HMNS': np.int64(0), 'Prompt BH':
np.int64(1), 'Short-lived HMNS': np.int64(2), 'Stable NS':
np.int64(3)}
Confusion Matrix:
 [[462   0   0   0]
 [  0 119   0   0]
 [  0   0  13   0]
 [  0   0   0 214]]

Classification Report:
                   precision    recall  f1-score   support

 Long-lived HMNS       1.00      1.00      1.00       462
       Prompt BH       1.00      1.00      1.00       119
Short-lived HMNS       1.00      1.00      1.00        13
       Stable NS       1.00      1.00      1.00       214

        accuracy                           1.00       808
       macro avg       1.00      1.00      1.00       808
    weighted avg       1.00      1.00      1.00       808

MCC: 1.0
```


Feature Importances (Classifier C - 4 Class)
None
lambda_tilde
m1
mass_ratio
m2
chirp_mass
spin2
lambda2
cos_tilt1
lambda1
cos_tilt2
spin1
0.0
0.2
0.4
0.6
0.8
1.0