CLASSIFICATION OF GRAVITATIONAL WAVES

AYÇA YERLİKAYA
ARZUNUR KÖROĞLU

Project Overview

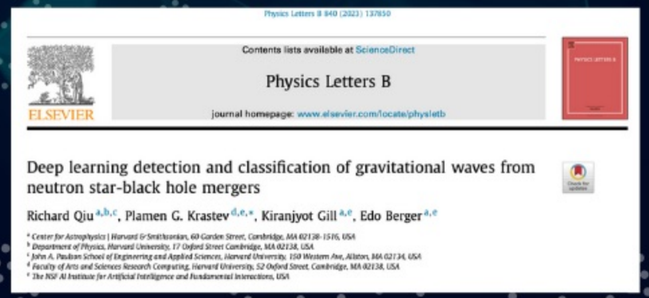
1. Introduction
2. Methodology
3. Data Preprocessing
   i. Data Preparation for Model Training
      - Synthetic Data Generation
      - Noise Generation
4. Model Initialization
5. Model Training
6. Model Testing
   ii. Data Preparation for Model Testing
      - GWOSC Data Acquisition
      - Bandpass Filtering
7. Conclusion

the aims and goals of the project are mentioned.

The articles, data and github pages used are mentioned.

1. Introduction

The aim of the project is to generate synthetic gravitational wave signals, including BBH, BNS, and NSBH events, and classify these events using Long Short-Term Memory (LSTM) networks.


Physics Letters B 840 (2023) 137850

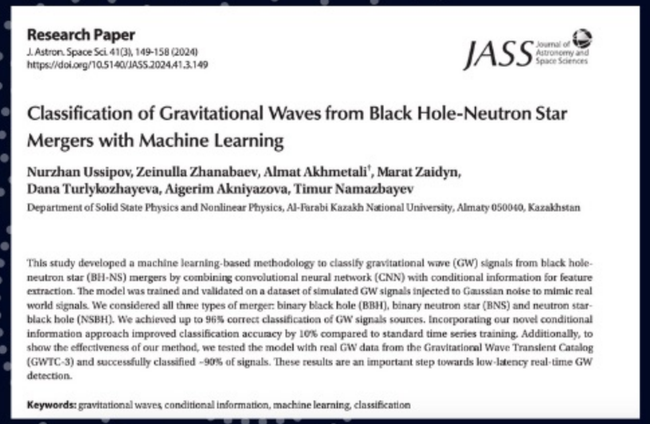
Contents lists available at ScienceDirect

Physics Letters B

journal homepage: www.elsevier.com/locate/physletb

Deep learning detection and classification of gravitational waves from neutron star-black hole mergers

Richard Qiu [a,b,c], Plamen G. Krastev [d,e,*], Kiranjyot Gill [a,e], Edo Berger [a,e]


Research Paper
J. Astron. Space Sci. 41(3), 149-158 (2024)
https://doi.org/10.5140/JASS.2024.41.3.149

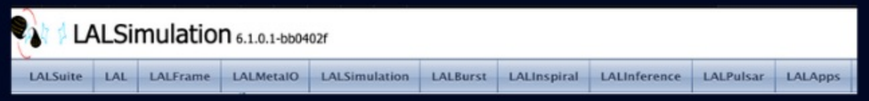
JASS Journal of Astronomy and Space Sciences

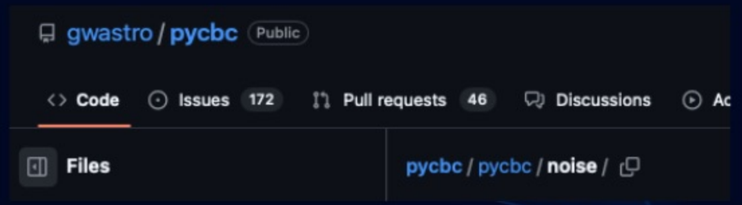
Classification of Gravitational Waves from Black Hole-Neutron Star Mergers with Machine Learning

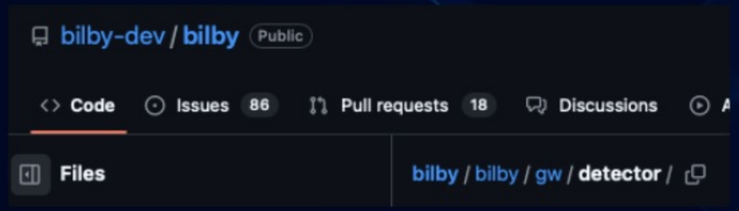
Nurzhan Ussipov, Zeinulla Zhanabaev, Almat Akhmetali†, Marat Zaidyn, Dana Turlykozhayeva, Aigerim Akniyazova, Timur Namazbayev
Department of Solid State Physics and Nonlinear Physics, Al-Farabi Kazakh National University, Almaty 050040, Kazakhstan

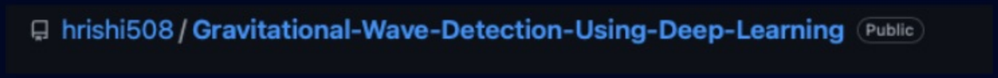
This study developed a machine learning-based methodology to classify gravitational wave (GW) signals from black hole-neutron star (BH-NS) mergers by combining convolutional neural network (CNN) with conditional information for feature extraction. The model was trained and validated on a dataset of simulated GW signals injected to Gaussian noise to mimic real world signals. We considered all three types of merger: binary black hole (BBH), binary neutron star (BNS) and neutron star-black hole (NSBH). We achieved up to 96% correct classification of GW signals sources. Incorporating our novel conditional information approach improved classification accuracy by 10% compared to standard time series training. Additionally, to show the effectiveness of our method, we tested the model with real GW data from the Gravitational Wave Transient Catalog (GWTC-3) and successfully classified ~90% of signals. These results are an important step towards low-latency real-time GW detection.

Keywords: gravitational waves, conditional information, machine learning, classification









overviewdan farklı olarak proje süreci anlatılır(kısaca özet)

2. Methodology

Data Preprocessing

Simulated Signal Generation

Noise Generation

Normalization

Tools Used: LALSimulation, Bilby, PyCBC, GWpy.

Model Initialization

Preprocessing Pipeline

Bandpass Filtering

Parameter Tuning

LSTM Architecture

3. Data Preprocessing

i. Data Preparation for Model Training

Synthetic Data Generation

```
# parameters
solar_mass = bilby.core.utils.constants.solar_mass
sampling_frequency = 4096
deltaT = 1 / sampling_frequency
duration = 32
post_trigger_duration = 0.5
pre_trigger_duration = duration - post_trigger_duration
```

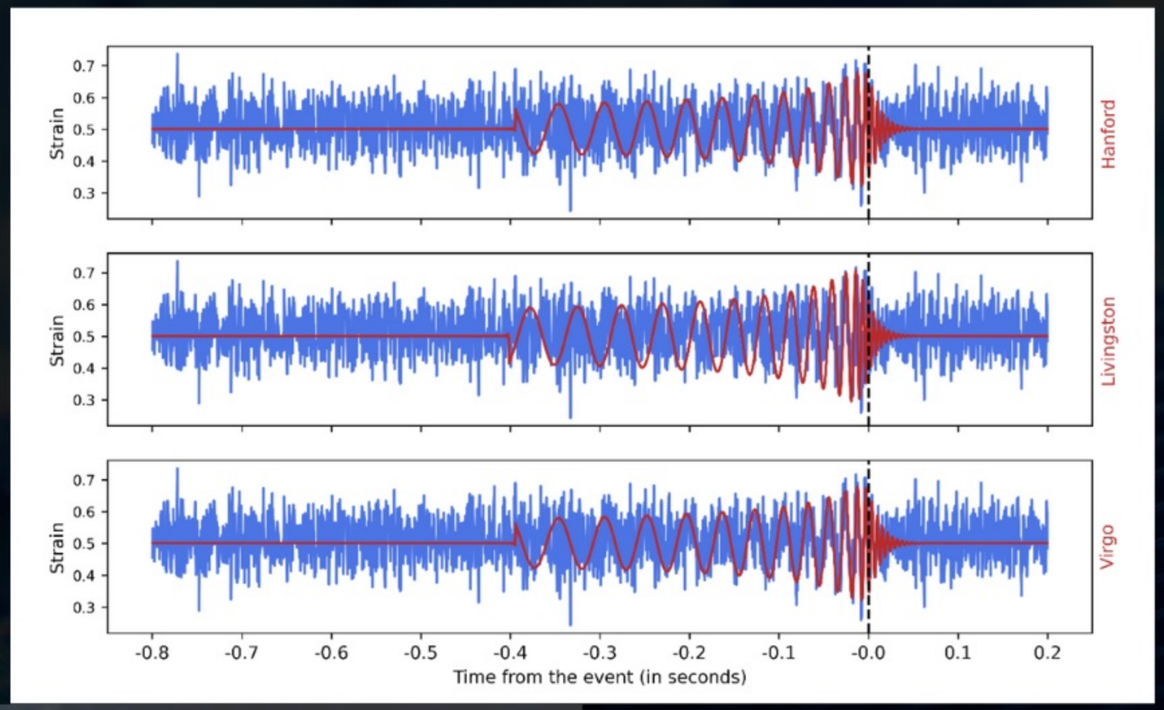
```
h_plus = h_plus_timeseries.data.data
h_cross = h_cross_timeseries.data.data
h_plus_time = np.arange(len(h_plus)) * h_plus_timeseries.deltaT + float(h_plus_timeseries.epoch)
```

```
h_plus_timeseries, h_cross_timeseries = lalsim.SimInspiralChooseTDWaveform(
    mass_1, mass_2, spin_1x, spin_1y, spin_1z, spin_2x, spin_2y, spin_2z,
    luminosity_distance, theta_jn, phase, longAscNodes, eccentricity, meanPerAno,
    deltaT, f_min, f_ref, LALParams, approximant
)
```

i. Data Preparation for Model Training

Synthetic Data Generation

```
# detectors
#H1 = bilby.gw.detector.get_empty_interferometer("H1")
L1 = bilby.gw.detector.get_empty_interferometer("L1")
#V1 = bilby.gw.detector.get_empty_interferometer("V1")
```




```
# interpolation
#_, h_interp_H1 = interpolate_strain(strain_H1, h_plus_time, geocent_time, sampling_frequency, duration)
_, h_interp_L1 = interpolate_strain(strain_L1, h_plus_time, geocent_time, sampling_frequency, duration)
#_, h_interp_V1 = interpolate_strain(strain_V1, h_plus_time, geocent_time, sampling_frequency, duration)
```

i. Data Preparation for Model Training

Synthetic Data Generation

Binary Black Hole (BBH)

```
for i in range(num_samples):
    geocent_time = get_random_geocentric_time()
    mass_1 = np.random.uniform(2, 95) * solar_mass
    mass_2 = np.random.uniform(2, min(mass_1 / 10, 95)) * solar_mass
    spin_1z = np.random.uniform(-0.70, 0.70)
    spin_2z = np.random.uniform(-0.70, 0.70)
    spin_1x, spin_1y = 0, 0
    spin_2x, spin_2y = 0, 0
    luminosity_distance = np.random.uniform(50, 500)
    theta_jn = np.random.uniform(0, np.pi)
    phase = np.random.uniform(0, 2 * np.pi)
    longAscNodes = 0
    eccentricity = 0
    meanPerAno = 0
    LALParams = lal.CreateDict()
    waveform_approximant = "SEOBNRv4"
    approximant = lalsim.GetApproximantFromString(waveform_approximant)
```

Binary Black Hole (BBH)

```python
bbh_data.head()
```

[43] ✓ 0.0s Python

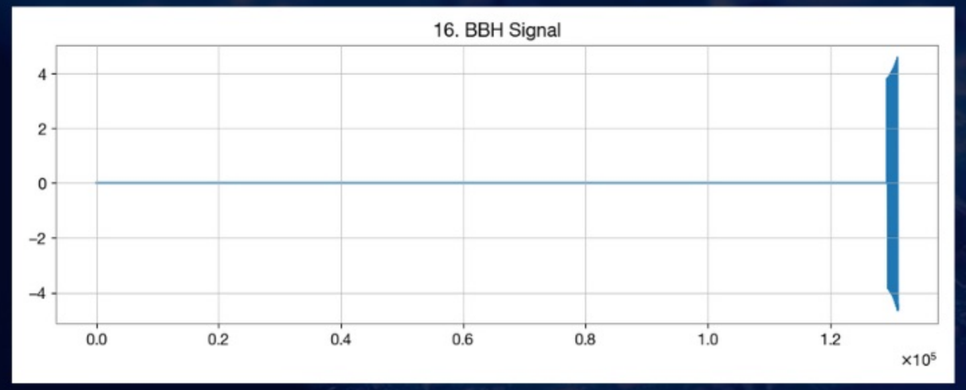
	L1_0	L1_1	L1_2	L1_3	L1_4	L1_5	L1_6	L1_7	L1_8	L1_9	...	L1_131063	L1_131064	L1_131065	L1_131066	L1_131067	L1_131068	L1_131069	L1_131070	L1_131071	label
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

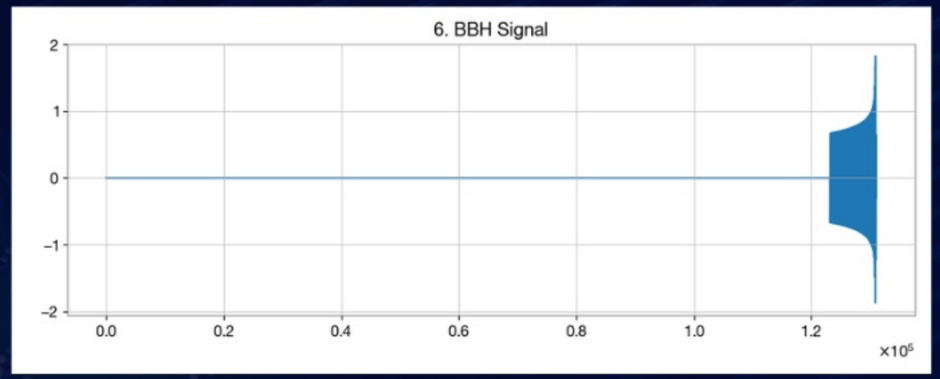
5 rows × 131073 columns

```python
bbh_data.shape
bns_data.shape
nsbh_data.shape
```

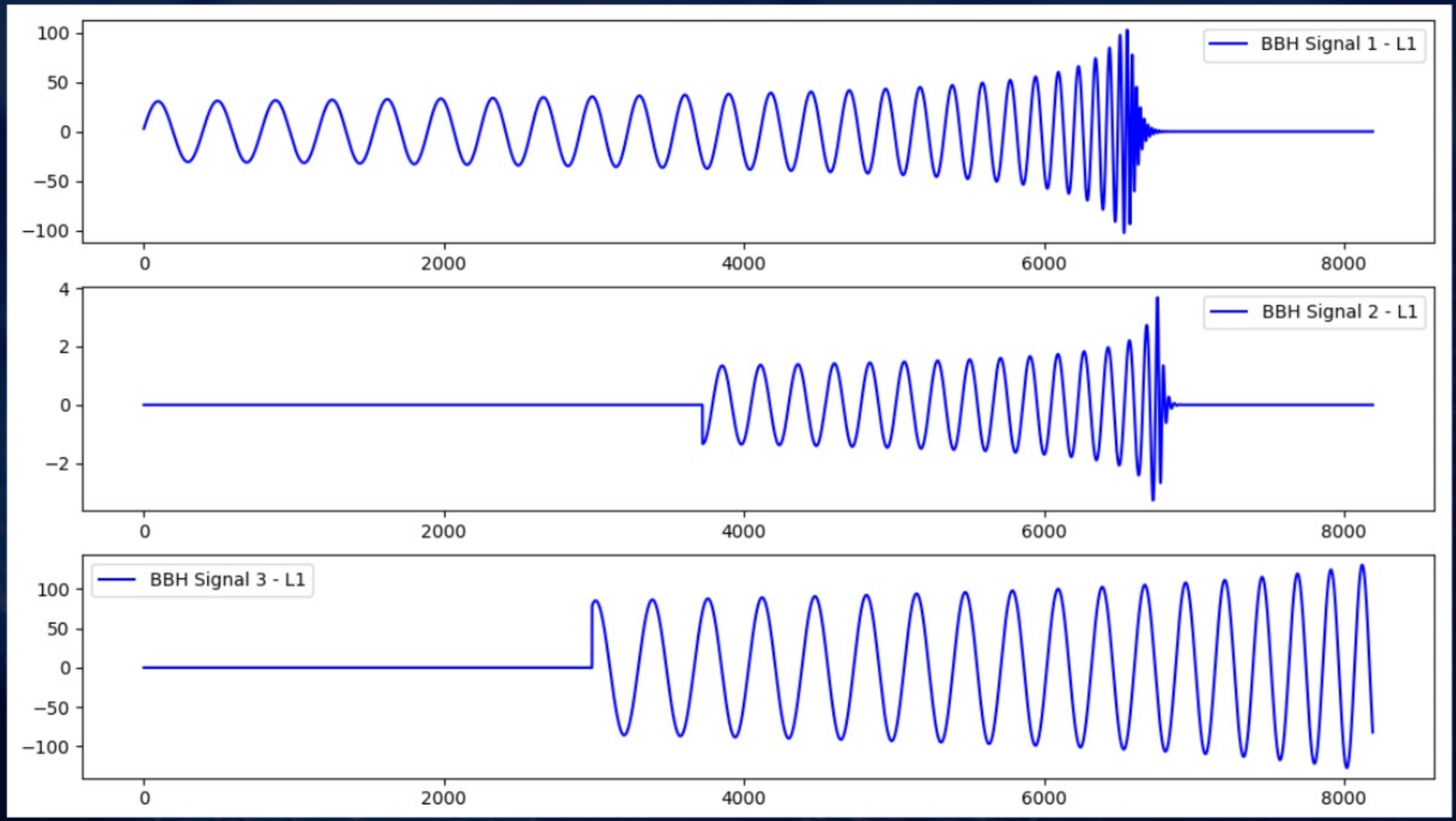
[51] ✓ 0.0s

```
(1000, 131073)
```





Binary Black Hole (BBH)



i. Data Preparation for Model Training

Synthetic Data Generation

Binary Neutron Star (BNS)

```
for i in range(num_samples):
    geocent_time = get_random_geocentric_time()
    mass_1 = np.random.uniform(1.1, 1.6) * solar_mass
    mass_2 = np.random.uniform(1.1, 1.6) * solar_mass
    spin_1z = np.random.uniform(-0.02, 0.02)
    spin_2z = np.random.uniform(-0.02, 0.02)
    spin_1x, spin_1y = 0, 0
    spin_2x, spin_2y = 0, 0
    luminosity_distance = np.random.uniform(50, 500)
    theta_jn = np.random.uniform(0, np.pi)
    phase = np.random.uniform(0, 2 * np.pi)
    longAscNodes = 0
    eccentricity = 0
    meanPerAno = 0
    LALParams = lal.CreateDict()
    waveform_approximant = "IMRPhenomPv2_NRTidal"
    approximant = lalsim.GetApproximantFromString(waveform_approximant)
```

Binary Neutron Star (BNS)

```python
bns_data.head()
```

[53] ✓ 0.0s Python

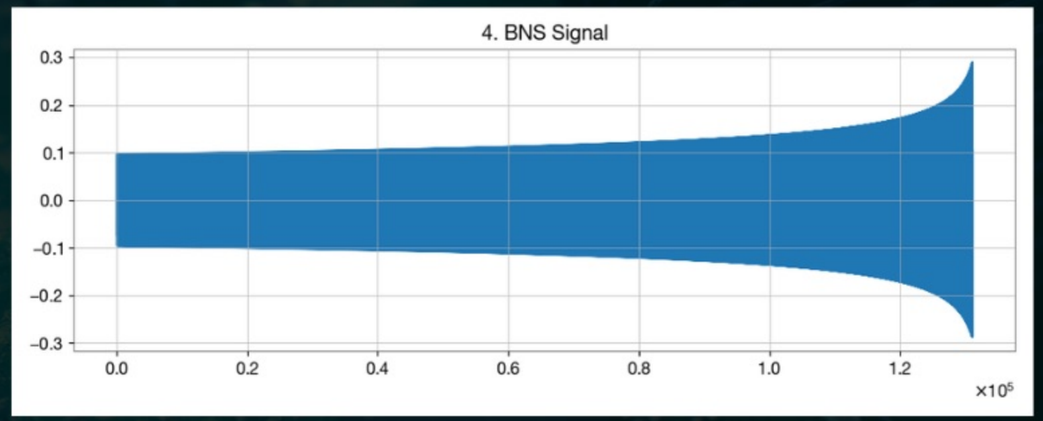
	L1_0	L1_1	L1_2	L1_3	L1_4	L1_5	L1_6	L1_7	L1_8	L1_9	...	L1_131063	L1_131064	L1_131065	L1_131066	L1_131067	L1_131068	L1_131069	L1_131070	L1_131071	label
0	-0.045258	-0.047824	-0.050201	-0.052354	-0.054301	-0.056005	-0.057487	-0.058714	-0.059707	-0.060436	...	-0.125306	-0.187208	-0.212352	-0.195306	-0.139740	-0.056213	0.038373	0.125628	0.187801	1
1	-0.003312	-0.004254	-0.005180	-0.006087	-0.006972	-0.007833	-0.008665	-0.009465	-0.010231	-0.010961	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1
2	-0.129017	-0.136961	-0.144463	-0.151532	-0.158113	-0.164220	-0.169799	-0.174867	-0.179374	-0.183341	...	-0.541419	-0.470747	-0.359498	-0.218981	-0.058980	0.105065	0.261356	0.394722	0.495913	1
3	-0.073447	-0.073040	-0.072405	-0.071545	-0.070463	-0.069162	-0.067645	-0.065918	-0.063986	-0.061855	...	-0.087052	-0.032275	0.024981	0.080363	0.129652	0.169089	0.195662	0.207334	0.203201	1
4	-0.052902	-0.053543	-0.053990	-0.054259	-0.054332	-0.054225	-0.053922	-0.053441	-0.052766	-0.051918	...	-0.198778	-0.295427	-0.209970	-0.011157	0.206066	0.291710	0.214297	-0.003272	-0.209450	1

5 rows × 131073 columns

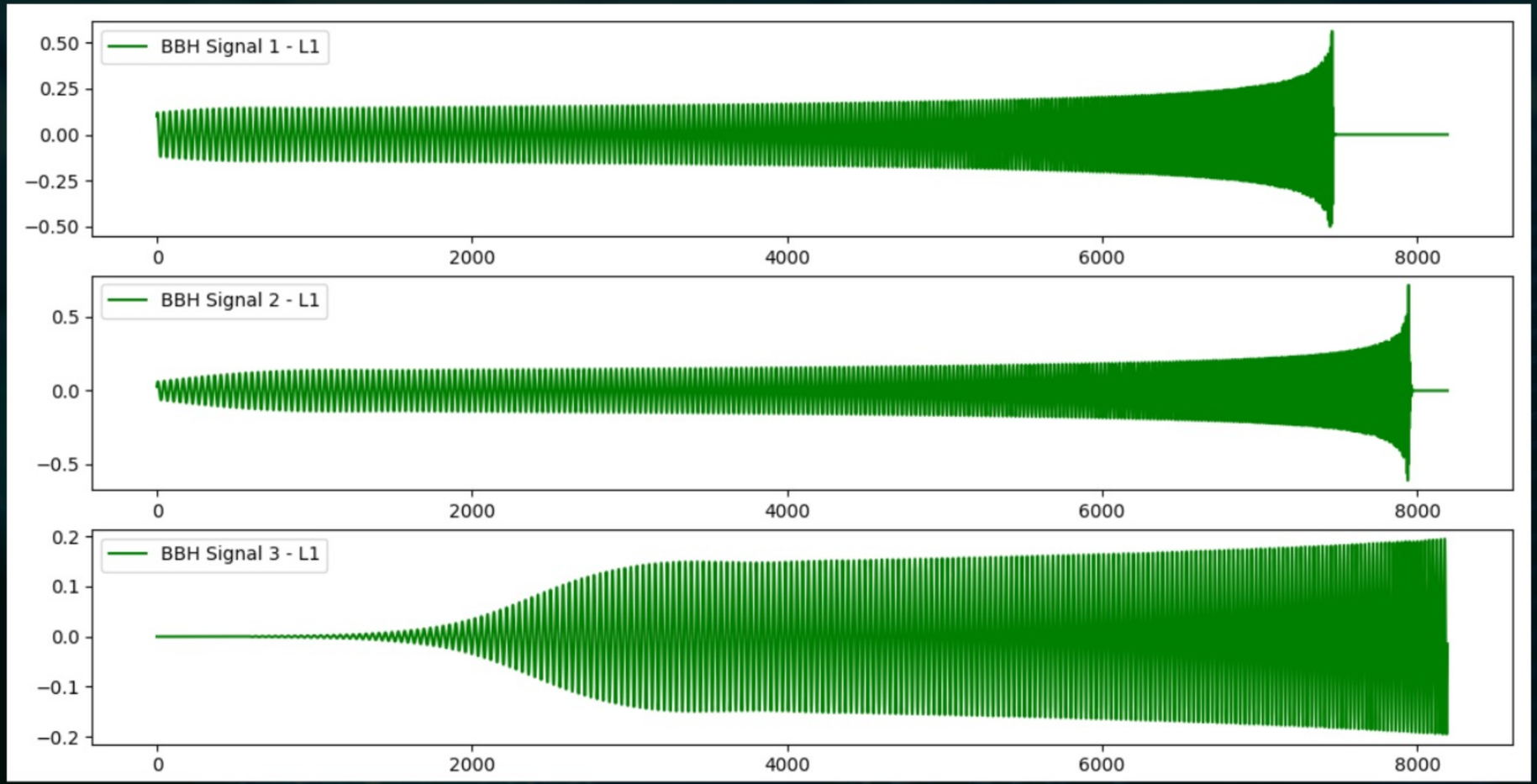
```python
bbh_data.shape
bns_data.shape
nsbh_data.shape
```

[51] ✓ 0.0s

```
(1000, 131073)
```



Binary Neutron Star (BNS)



Neutron Star-Black Hole (NSBH)

```
nsbh_data.head()
```

[45] ✓ 0.0s Python

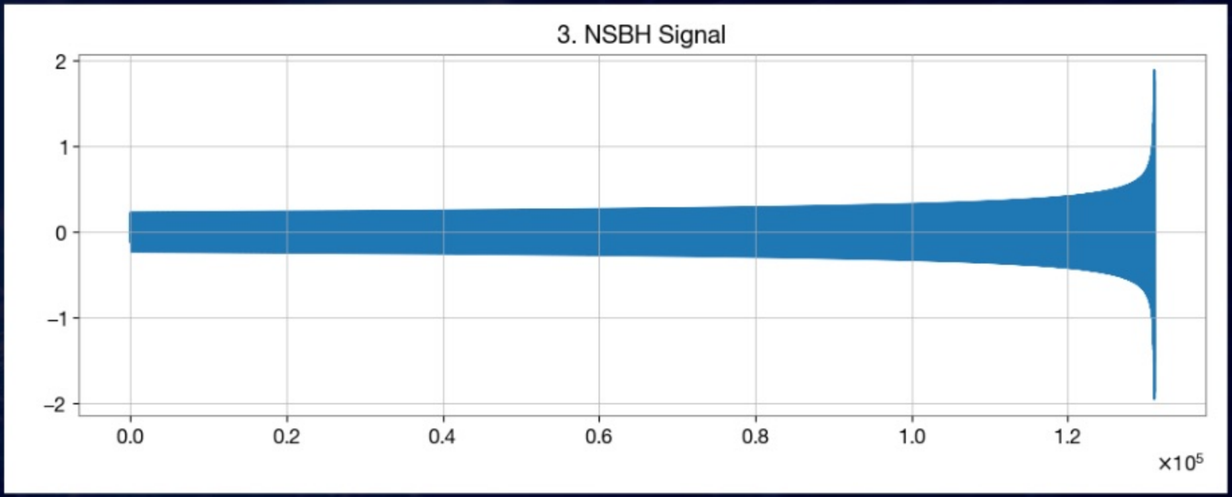
	L1_0	L1_1	L1_2	L1_3	L1_4	L1_5	L1_6	L1_7	L1_8	L1_9	...	L1_131063	L1_131064	L1_131065	L1_131066	L1_131067	L1_131068	L1_131069	L1_131070	L1_131071	label
0	-0.040690	-0.040551	-0.040364	-0.040131	-0.039853	-0.039527	-0.039157	-0.038741	-0.038281	-0.037776	...	0.000221	-0.000124	0.000101	-0.000176	-0.000032	-0.000120	-0.000020	-0.000040	0.000000	2
1	-0.046601	-0.043142	-0.039637	-0.036090	-0.032504	-0.028885	-0.025235	-0.021558	-0.017858	-0.014140	...	0.300855	0.270546	0.232435	0.187569	0.137283	0.082976	0.026261	-0.031271	-0.087908	2
2	-0.004760	-0.003904	-0.003044	-0.002181	-0.001316	-0.000450	0.000417	0.001283	0.002148	0.003011	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2
3	0.025845	0.027164	0.028452	0.029709	0.030933	0.032122	0.033275	0.034392	0.035469	0.036508	...	0.010670	0.036751	0.061589	0.084301	0.104122	0.120338	0.132405	0.139879	0.142516	2
4	0.054375	0.054063	0.053689	0.053252	0.052753	0.052193	0.051572	0.050890	0.050150	0.049351	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2

5 rows × 131073 columns

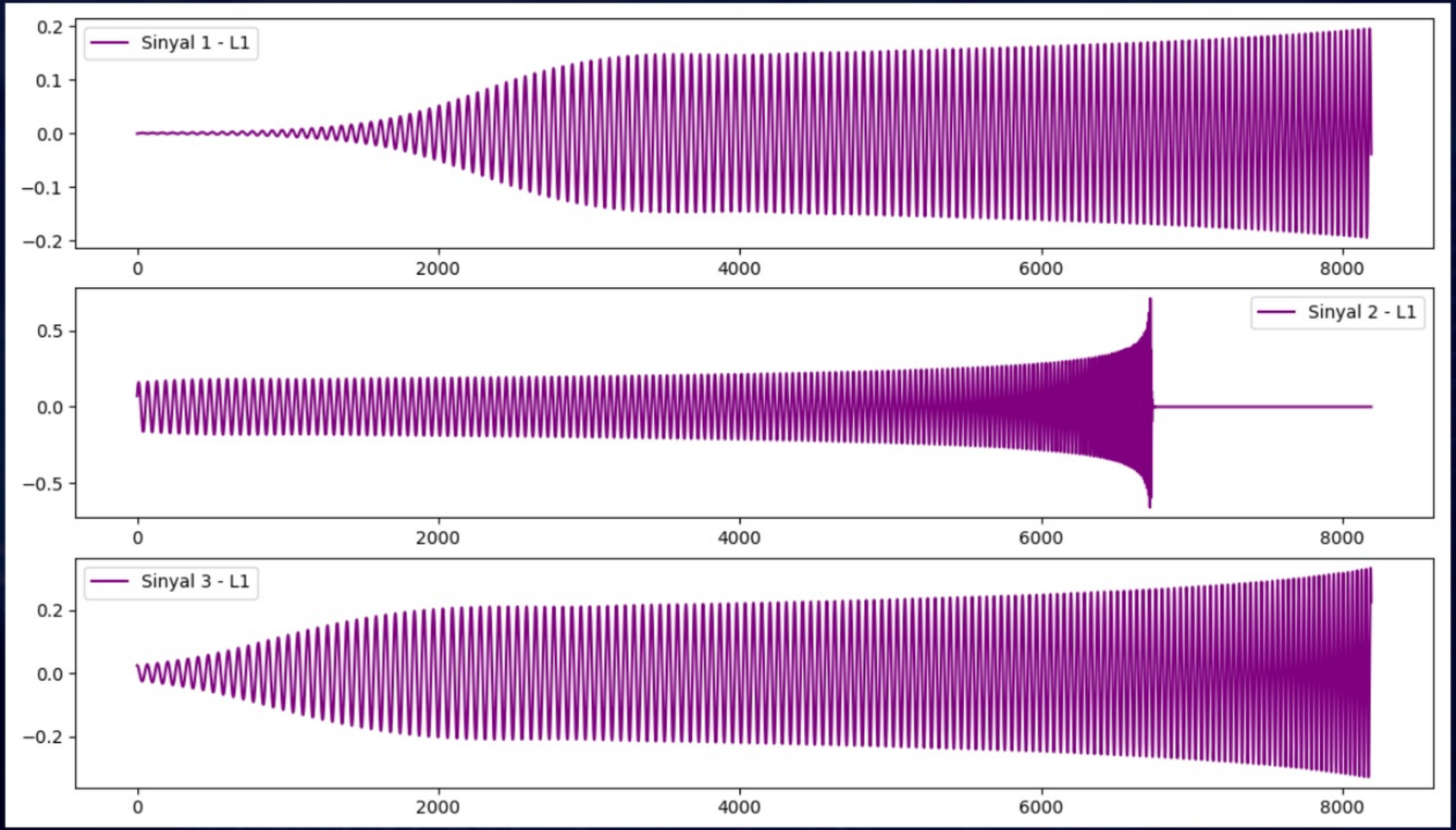
```
bbh_data.shape
bns_data.shape
nsbh_data.shape
```

[51] ✓ 0.0s

```
(1000, 131073)
```



Neutron Star-Black Hole (NSBH)



i. Data Preparation for Model Training

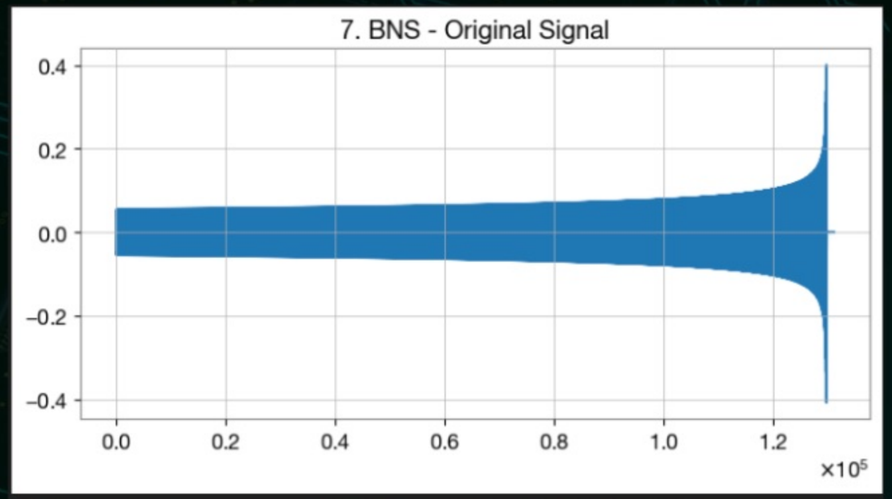
Noise Generation

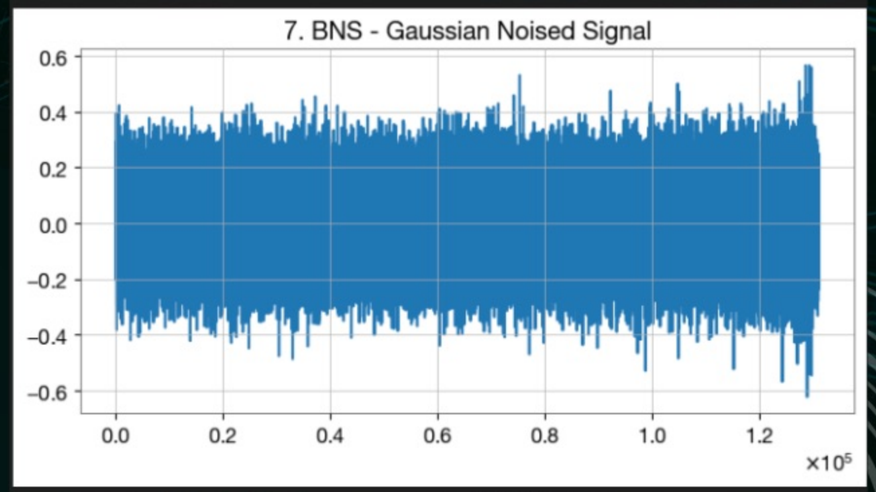
```python
def add_gauss_noise_to_signal(signal, snr=10.0, flow=30.0, delta_f=1.0/16, delta_t=1.0/4096, duration=32):

    flow = np.random.uniform(20.0, 40.0)
    delta_f = np.random.uniform(1.0/32, 1.0/8)
    flen = int(2048 / delta_f) + 1
    psd = pycbc.psd.aLIGOZeroDetHighPower(flen, delta_f, flow)


    delta_t = 1.0 / 4096
    tsamples = int(duration / delta_t)

    # Gaussian noise
    noise = pycbc.noise.noise_from_psd(tsamples, delta_t, psd)

    signal_power = np.mean(np.abs(signal) ** 2)
    noise_power = np.mean(np.abs(noise) ** 2)

    scaling_factor = np.sqrt(signal_power / (noise_power * (10 ** (snr / 10))))
    noise *= scaling_factor

    noise = TimeSeries.from_pycbc(noise)

    signal = TimeSeries(signal, sample_rate=1.0 / delta_t)
    noise_signal = noise.inject(signal)

    return noise_signal.value
```

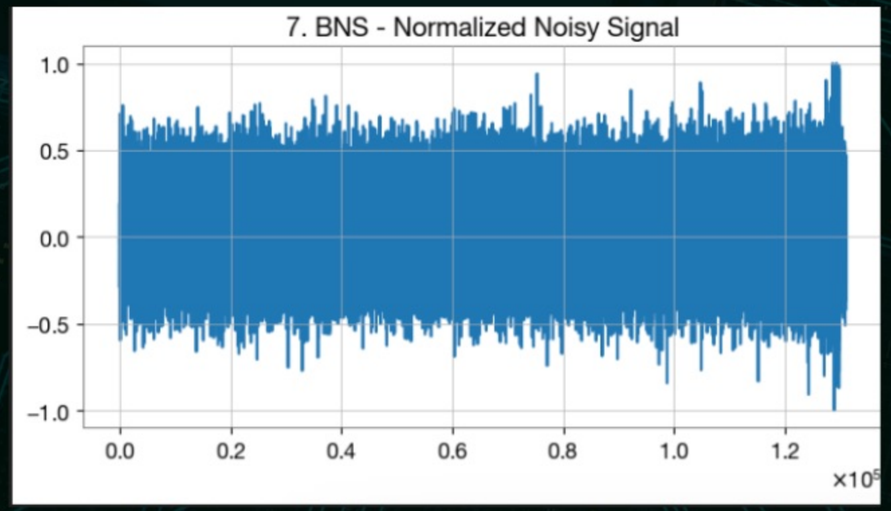
i. Data Preparation for Model Training

Noise Generation

BNS



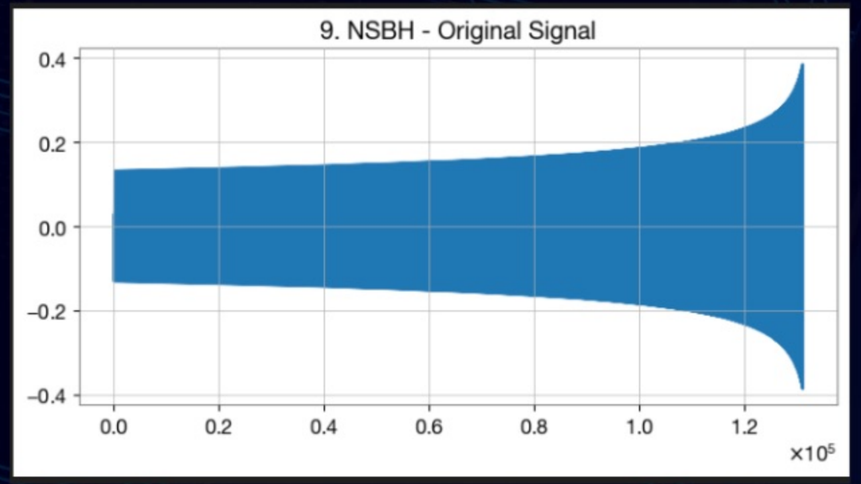


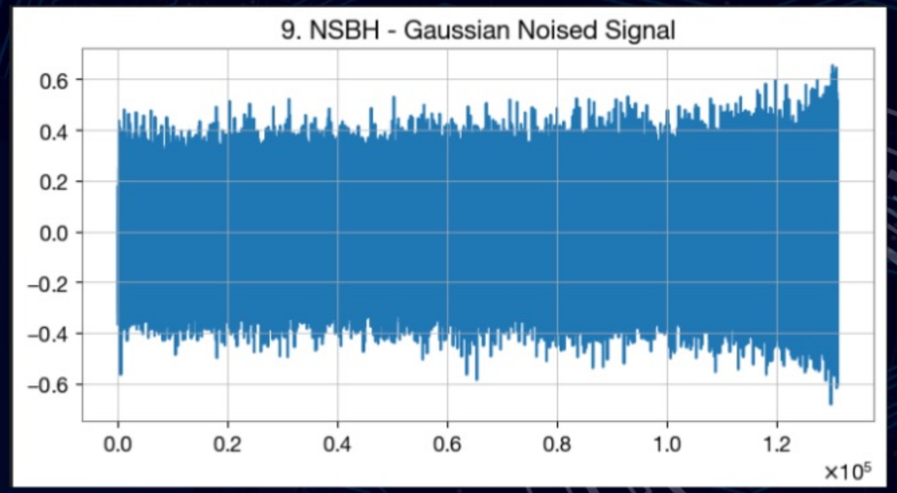


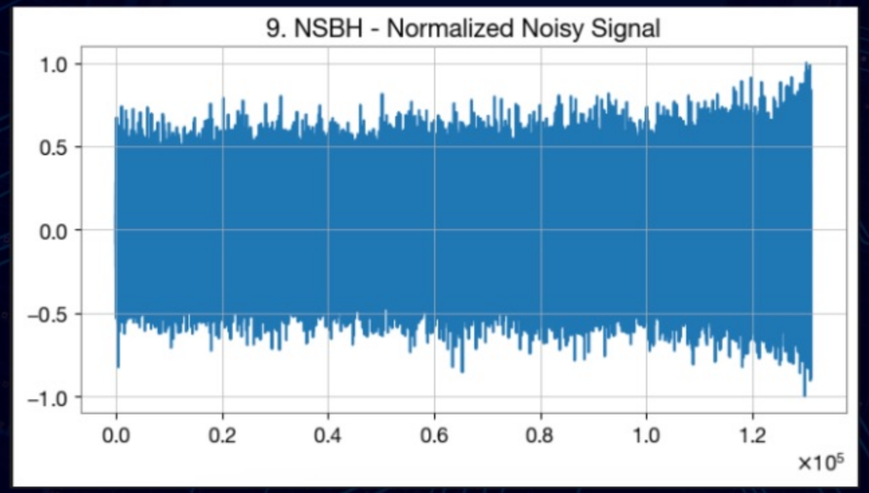
i. Data Preparation for Model Training

Noise Generation

NSBH



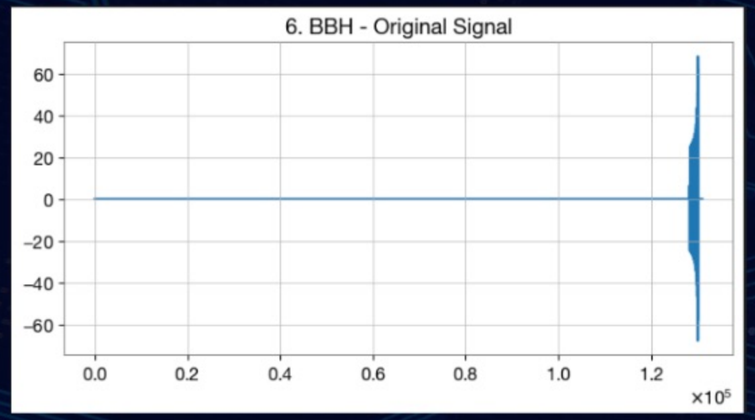


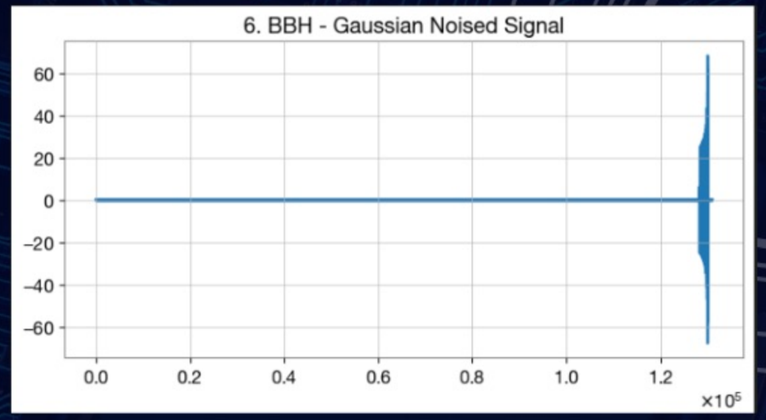


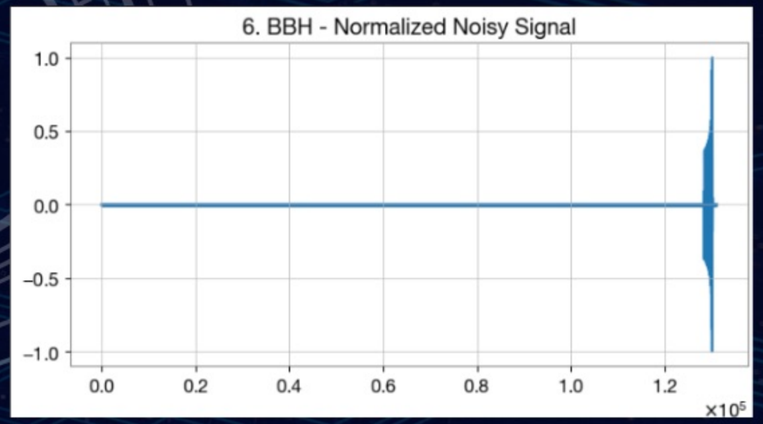
i. Data Preparation for Model Training

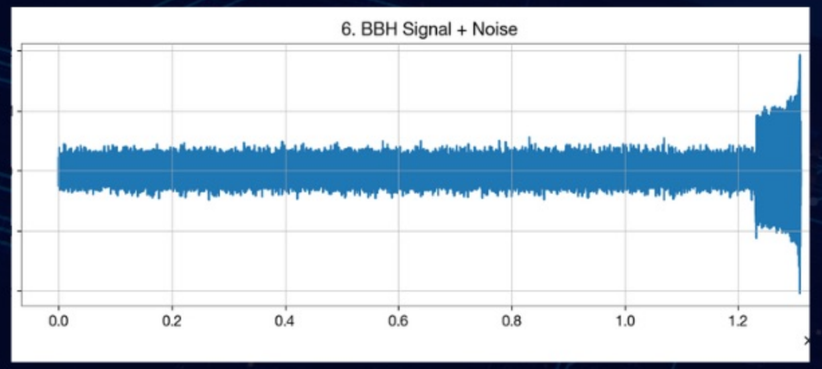
Noise Generation

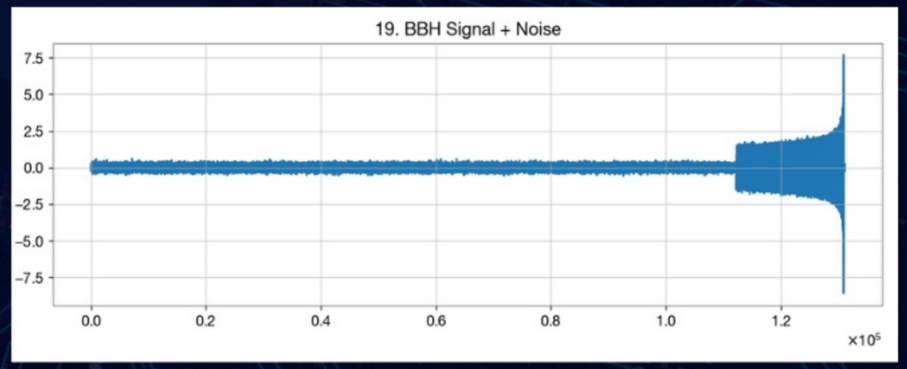
BBH



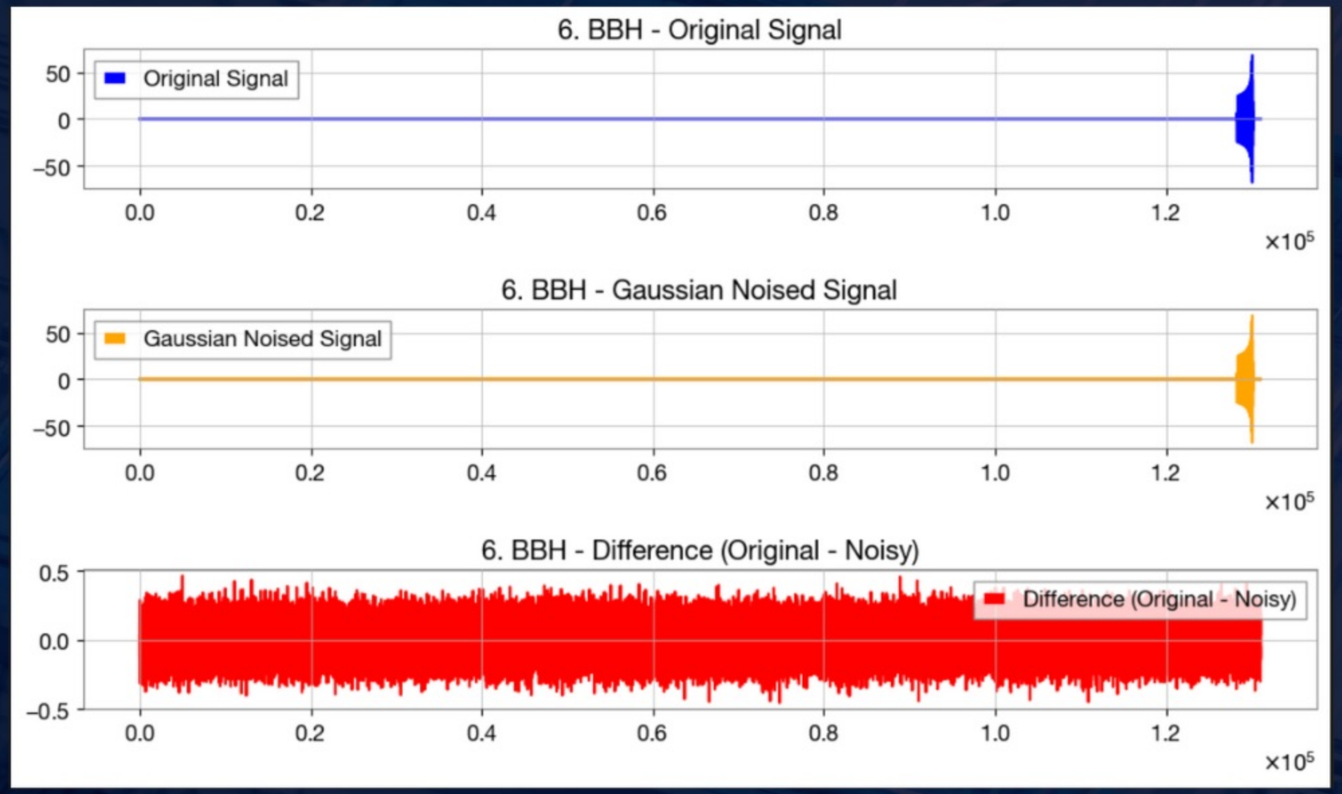






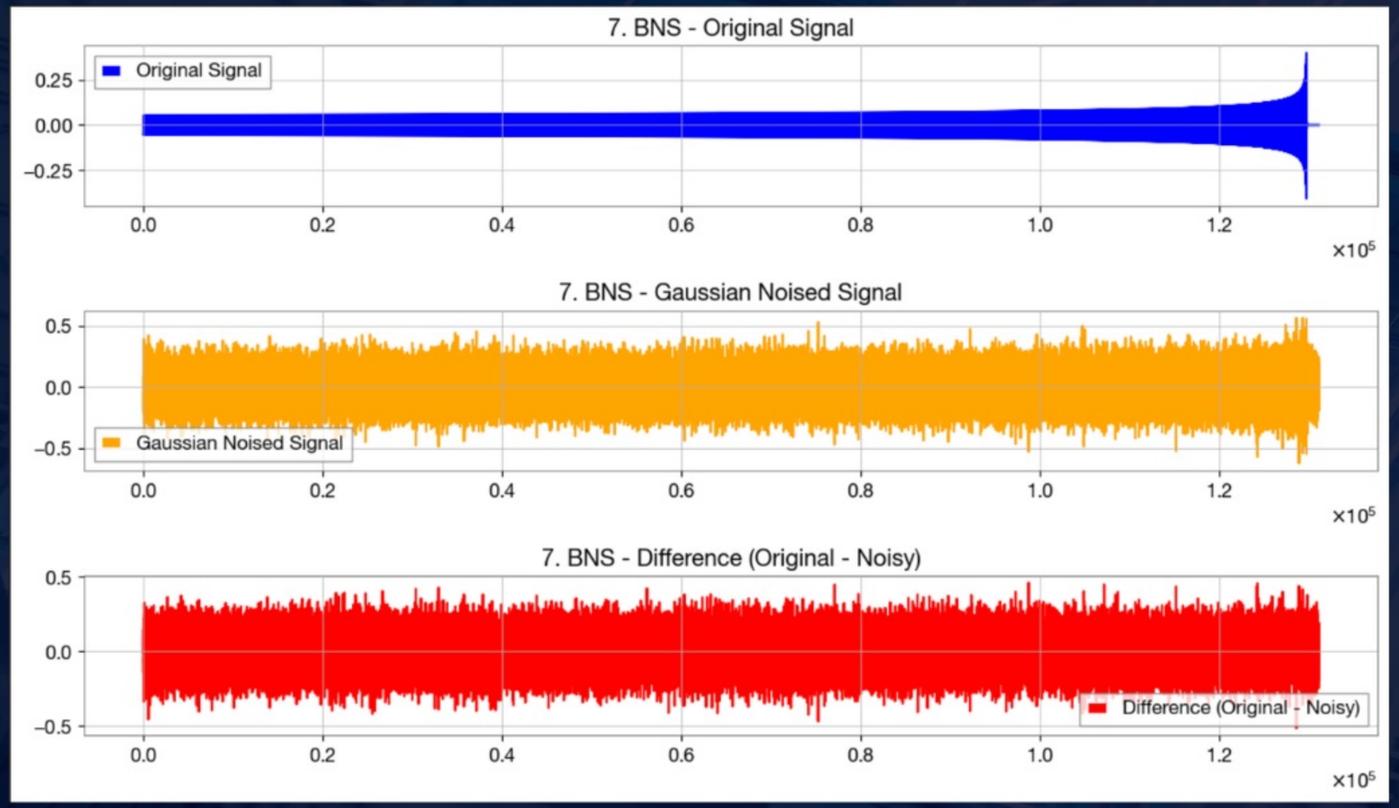


DIFFERENCE



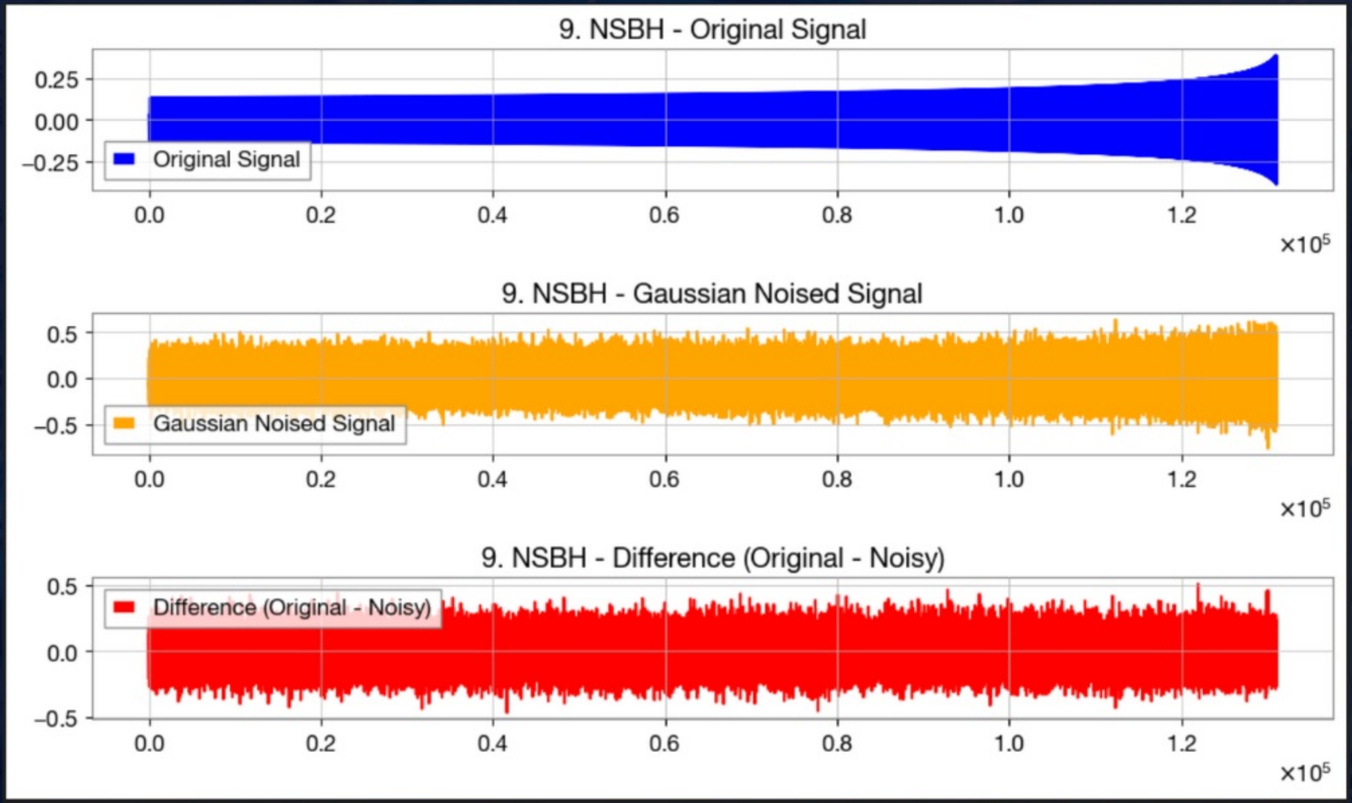
Cosine Similarity (Original vs. Noisy): 0.8925368314523147

DIFFERENCE



Cosine Similarity (Original vs. Noisy): 0.6468039599556746

DIFFERENCE

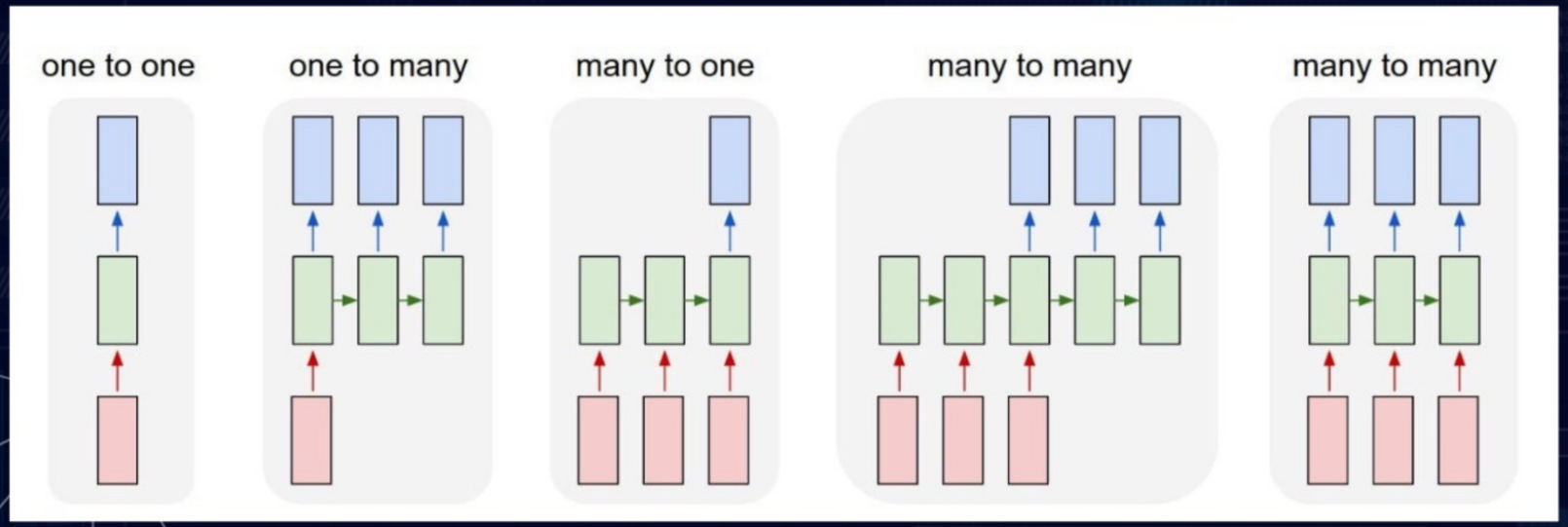


Cosine Similarity (Original vs. Noisy): 0.7544811460474754

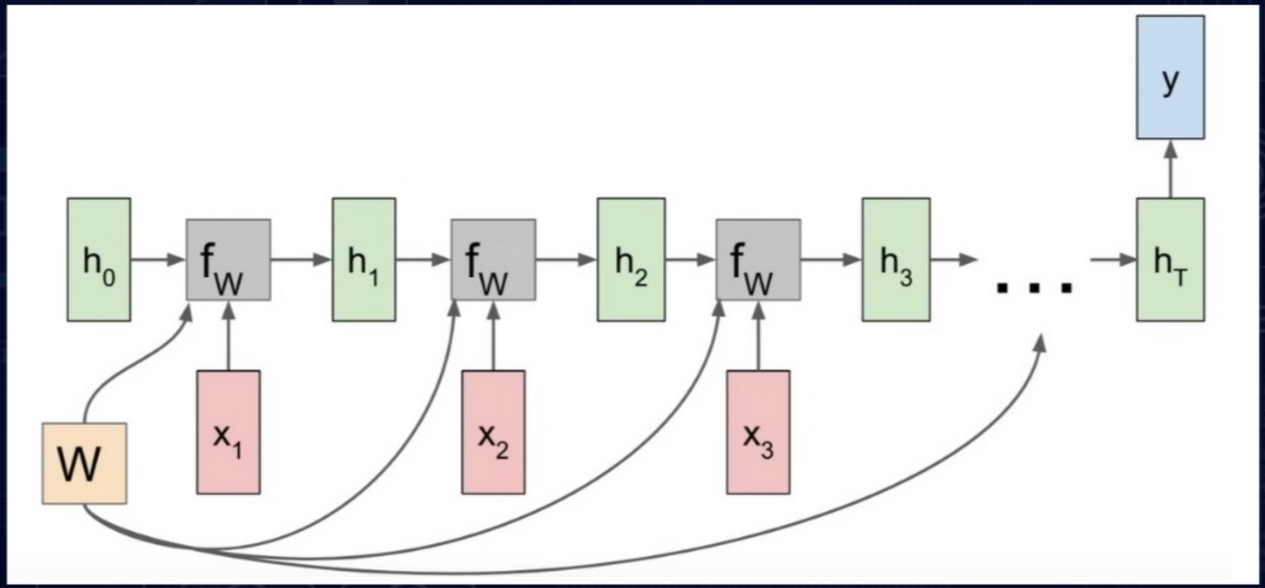
4. Model Initialization

Recurrent Neural Networks (RNNs)

1. RNNs offer a flexible architecture for handling both input and output sequences.
2. They are particularly well-suited for processing sequential data and time-series information.
3. RNNs excel at retaining information from earlier steps in a sequence, enabling them to capture temporal dependencies effectively.



!



- RNNs are capable of retaining information from earlier steps in a sequence, making them effective for capturing temporal dependencies.
- However, they are prone to challenges such as **exploding** or **vanishing** gradients during backpropagation.

!

Long Short Term Memory

- Advanced RNN structures, such as those with cell states and gates, allow for more selective retention of past information.

- These structures are designed to effectively handle long-term dependencies in sequences.

- They also improve gradient flow, minimizing the risk of the vanishing gradient problem.

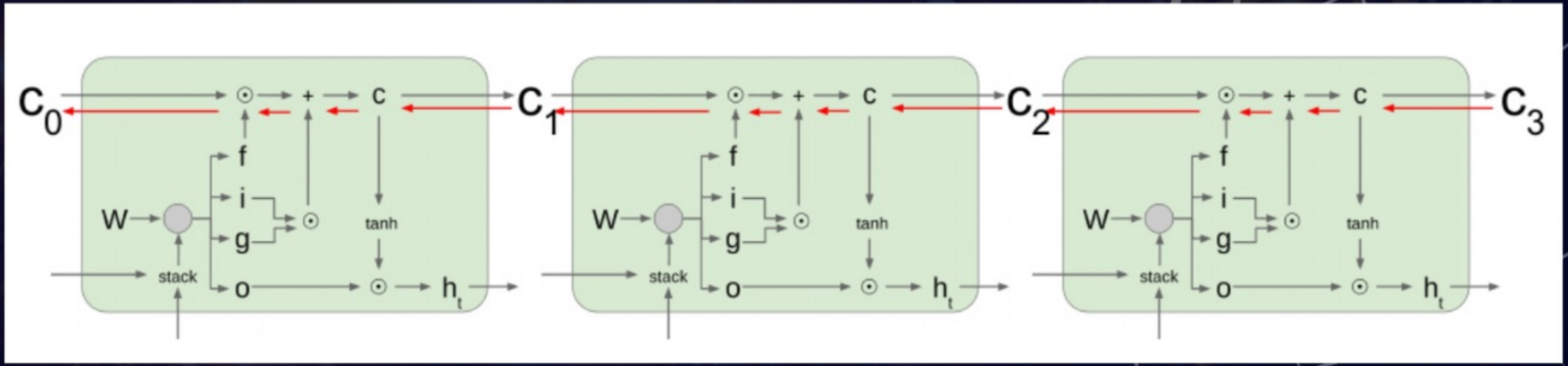
Input Gate
Forget Gate
Output Gate
Candidate Gate

LSTM

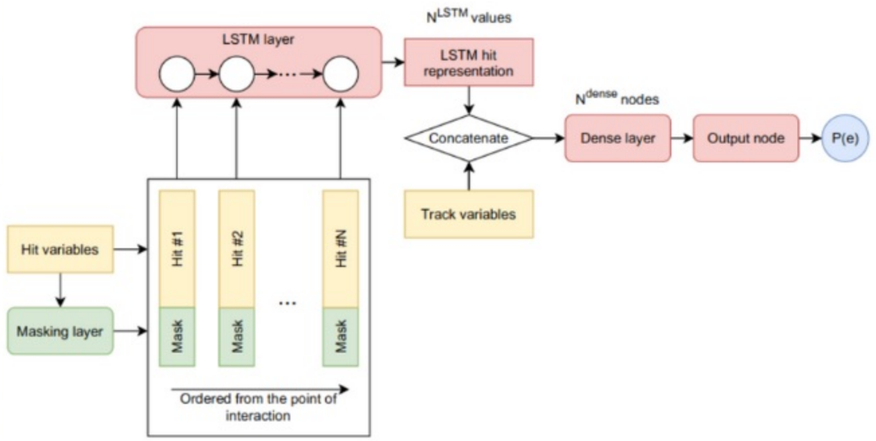
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$



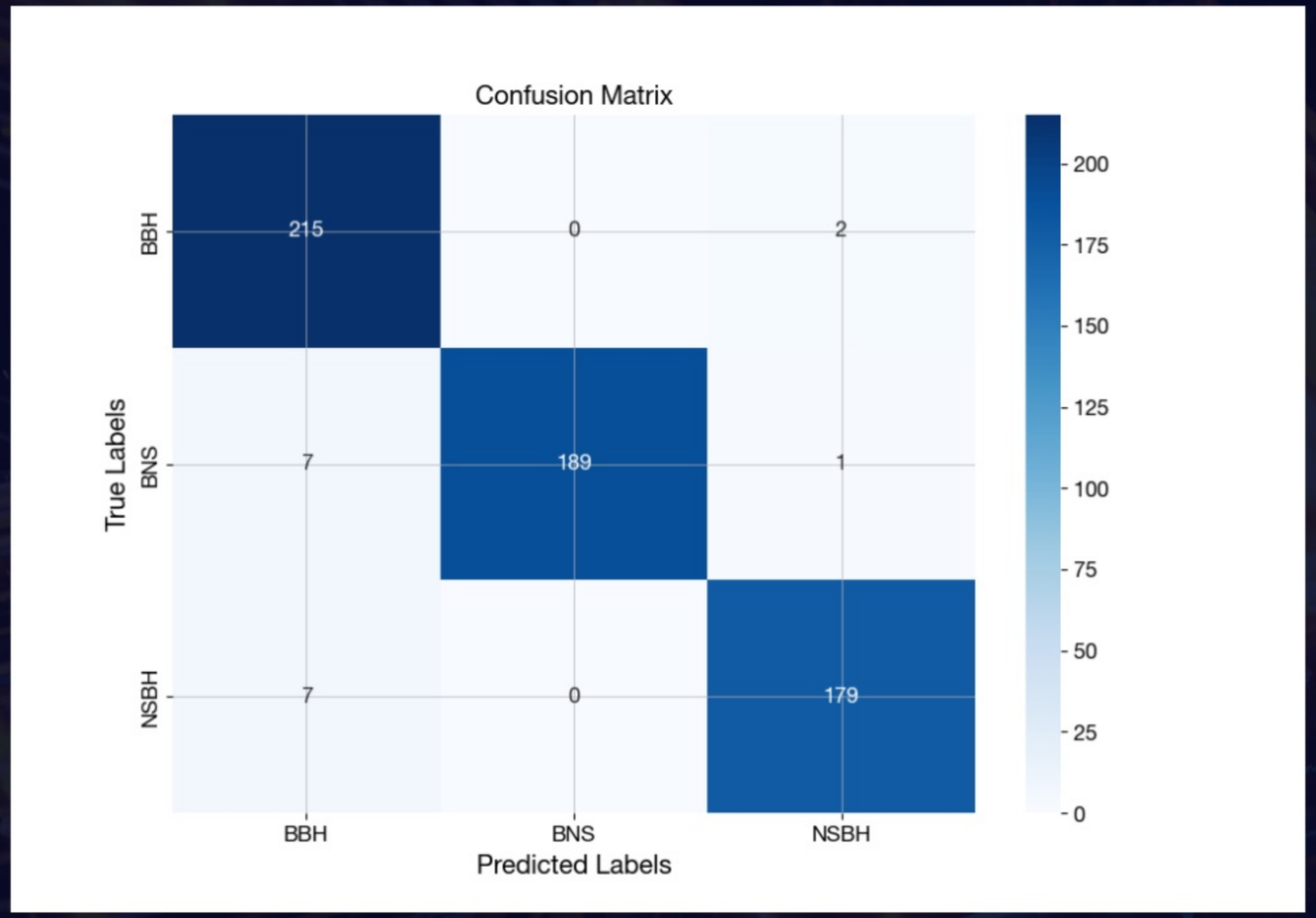
5. Model Training



Hyperparameter	Value
LSTM Layer	
Number of Nodes	16
Activation Function	tanh
Recurrent Activation	hard sigmoid
Dense Layer	
Number of Nodes	16
Activation Function	ReLU
Output Layer	
Number of Nodes	1
Activation Function	sigmoid
Learning Rate	0.0001
Batch Size	256

LSTM Model Hyperparameters

	Component	Hyperparameter	Value
1	LSTM Layer	Number of Layers	2
2	LSTM Layer	Hidden Size	128
3	LSTM Layer	Dropout Probability	0.4
4	LSTM Layer	Bidirectional	True
5	Dense Layer	Number of Nodes	128
6	Dense Layer	Activation Function	ReLU
7	Output Layer	Number of Nodes	3
8	Output Layer	Activation Function	Softmax
9	Training Configuration	Learning Rate	0.001
10	Training Configuration	Batch Size	16
11	Training Configuration	Number of Epochs	20



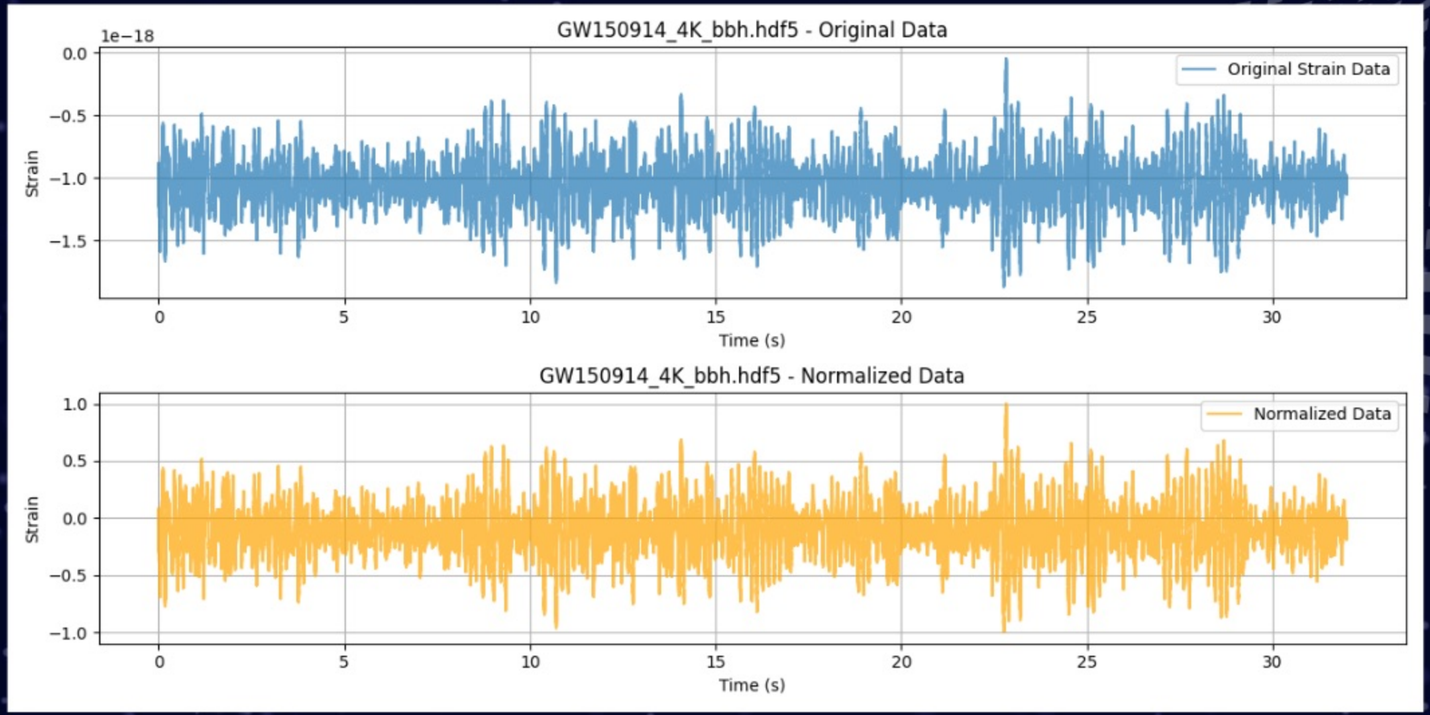

```
Epoch 18, Loss: 0.0824, Train Acc: 97.42%, Test Acc: 97.67%
New best model saved with validation accuracy: 97.67%
```

```
Classification Report:
              precision    recall  f1-score   support

         BBH       0.94      0.99      0.96       217
         BNS       1.00      0.96      0.98       197
        NSBH       0.98      0.96      0.97       186

    accuracy                           0.97       600
   macro avg       0.97      0.97      0.97       600
weighted avg       0.97      0.97      0.97       600
```

6. Model Testing

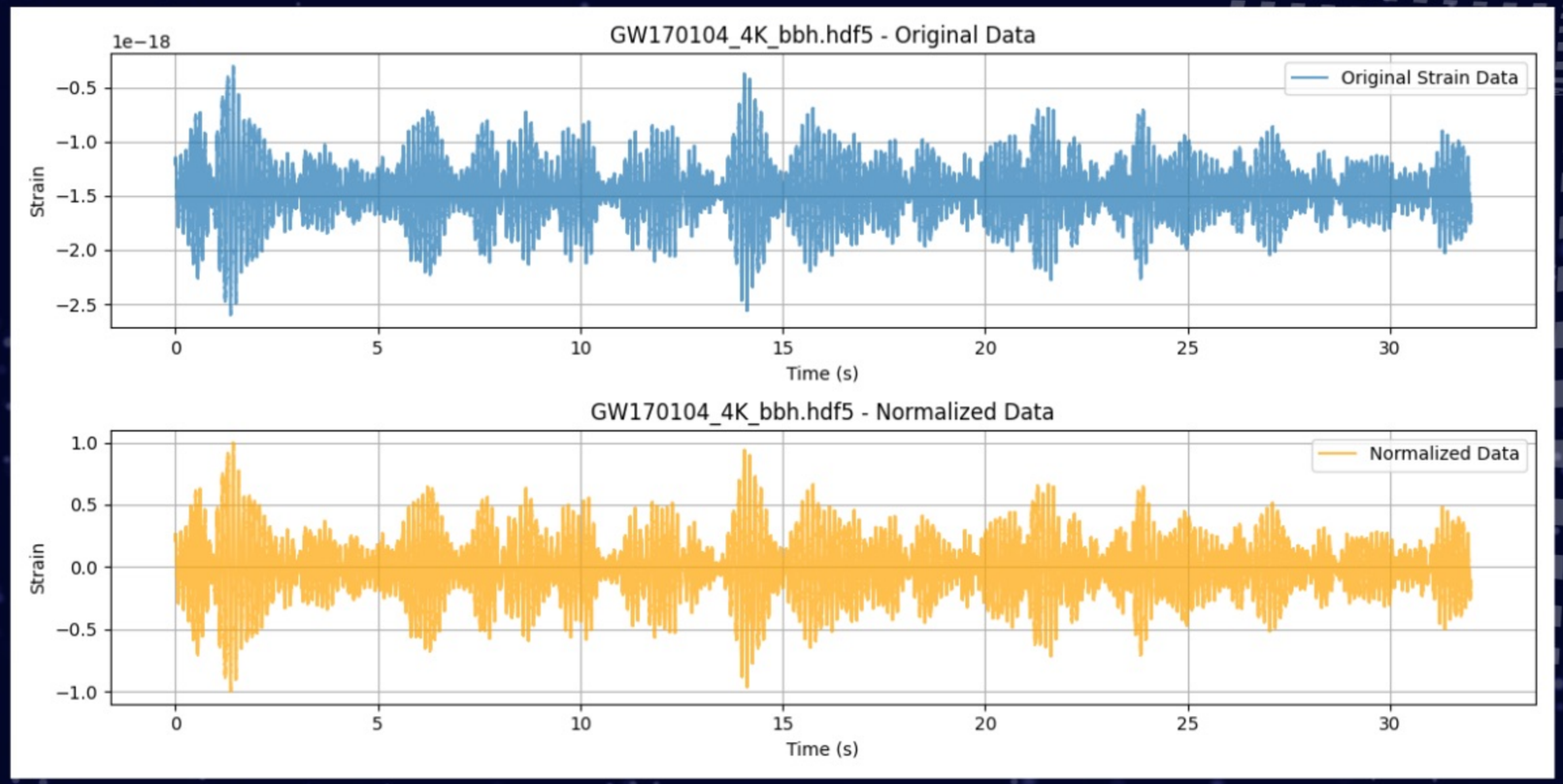
ii. Data Preparation for Model Testing

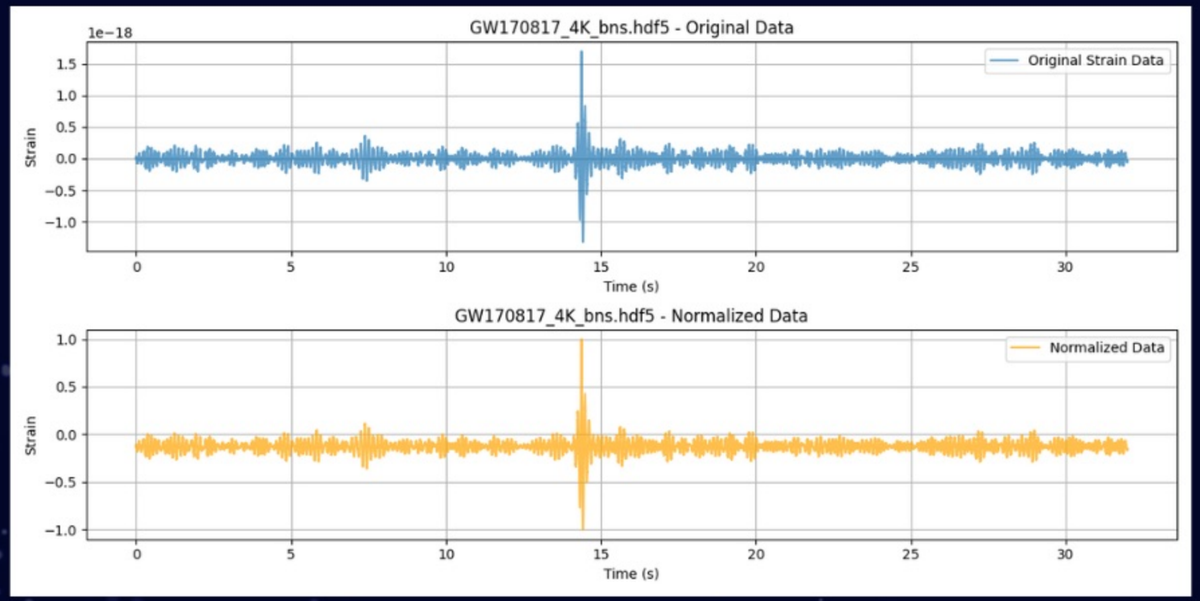
GWOSC Data Acquisition

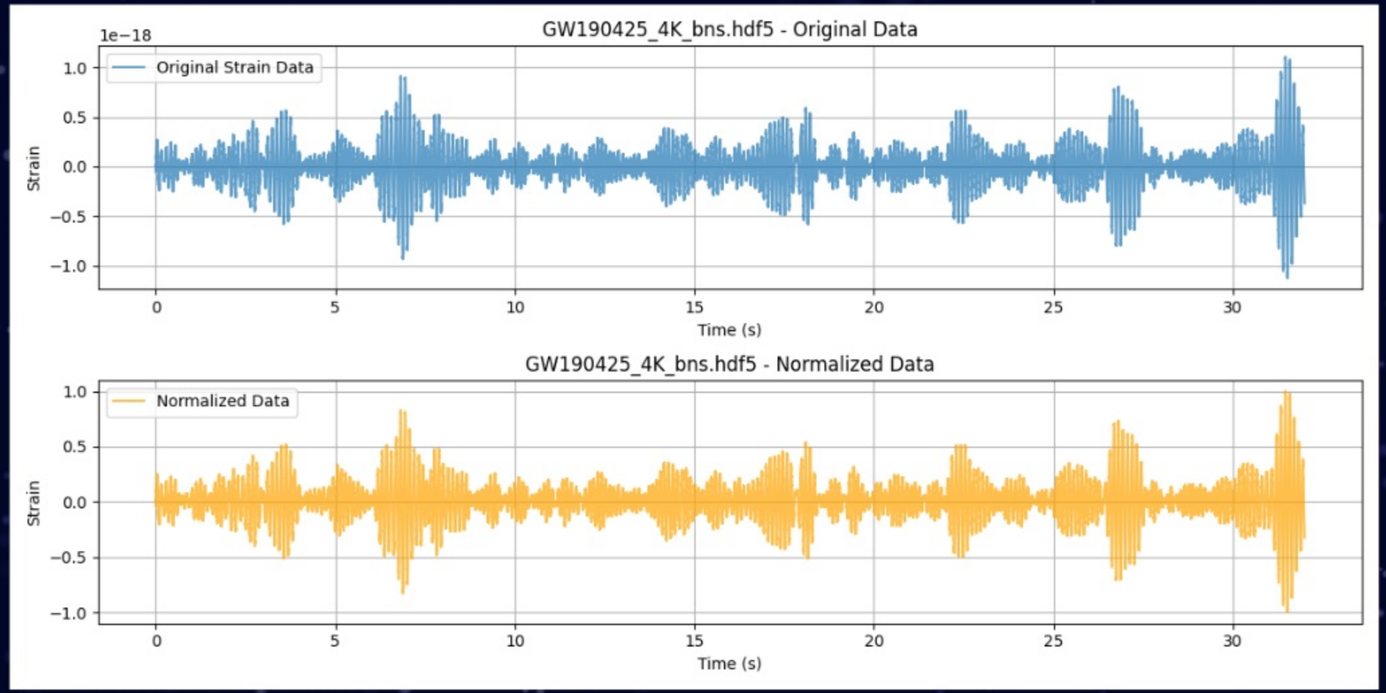


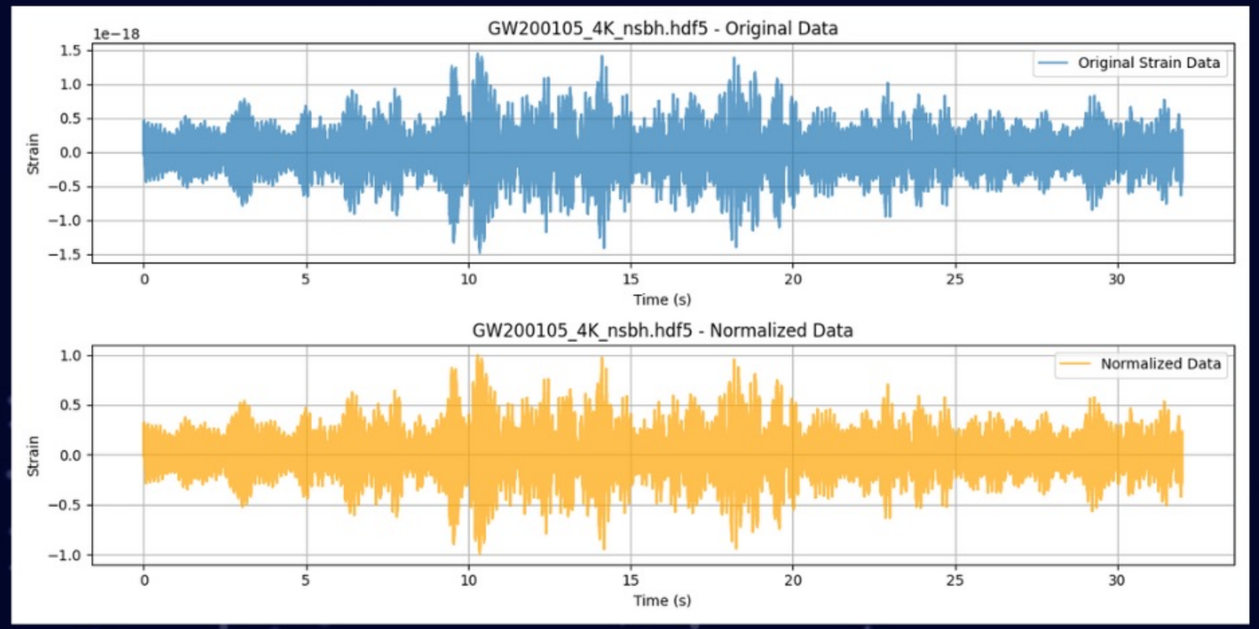
ii. Data Preparation for Model Testing

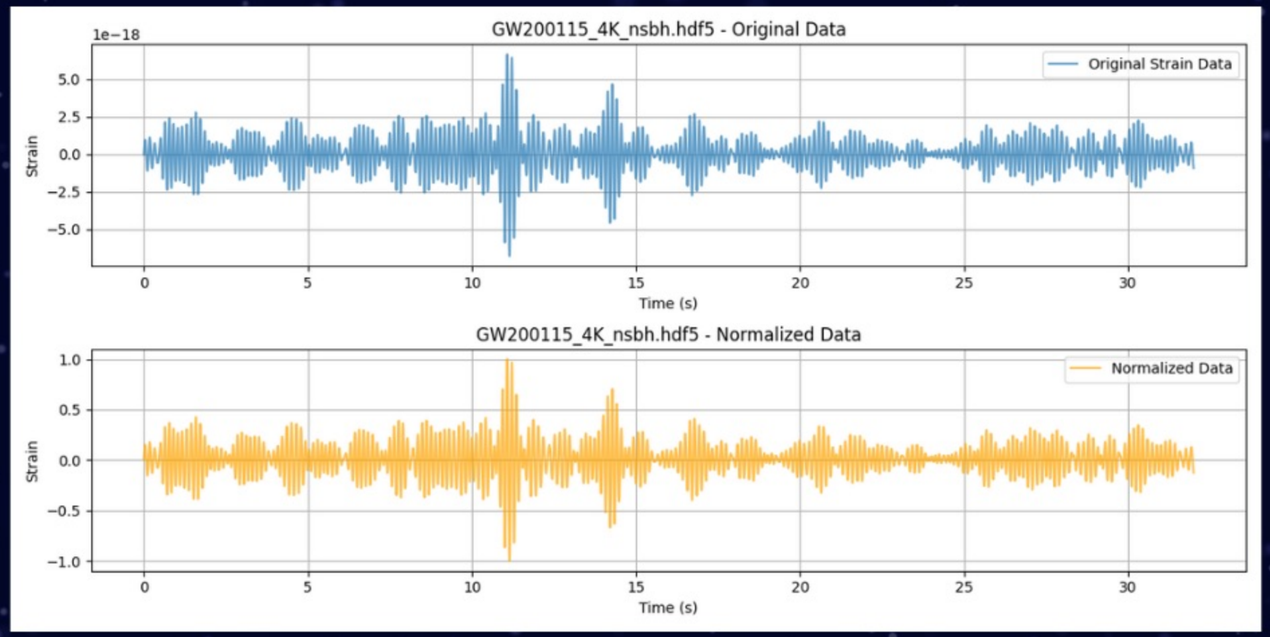
GWOSC Data Acquisition




GW170817_4K_bns.hdf5 - Original Data
1e−18
Original Strain Data
Strain
Time (s)
GW170817_4K_bns.hdf5 - Normalized Data
Normalized Data
Strain
Time (s)


GW190425_4K_bns.hdf5 - Original Data
1e−18
Original Strain Data
Strain
Time (s)
GW190425_4K_bns.hdf5 - Normalized Data
Normalized Data
Strain
Time (s)


GW200105_4K_nsbh.hdf5 - Original Data
1e−18
Original Strain Data
Strain
Time (s)
GW200105_4K_nsbh.hdf5 - Normalized Data
Normalized Data
Strain
Time (s)


GW200115_4K_nsbh.hdf5 - Original Data
1e−18
Original Strain Data
Strain
Time (s)
GW200115_4K_nsbh.hdf5 - Normalized Data
Normalized Data
Strain
Time (s)

i. Test Model on Real Data

```
Predicted Labels for GW150914_4K_bbh.hdf5: ['BNS']
Predicted Labels for GW170104_4K_bbh.hdf5: ['BNS']
Predicted Labels for GW170817_4K_bns.hdf5: ['BBH']
Predicted Labels for GW190425_4K_bns.hdf5: ['BNS']
Predicted Labels for GW200105_4K_nsbh.hdf5: ['BNS']
Predicted Labels for GW200115_4K_nsbh.hdf5: ['NSBH']
```

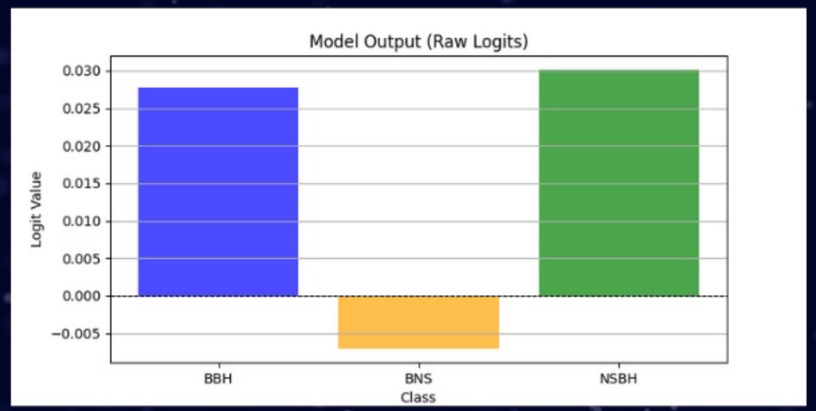
```
Predicted Labels for GW150914_4K_bbh.hdf5: ['NSBH']
Predicted Labels for GW170104_4K_bbh.hdf5: ['NSBH']
Predicted Labels for GW170817_4K_bns.hdf5: ['BBH']
Predicted Labels for GW190425_4K_bns.hdf5: ['NSBH']
Predicted Labels for GW200105 4K nsbh.hdf5: ['NSBH']
Predicted Labels for GW200115_4K_nsbh.hdf5: ['NSBH']
```

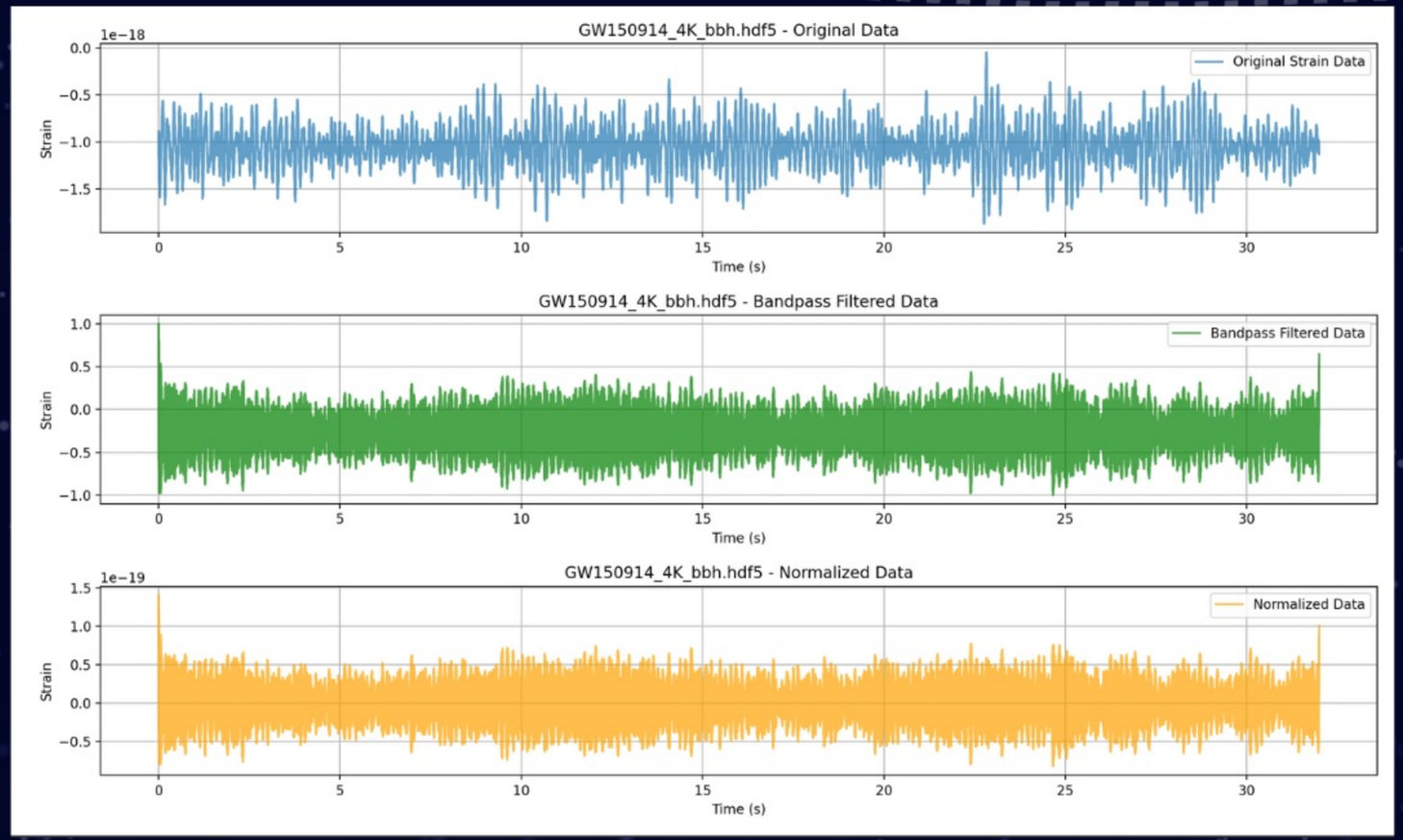
```
Predicted Labels for GW150914_4K_bbh.hdf5: ['NSBH']
Predicted Labels for GW170104_4K_bbh.hdf5: ['NSBH']
Predicted Labels for GW170817_4K_bns.hdf5: ['BBH']
Predicted Labels for GW190425_4K_bns.hdf5: ['NSBH']
Predicted Labels for GW200105_4K_nsbh.hdf5: ['NSBH']
Predicted Labels for GW200115_4K_nsbh.hdf5: ['NSBH']
```

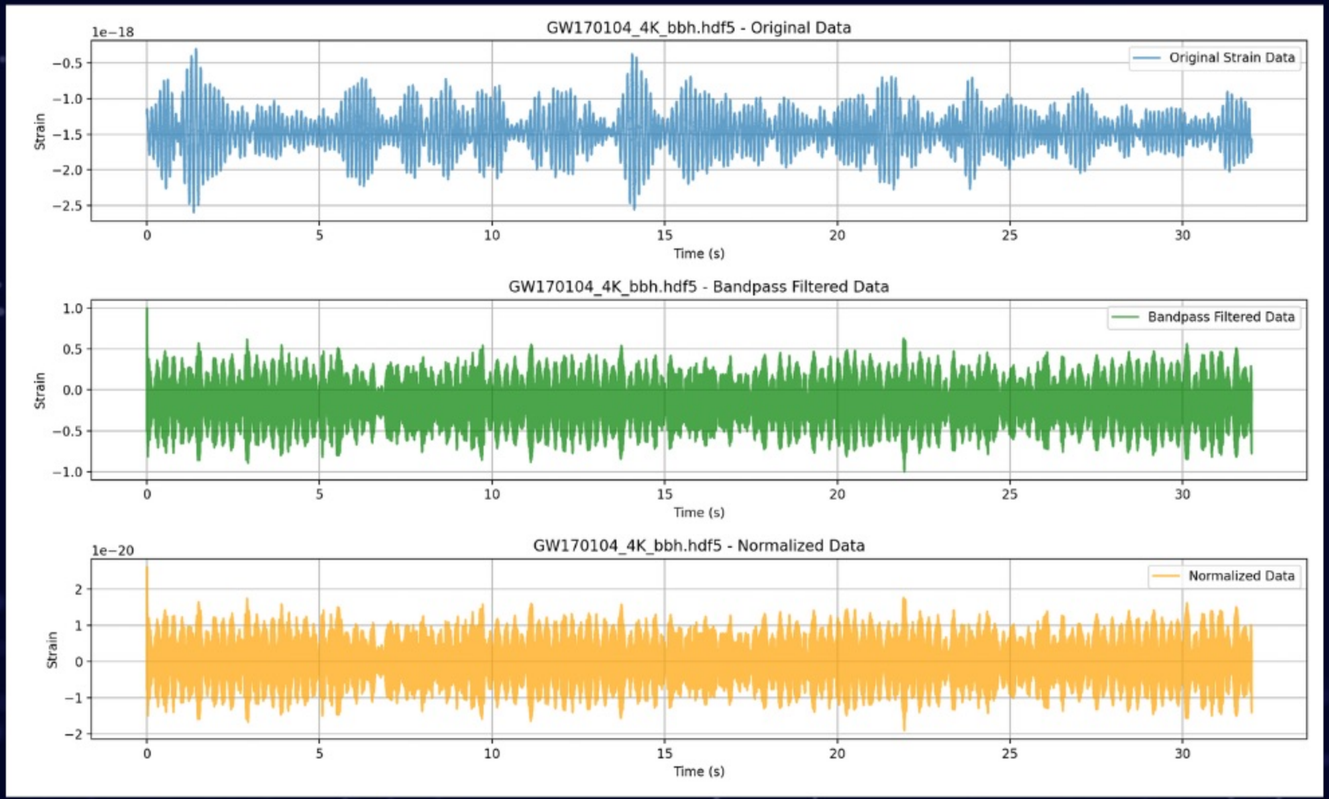
```
Predicted Labels for GW150914_4K_bbh.hdf5: ['NSBH']
Predicted Labels for GW170104_4K_bbh.hdf5: ['NSBH']
Predicted Labels for GW170817_4K_bns.hdf5: ['BBH']
Predicted Labels for GW190425_4K_bns.hdf5: ['NSBH']
Predicted Labels for GW200105_4K_nsbh.hdf5: ['NSBH']
Predicted Labels for GW200115_4K_nsbh.hdf5: ['NSBH']
```

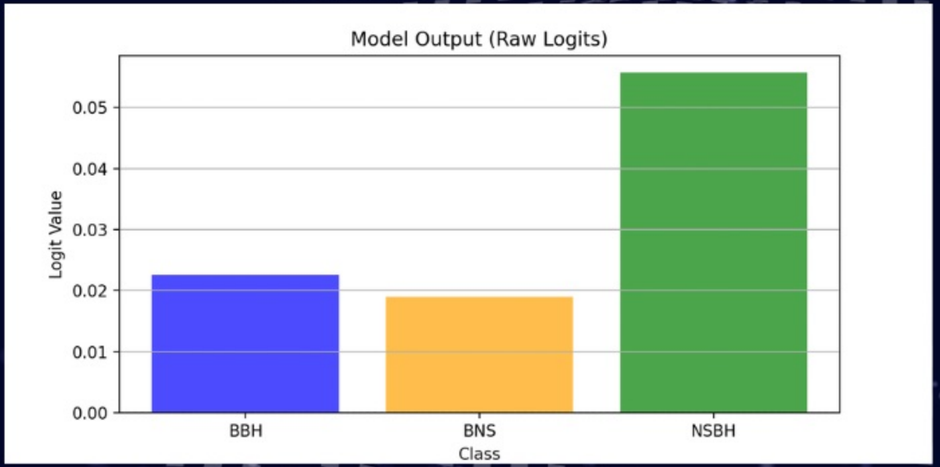
ii. Test Model on Bandpass Filtering Data

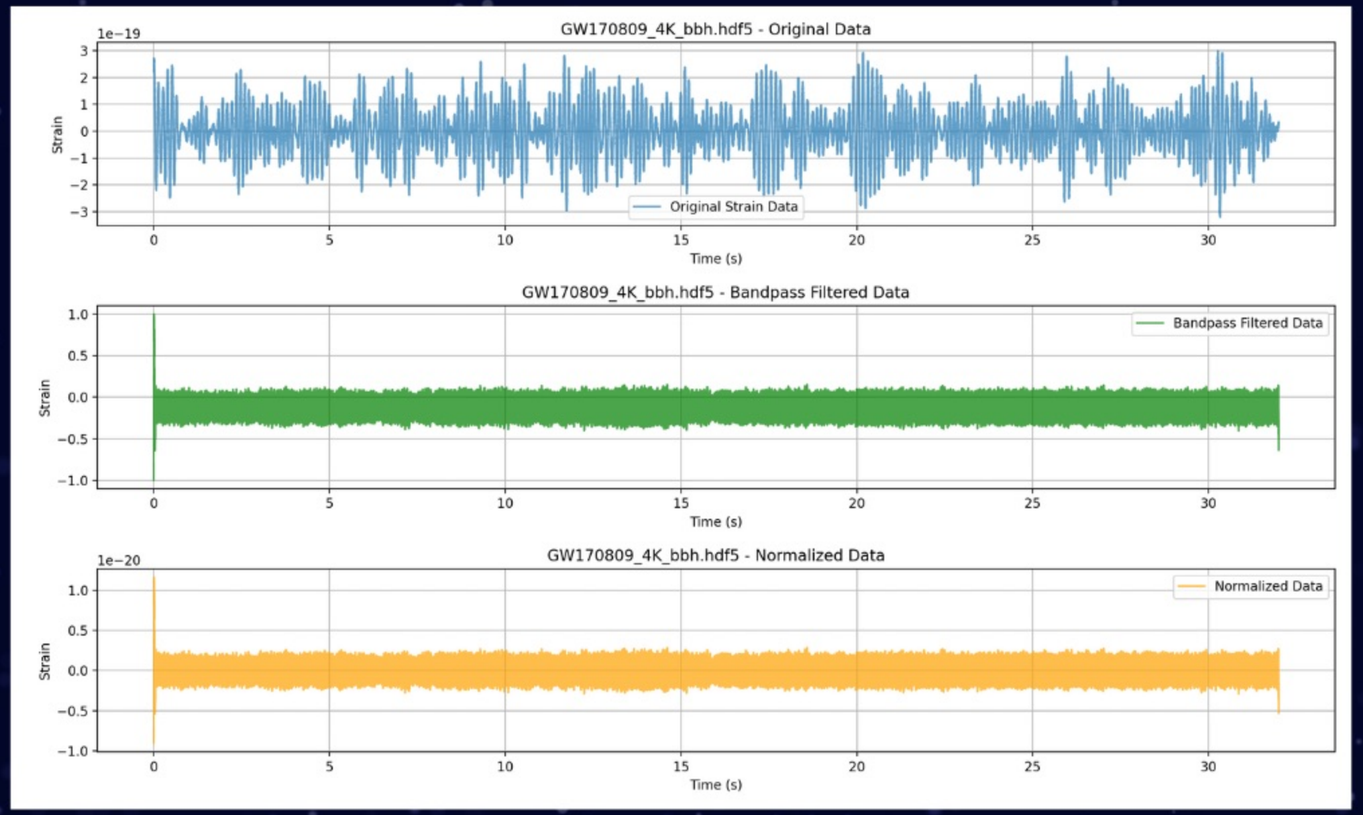
```python
def bandpass_filter(data, lowcut, highcut, fs, order=4):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    y = filtfilt(b, a, data)
    return y
```

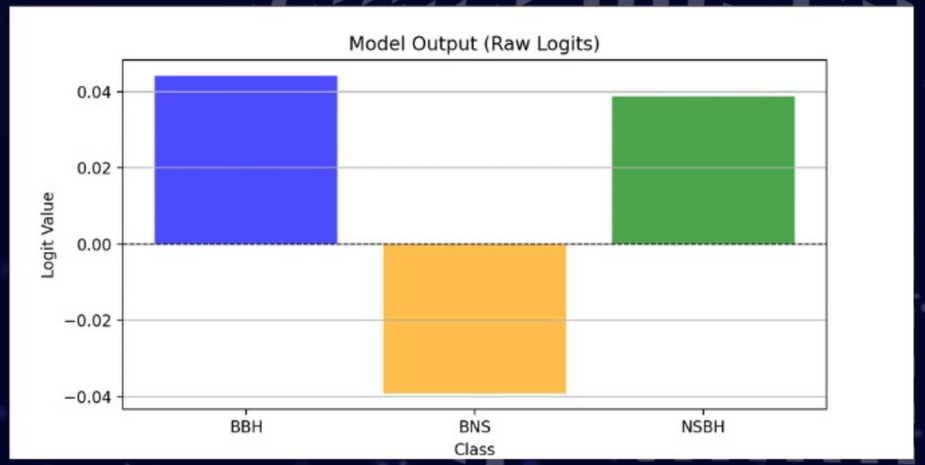


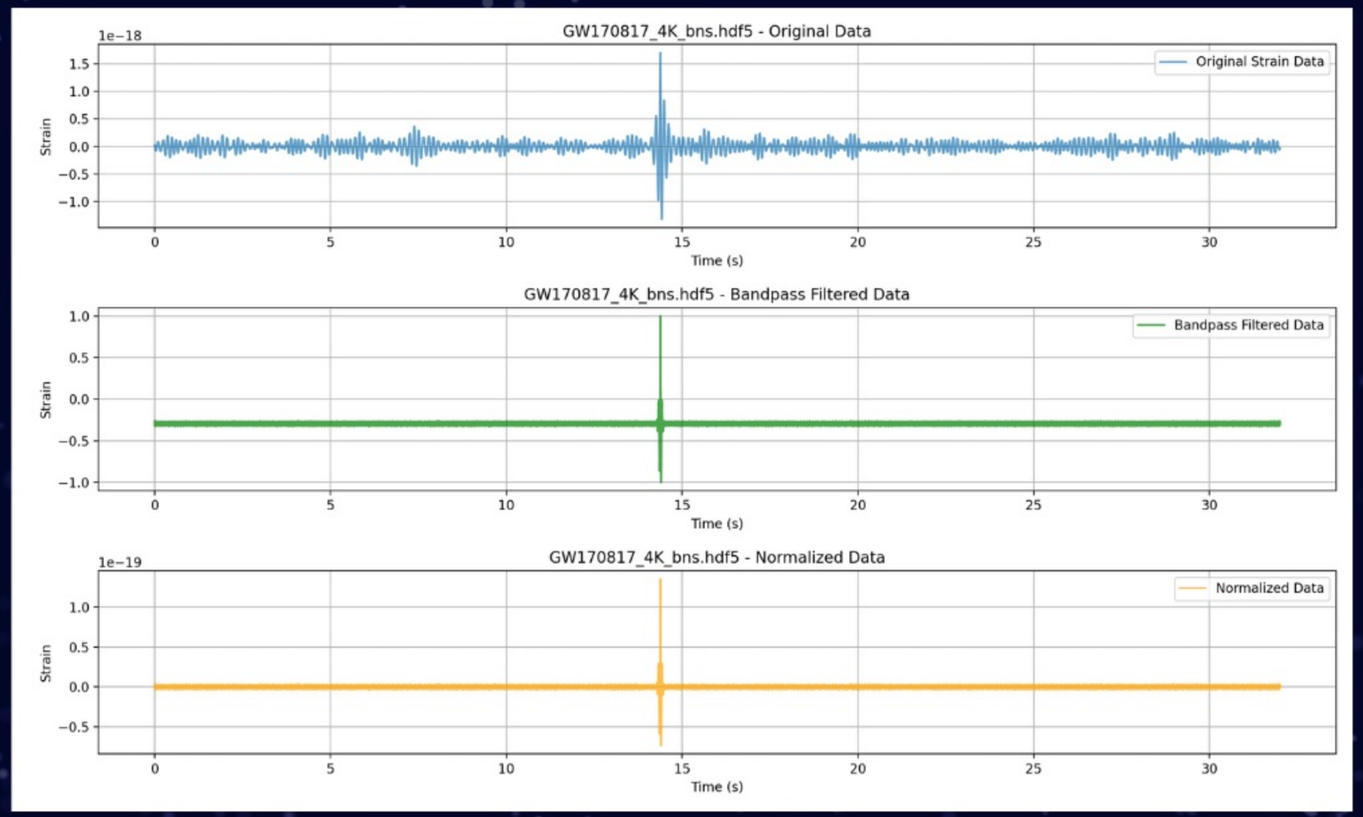


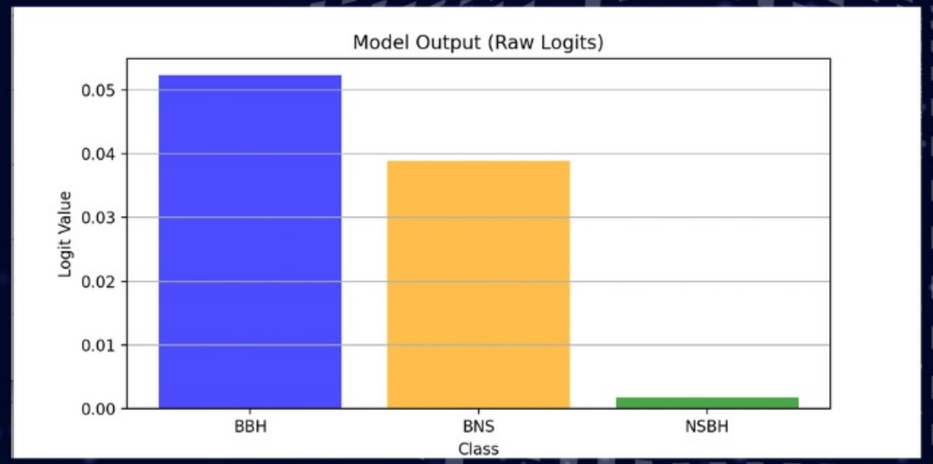

GW170104_4K_bbh.hdf5 - Original Data
1e−18
Original Strain Data
Strain
Time (s)
GW170104_4K_bbh.hdf5 - Bandpass Filtered Data
Bandpass Filtered Data
Strain
Time (s)
GW170104_4K_bbh.hdf5 - Normalized Data
1e−20
Normalized Data
Strain
Time (s)

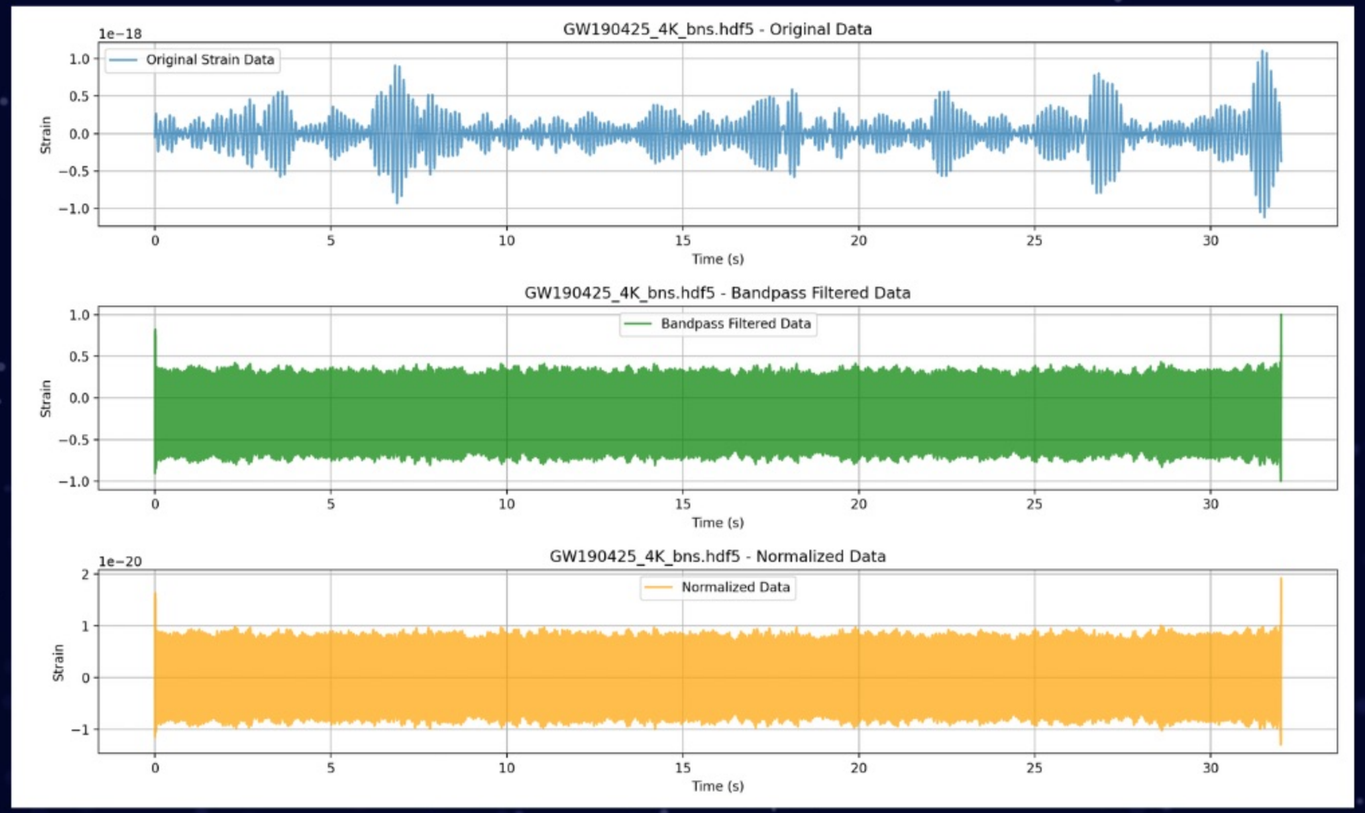

Model Output (Raw Logits)
Logit Value
BBH
BNS
NSBH
Class

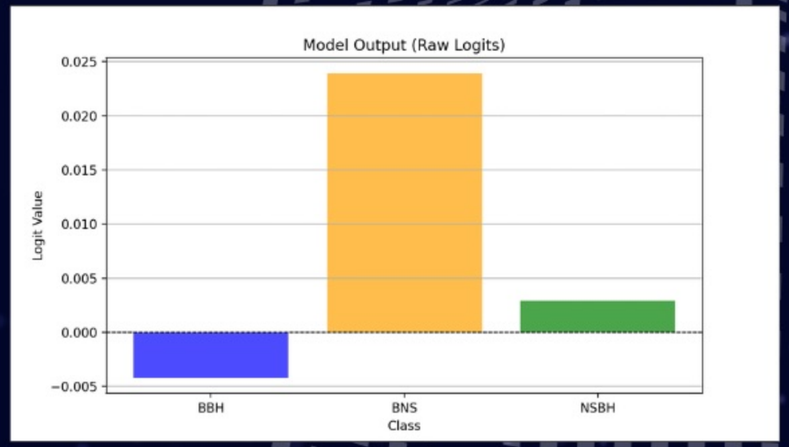

GW170809_4K_bbh.hdf5 - Original Data
1e−19
Strain
Original Strain Data
Time (s)
GW170809_4K_bbh.hdf5 - Bandpass Filtered Data
Bandpass Filtered Data
Strain
Time (s)
GW170809_4K_bbh.hdf5 - Normalized Data
1e−20
Normalized Data
Strain
Time (s)

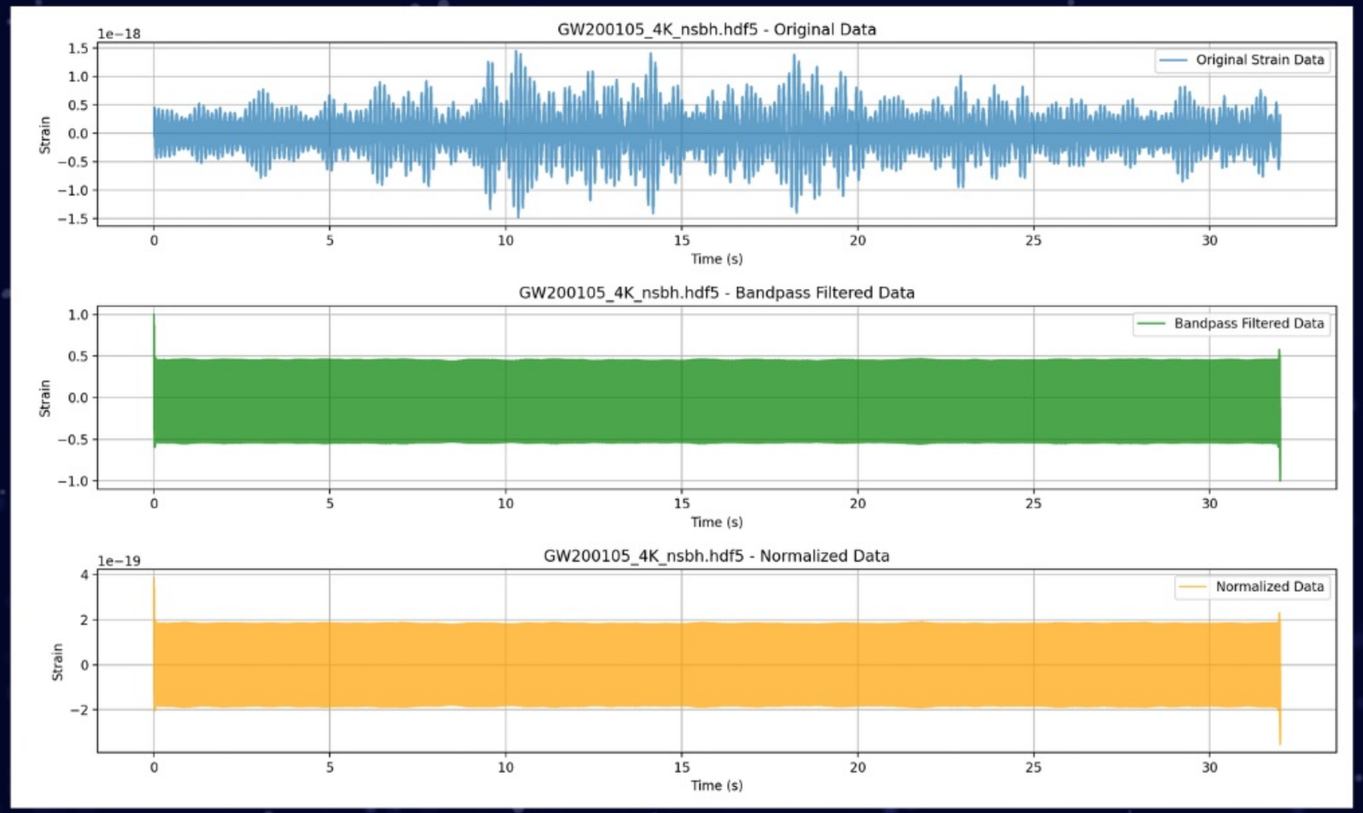

Model Output (Raw Logits)
Logit Value
BBH
BNS
NSBH
Class

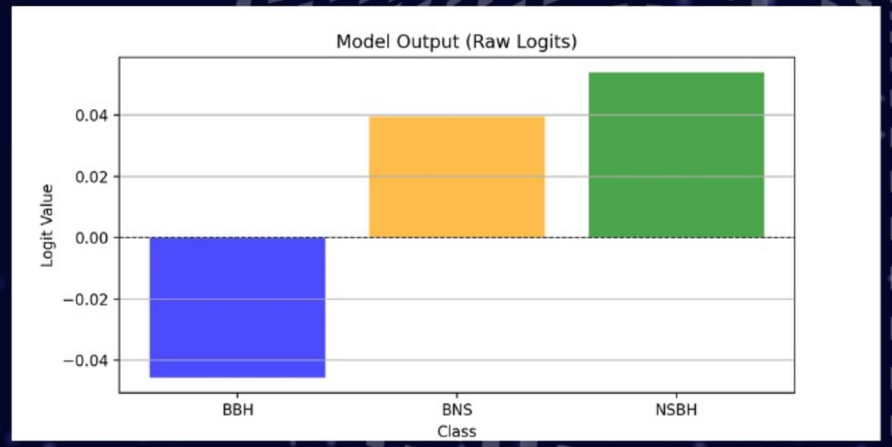

GW170817_4K_bns.hdf5 - Original Data
1e−18
Original Strain Data
Strain
Time (s)
GW170817_4K_bns.hdf5 - Bandpass Filtered Data
Bandpass Filtered Data
Strain
Time (s)
GW170817_4K_bns.hdf5 - Normalized Data
1e−19
Normalized Data
Strain
Time (s)

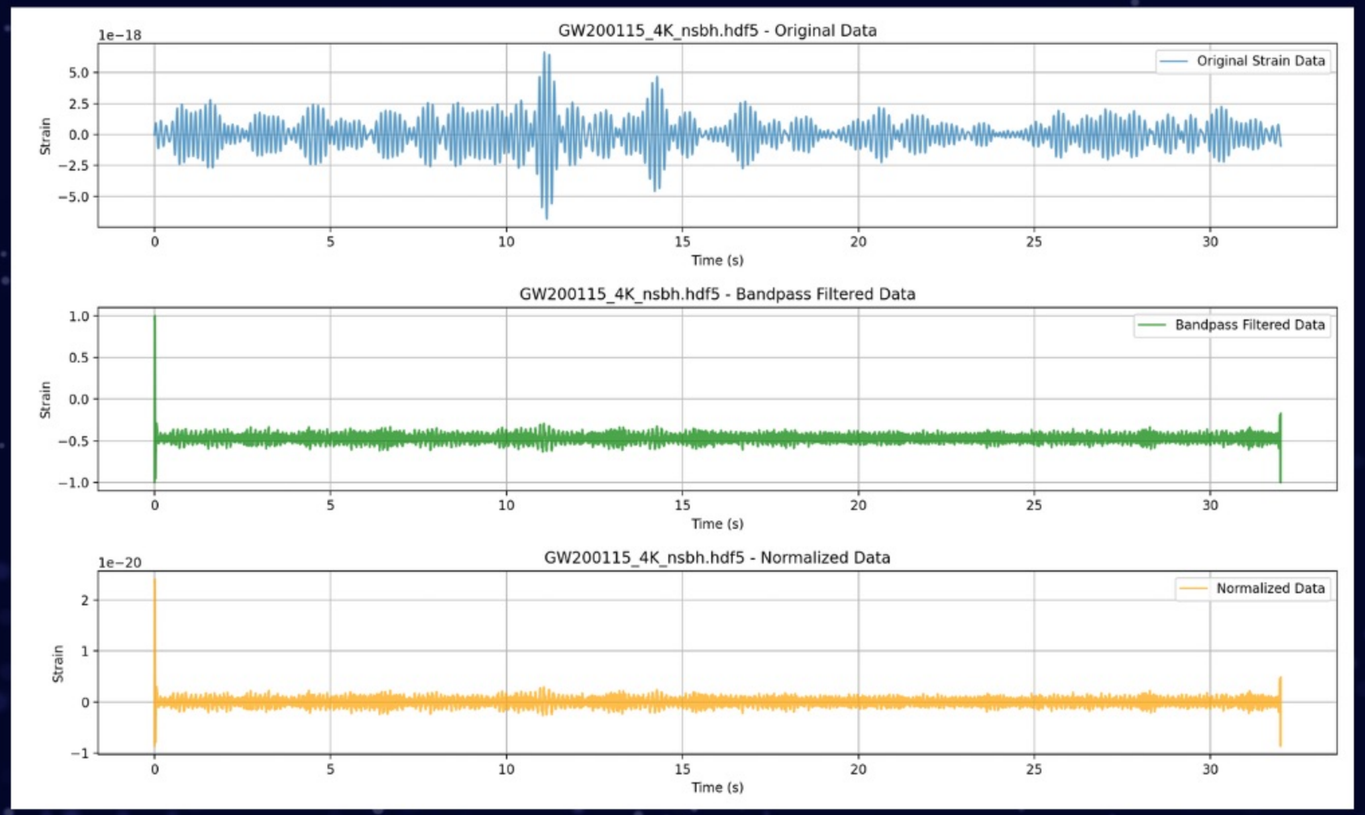

Model Output (Raw Logits)
Logit Value
BBH
BNS
NSBH
Class

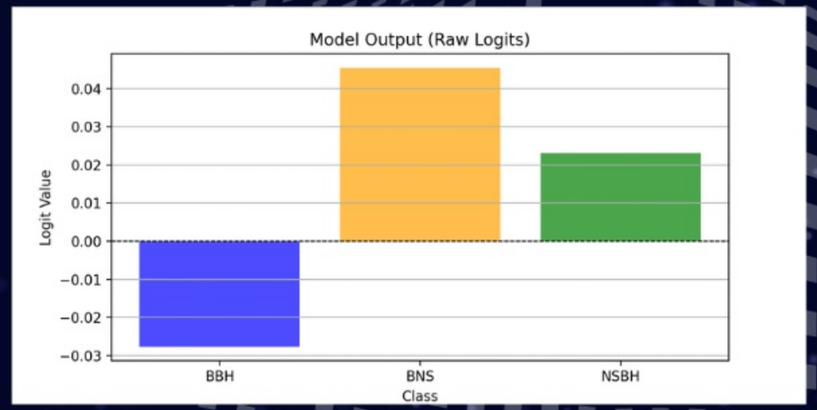

GW190425_4K_bns.hdf5 - Original Data
1e−18
Original Strain Data
Strain
Time (s)
GW190425_4K_bns.hdf5 - Bandpass Filtered Data
Bandpass Filtered Data
Strain
Time (s)
GW190425_4K_bns.hdf5 - Normalized Data
1e−20
Normalized Data
Strain
Time (s)


Model Output (Raw Logits)
Logit Value
BBH
BNS
NSBH
Class


GW200105_4K_nsbh.hdf5 - Original Data
1e−18
Original Strain Data
Strain
Time (s)
GW200105_4K_nsbh.hdf5 - Bandpass Filtered Data
Bandpass Filtered Data
Strain
Time (s)
GW200105_4K_nsbh.hdf5 - Normalized Data
1e−19
Normalized Data
Strain
Time (s)


Model Output (Raw Logits)
Logit Value
BBH
BNS
NSBH
Class


GW200115_4K_nsbh.hdf5 - Original Data
1e−18
Original Strain Data
Strain
Time (s)
GW200115_4K_nsbh.hdf5 - Bandpass Filtered Data
Bandpass Filtered Data
Strain
Time (s)
GW200115_4K_nsbh.hdf5 - Normalized Data
1e−20
Normalized Data
Strain
Time (s)


Model Output (Raw Logits)
Logit Value
BBH
BNS
NSBH
Class

REFERENCES

THANK YOU