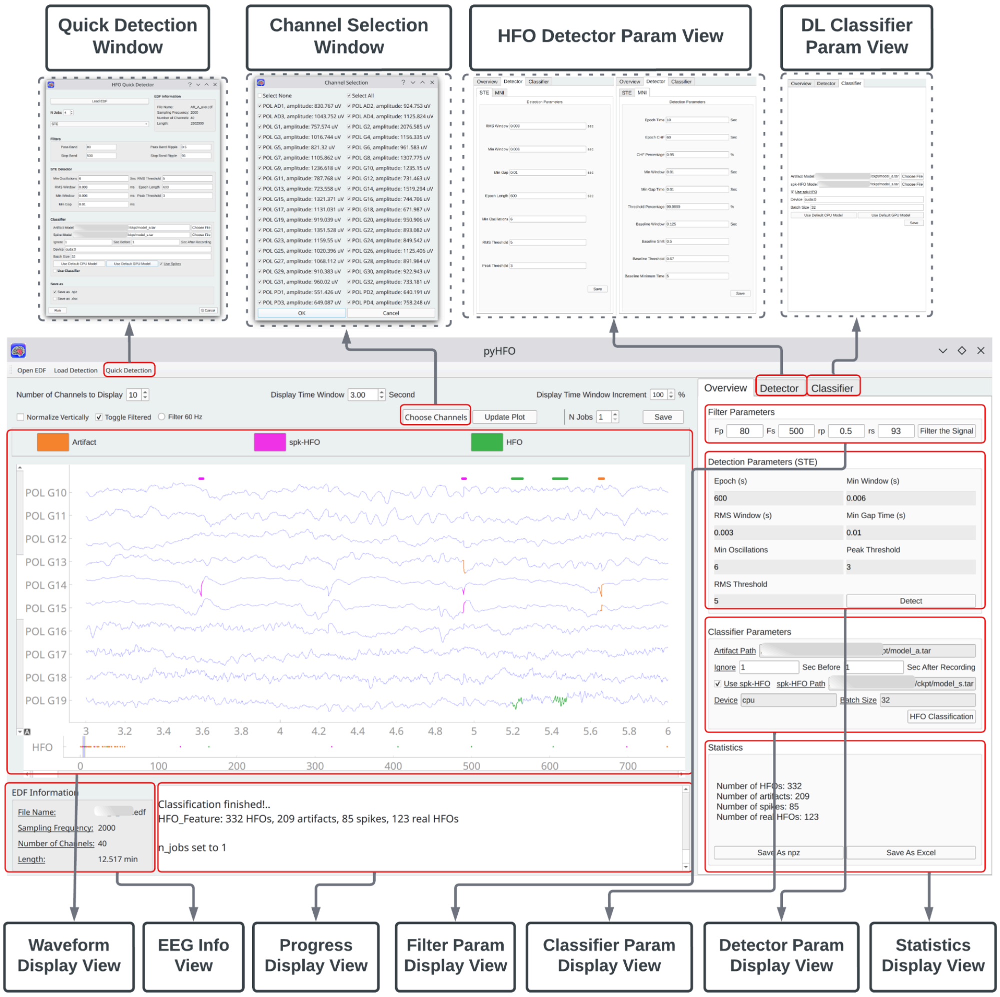
Window overview:

The GUI interface was implemented in PyQt version 5.15 to make it compatible with different hardware platforms such as OS X, Linux, and Windows.
Here we need to name our window in each page and explain its functionality



Number of Channels to Display 10 | Display Time Window 3.00 Second | Display Time Window Increment 100 % | N Jobs 1

Filter Parameters

Fp 80 Fs 500 rp 0.5 rs 93

Detection Parameters (STE)

Epoch (s)	Min Window (s)
600	0.006
RMS Window (s)	Min Gap Time (s)
0.003	0.01
Min Oscillations	Peak Threshold
6	3
RMS Threshold	
5	

Classifier Parameters

Artifact Path …pt/model_a.tar
Ignore 1 Sec Before 1 Sec After Recording
☑ Use spk-HFO spk-HFO Path …/ckpt/model_s.tar
Device cpu Batch Size 32

Statistics

Number of HFOs: 332
Number of artifacts: 209
Number of spikes: 85
Number of real HFOs: 123

EDF Information

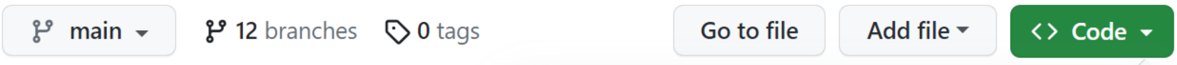
File Name: …edf
Sampling Frequency: 2000
Number of Channels: 40
Length: 12.517 min

```
Classification finished!..
HFO_Feature: 332 HFOs, 209 artifacts, 85 spikes, 123 real HFOs

n_jobs set to 1
```

Quick start

1. Installing the software
   pyHFO scripts can be found in https://github.com/roychowdhuryresearch/pyHFO .
   To access the link, please select the [Download ZIP] option.

   Select a location to unzip HFO-APP files
2. System Configuration
   pyHFO was developed using Python.
   Follow the steps in the README file to configure the environment and all libraries in requirements.txt.
   Run the hfo_main_window.py script in Python to open the GUI of the app.

How to use pyHFO

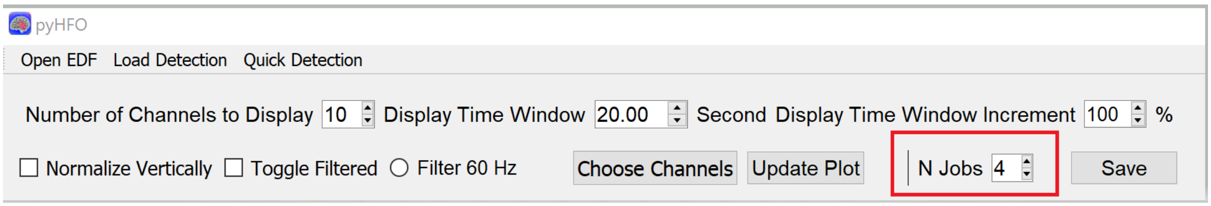
The complete procedure for detection and analysis of HFOs using pyHFO following steps as below:

1. set N_jobs to run in parallel
2. Import and load EEG *edf file
3. Pre-processing with signal filtering
4. Automatic HFO detection by choosing the detector and parameters
5. Artifact, spike, and HFO validation by DL-based classification
6. Save analysis

Following filtering the signal, the buttons to save the detector parameters will become enabled, since those detectors rely on the filter parameters. Likewise, the filter button will only be enabled once an edf has been loaded.

Set N_jobs:

pyHFO supports multiprocessing, so at the top of the parameters window, there is an input where the user can set the number of jobs, which specifies the number of concurrently running workers. The user can set this number from between 1 (so no multiprocessing) to the max of the number of cpu cores in the machine. The number of cores can be changed between operations, to do so simply set the number of cores the user desires, and click the Save button



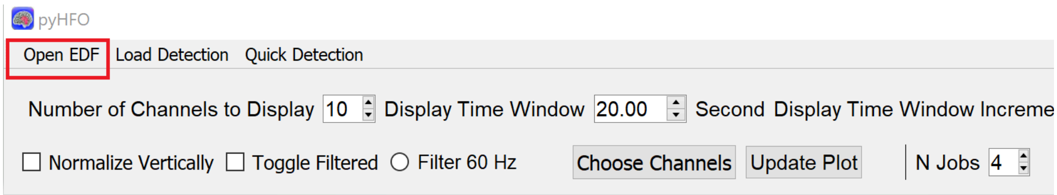
Number of Channels to Display 10 Display Time Window 20.00 Second Display Time Window Increment 100 %

□ Normalize Vertically □ Toggle Filtered ○ Filter 60 Hz N Jobs 4

Load EDF:

The user can select an *.edf file to import and load for the analysis of EEG data.

a. To load an edf, the user should click on the load edf button on the upper left bar or go to File > Open EDF on upper left on the pyHFO's main window.



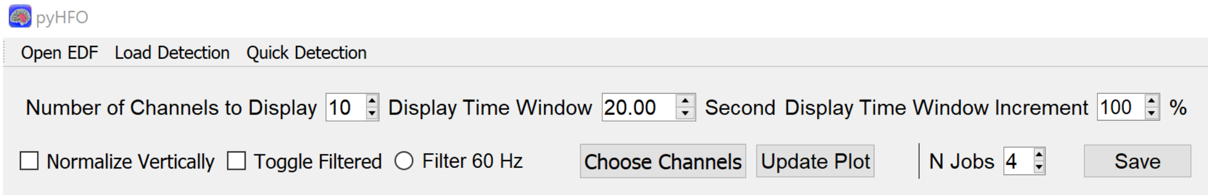
Number of Channels to Display 10 Display Time Window 20.00 Second Display Time Window Increme

□ Normalize Vertically □ Toggle Filtered ○ Filter 60 Hz N Jobs 4

b. Then, a window will pop up where the user can select the edf file to open. Only valid file(*.edf) could be loaded and the loading progress will be shown in the information window.

c. After the edf has been successfully loaded, the waveforms for each channel will be plotted in the waveform plotting region. Meanwhile, the detailed information of the edf file will be shown in the bottom of the main window.

Furthermore, with the EEG data loaded, we also give users options to play around and visually explore the signal plotted in the window.

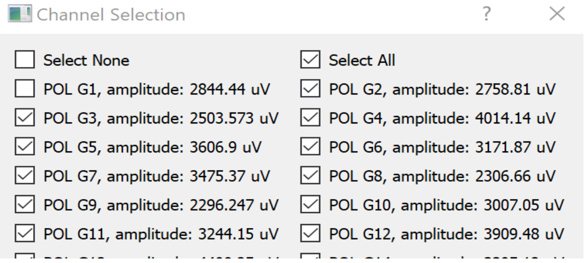


Number of Channels to Display 10 Display Time Window 20.00 Second Display Time Window Increment 100 %

☐ Normalize Vertically ☐ Toggle Filtered ○ Filter 60 Hz N Jobs 4

a. Users can choose any channel to display. One or multiple electrodes can be selected for further analysis. To include channels to process, click button.

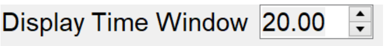
With the *Choose Channels* button clicked, a window with all channels will pop up -> Select channels -> click *OK* button.

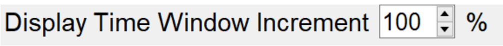


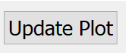
☐ Select None
☑ Select All
☐ POL G1, amplitude: 2844.44 uV
☑ POL G2, amplitude: 2758.81 uV
☑ POL G3, amplitude: 2503.573 uV
☑ POL G4, amplitude: 4014.14 uV
☑ POL G5, amplitude: 3606.9 uV
☑ POL G6, amplitude: 3171.87 uV
☑ POL G7, amplitude: 3475.37 uV
☑ POL G8, amplitude: 2306.66 uV
☑ POL G9, amplitude: 2296.247 uV
☑ POL G10, amplitude: 3007.05 uV
☑ POL G11, amplitude: 3244.15 uV
☑ POL G12, amplitude: 3909.48 uV

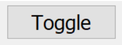
a. To select the number of channels to display, enter or change

Number of Channels to Display 10

b. To change the number of seconds of the waveform to display, enter or change

Display Time Window 20.00

c. To change display time window increment, enter or change percentage

Display Time Window Increment 100 %

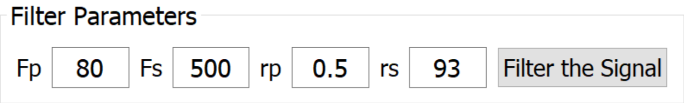
d. With any change of the signal display, click  button to view the newly updated signal display.

b. To normalize the signal, use  button to toggle between the unfiltered and filtered waveform (after filtering the signal) and after detection, the HFOs will be highlighted.

Signal Filtering

If raw data has been selected to be displayed, raw data would be filtered for pre-processing.

1. Filter Parameter selection
   The filter used in pyHFO is the Chebyshev type II filter. In the filter operation, the user needs to specify the PassBand, StopBand, Passband Ripple, and StopBandAttenuation. The Filter Parameters block (Fig X) in the Overview panel allows users to set filter parameters. Here, we also provide default parameters.

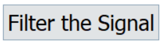


Filter Parameters

Fp 80 Fs 500 rp 0.5 rs 93

Fp: passband frequency
Fs: stopband frequency
rp: passband ripple
rs: stopband attenuation

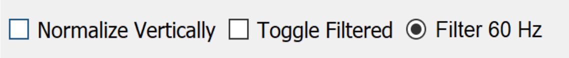
2. Filter and display the signal



After the user specifies the parameters in the filter window and clicks button, the filter will filter the EEG signal in each channel (sequentially when n_jobs =1, in parallel when n_jobs > 1).

    a. The filtered signal will be displayed in the main window.
    b. Filtering completion will be shown in the information window, after the filtering process is done.

3. Explore the filtered signal



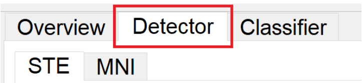
☐ Normalize Vertically ☐ Toggle Filtered ● Filter 60 Hz

pyHFO gives options to explore signal, filtered and normalized signal and signal filtering 60Hz to explore and display. Check options for further analysis.

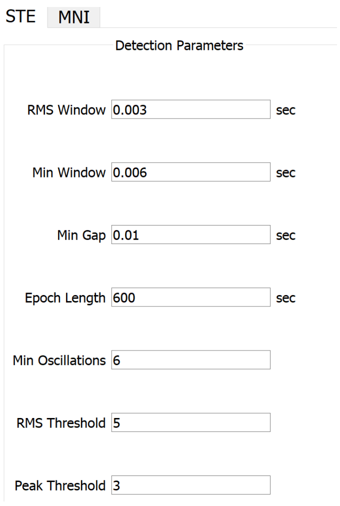
Automatic HFO Detection

Automatic HFO detection will be processed after filtering the signal for HFO detection. pyHFO provides two automatic HFO detection methods: Short Time Energy detector (STE) and MNI detector (MNI).

1. To detect HFOs, go to the ***Detector*** panel on the right part of the main window page -> Select one method for detection.



2. To select one method for HFO detection, set the ***Detection Parameters*** of the method in the detection selection panel
   Each of the two methods have their own specific parameters. Default values are provided for each method(Fig X) and users can set values manually. Detailed description of each method would be stated in the Implementation details part.



STE | MNI

Detection Parameters

RMS Window 0.003 sec

Min Window 0.006 sec

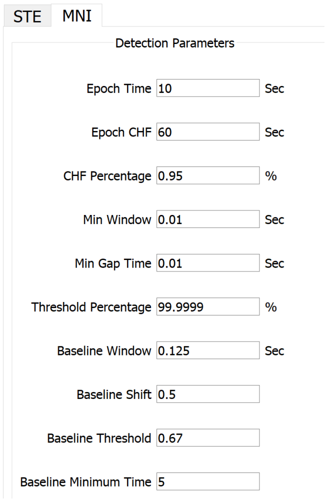
Min Gap 0.01 sec

Epoch Length 600 sec

Min Oscillations 6

RMS Threshold 5

Peak Threshold 3



STE | MNI

Detection Parameters

Epoch Time 10 Sec

Epoch CHF 60 Sec

CHF Percentage 0.95 %

Min Window 0.01 Sec

Min Gap Time 0.01 Sec

Threshold Percentage 99.9999 %

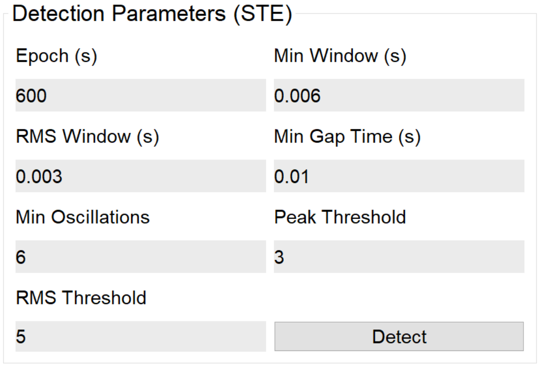
Baseline Window 0.125 Sec

Baseline Shift 0.5

Baseline Threshold 0.67

Baseline Minimum Time 5

3. To save the detector parameters, click the ***save*** button Save in the detection selection panel, the specific detector and its corresponding parameters will be shown in the ***Overview*** panel on the main window page.
4. To start an automatic HFO detection process, go back to the ***Overview*** panel, the parameters saved for method will shown in ***Detection Parameters*** section, the user needs to click the ***detect*** button Detect in the section to start HFO detection.



Detection Parameters (STE)

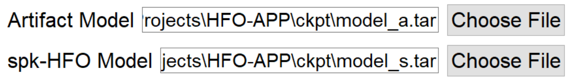
Epoch (s)	Min Window (s)
600	0.006
RMS Window (s)	Min Gap Time (s)
0.003	0.01
Min Oscillations	Peak Threshold
6	3
RMS Threshold	
5	Detect

The HFO detection will be run in each channel (sequentially when n_jobs =1, in parallel when n_jobs > 1). The progress of the detection should be shown in the information window. After the detection is completed, the detected HFO will be marked as orange in the waveform visualization window.

DL-based Classification

With all HFOs detected, 'HFOs detected' will be shown in the progress window in the bottom of the main window. And the ***HFO Classification*** button would be valid to click.

a. Go to Classifier panel Overview | Detector | Classifier for further setting for classification model.
b. pyHFO provides two deep learning models to classify the detected HFOs, artifact and a spike detector. To use the user must choose the files containing the model weights, this can be done by clicking on the choose file button, there are two of these buttons corresponding to the two models.



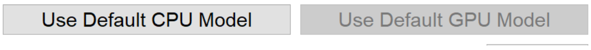
Artifact Model rojects\HFO-APP\ckpt\model_a.tar

spk-HFO Model jects\HFO-APP\ckpt\model_s.tar

c. Next the user must specify if they want the spike detector to run or not, they can choose this by clicking on the use spike checkbox. and specify which device the models are run on. These models can be run on either CPU, or GPU if the machine pyHFO is being on has a cuda capable GPU. Running the models on GPU can significantly speed up the runtime. To specify which device, the user can input it in (for instance the user would input in “CPU” for CPU, and “cuda:0” for the first GPU. The user must specify the batch size, this is the number of HFOs to classify simultaneously.

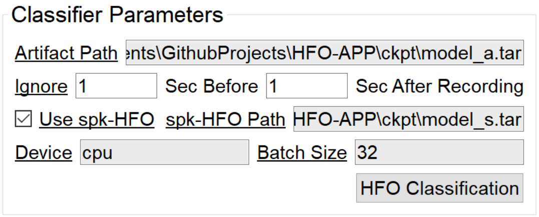
d. We provided default model weights and parameters for both CPU and GPU inference, to quickly load these default parameters, the user can click on the **Use Default CPU** and **Use Default GPU** parameters.

To use default models, click



e. Once users input all these parameters, they can save them by clicking the ***save button*** in the panel.

f. Go back to the ***Overview*** panel -> Now, the parameters for the detector are shown in the ***Classifier Parameters*** section, and from here, the user can classify it by clicking the ***HFO Classification*** button.



Classifier Parameters

Artifact Path ents\GithubProjects\HFO-APP\ckpt\model_a.tar

Ignore 1 Sec Before 1 Sec After Recording

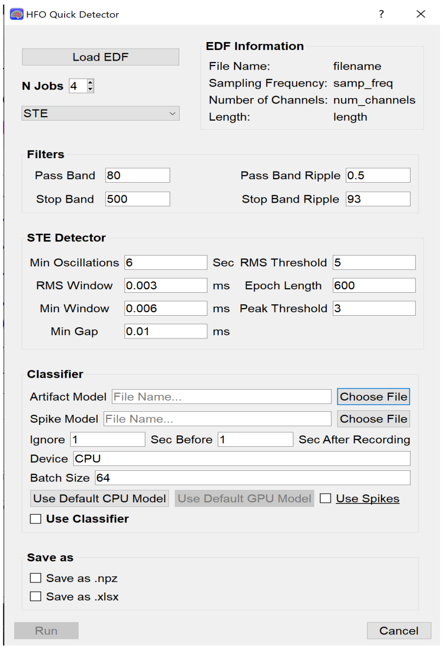
☑ Use spk-HFO spk-HFO Path HFO-APP\ckpt\model_s.tar

Device cpu Batch Size 32

Quick Detection

In the cases where the user does not need to visualize the waveform and rather just wants to detect the HFOs/classify it, we provide the Quick Detection functionality using STE Detector.

1. To use quick detection, users should click on the ***Quick Detection*** button, upon which the quick detection window will appear.
2. In the ***HFO Quick Detector*** window, users can quickly do all actions mentioned before. For example, setting the n_jobs, loading the edf file, setting the filter parameters and selecting detector and parameters, Classifying HFOs, and specifying ways of saving the results.
    a. To start quick detection, load a valid *edf file first by clicking the button ***Load EDF***



HFO Quick Detector

N Jobs 4

STE

EDF Information

File Name: filename

Sampling Frequency: samp_freq

Number of Channels: num_channels

Length: length

Filters

Pass Band 80 Pass Band Ripple 0.5

Stop Band 500 Stop Band Ripple 93

STE Detector

Min Oscillations 6 Sec RMS Threshold 5

RMS Window 0.003 ms Epoch Length 600

Min Window 0.006 ms Peak Threshold 3

Min Gap 0.01 ms

Classifier

Artifact Model File Name...

Spike Model File Name...

Ignore 1 Sec Before 1 Sec After Recording

Device CPU

Batch Size 64

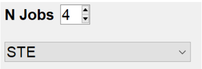
☐ Use Spikes

☐ Use Classifier

Save as

☐ Save as .npz

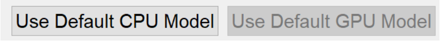
☐ Save as .xlsx

b. Select ***N jobs*** and detector type through 

N Jobs 4

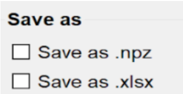
STE

c. Besides default parameters for filters and detectors, users can also specify parameters.

d. To classify detected HFOs, upload Artifact Model and Spike Model by choosing files that the user has. Or users can click 

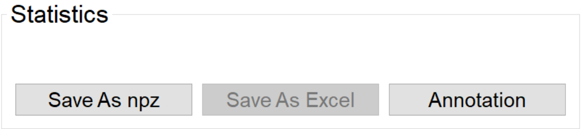
button to use our default Model. The models will be loaded automatically after one click.

e. Choose the result the user wants to save. Users can select whether to use spike or not, use the classifier or not and can also choose ways to save results.



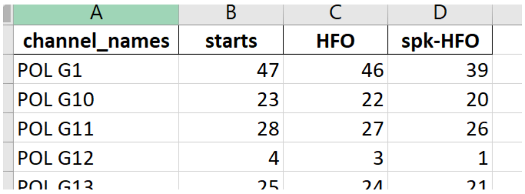
f. Click the ***Run*** button in the window to detect HFOs and save the result.

Save Results



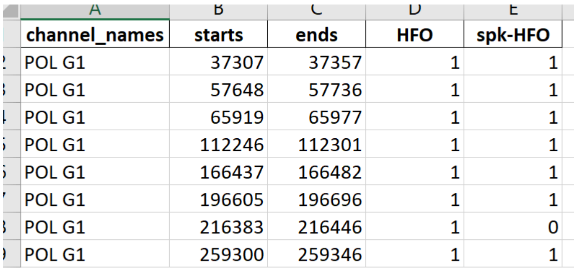
pyHFO provides two ways to save the HFO analysis results as below.

1. Save as npz
   With the .npz file format, a zipped archive of files named after the variables they contain and can be compressed later for further use efficiently and securely.
   Users can save results with all HFOs information before classification, and can also save all artifacts and spikes information if using Classifiers.
2. Save as Excel:
   With the .xlsx file format, users can quickly and easily look into the file and explore all information about each channel's HFOs getting rid of artifacts and spike HFOs.



channel_names	starts	HFO	spk-HFO
POL G1	47	46	39
POL G10	23	22	20
POL G11	28	27	26
POL G12	4	3	1
POL G13	25	24	21

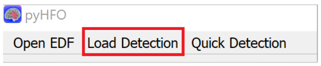
Users can also know the intervals in EEG raw data of the HFO for each channel and whether the HFO is spike HFO.



channel_names	starts	ends	HFO	spk-HFO
POL G1	37307	37357	1	1
POL G1	57648	57736	1	1
POL G1	65919	65977	1	1
POL G1	112246	112301	1	1
POL G1	166437	166482	1	1
POL G1	196605	196696	1	1
POL G1	216383	216446	1	0
POL G1	259300	259346	1	1

pyHFO also give you option to get all annotation of

Load results

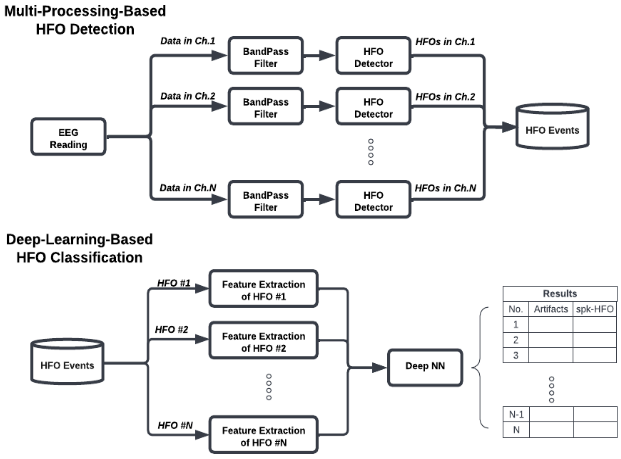
With a valid .npz file saved in your local, pyHFO gives you the option to load the file to the app and visualize the result. To visualize your .npz file HFO result, users can click the button Load Detection  to load the result.

Implementation details:

In this part, we also give users a detailed implementation of our pipeline to detect HFO with an EEG recording given.

System overview:

Four major steps are conducted once an EEG recording is sent into the pipeline, including EEG signal reading, data filtering, HFO detector, and DL-based HFO classification algorithm. As shown in Figure 1. The results of this pipeline are the detected event based on the detection algorithm with annotation of the real HFO, artifacts, and HFO-with-spike (spk-HFO) using pre-trained DL-based HFO classification models.
For the overall data processing workflow shown in the flowchart, we adopt a multi-processing mechanism in both HFO detection and feature extraction for deep learning networks, which significantly increases the efficiency of the HFO analysis.
Nest, we will introduce each steps in details.

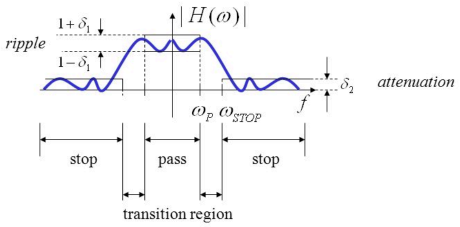


Results		
No.	Artifacts	spk-HFO
1		
2		
3		
N-1		
N		

Filtering:

With EEG raw data file loaded, the signal will be preprocessed by filtering the signal.
The filter used in pyHFO is the Chebyshev type II IIR bandpass filter.
The parameter of this filter consists of Passband Frequency, Stopband Frequency, Passband Ripple, and Stopband Attenuation. The definition of the parameters are shown in Figure X.
For constructing such a filter, the order of the filter is first estimated, and then the frequency response is constructed. In the Python implementation of pyHFO, we choose to use Scipy to

construct the filter to produce a more accurate frequency response, and specify the filter parameter as PassBand = 80, StopBand = 500, PassBandRipple=0.5, and StopBandAttenuation = 93.



LPF specified by:

- passband frequency ω_P
- passband ripple δ_1 or $R_P = 20\log_{10}\frac{1+\delta_1}{1-\delta_1}\ dB$
- stopband frequency ω_{STOP}
- stopband attenuation δ_2 or $R_S = 20\log_{10}\delta_2\ dB$

HFO detection:

Two automatic HFO detection algorithms in pyHFO, Short Time Energy (STE) and Montreal Neurological Institutes detector (MNI) are used to detect HFOs in PyHFO. They were selected based on the two mainstream detection algorithms from a popular Matlab-based HFO analysis tool RIPPLELAB, and they have been proven to be successful detectors in many previous works. Nonetheless, because of the pyHFO's modular structure and the open-source nature of this project, by just following a simple interface, other HFO detection methods can be easily integrated into the application if desired and can be benefited from the multi-processing paradigm.

STE detector

Short Time Energy (STE), proposed by Staba et al. detects HFO events by selecting the energy of the filtered raw EEG signal with an estimated energy threshold for each 10-min epoch. In detail, the EEG signal is processed through a bandpass filter in frequencies between 80 and 500 Hz. The energy of the filtered signal is then computed based on RMS with 3 ms window defined by the equation

$$E(t) = \sqrt{\frac{1}{N}\sum_{k=t-N+1}^{t} x^2(k)}$$

The estimation of the energy threshold is 5 standard deviations (SD) above the overall RMS mean. Finally, all HFO events selected should be with a duration of more than 6 ms and contain more than 6 peaks greater than 3 SD above the filtered signal mean value.

The computational flowchart of MNI is shown in Supplementary Fig X.

MNI detector

MNI detector (MNI) was proposed by Zelmann et al.. For this approach, similar to the STE algorithm, the raw EEG signal is also filtered by the bandpass filter and the energy of the filtered signal is then computed using root mean square (RMS) with 2 ms window. A key block for MNI algorithm is the baseline detector, designed to construct the baseline interval. The baseline, defined as EEG segments with no oscillation, is detected by computing the wavelet entropy of the autocorrelation of the filtered signal. For each 125 ms EEG segment with 50% shift, the segment is considered as baseline if the wavelet entropy is greater than the threshold. From the baseline detector, possible HFO events could be detected by selecting energy of the filtered signal using RMS above the energy threshold for each epoch. The energy threshold is computed in two different methods if more than or less than 5 s/min of the amount of all detected baseline. If a sufficient baseline was found, the energy threshold is selected from rms values for each 10s baseline segment at the 99.9999 percentile of its empirical cumulative distribution function. If less baseline was detected, we consider the channel with continuous high-frequency oscillatory activity. The energy threshold is iteratively selected from rms values for each 60s segment at the 95 percentile of its empirical cumulative distribution function. The value is found by continuously detecting and removing the highest energy till no more new HFO events are detected. All possible HFO events from the two situations are finally selected with a duration of more than 6 ms. The computational flowchart of MNI is shown in Supplementary Fig X.

Deep Learning-based classifier:

From our previous study, we proposed a deep neural network (DNN) model that has shown promising performance against expert annotation. For pyHFO, we follow the same artifact and spk-HFO classifier design in our previous study and slightly modified the neural network due to limitations for this study, such as the time-consuming of large-scale network using CPU. The network is modified by reducing the computational cost of the model and using a data-augmentation strategy. The data augmentation strategy was applied by randomly flipping the EEG signals and randomly shifting the center of the HFO event forward and backward 50 ms. The reduction of the computational cost was applied by first reducing the redundancy in the dimension of the input and second by pruning the neural network using DepGraph.

Artifact detector:

For the artifact detector, we differentiate between artifacts and ‘Real HFOs’, defined as the union of spk-HFOs and non-spk-HFOs. All ‘Real HFOs’ were labeled as the positive samples and the artifacts were the negative samples.

To reduce the computational cost of the artifact detector, only one time-frequency plot was used for the artifact detector. Then, with the reduction of the input dimension, the architecture of the artifact detector is simplified by pruning the neural network using DepGraph (Figure X).

spk-HFO detector:

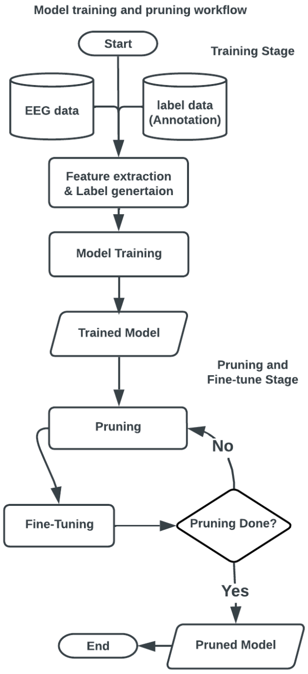
The spk-HFO detector classified the ‘Real HFOs’ into spk-HFO and non-spk-HFO; the spk-HFO were defined as positive samples, and the non-spk-HFO were defined as negative samples.

To reduce the computational cost of the artifact detector, concatenation of only the time-frequency plot and amplitude coding plot was used as input for the spk-HFO detector. Then, same as the artifact detector, with the reduction of the input dimension, the architecture of the spk-HFO detector is also simplified by pruning the neural network using DepGraph.

The interactive pruning was conducted for 5,000 iterations, and the model was fine-tuned every 250 iterations by five epochs. By pruning and fine-tuning the model, we get our simplified models with the best trade-off.

Training your own detectors

pyHFO allows users to train and prune their own detectors and classify detected HFOs as what our default artifact detector and spk-HFO detector do. To get the pruned models, users need to feed data and annotation files into the pipeline(figure), and follow the process of feature extraction, train model and prune, and fine-tune the trained model to get the user's own detectors.



For more details, users can follow the instructions in our Github Repo(https://github.com/roychowdhuryresearch/HFO-Classification/tree/7b88b55f43771b3894cd040011f2d4b71aa6124b/Pruning-pipeline).