An investigation into the development of a low cost, easy to use seizure analysis tool.

Cody Dean
Grand Valley State University

An investigation into the development of a low cost, easy to use seizure analysis tool.

Cody Dean

A Thesis Submitted to the Graduate Faculty of

GRAND VALLEY STATE UNIVERSITY

In

Partial Fulfillment of the Requirements

For the Degree of

Master of Science in Engineering

Biomedical Engineering

Seymour and Esther Padnos College of Engineering and Computing

December 2020

Dedication

To my loving and supportive wife, I am done with school now, I promise!

Acknowledgements

I would like to thank my thesis advisor Dr. Samhita Rhodes for her extreme patience and support as I navigated a less “traditional” Masters path. Without her guidance and insight, I truly would have never finished this thesis. A big thanks to Dr. Paul Fishback for his help and insight as we navigated the world of Python together. I would also like to thank Dr. David Zeitler for his expertise and guidance. Thank you to Dr. Mohamad Haykal for his expertise and time as I learned about the amazing work that the Spectrum Health Epilepsy Monitoring Unit does. I would like to thank Dr. Konstantin Elisevich and the numerous other researchers I met at Spectrum Health for their continued support and collaboration with the Biomedical Engineering program at Grand Valley State University.

Abstract

The need for collaborating and sharing data and research between doctors, researchers, universities and patients has never been more necessary. We are seeing firsthand how a deadly virus can completely devastate the world in a matter of months and being able to react quickly is the top priority. Open source tools are making it possible to share research and learnings about viruses like COVID-19 across countries, industries, and universities and these tools and philosophies extend across all areas of medical research.

The amount of data that is being collected within the medical industry is increasing at an exponential rate and there aren’t enough analysts and researchers to work through it all. Partnering with universities to leverage the capabilities of the students starts to address this issue. Not only can the institution glean more insights from their data, the students receive much needed experience and exposure to real world scenarios. The problem then becomes how to effectively develop the university research, so the clinicians can apply the learnings to their current work – how do we go from bench to bedside?

By partnering with the Spectrum Health Epilepsy Monitoring Unit (EMU) doctors, and leveraging previous research from Grand Valley State University Biomedical students, we introduced a framework for using open source development tools to build out and test novel algorithms in a way that makes it easier for clinicians to provide feedback for rapid development. By embracing a collaborative culture, we will be able to build and test custom tools in a clinical setting to better assist the clinician in helping their patients.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Problem and Clinical Significance

A revolution in health care is coming, but it may not be the one that people think. The Artificial Intelligence (AI) hype train has left the station and the medical industry is one of its many stops. People or patients are excited for the promise that soon they will be able to go into the doctor's office, get scanned with a tricorder device like the one from Star Trek, and instantly they will know exactly what is wrong with them. The reason this hope is so strong is because cost and time are increasing for the patient. Patients experience longer waits, invasive procedures, and the cost of healthcare has skyrocketed. Commercial startup companies are now starting to step in to address some of these issues. There has been an explosion of health-related wearable monitors, telemedicine is becoming more realistic and convenient, allowing patients to pay less, make it easier to see the doctor, and now the doctor can see more people. (The Economist, 2018) This is the true revolution that we are experiencing. There is no immediate future where we will be replacing our doctors with an AI robot that can diagnose every disease known to man. What we will be doing is making the whole healthcare experience more efficient using tools, such as the internet infrastructure, big data storage and computing, machine learning, and artificial intelligence. The real promise of all these tools is to make patient care more efficient and minimize the risks. Repetitive tasks can become automated to then free up health professionals to focus on the human interaction with the patient.

The website healthcare.digital is a thought leadership forum covering all things relating to digital health and health technology. They recently posted an article talking about the digital health hype cycle of 2018. A hype cycle is a branded graphical presentation for representing the adoption, maturity and application of different technologies. (Price, 2018) Figure 1 shows the hype cycle created by healthcare.digital.

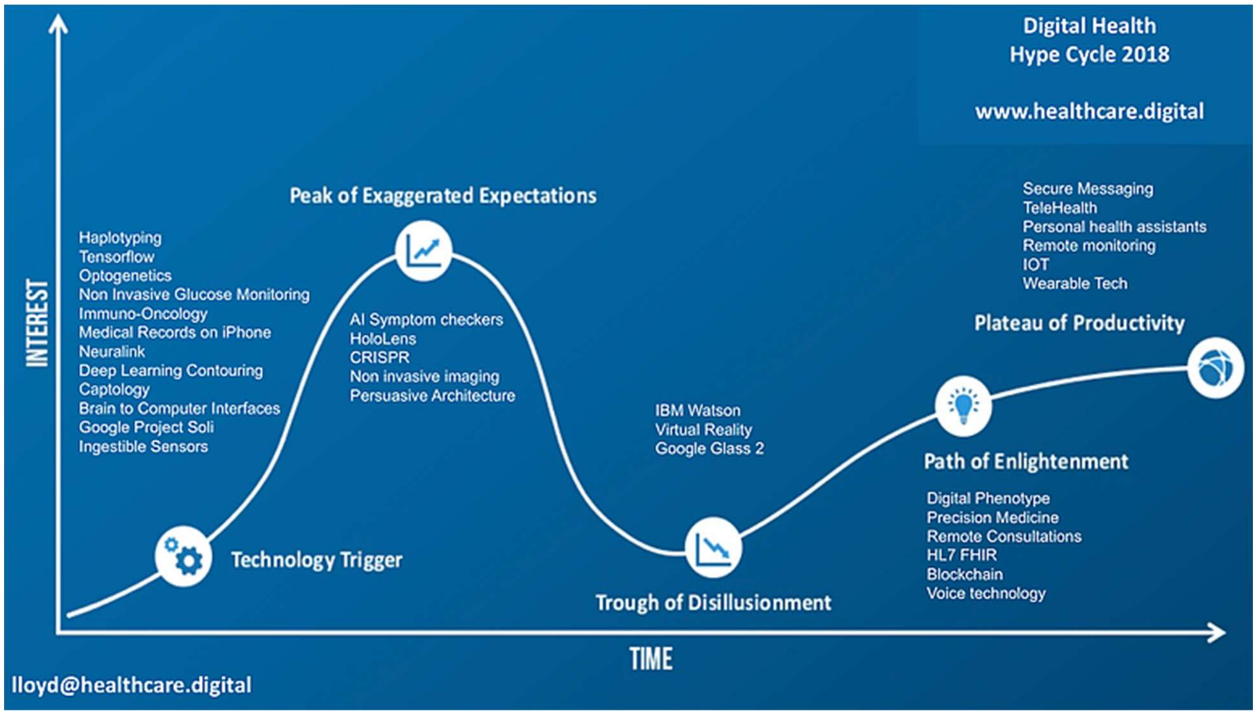


Figure 1: Digital Health Hype Cycle from www.healthcare.digital

Each hype cycle has 5 key phases of a technology life cycle. The Innovation Trigger is when a potential technology breakthrough kickstarts the process. The Peak of Exaggerated Expectations is when early publicity produces several success stories, driving up everyone's expectations. For the digital health hype cycle this is the AI Symptom checkers, where the patient can plug in all their symptoms and the AI machine diagnoses the patient. The next phase is the Trough of Disillusionment, where people are starting to realize maybe the promise of AI super doctors aren't an actual thing. The final two stages are the Slope of Enlightenment and the Plateau of Productivity. These phases are where the healthcare revolution will spring from. Things like TeleHealth, personal health assistants, remote monitoring, Internet of Things, and wearable tech. These are all pieces of technology that will help make the healthcare industry cheaper, faster and more efficient. (Price, 2018) These technologies aren't meant to replace healthcare professionals, rather they will allow doctors and nurses to have more time to spend on the patients that truly need help. It will give doctors more information to make informed decisions and helping to diagnose and treat correctly the first time. They will also be able to monitor their patients remotely to see how efficient the treatment was. This data will travel with the patient as they go from one doctor to the next, allowing everyone to use the same information to treat the patient. This revolution will depend on understanding the strengths and weaknesses of these new technologies and applying them where they make sense. An analogy would be going from a shovel to a backhoe. There is a lot of dirt that needs to be moved and worked through, and while we still need someone to run the backhoe, they now have a new technology to do it faster and more efficiently.

One potential roadblock that will slow this revolution is the gap between academia and the medical industry. There is a lot of research happening at the undergraduate and graduate level around using machine learning and AI for detection and diagnosis of pathophysiologies from medical signal and image processing. While these results are great, they remain entrenched in the research setting. There is still a need to transform these results into something that a medical professional can use clinical settings - a need to bring these techniques from bench to bedside.

Novartis is one of the largest multinational pharmaceutical companies in the world, based in Basel, Switzerland. They recently put out an article addressing the culture gap between academia and industry in biomedicine. In the article they reported on a panel discussion that explored the relationship between academic and industrial research, which was held at the Novartis Institutes for Biomedical Research (NIBR). Much of the discussion centered around how medical students graduate with misconceptions about how R&D works. Ken Kaitin, Director of the Tufts Center for Study of Drug Development said, “One of the things that I think is so critical now is an increase in education and understanding within the academic environment of the entire process of innovation.” (Kneller, 2016)

In the health section of the Stanford Social Innovation Review, there is an interesting interview from 2011 with Stephen Friend, an Ashoka Fellow in the United States and president of Sage Bionetworks. In the interview they address the issue of the medical information system being closed and how the method to share and collaborate on work is slow and primitive. The culture is a them-against-the-world mentality, and the “medical-industrial complex” is not incentivized to share amongst each other, let alone with patients. (Clay, 2011) Stephen goes on to describe their open source platform that allows researchers to share data and research to evolve their models as more data is collected. Their goal is a world, “where citizens could follow disease-related projects, become fans and join as followers or even funders.” (Clay, 2011) This further highlights the problem of the current culture around biomedical research being a closed and competitive system. All these examples point to a need for using open source development tools and philosophies within academia and the medical industry to foster a more collaborative and creative culture.

The specific problem that this thesis will address is the fact that there is currently a gap between the research being conducted by the Grand Valley State University (GVSU) Biomedical

Engineering (BME) program, and the Spectrum Health Epilepsy Monitoring Unit (EMU). There has been a lot of progress over the last couple years, specifically in new signal processing and analysis techniques applied to the intracortical EEG data routinely collected at the EMU. The issue has been how to get this research and these tools into the hands of the clinicians - EMU doctors and our efforts to move the research from bench to bedside are hampered by technological and administrative obstacles. But with new open source platform capabilities and cloud computing solutions, the technical issues surrounding working with big data in an efficient manner, can be addressed.

By helping move academic research to the clinical or industry setting, advances in new machine learning algorithms can improve the medical community's abilities to diagnose and treat pathophysiologies more efficiently. One example from Healthcare IT News talks about how the Hill Physicians Medical Group used AI and machine learning algorithms to increase the number of patient charts collected for risk adjustment effort by more than 200 percent. The medical group was able to take the electronic health records (EHR) they had access to and used the algorithms to extract important data that they had to collect manually before. (Siwicki, 2018) This is one example of how machine learning can be used help doctors be more efficient in their day to day work.

1.2 Purpose

The purpose of this thesis is to develop a process and system that allows research and algorithms developed by GVSU students and faculty to be easily used by researchers at the EMU as well as future students and staff at GVSU and other universities to assist clinicians in their diagnosis. As the researchers get access to new data, the old algorithms can evolve and become more robust by being tested on multiple sources of data. Collaboration between EMU and GVSU will guarantee that any findings can be quickly adapted to assist the EMU clinicians.

1.3 Objectives

The specific objects for this study are provided in Table 1 and we discuss how each was fulfilled throughout this thesis.

Objectives
Review of current seizure analysis tools both commercial and open source.
Investigation into open source software to analyze EEG signals in different formats.
Develop a GUI that allows user to load at least 1 data file in edf format for any patient.
Research and implement efficient methods for speed.
Develop an output format based on the customer's (Epilepsy Monitoring Unit clinicians) requirements.

Table 1: Objectives of the research

1.4 Thesis Roadmap

This thesis is organized in the following manner. Chapter 2 discusses the scientific literature regarding epilepsy and the tools and methods used for analyzing data from epileptic patients. In this section we will also discuss the benefits of open source software and how it can be used for better collaboration and development between academia and the medical industry. In Chapter 3 we will discuss the algorithm that we translated to the open source language Python and using these tools how we were able to focus on performance optimization and developing a framework to test ideas quickly within this environment. We will then review the results of the Python algorithm compared to the original algorithm developed within MATLAB using sample data from (Brunos, et al., 2014) and data collected from the Spectrum Health EMU. Finally, the findings will be discussed, future work will be proposed, and conclusions will be made.

2 Background and Literature Review

2.1 Epilepsy Diagnosis and Treatment

Epilepsy is a neurological condition which affects the nervous system. Some epilepsies are caused by problems in the early formation of the fetal brain while others occur as a result of brain trauma, stroke, infection, tumor or genetic susceptibility. (Johns Hopkins Medicine, 2020) Another term for epilepsy is seizure disorder. A person is considered to have epilepsy if they meet any of the following conditions:

1. At least two unprovoked (or reflex) seizures occurring greater than 24 hours apart.
2. One unprovoked (or reflex) seizure and a probability of further seizures similar to the general recurrence risk (at least 60%) after two unprovoked seizures, occurring over the next 10 years.
3. Diagnosis of an epilepsy syndrome. (Epilepsy is resolved for individuals who had an age-dependent epilepsy syndrome but are now past the applicable age or those who have remained seizure-free for the last 10 years, with no seizure medicines for the last 5 years.) (Fisher, 2014)

Seizures are manifested as disturbances in electrical activity within the brain. The seizures in epilepsy may be related to a brain injury or a family tendency, but in 6 out of 10 cases, epilepsy is idiopathic – meaning the cause is unknown. (Johns Hopkins Medicine, 2020) Epilepsy is more common than people know with 1 in 26 Americans developing epilepsy at some point in their lifetime. (Sirven & Shafer O. Patricia, 2014)

Antiepileptic drugs are the most common treatment to control seizures. The dose of the medicine is adjusted for better seizure control. Other treatments include vagus nerve stimulation where an electrical device is placed in the shoulder to stimulate a cranial nerve, along with special diets, such as the ketogenic diet, have shown promise. However, for more than a third of patients, breakthrough seizures, or seizures that continue despite the treatment, will occur. This condition is known as intractable or refractory epilepsy. The Epilepsy Monitoring Unit at Spectrum Health in Grand Rapids, Michigan, is one place where advanced medical and surgical therapy is offered. Patients stay in a private room for 5 to 7 days while a microphone and video equipment are setup to track seizure activity. Electrodes are placed on the surface of the scalp and implanted on the surface of the brain to track detailed brain activity over time. The clinical team changes the environmental conditions of the patient by introducing stressors, such as light, sound or insomnia to induce seizure activity so the events can be recorded. Millions of data points are recorded over this period and reviewed by a team of epileptologists and epilepsy neurosurgeons to determine which zone of the brain is the epileptogenic focus and must be resected to reduce or stop the seizures all together. (Treatments for Epilepsy, 2019)

2.2 Electroencephalogram (EEG)

Epilepsy is one of the few common clinical problems routinely diagnosed using an Electroencephalography (EEG) evaluation. EEG measures the electrical activity coming from thousands of neurons firing in the brain. Neurons communicate by sending electrical impulses between each other in a complex network structure throughout the brain. By “listening” to certain parts of the brain, neurologists can get a better understanding of how information is shared and if there are issues with the messages being transmitted or received. EEG can reveal specific features that are unique to several epilepsy syndromes. Rarely are these features recorded in healthy, particularly young patients. EEG recordings are critical pieces of information when it comes to identifying the seizure onset zone (SOZ). Epileptologists look at the first clinical signs and symptoms of a seizure along with the evolutions of the seizure symptomatology to identify important localizing clues. Noninvasive EEG recordings are taken from surface electrodes placed directly on the scalp. While the recording is less sensitive than invasive studies, it can help to provide a general area of where the SOZ is located.

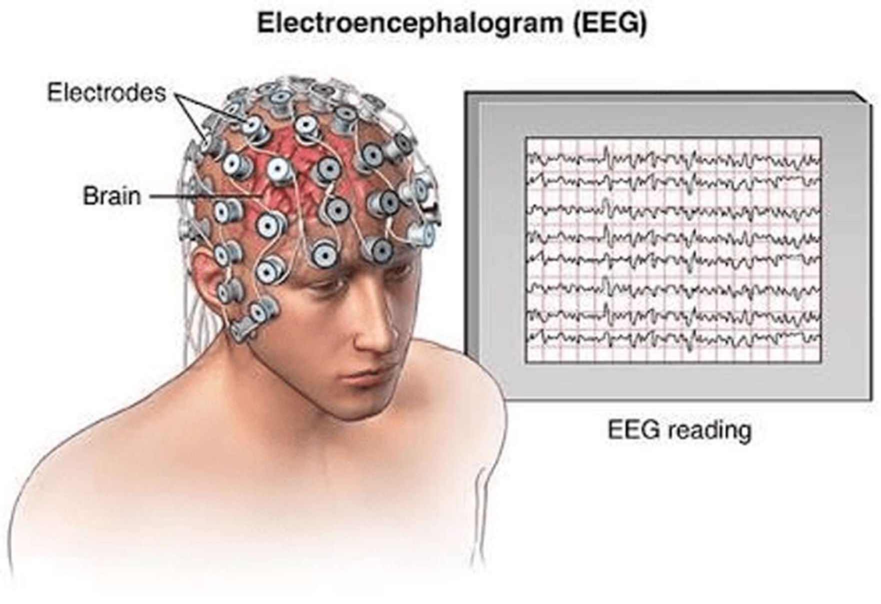


Figure 2: An illustration of EEG recording (Ref. EEG, Saint Luke's Health System)

Intracranial EEG (iEEG) is an invasive procedure where electrodes are placed inside the brain through surgery. This type of electrophysiological monitoring is also referred to as electrocorticography (ECoG). The electrodes are placed on the exposed cortex and in order to access this region the surgeon must perform a craniotomy, removing part of the skull to expose the brain surface. The electrodes can be in an array that is placed directly on the surface of the brain, or long needle electrodes that are placed deep within the brain. The precision and resolution with these methods are much more accurate than normal EEG and they can be left in over longer periods of time - from 5 to 10 days. The system is continuously sampling at 1000-2000Hz and can have between 4 to 256 different channels collecting signals simultaneously. This provides the clinicians with massive amounts of data that they can then use to try and locate the SOZ. With so much data, it becomes essentially that the clinicians are armed with methods to efficiently clean and analyze all the iEEG signals.

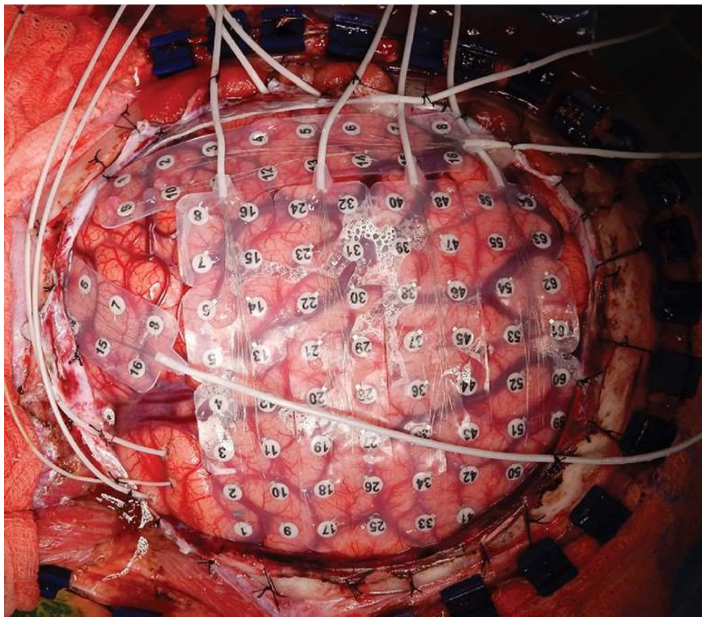


Figure 3: Intracranial EEG electrode placement (Journal of Neurosurgery)

2.3 Current SOZ Identification Methods

There are numerous ways that epileptologists use the iEEG signals to try and locate the SOZ, like analyzing features between the phase of the amplitude of the high gamma activity and the phase of lower frequency rhythms (Elahian, Yeasin, Mudigoudar, Wheless, & Babajani-Feremi, 2017) or using directed information to infer the causal connectivity graph between ECoG signals (Malladi, Kalamangalam, Tandon, & Aazhang, 2016). However, one method that has been continuing to generate a lot of attention are High Frequency Oscillations (HFOs). HFOs are defined as spontaneous EEG patters in the frequency range of 80-500Hz, consisting of at least four cycles that can be “clearly” distinguished from background noise. (Buzsaki, Horvath, Urioste, & et, 1992) By isolating the channels that have the most HFOs, the research might be able to better zero in on the SOZ. One method for HFO detection was presented by Sergey Brunos and team in the April 2014 issue of PLOS ONE. (Brunos, et al., 2014) This method was also adapted by a prior graduate student and applied to patients at the Spectrum Epilepsy Monitoring Unit (EMU) in fulfillment of their graduate work. Both methods break the identification of the HFO into stages. In the first stage, Epochs of Interest (EoIs) are identified by creating an envelope of the band-pass filtered signal using the Hilbert transform and then building an amplitude threshold from the standard deviation of the envelope. These EoIs are then merged if they are less than 10 ms apart and rejected if they don’t have a minimum of 6 peaks. In the second stage, the EoIs are transformed to the time-frequency space using the Stockwell transform and then HFOs are accepted or rejected based on their high and low frequency peaks. (Brunos, et al., 2014) Riazul adapted this method to MATLAB and while he kept the overall process flow the same, there were changes made to thresholds and frequency ratios to better handle the data coming from the EMU. (Islam, 2015)

Another HFO detection method for SOZ identification that seems promising is referred to as the CS algorithm (Cimbàlnìk-Stead). One of the big differences between the CS method and a method like the one described above (Brunos, et al., 2014) is based on the thresholding. The CS algorithm states that their method considers the non-stationarity of the EEG signal by using adaptive thresholds that are built from renormalizing metrics at specific frequencies in 10 second blocks. (Cimbàlnìk, Hewitt, Worrell, & Stead, 2018) While this paper doesn't go into depth on these other methods, they are worth mentioning, since they could also be adapted and built into an open source system that epileptologists and researchers could use to analyze their iEEG data.

2.4 Commercial vs Open Source Software

To have something be open source is a mindset and way of thinking. This is saying that through transparency and collaboration the output that can be created is far more impactful than if the work and research was closed off. For doctors and patients to be on the same page with diagnoses and treatment there has been a big push in the medical field to try and open more data information to patients and researchers. One great example of this is an initiative called OpenNotes. This is an international movement supporting and studying the effects of transparent communication by providing patients and families the meaningful notes described in a telehealth or office visit (Open Notes, 2020). Another example is an open source tool called Mendel, MD that could help doctors analyze patient's genetic data in order to diagnose disease caused by mutations. (Cardenas & et, 2017) The researchers designed this tool to be “intuitive enough to be used directly by physicians, even those who are not proficient in bioinformatics.” This is a value benefit of open source software vs. commercial software, the ability to customize a product to fit the needs of specific doctors, or clinicians. Commercial products tend to be inflexible. If the researcher finds the exact software for their exact problem, then the commercial version is great, but for everyone else, the data and problem are less defined, and the methods of analysis are constantly changing. By partnering with a research institute or university, the doctors can continuously refine and update their open source version with the latest methods. Also, crowd sourcing solutions and sharing information is a proven effective way to ensure the validity of the techniques leading to more robust algorithm development.

3 Methodology

There were two sets of iEEG data used to validate the open source HFO detection algorithm. The first was a single channel 30 second data set collected from (Brunos, et al., 2014) sampled at 2000Hz. The second was data collected from the Spectrum Health EMU. Data was recorded for evaluation starting the day after electrode implantation and was sampled at 1000Hz across 88 channels. The anti-aliasing filter was set at 300Hz on the EEG recording device.

3.1 HFO detection algorithm

Sergey Brunos and team published their HFO detection method in 2014 and a major focus of their algorithm was to identify and discard spurious detections caused by artifacts or sharp epileptic activity. They accomplished this by focusing on analysis in the time-frequency domain in the second stage of their algorithm. In prior graduate work (Islam, 2015) this HFO detection algorithm was adapted to MATLAB with some modifications that will be discussed later in this section. Ultimately, this paper will focus on the translation of prior graduate work on the Brunos HFO detection algorithm into a version written in the Python open source development language. We also explore the benefits of using an open source language for flexibility and optimization.

In the first stage of the algorithm, events of interest (EoIs), were identified from the filtered iEEG signal. The signal was first passed through a band-pass filter from 80-488Hz. An Infinite Impulse Response (IIR) Elliptic or Cauer filter was used with 60 dB minimum lower and upper stop band attenuation, 0.5 dB maximum pass band ripple, 10 Hz lower and upper transition width. (Islam, 2015) The one difference with the Python method was the filter order. The Python package, *scipy*, used to build the filter has a function called iirdesign, which will select the minimum order based on the requirements passed to it. The filtered signal was then scanned for events above the chosen threshold and sufficient duration to qualify as EoIs. (Brunos, et al., 2014) The envelope of the filtered signal was calculated using the Hilbert Transform and the envelope was then scanned for events. Figure 4 shows a diagram of the first stage.

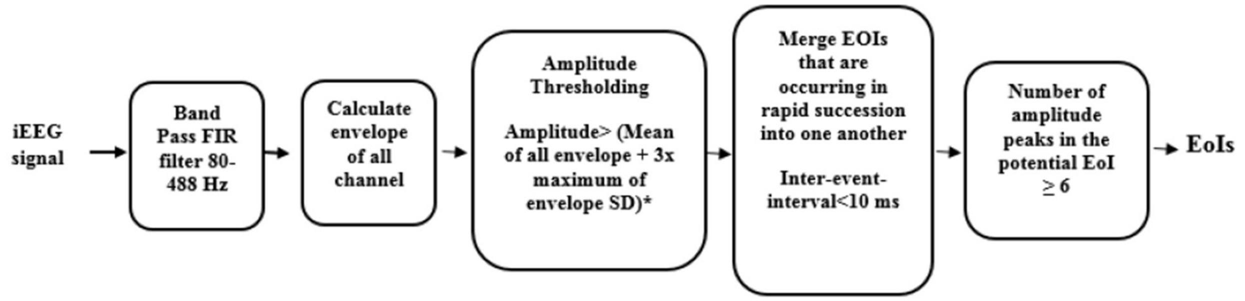

iEEG signal
Band Pass FIR filter 80-488 Hz
Calculate envelope of all channel
Amplitude Thresholding
Amplitude> (Mean of all envelope + 3x maximum of envelope SD)*
Merge EOIs that are occurring in rapid succession into one another
Inter-event-interval<10 ms
Number of amplitude peaks in the potential EoI ≥ 6
EoIs

Figure 4: EoI detection (stage 1 of HFO detection algorithm)

The first difference between the Brunos method and the prior graduate work is in the amplitude thresholding, shown in step 3 above. Brunos sets a different threshold for each channel, using the mean of the channel plus 3 standard deviations. In prior graduate research, they found that this identified too many EoIs for each channel and ultimately made the SOZ detection non-specific. They proposed that a threshold could be set using the mean plus 3 standard deviations of all the channels. The researcher argues this identifies the SOZ more effectively and is also more computationally efficient. (Islam, 2015) An event was marked when the envelope exceeded the threshold. The duration of the event was calculated using the upward and downward crossing of half the threshold. If the duration exceeded 6ms, it qualified as an EoI. Events with inter-event-intervals of less than 10ms were merged into one EoI. Events not having a minimum of 6 peaks greater than 2 SD from the mean baseline signal were rejected. (Brunos, et al., 2014) An example of an EoI from (Islam, 2015) is shown in figure 5.

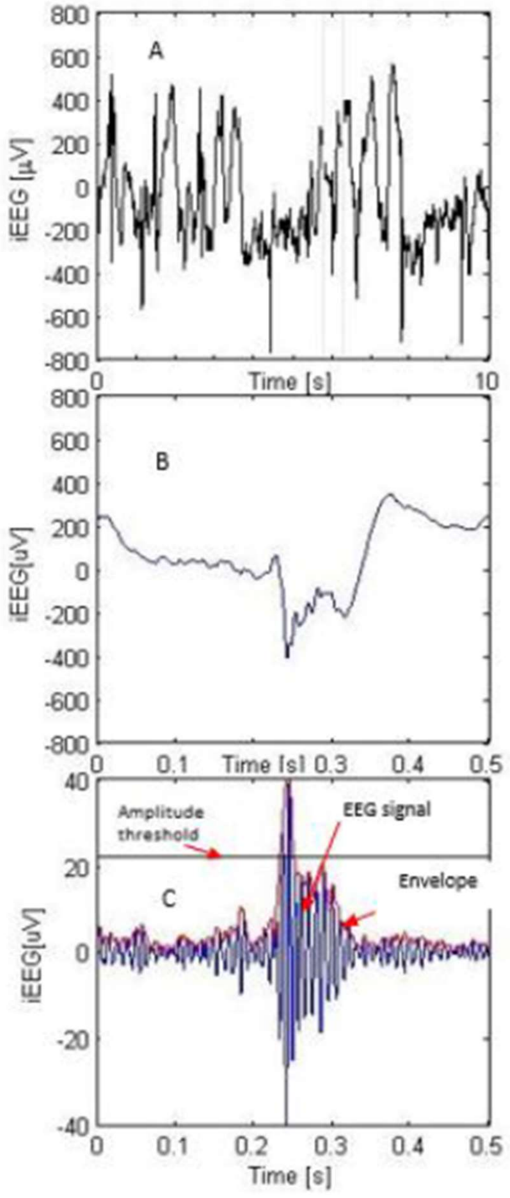


Figure 5: (a) 10s raw iEEG signal (b) 0.5s of iEEG signal containing an EoI (c) 80 to 488 Hz band passed filtered signal and envelope of the signal

In stage 2 the algorithm employed the Stockwell Transform to convert the signal to the time-frequency domain. In order to qualify as an HFO, the EoI detected in stage 1 must exhibit a high frequency peak, which is isolated from the low frequency activity by a spectral trough. Figure 6 shows a diagram of stage 2.

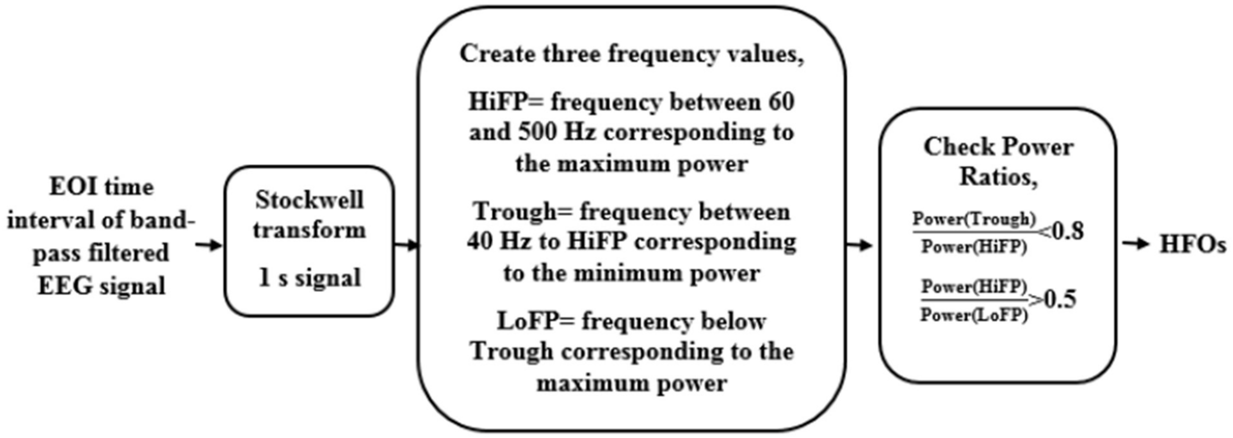

EOI time interval of band-pass filtered EEG signal
Stockwell transform
1 s signal
Create three frequency values,
HiFP= frequency between 60 and 500 Hz corresponding to the maximum power
Trough= frequency between 40 Hz to HiFP corresponding to the minimum power
LoFP= frequency below Trough corresponding to the maximum power
Check Power Ratios,
Power(Trough)/Power(HiFP) <0.8
Power(HiFP)/Power(LoFP) >0.5
HFOs

Figure 6: Time-frequency space analysis for recognition of HFOs among EoIs

The instantaneous power spectra were computed for all time points of the envelope within the full width at half maximum above threshold (FWHM). This boundary assures that the maximum of the envelope and its neighborhood above the threshold is taken into analysis. The instantaneous power spectrum for each time point was parameterized by three frequency bins in the following way. (Brunos, et al., 2014)

1. The high frequency peak (HiFP) was selected as the spectral peak of the HFO. This HiFP was selected in the spectral range from fmin (HiFP) = 60 Hz to 500 Hz. For the HFO in figure 7, the HiFP is 116 Hz because there is a spectral peak at that frequency.

2. The trough is defined as the minimum in the range between fmin (trough) = 40 Hz and the HiFP. For the HFO in Figure 3.6 there is a visible power gap around 80 Hz. The trough frequency is therefore 83 Hz.
3. The low frequency peak (LoFP) is defined as the closest local maximum below the trough. For figure 7, this is 47 Hz.

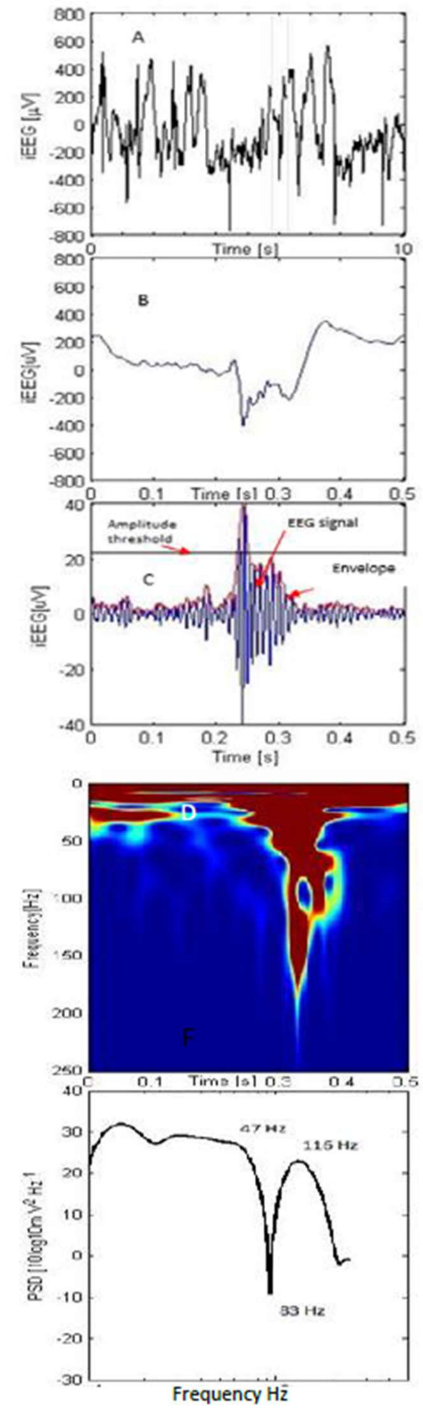


Figure 7: Time-frequency space representation of an HFO

These three frequency bins were used to distinguish HFOs in the instantaneous spectrum at each time point with the FWHM. To qualify as an HFO, the trough of sufficient depth

$$\text{Power (Trough)/Power (HiFP)} < 0.8$$

and a HiFP peak of sufficient height

$$\text{Power (HiFP)/Power (LoFP)} > \text{Rthr} = 0.5$$

These two conditions have to be satisfied by all instantaneous power spectra within the FWHM.

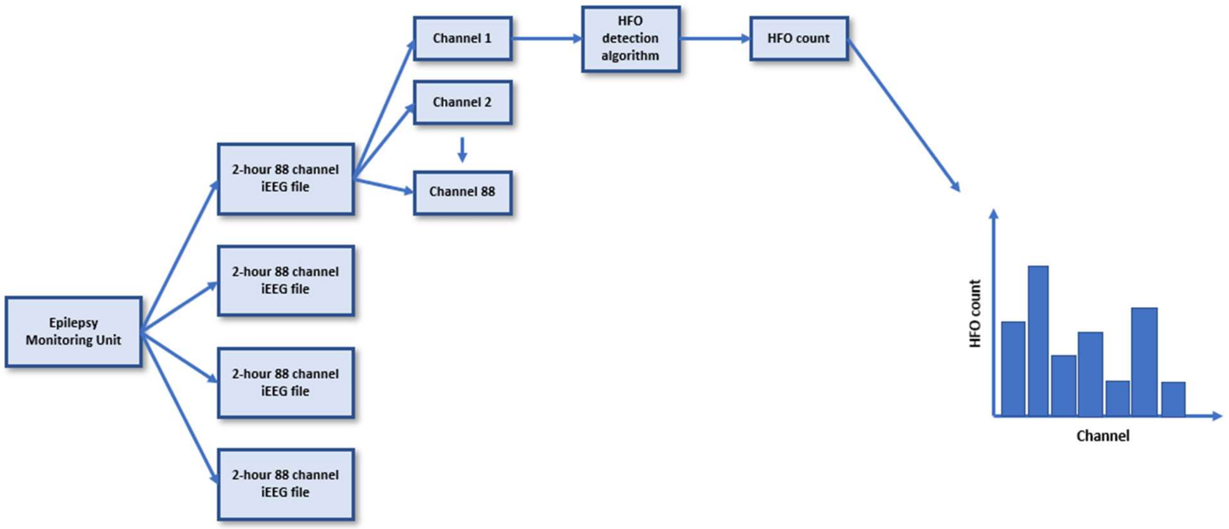
The EEG channels were ranked depending on total number of HFOs detected over two hours of study, and the channels with HFO rates higher than half the maximum rate contributed to the HFO region.

The method described above, which was originally written in MATLAB for prior graduate research, can now be thought of as a code block, the HFO detection block. From this point on, that block can be inserted into different workflows using iEEG signals as inputs. The following sections will cover how the MATLAB block was converted to a Python block, and then how this new open source version of the block was plugged into a more efficient processing framework for EMU clinicians in partnership with researchers at GVSU.

3.2 Performance optimization

There are two main ways in which we focused on optimizing the HFO detection algorithm. As stated above, if we think about the algorithm as a block that takes an input of raw iEEG data and outputs the HFO count of that data, then by optimizing the block itself, it can more efficiently analyze multiple channels of iEEG data. This is the first method and is shown in figure

8 below. By optimizing the HFO detection block, this could speed up the time it takes for the algorithm to process one 2-hour channel of data. This area was not the focus of performance optimization. To truly do this well would require a Python developer who knows the efficiencies of the code and could then optimizing each section of the algorithm. While this is a beneficial exercise, it should be the focus of future work as this system gets closer to production level.



Epilepsy Monitoring Unit
2-hour 88 channel iEEG file
2-hour 88 channel iEEG file
2-hour 88 channel iEEG file
2-hour 88 channel iEEG file
Channel 1
Channel 2
Channel 88
HFO detection algorithm
HFO count
HFO count
Channel

Figure 8: HFO detection algorithm flow chart

The areas that were explored deal more with a process re-arrangement and how we think about arriving at the final output that the researcher or clinician use. Since we are sampling between 1000 Hz and 2000 Hz, one channel from one 2-hour file becomes a very large dataset, anywhere from 7,200,000 to 14,400,000 data points, and this is just for one channel. Bring in 88 channels, we have over 1 billion data points. Trying to come up with a more efficient way to process a 2-hour file is not only helpful, it is necessary. The way we attempted to solve this was to add an extra step into the process flow as a channel modification block. Within this block we attempted three different ways to either reduce total channels, or randomly sample the channels with the goal of processing less data. From here the HFO detection algorithm is run on the modified channels and our processing speeds are much faster since we are reducing the data size by up to a factor of 100 in some instances.

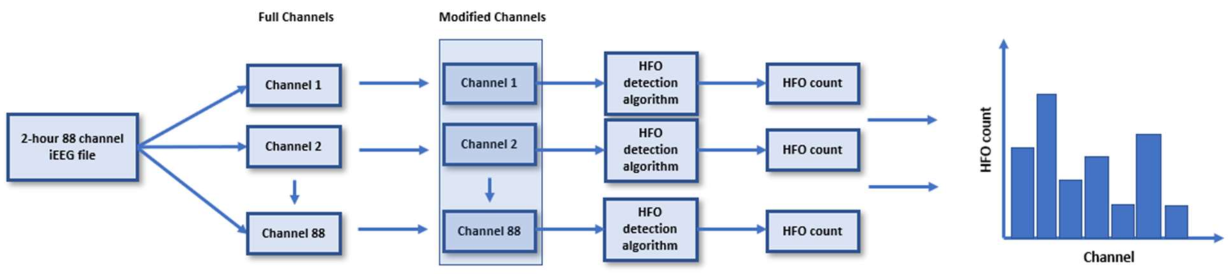
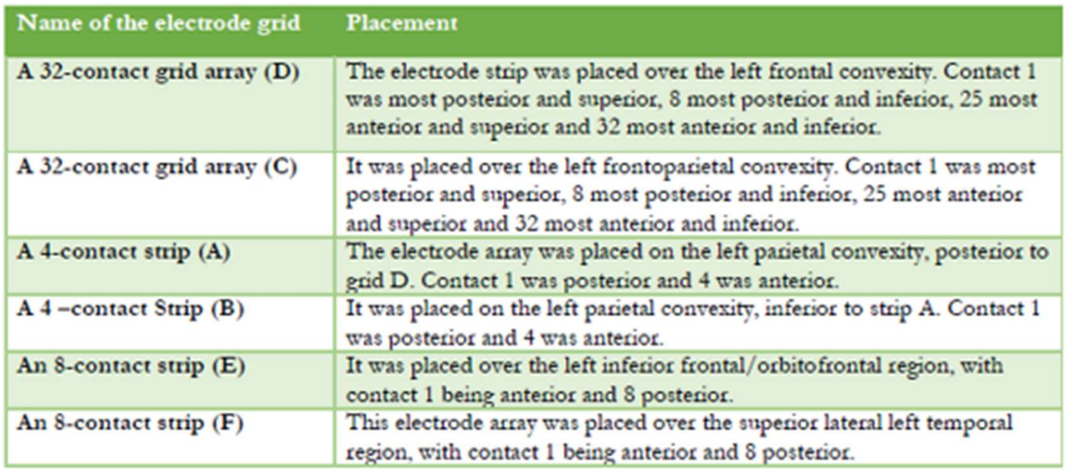


Figure 9: New process flow with extra screening stage added in.

For the channel aggregation method, the hypothesis was that if we could identify groups of electrodes that were similar, we could average these signals over the full 2-hour sample to decrease the total number of channels we would run through. Instead of processing all 88, there would be anywhere from 5-15 new aggregated channels to process. To test this idea, we looked at electrodes that were all on the same grid array and would then be on the same region when placed on the brain. After averaging the grid arrays below the 6 modified channels were run through the HFO detection algorithm to see if any patterns emerged between the top HFO channels from the full analysis and the grid array regions.



Name of the electrode grid	Placement
A 32-contact grid array (D)	The electrode strip was placed over the left frontal convexity. Contact 1 was most posterior and superior, 8 most posterior and inferior, 25 most anterior and superior and 32 most anterior and inferior.
A 32-contact grid array (C)	It was placed over the left frontoparietal convexity. Contact 1 was most posterior and superior, 8 most posterior and inferior, 25 most anterior and superior and 32 most anterior and inferior.
A 4-contact strip (A)	The electrode array was placed on the left parietal convexity, posterior to grid D. Contact 1 was posterior and 4 was anterior.
A 4 –contact Strip (B)	It was placed on the left parietal convexity, inferior to strip A. Contact 1 was posterior and 4 was anterior.
An 8-contact strip (E)	It was placed over the left inferior frontal/orbitofrontal region, with contact 1 being anterior and 8 posterior.
An 8-contact strip (F)	This electrode array was placed over the superior lateral left temporal region, with contact 1 being anterior and 8 posterior.

Table 2: Table from (Islam, 2015) showing location of grid arrays for patient WDH-022

The other two methods that were explored focused on random sampling of each channel. The assumption being made was if a channel had multiple HFOs across the whole signal, then by taking a random subset of the signal and running that through, it should still have more HFOs relative to the other channels, and ultimately be identified as a channel of interest even without processing the whole 2 hour signal.

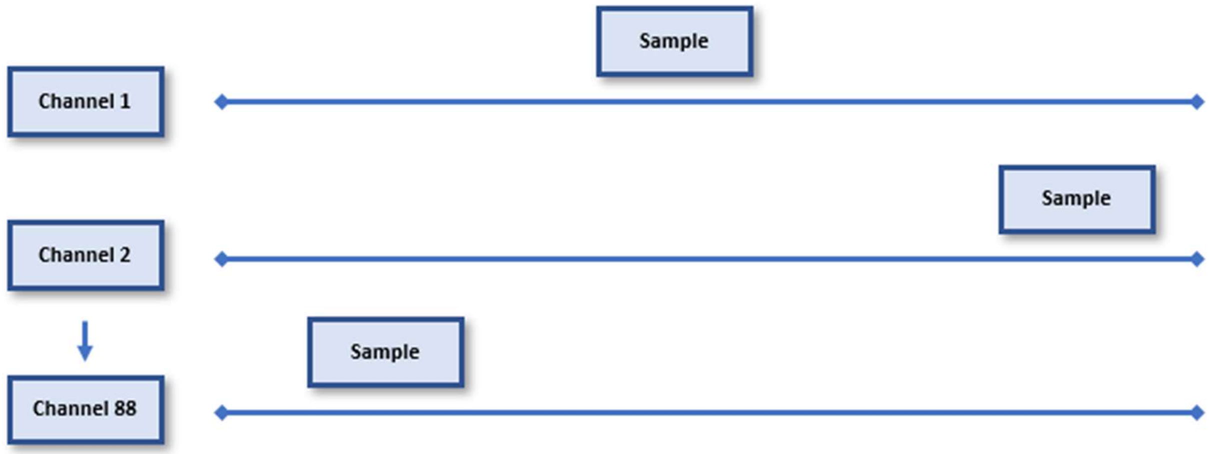


Figure 10: Diagram of how each channel was randomly sampled. Each channel is 120 minutes long and samples range from 1 to 30 minutes.

The final method that was explored was a stratified random sampling approach, which samples across intervals added in to make the sampling more uniform across the entire signal.

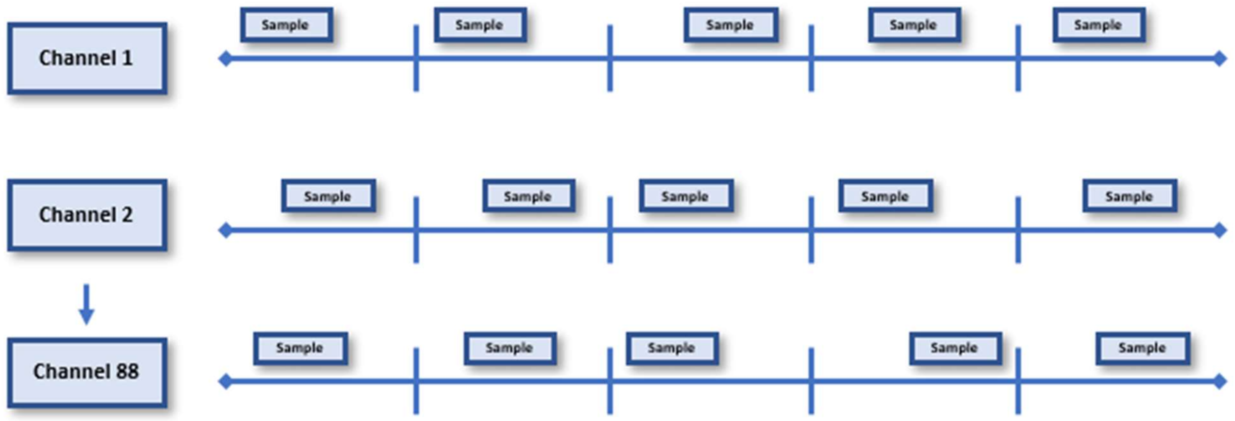
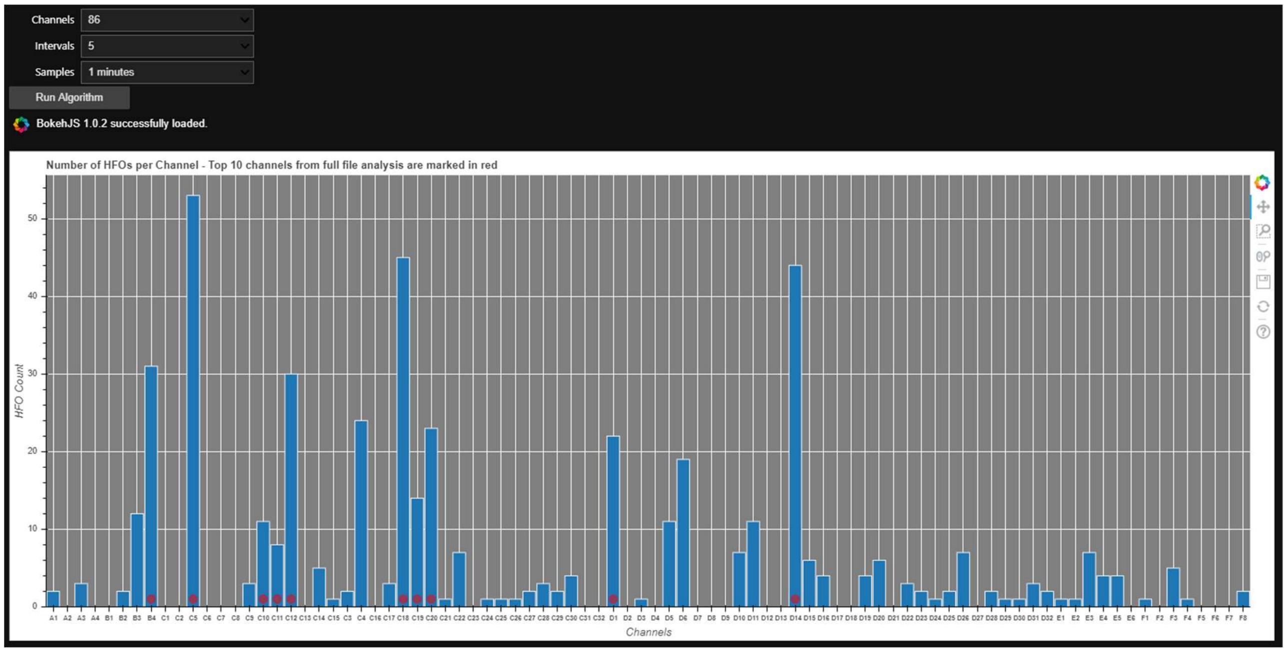


Figure 11: Diagram showing how each channel is broken into intervals before sampling.

By adding in this modification block we were able to greatly reduce the overall computation time of a 2-hour data file. In the results section we will discuss by how much we were able to reduce the computation time, and any trade-offs in resolution that were seen.

3.3 Development Tools

Open source development languages, such as Python, give the programmer flexibility and the ability to test and iterate through multiple solutions quickly. Jupyter Notebooks are another tool that we used to test out a framework for quick and agile development. Jupyter is an open source web application that allows interactive computing across multiple languages. Using these notebooks in combination with Python allowed us to create and test process flows that could allow for rapid prototyping between GVSU students and Spectrum researchers. Using interactive widgets within our code, we can test different user interfaces and also graphical outputs with the clinicians and researchers to see which versions are the easiest to use and also understand.


Channels 86

Intervals 5

Samples 1 minutes

Run Algorithm

BokehJS 1.0.2 successfully loaded.

Number of HFOs per Channel - Top 10 channels from full file analysis are marked in red
HFO Count
Channels
0
10
20
30
40
50

Figure 12: Example of a user interface built within Jupyter to test different optimization methods.

4 Results

4.1 Example Data

In order to validate how well the Python HFO detection version compared against the HFO detection version written in MATLAB, we ran two iEEG datasets through each and compared the results. The first dataset that was collected from (Brunos, et al., 2014) and was sampled at 2000 Hz. In both the Brunos paper and prior graduate work (Islam, 2015) each detected a total of 7 HFOs over a 30 second period. Figure 13 below is taken from (Islam, 2015) and clearly shows the 7 locations where HFOs were detected. These data sets were used to see how closely the Python version matched the "gold-standard" (Islam, 2015) version.

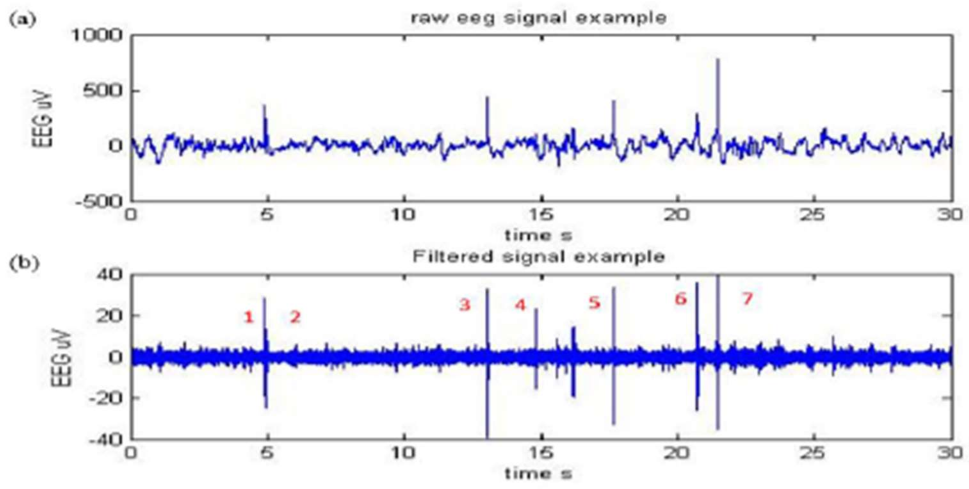


Figure 13: (a) 30 s example iEEG data sampled at 2000 Hz (b) Example data after 80 to 488 Hz band-pass filtering

In the Python version, the same filtered chart was generated and the HFO location was marked with red vertical lines. Since HFO 1 and 2 are so close, the lines overlap, however the same 7 HFOs were detected in the Python version as well.

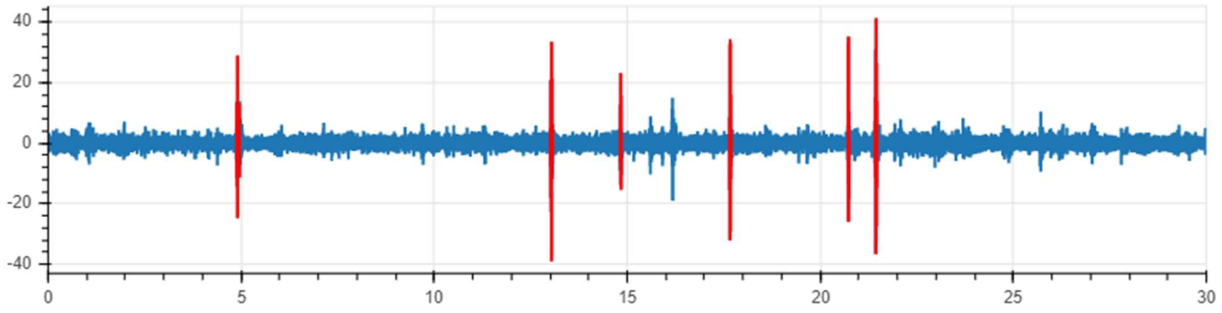
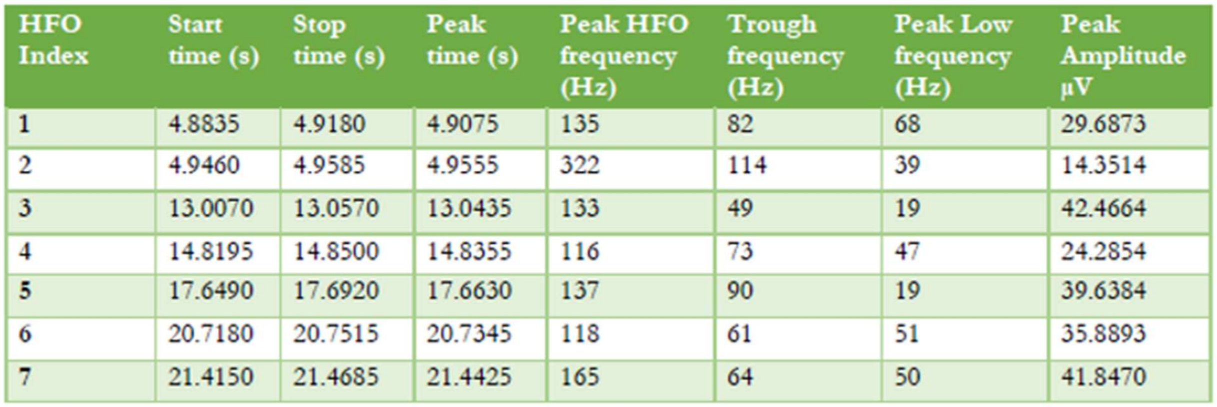
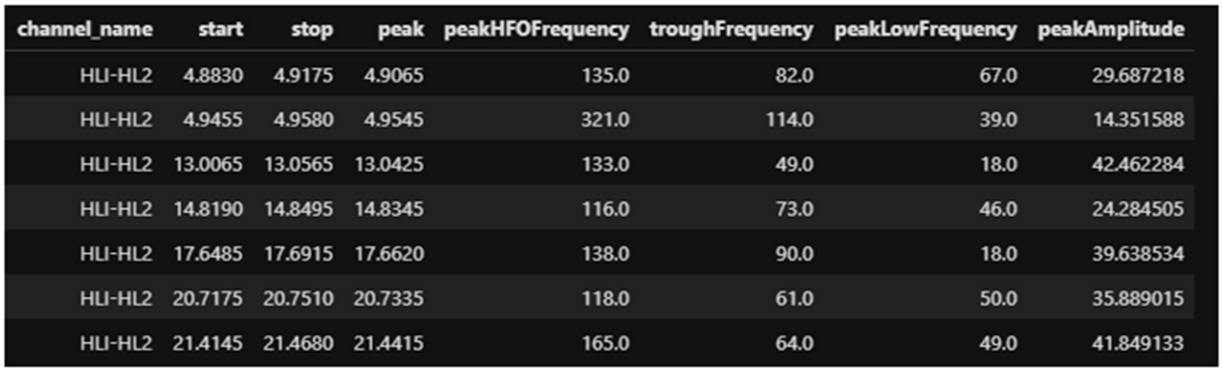


Figure 14: Filtered 30 s iEEG data sample using the Python HFO detection method. The vertical red bars indicate where HFOs were detected

In order to make sure the Python version is not only identifying the HFO location, but also the frequency content of each, we compared our results to the HFO result table in (Islam, 2015).



HFO Index	Start time (s)	Stop time (s)	Peak time (s)	Peak HFO frequency (Hz)	Trough frequency (Hz)	Peak Low frequency (Hz)	Peak Amplitude µV
1	4.8835	4.9180	4.9075	135	82	68	29.6873
2	4.9460	4.9585	4.9555	322	114	39	14.3514
3	13.0070	13.0570	13.0435	133	49	19	42.4664
4	14.8195	14.8500	14.8355	116	73	47	24.2854
5	17.6490	17.6920	17.6630	137	90	19	39.6384
6	20.7180	20.7515	20.7345	118	61	51	35.8893
7	21.4150	21.4685	21.4425	165	64	50	41.8470

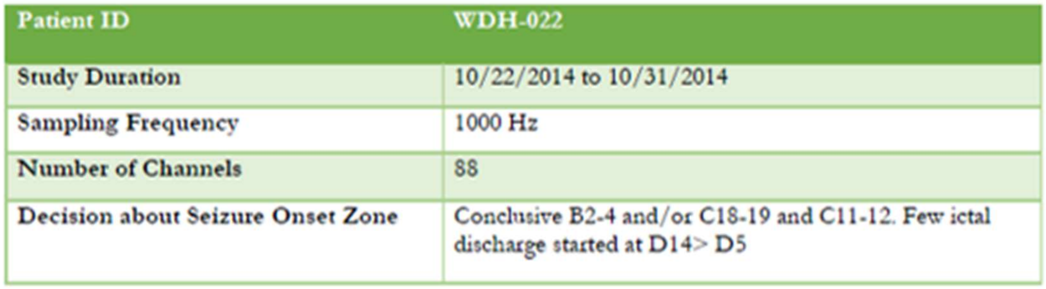


channel_name	start	stop	peak	peakHFOFrequency	troughFrequency	peakLowFrequency	peakAmplitude
HLI-HL2	4.8830	4.9175	4.9065	135.0	82.0	67.0	29.687218
HLI-HL2	4.9455	4.9580	4.9545	321.0	114.0	39.0	14.351588
HLI-HL2	13.0065	13.0565	13.0425	133.0	49.0	18.0	42.462284
HLI-HL2	14.8190	14.8495	14.8345	116.0	73.0	46.0	24.284505
HLI-HL2	17.6485	17.6915	17.6620	138.0	90.0	18.0	39.638534
HLI-HL2	20.7175	20.7510	20.7335	118.0	61.0	50.0	35.889015
HLI-HL2	21.4145	21.4680	21.4415	165.0	64.0	49.0	41.849133

Table 3: Tables comparing time location and frequency content between (Islam, 2015) version (top) and open source Python version (bottom)

4.2 Spectrum Health Epilepsy Monitoring Unit Data

The second file that was used to test the Python version came from the Epilepsy Monitoring Unit at Spectrum. This file was from Patient ID WDH-022 and the data was collected in October of 2014. A description of where the clinician identified the SOZ is detailed in Table 4.



Patient ID	WDH-022
Study Duration	10/22/2014 to 10/31/2014
Sampling Frequency	1000 Hz
Number of Channels	88
Decision about Seizure Onset Zone	Conclusive B2-4 and/or C18-19 and C11-12. Few ictal discharge started at D14> D5

Table 4: Table from (Islam, 2015) showing information about the patient file used to test the open source HFO detection algorithm

The 2-hour file from WDH-022 above was ran through both versions of the HFO detection algorithm. Each version identified the HFOs in all 88 channels along with their frequency content. The output of each was a bar chart showing channels across the x-axis and HFO count along the y-axis. As can be seen in the figure below, the open source Python version matches the MATLAB version almost perfectly. It is consistently lower by about 5%, which the reasons for will be reviewed in the discussion section. This is one way to compare the output, but a more realistic way is not to look at the exact HFO counts, but the channels with the highest HFO counts since these will be what the clinician uses to help identify the SOZ. If we look at the top 20 channels with the highest HFO counts from the MATLAB version, our Python version identifies the exact same 20 channels as having the highest HFO counts. Based on these results, we can confidently say that the Python version can be used as a replacement HFO detection method for the MATLAB version moving forward.

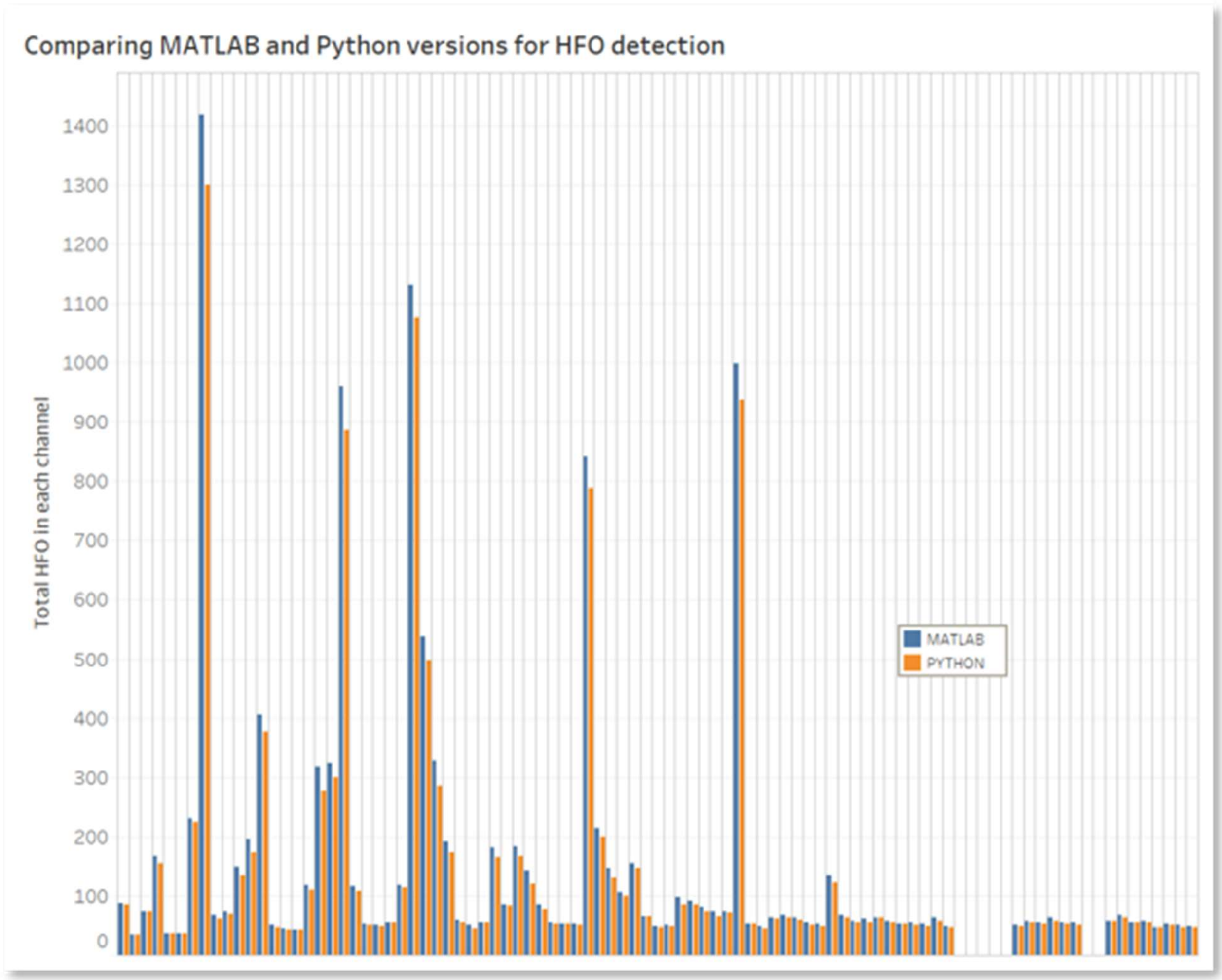


Figure 15: Bar charts comparing HFO counts across channels for the MATLAB and Python versions of the HFO detection algorithm

4.3 Performance optimization

As was mentioned in the Methodology section, processing time becomes an issue when trying to analyze a 2-hour file. For reference, running a full 2-hour file through the MATLAB HFO detection algorithm on the GVSU Blade server took between 4-5 hours. To run the same file through the Python HFO detection algorithm on a very large personal server took almost 3-4 times as long, anywhere between 15-18 hours. The Blade server specifications on-line state that it has up to 48 CPUs and 256GB of RAM, whereas the personal server has 8 CPUs and 64GB of RAM. This server size difference probably accounts for most of the processing time difference, which means clinicians and researchers in the future, who probably have machines with 4 CPUs and 16-32GB of RAM, will run into processing times lasting days. Based on that, this tool doesn't become practical and an exploration of ways to decrease processing time becomes necessary.

As was mentioned in the methodology section. Three methods of processing time reduction were explored. The first was an aggregation method. Averaging out the signals from the electrodes in the array gave us 6 channels which were then run through the HFO detection algorithm. For the grid array D, 60 HFOs were detected, C had 67, A had 53, B had 31, E had 52 and F had 51. It took approximately 10 minutes to run all 6 aggregated channels through the HFO detection algorithm. These outputs were much lower than expected and for the most part there wasn't one array that was drastically different than the others. Array C and D were the highest and if we look at the 20 highest individual channels, most of them are in the C or D grid, however this may just be a function of C and D also having the most electrodes on their arrays.

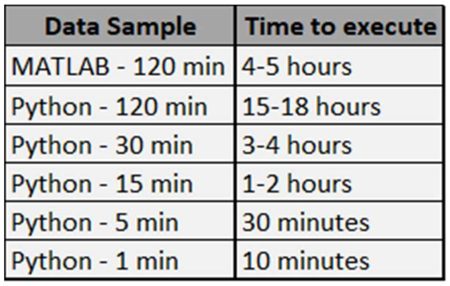
Reasons for this method not having the resolution we hoped for will be reviewed more in the discussion section.

The other processing time reduction methods were two ways to randomly sample the full signal. In the first approach, we looked at 4 different interval durations, 30 minutes, 15 minutes, 5 minutes and 1 minute. For each channel we randomly selected a subset from it, and ran that through the HFO detection algorithm, we then compared the highest 20 channels based on HFO count to the highest 20 channels on the full signal.

Top 20 channels based on HFO count					
MATLAB-120 min	**Python-120 min**	**Python-30 min**	**Python-15 min**	**Python-5 min**	**Python-1 min**
B4	B4	B4	C12	C12	C18
C18	C18	C12	D1	D14	C12
D14	D14	D14	C27	B4	B4
C12	C12	D1	C19	F5	C11
D1	D1	C27	D14	D19	D22
C19	C19	B3	B4	C5	D14
C5	C5	C21	D10	D6	D1
C20	C11	C28	C18	D1	C19
C11	C20	A4	C10	C10	C1
C10	C10	C19	C21	C20	C26
B3	B3	C9	F2	B3	D27
D2	D2	D9	C14	E4	E1
C4	C4	C5	B3	C19	D25
C21	C21	C13	A1	C22	C15
C27	C27	D6	E1	D5	C20
C25	C25	F4	C5	C25	C14
A4	A4	D31	C17	D3	D5
D5	D5	C25	C30	D32	C25
C3	C3	F3	E5	A4	C17
D3	D3	D15	D30	F8	C4

Table 5: Comparing top 20 channels based on HFO count for each sample subset

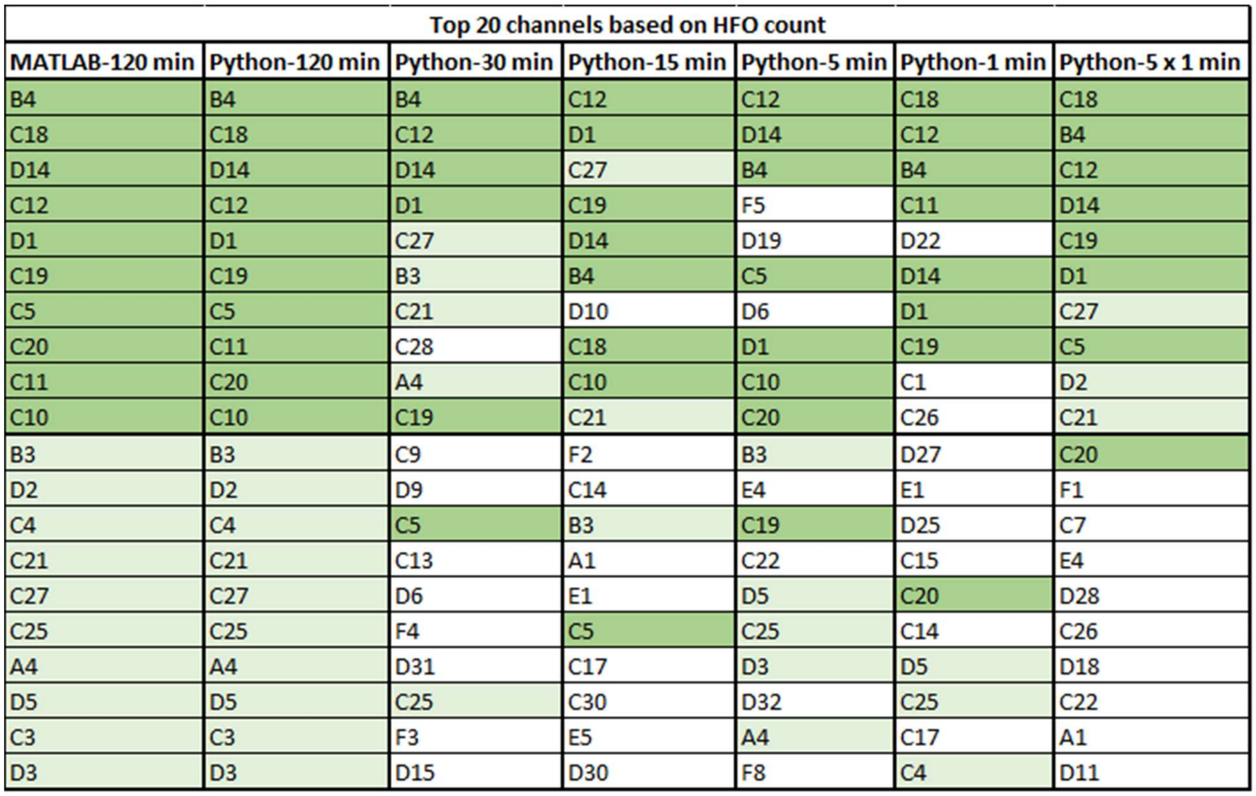
Each sample subset correctly identified over 50% of the top HFO channels when compared to the full signal. This becomes even more impressive when looking at the time required to run each subset through the HFO detection algorithm.



Data Sample	Time to execute
MATLAB - 120 min	4-5 hours
Python - 120 min	15-18 hours
Python - 30 min	3-4 hours
Python - 15 min	1-2 hours
Python - 5 min	30 minutes
Python - 1 min	10 minutes

Table 6: Approximate time it took to run 88 channels through the HFO detection algorithm using various sample subsets

For the stratified random sampling method, we broke each channel into 5 intervals and then took a random 1-minute sample within each interval to run through the HFO detection algorithm. The HFO count for each sample was added together for what was effectively a 5-minute sample. Processing time was a little longer compared to the 5-minute signal with no intervals, approximately 40 minutes verse 30 minutes, however, the output looked much more promising.



Top 20 channels based on HFO count						
MATLAB-120 min	Python-120 min	Python-30 min	Python-15 min	Python-5 min	Python-1 min	Python-5 x 1 min
B4	B4	B4	C12	C12	C18	C18
C18	C18	C12	D1	D14	C12	B4
D14	D14	D14	C27	B4	B4	C12
C12	C12	D1	C19	F5	C11	D14
D1	D1	C27	D14	D19	D22	C19
C19	C19	B3	B4	C5	D14	D1
C5	C5	C21	D10	D6	D1	C27
C20	C11	C28	C18	D1	C19	C5
C11	C20	A4	C10	C10	C1	D2
C10	C10	C19	C21	C20	C26	C21
B3	B3	C9	F2	B3	D27	C20
D2	D2	D9	C14	E4	E1	F1
C4	C4	C5	B3	C19	D25	C7
C21	C21	C13	A1	C22	C15	E4
C27	C27	D6	E1	D5	C20	D28
C25	C25	F4	C5	C25	C14	C26
A4	A4	D31	C17	D3	D5	D18
D5	D5	C25	C30	D32	C25	C22
C3	C3	F3	E5	A4	C17	A1
D3	D3	D15	D30	F8	C4	D11

Table 7: Comparing top 20 channels based on HFO count for each sample subset with 5-minute interval method.

In this table, the top 10 channels are colored a dark green and channels 11-20 are colored light green. For the 5 x 1-minute interval method, of the first 10 channels identified, all of them are within the top 20 from the full analysis, and 7 out of 10 are within the top 10 from the full analysis. There is a distinct drop off after the top 11 channels which we will discuss further in the next section. While the aggregation method for time reduction needs continued exploration, the random sampling methods have potential to greatly reduction processing time for clinicians.

5 Discussion

Based on preliminary results there seems to be high potential to use an open source language, such as Python, in combination with Jupyter to build out a seizure analysis tool. Not only were we able to translate the MATLAB HFO detection algorithm to Python, we were then able to build on it in numerous ways and also test different user interfaces and visual outputs. This is the true benefit of open source development tools. A developer can continuously build the latest algorithms into their software, but also have the flexibility to change them for their specific application. In this instance, the HFO detection algorithm is primarily to help identify channels of interest, not necessarily to identify each HFO with 100% accuracy and because of this we are able to focus on building out the screening aspects of the tool more. With further researchers, clinicians might decide that they also need to understand the frequency content of each HFO in the channel, and we would able to modify this tool to also accomplish this. Another benefit of using a language like Python is its popularity. There are millions of developers creating free packages for the language and publishing them for everyone to use. For this HFO detection algorithm both (Brunos, et al., 2014) and (Islam, 2015) used a Stockwell transform or s-transform, which is a generalized short-time Fourier transform, to identify frequency content of the HFO over the specified window. This is a function that comes with the MATLAB signal processing toolbox and is straight-forward to use, however it was more difficult to find a similar function in Python. While this may be considered a limitation, we were able to find the same function, written for Python, on a GitHub repository, free for anyone to use. After verifying the accuracy was within 1-2Hz, I was able to build this into our algorithm and get results within 5% of the MATLAB version. Since the functions weren't identical, we believe this is the reason for the small difference in overall HFO detections, but the difference was consistent and could be compensated for in future versions.

Since this is meant to be a tool used by clinicians, speed and processing time become top priorities for development. In the results section we discussed two methods to try and deal with this. The aggregation method seemed promising at first, however, by averaging the signals this mostly likely smoothed them to the point that the high frequency content was removed. This would explain why only 50-60 HFOs were detected in the aggregated signals while some of the highest individual channels had over 1000. One way to try and get better resolution would be to reduce the number of signals aggregated into one channel. Some of the grid arrays contained 32 electrodes which would lend itself to signal smoothing, and maybe only 2-3 channels per aggregated signal would make more sense, but at this point the time reductions starts to become minimal and ultimately may not be worth the loss of signal information to averaging. For long term development, stratified random sampling has the most potential for significant processing time reduction.

From Table 5 we can see that each sampling subset identifies at least 50% of the top 20 channels (as identified by the full data analysis gold standard) for HFOs, and all of them identify the topmost channel. Even the 1-minute subset identifies the top 5 channels within the 20. These results help initiate a conversation about what is most important for the clinician to see when using this tool. Do they need to perfectly identify the top 20 channels, or maybe just the channels that are outliers? By zeroing in on what is most beneficial for the SOZ detection, we can start to tune the algorithm accordingly.

Combining 1-minute samples across equally spaced intervals appears to be an option that has good resolution with greatly reduced processing time. Using 5 intervals, the algorithm identified 7 of the top 10 HFO channels and took roughly 40 minutes to run through the entire 86 channels. Compared to almost 18 hours for the full 2-hour analysis. It is interesting to note that after the top 11 channels, none of the other 12-20 spots identify a top HFO channel. It seems that for this method to work, there needs to be a certain number of HFOs within the channel. It is unclear what the number of HFOs needs to be but should be the subject of future work.

6 Future Work

One area of focus for future research should be around algorithm optimization, specifically parallel processing methods. Since the detection algorithm is on a for loop and performs the same operation on each channel in series, we should be able to leverage more cores and have the algorithm running in parallel. One Python package that allows an individual to use all their computer cores for processes like this is called DASK and should be explored (DASK, 2020).

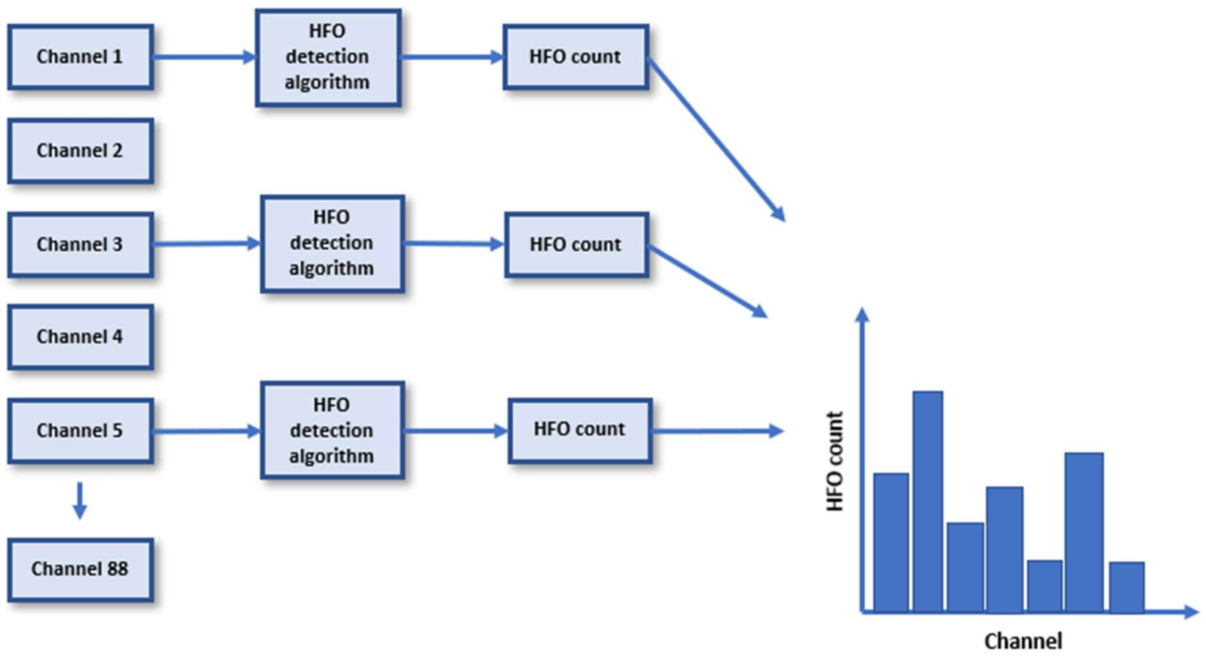


Figure 16: HFO detection algorithm with parallel processing built in

Another area that needs to be explored is how the researchers and clinicians would use the output from the HFO Detection Algorithm. Now that we have an actual development tools, we could have discussions with the EMU team and change the algorithm in real-time. This would give us more tangible feedback and would also make it easier for the clinicians to understand what features they have at their disposal. The focus of this research was around HFO detection, but there are many other ways that we can visualize and analyze the EEG signals that can also be incorporated into a tool like this.

The addition of features like network connectivity become much easier by embracing python and Jupyter as a development framework. We can also begin to tweak parameters within the original HFO detection algorithm. For example, some of the rejection rations are based on real HFO parameters and can be tuned to the researcher or clinician. They may find that a higher peak, or lower trough is more appropriate to identify an HFO which can be easily changed with this new tool. New functions and modules can be added in quickly by multiple researchers, students, and teachers providing a truly collaborative and adaptive tool.

7 Conclusions

This thesis contributes to bridging the gap between industry and academia and shows that we can take the work done by previous GVSU graduates, translate it into an open source version, and quickly iterate and build upon it to meet the needs of the researchers working in the Spectrum Epilepsy Monitoring Unit.

We have shown that the Python version we developed gets comparable accuracy to the MATLAB version from prior graduate work (Islam, 2015) as well as speed when balancing for server specifications. From here we were able to explore the suite of open source development tools available with Python and Jupyter and laid out a framework for future development. Using these tools, students and teachers can began to rapidly prototype different algorithms related to data coming out of the EMU and work with clinicians and researchers to test better ways to visualize and use the output coming out of these tools.

Moving forward it will be the responsibility of the clinicians, doctors, teachers and students to embrace the use of these tools for the greatest impact. By making these processes standard practice, sharing ideas and methodologies will become second nature. Spectrum Health will have access to the latest advancements in signal and image processing while the GVSU students will be receiving invaluable experience and helping to create a real impact in patients' lives.

8 Appendices

8.1 Full HFO Count Table

HFO Count with subsets							
Channels	Full Sample (Riazul)	Full Sample (Cody)	30 minutes	15 minutes	5 minutes	1 minute	5 - 1-minute intervals
C18	1131	1075	3	61	1	27	73
B4	1418	1299	372	63	42	12	58
C12	960	886	371	204	73	12	48
D14	999	937	196	70	47	10	41
C19	538	496	77	76	8	9	31
D1	841	789	120	96	16	9	24
C27	183	167	108	95	1	0	20
C5	406	377	62	20	18	0	19
D2	213	200	15	9	1	0	17
C21	191	173	90	41	0	0	15
C20	327	285	13	6	10	2	11
F1	57	57	6	3	1	0	10
C7	45	42	3	2	1	0	10
E4	63	58	7	2	9	0	9
D28	53	53	40	3	0	0	8
C26	85	83	7	2	0	8	7
D18	68	64	1	0	1	0	7
C22	59	56	3	1	7	0	7
A1	88	85	13	22	2	0	5
D11	81	73	6	6	2	1	5
C24	56	54	2	1	0	1	5
C11	323	300	17	7	2	11	4
C10	318	278	6	48	16	0	4
C4	196	174	29	10	2	1	4
C25	181	165	41	9	5	1	4
A4	167	154	82	11	3	0	4
C28	142	120	88	5	1	0	4
D10	91	86	1	62	1	1	4
B2	37	37	1	0	1	0	4
D5	155	146	13	5	6	1	3
D22	134	122	10	12	0	11	3
C9	119	109	70	10	2	0	3
C17	118	115	23	19	2	1	3
F3	56	54	40	2	1	0	3

D7	49	46	2	1	0	0	3
C13	116	108	61	2	0	0	2
F2	67	64	4	37	1	0	2
C30	55	52	3	19	0	0	2
E3	54	52	4	2	1	0	2
C14	53	50	38	32	0	2	2
B3	231	224	105	22	9	0	1
C3	149	134	9	4	1	1	1
A3	74	73	5	3	2	0	1
D23	67	63	10	0	1	0	1
D17	64	62	5	0	1	0	1
D26	64	63	3	0	0	0	1
D31	63	58	42	2	0	0	1
D25	61	55	2	1	1	5	1
D27	58	56	6	1	0	8	1
F4	58	55	43	2	1	0	1
E2	57	54	5	2	0	0	1
D29	56	51	8	4	1	0	1
E6	56	51	39	2	0	0	1
C31	53	53	4	13	0	0	1
D21	52	49	1	3	0	0	1
D30	52	49	6	15	1	0	1
E1	51	49	4	21	2	7	1
F8	48	47	32	2	3	0	1
F5	46	46	3	1	19	0	1
B1	37	37	32	0	0	0	1
A2	34	34	23	1	1	0	1
D3	147	131	13	9	5	0	0
D4	106	99	5	2	3	0	0
D9	97	85	65	0	0	0	0
C29	86	78	5	0	0	0	0
D12	74	66	7	5	0	1	0
C2	73	70	35	4	1	0	0
D13	73	71	2	1	0	0	0
C1	67	62	8	3	0	9	0
D6	66	65	43	3	17	0	0
D19	64	59	6	0	19	0	0
D24	58	54	6	2	0	0	0
C16	55	54	3	1	0	0	0
D20	55	50	2	3	1	0	0
E5	55	52	17	15	2	0	0

D15	53	53	40	2	1	0	0
F6	53	51	4	2	1	0	0
C32	52	50	3	2	0	0	0
C6	51	47	4	2	0	0	0
D8	51	48	37	0	1	0	0
C15	50	49	3	11	0	3	0
C23	50	45	1	2	0	0	0
F7	50	46	3	11	0	0	0
D32	49	46	5	2	4	0	0
D16	48	44	31	2	0	0	0
C8	43	43	2	1	1	0	0

8.2 Python HFO Detection Code

HFO Detection Notebook

GVSU Biomedical Engineering- November 30th, 2020

Before using this notebook, go to the Kernal tab above and select 'Restart & Run All'. The interactive widgets will allow the user to change the number of channels to analyze, the number of intervals to break each channel into, and finally the sample size used within each interval to calculate the number of HFOs. This notebook and code were developed for the Biomedical Engineering department at GVSU as a part of the thesis for Cody Dean.

```
#import all packages and set options for the notebook and the cell outputs
import warnings
warnings.filterwarnings('ignore')

from IPython.display import HTML, display
from ipywidgets import interact, interactive, fixed, interact_manual, Button, Layout, IntSlider
import ipywidgets as widgets

import pandas as pd
import numpy as np
from numpy import random
import time

import scipy.special
from scipy import signal
from stockwell import st
import pyedflib

from bokeh.plotting import figure, show, output_notebook, save, output_file, reset_output
from bokeh.models import HoverTool, value, LabelSet, Legend, ColumnDataSource, Range1d
from bokeh.transform import dodge

pd.set_option('display.max_columns', 999)
pd.set_option('display.max_rows', 999)

#Load data from edf file and convert to a pandas dataframe for easier use
f = pyedflib.EdfReader('BA2681LD_1-1.edf')
n = f.signals_in_file
signal_labels = f.getSignalLabels()
Fs = int(f.samplefrequency(0))
Indices = np.arange(n)
#convert edf data into dataframe
df = pd.DataFrame(columns=signal_labels)
for i in Indices:
    df[signal_labels[i]] = f.readSignal(i)
#drop all channels we don't need for this analysis
df.drop(['E','DC01','DC02','DC03','DC04','EEG Mark1','EEG Mark2','REF','REF','ECG','ECG','Events/Markers']
, axis=1, inplace=True)

#for this current version we will reference data that is already loaded, for future versions we can explor
e loading that data within a separte function and ipywidgets
#This will also reference data from the file loaded such as sampling frequency
def HFO_Detection(Channels,Intervals,Samples):
    #full limit goes to 86
    start = time.time()
    HFOcount = []
    sig_length = df.shape[0]
    interval_len = int(sig_length/Intervals)

    for col in list(df.columns[0:Channels]):
        hfocount=0 #initialize hfocounter for intervals
        #sample is in minutes times 60 seconds/minutes
        seconds = Samples * 60
        #period is seconds times sampling frequency to get total data points
        period = seconds * Fs
        #generate random number with enough space on end to be able to rand select last data point
        #add a 1 for when 120 minutes is selected, this will be randint(0,0) without, don't think it will
affect other samples
        #using array of intervals to select a random sample within each one
        for i in np.arange(0,Intervals):
            x = random.randint(i*interval_len,((i+1)*interval_len-period)+1)
            results = nbt_doHFO_revised4(df[col][x:x+period])
            #put this in to remove blank rows, not sure why those are adding, need to revisit
            results = results[results['start'] > 0]
            #count hfos for each interval, sum up for total for channel
            hfocount = hfocount+results.shape[0]

        HFOcount.append({'Channel':col, 'HFO count':hfocount})

    HFOcount = pd.DataFrame(HFOcount)
    #need to reset notebook each time so it plots a new chart when Run Algorithm is selected
    reset_output()
    output_notebook()

    channels = HFOcount['Channel']
    counts = HFOcount['HFO count']

    p = figure(x_range=channels, title='Number of HFOs per Channel - Top 10 channels from full file analys
is are marked in red', plot_width=1550, plot_height=600, background_fill_color="grey")

    p.vbar(x=channels, top=counts, width=0.9, line_color='white')
    # add a circle for the top 10 HFO channels from initial analysis of full file
    p.circle(['B4','C18','D14','C12','D1','C19','C5','C20','C11','C10'], [1,1,1,1,1,1,1,1,1,1], size=10, c
olor="red", alpha=0.5)

    p.y_range.start = 0
    p.xaxis.axis_label = 'Channels'
    p.xaxis.major_label_text_font_size = '6pt'
    p.xaxis.major_label_text_font_style = 'bold'
    p.yaxis.axis_label = 'HFO Count'
    p.grid.grid_line_color="white"

    show(p)
    end = time.time()
    print('It took approximately %s seconds for the HFO Detection Algorithm to analyze the selected data'
%(round(end-start,2)))

    return;

#main function that allows us to dynamically set variables
im = interact_manual(HFO_Detection, Channels=np.arange(1,87), Intervals=np.arange(1,21), Samples=[('2 Hour
s',120),('30 minutes',30),('15 minutes',15),('5 minutes',5),('1 minutes',1)])
im.widget.children[3].description = 'Run Algorithm'
display(im)

#automatic time-frequency algorithm for detection of HFOs
#This was translated to python from the origianl publication at http://www.plosone.org/article/info%3Adoi%
2F10.1371%2Fjournal.pone.0094381
# ========================================================================
def nbt_doHFO_revised4(Signal):
    #set initial parameters - 30sec
    #fs = 2000
    #hp = 80
    #lp = 500

     #set initial parameters - BA2681LD_1-1
    fs = 1000
    hp = 80
    lp = 488

    #set initial parameters - 11.edf
    #fs = 200
    #hp = 58
    #lp = 62

    channel_name = 'Test'
    time_thr = np.ceil(0.006*fs)

    #parameters for filtering
    Fst1 = (hp-10)/(fs/2)
    Fp1 = hp/(fs/2)
    Fp2 = lp/(fs/2)
    Fst2 = (lp+10)/(fs/2)
    Ast1 = 40
    Ap = 0.5
    Ast2 = 40

    #merge IoEs
    maxIntervalToJoin = 0.01*fs #10 ms

    #reject events with less than 6 peaks
    minNumberOscillations = 6
    dFactor = 2

    #Stage 2
    bound_min_peak = 40 #Hz, minimum boundary for the lowest ("deepest") point
    ratio_thr = 0.5 #threshold for ratio
    min_trough = 0.2 #20%
    limit_fr = 500
    start_fr = 60 #limits for peak frequencies

    # 1. filtering ------------------------------------------------------------------------a
    #see end of notebook for notes on conversion
    wp = [Fp1,Fp2]
    ws = [Fst1,Fst2]
    gpass = Ap
    gstop = Ast1

    #bandpass filter
    [b,a] = signal.iirdesign(wp, ws, gpass, gstop, analog=False, ftype='ellip')
    Signal_filtered = signal.filtfilt(b,a,Signal)

    # 2. envelope ---------------------------------------------------------------------
    env = np.abs(signal.hilbert(Signal_filtered))

    # 3. threshold --------------------------------------------------------------------
    #THR = 3 * np.std(env) + np.mean(env) - use this threshold for 30sec file
    THR = 27.19761355 #use this threshold for 2 hour EMU files

    # 4. Stage 1 - detection of EoIs --------------------------------------------------
    #assign the first and last positions at 0 point
    env[0] = 0
    env[-1] = 0

    pred_env = np.zeros(len(env))
    pred_env[1:len(env)] = env[0:len(env)-1]
    pred_env[0] = pred_env[1]

    if np.size(pred_env,0) != np.size(env,0):
        pred_env = pred_env.T

    #np.where is python equivalent to matlab find(), returns index where conditions are true
    #for some reason returns array in one cell, explore later, but if we take the first element that gives
us what we want
    t1 = np.where((pred_env < (THR/2)) & (env >= (THR/2)))[0] #find zero crossings rising
    t2 = np.where((pred_env > (THR/2)) & (env <= (THR/2)))[0] #find zero crossings falling

    trig = np.where((pred_env < THR) & (env >= THR))[0] #check if envelope crosses the THR level rising
    trig_end = np.where((pred_env >= THR) & (env < THR))[0] #check if envelope crosses the THR level falli
ng
```

```
    Detections = pd.DataFrame(0.0, index=range(len(trig)), columns=['channel_name','start','peak','stop','
peakAmplitude'])
    nDetectionCounter = 0

    # check every trigger point, where envelope crosses the threshold,
    # find start and end points (t1 and t2), t2-t1 = duration of event;
    # start and end points defined as the envelope crosses half of the
    # threshold for each EoIs

    for i in np.arange(0,len(trig)):

        if ((trig_end[i] - trig[i]) >= time_thr):

            nDetectionCounter = nDetectionCounter + 1
            #not sure why it is buried in layers of arrays, add the index at the end to get out int
            k = np.where((t1 <= trig[i]) & (t2 >= trig[i]))[0][0] #find the starting and end points of env
elope
            Detections['channel_name'][nDetectionCounter-1]=channel_name

            #check if it does not start before 0 moment
            if t1[k] > 0:
                Detections['start'][nDetectionCounter-1] = t1[k]
            else:
                Detections['start'][nDetectionCounter-1] = 1

            #check if it does not end after last moment
            if t2[k] <= len(env):
                Detections['stop'][nDetectionCounter-1] = t2[k]
            else:
                Detections['stop'][nDetectionCounter-1] = len(env)

            #calculate the max and where it occurs
            peakAmplitude = np.max(env[t1[k]:t2[k]])
            ind_peak = np.argmax(env[t1[k]:t2[k]])

            Detections['peak'][nDetectionCounter-1] = (ind_peak + t1[k])
            Detections['peakAmplitude'][nDetectionCounter-1] = peakAmplitude

    if (nDetectionCounter > 0):

        joinedDetections = joinDetections(Detections,trig)

        checkedOscillations = checkOscillations(joinedDetections, Signal_filtered)

        PSvalidated = PS_validation_all(checkedOscillations, Signal, env)

    else:
        PSvalidated = Detections

    return PSvalidated;

# ==========================================================================
# Any EoI that are close to each other and almost indistinguishable, are merged into one EoI
def joinDetections(Detections,trig):

    warnings.filterwarnings("ignore")
    #Merge EoIs with inter-event-interval less than 10 ms into one EoI
    nOrigDetections = len(Detections)
    #merge IoEs
    fs = 1000
    maxIntervalToJoin = 0.01*fs #10 ms

    #fill result with first detection
    joinedDetections = pd.DataFrame(0.0, index=range(len(trig)), columns=['channel_name','start','peak','s
top','peakAmplitude'])
    joinedDetections['channel_name'][0] = Detections['channel_name'][0]
    joinedDetections['start'][0] = Detections['start'][0]
    joinedDetections['stop'][0] = Detections['stop'][0]
    joinedDetections['peak'][0] = Detections['peak'][0]
    joinedDetections['peakAmplitude'][0] = Detections['peakAmplitude'][0]
    nDetectionCounter = 0

    for n in np.arange(1,nOrigDetections):

        #join detection
        if (Detections['start'][n] > joinedDetections['start'][nDetectionCounter]):
            nDiff = Detections['start'][n] - joinedDetections['stop'][nDetectionCounter]

            if (nDiff < maxIntervalToJoin):
                joinedDetections['stop'][nDetectionCounter] = Detections['stop'][n]

                if (joinedDetections['peakAmplitude'][nDetectionCounter] < Detections['peakAmplitude'][n])
:
                    joinedDetections['peakAmplitude'][nDetectionCounter] = Detections['peakAmplitude'][n]
                    joinedDetections['peak'][nDetectionCounter] = Detections['peak'][n]

            else:
                #initialize struct
                nDetectionCounter = nDetectionCounter + 1
                joinedDetections['channel_name'][nDetectionCounter] = Detections['channel_name'][n]
                joinedDetections['start'][nDetectionCounter] = Detections['start'][n]
                joinedDetections['stop'][nDetectionCounter] = Detections['stop'][n]
                joinedDetections['peak'][nDetectionCounter] = Detections['peak'][n]
                joinedDetections['peakAmplitude'][nDetectionCounter] = Detections['peakAmplitude'][n]

    #clear out zero rows, probably not efficient code, just trying to get everything working at this point
    joinedDetections = joinedDetections.replace(0,np.nan).dropna()
    #should go back and figure out why these go to floats
    joinedDetections['start'] = joinedDetections['start'].astype(int)
    joinedDetections['stop'] = joinedDetections['stop'].astype(int)
    joinedDetections['peak'] = joinedDetections['peak'].astype(int)

    return joinedDetections;

# ==========================================================================
# HFO needs to have a sufficient number of oscillations - this funtion verifies and cleans out ones that d
on't
def checkOscillations(Detections, Signal):

    # Reject events not having a minimum of 6 peaks above 2 SD
    # ----------------------------------------------------------------------
    # set parameters
    #reject events with less than 6 peaks
    channel_name = 'HLI-HL2'
    minNumberOscillations = 6
    dFactor = 2
    nDetectionCounter = -1
    AbsoluteMean = np.mean(np.abs(Signal))
    AbsoluteStd = np.std(np.abs(Signal))
    checkedOscillations = pd.DataFrame(0.0, index=range(len(Detections)), columns=['channel_name','start',
'stop','peak',
                                                                                          'peakHFOFrequency','troughFr
equency',
                                                                                          'peakLowFrequency','peakAmpl
itude'])

    for n in np.arange(len(Detections)):
        #get EEG for interval
        #add 1 since python indexes to 1 before last value
        intervalEEG = Signal[Detections['start'][n]:Detections['stop'][n]+1]
        # compute abs values for oscillation interval
        absEEG = np.abs(intervalEEG)
        #look for zeros
        zeroVec = np.where(np.multiply(intervalEEG[0:-1],intervalEEG[1:])<0)[0]
        nZeros = np.size(zeroVec)

        nMaxCounter = 0;
        if (nZeros > 0):
            #look for maxima with sufficient amplitude between zeros
            for ii in np.arange(nZeros-1):

                lStart = zeroVec[ii]
                lEnd = zeroVec[ii+1]
                dMax = np.max(absEEG[lStart:lEnd])

                if (dMax > AbsoluteMean + dFactor + AbsoluteStd):
                    nMaxCounter = nMaxCounter + 1

        if (nMaxCounter >= minNumberOscillations):
            nDetectionCounter = nDetectionCounter + 1
            checkedOscillations['channel_name'][nDetectionCounter] = Detections['channel_name'][n]
            checkedOscillations['start'][nDetectionCounter] = Detections['start'][n]
            checkedOscillations['stop'][nDetectionCounter] = Detections['stop'][n]
            checkedOscillations['peak'][nDetectionCounter] = Detections['peak'][n]
            checkedOscillations['peakHFOFrequency'][nDetectionCounter] = 0
            checkedOscillations['troughFrequency'][nDetectionCounter] = 0
            checkedOscillations['peakLowFrequency'][nDetectionCounter] = 0
            checkedOscillations['peakAmplitude'][nDetectionCounter] = Detections['peakAmplitude'][n]

    if (nDetectionCounter < 0):
        checkedOscillations['channel_name'][0] = channel_name
        checkedOscillations['start'][0] = -1
        checkedOscillations['stop'][0] = -1
        checkedOscillations['peak'][0] = -1
        checkedOscillations['peakHFOFrequency'][0] = 0
        checkedOscillations['troughFrequency'][0] = 0
        checkedOscillations['peakLowFrequency'][0] = 0
        checkedOscillations['peakAmplitude'][0] = 0

    checkedOscillations = checkedOscillations.iloc[checkedOscillations['channel_name'].nonzero()[0]]
    #should go back and figure out why these go to floats
    checkedOscillations['start'] = checkedOscillations['start'].astype(int)
    checkedOscillations['stop'] = checkedOscillations['stop'].astype(int)
    checkedOscillations['peak'] = checkedOscillations['peak'].astype(int)

    return checkedOscillations;

# ------------------------------------------------------------------------------
# Stage 2 - recognition of HFOs among EoIs
# ------------------------------------------------------------------------------
#==========================================================================
def PS_validation_all(Detections, Signal, env):

    # set parameters
    channel_name = 'HLI-HL2'
    fs = 1000
    bound_min_peak = 40 #Hz, minimum boundary for the lowest ("deepest") point
    ratio_thr = 0.5 #threshold for ratio
    min_trough = 0.2 #20%
    limit_fr = 500
    start_fr = 60 #limits for peak frequencies
    #THR = 3 * np.std(env) + np.mean(env)
    THR = 27.19761355 #use this threshold for 2 hour EMU files
    nDetectionCounter = -1
    PSvalidated = pd.DataFrame(0.0, index=range(len(Detections)), columns=['channel_name','start','stop','
peak',
                                                                              'peakHFOFrequency','troughFr
equency',
                                                                              'peakLowFrequency','peakAmpl
itude'])

    for n in np.arange(len(Detections)):

        if ((Detections['peak'][n] != -1) & ((Detections['stop'][n]-Detections['start'][n]) < fs*1)):
            #find the sec interval where the peak occurs
            det_start = Detections['peak'][n]-Detections['start'][n]
            det_stop  = Detections['stop'][n]-Detections['peak'][n]

            #define 0.5 sec interval where HFOs occur and take for
            #analysis 0.1 sec before + interval (0.5 sec) + 0.4 sec after
            #in total 1 sec around an HFO is analyzed

            if (np.floor(Detections['peak'][n]/(fs/2)) == 0):   #peak occured in first 0.5 sec

                det_peak = Detections['peak'][n]
                intervalST = Signal[0:fs]
                interval_env = env[0:fs]

            elif (np.floor(Detections['peak'][n]/(fs/2)) == len(Signal)/(fs/2)-1):  #peak occured last 0.5 
sec

                det_peak = np.mod(Detections['peak'][n], fs)
                intervalST = Signal[(len(Signal)-fs) : len(Signal)]
                interval_env = env[(len(Signal)-fs) : len(Signal)]

            else:                                                   #peak occured in middle of signal

                det_peak = int(np.mod(Detections['peak'][n], (fs/2))+np.floor(0.1*fs))
                t_peak_interval = int(np.floor(Detections['peak'][n] / (fs/2)))
                intervalST = Signal[int(t_peak_interval*fs/2-np.floor(0.1*fs)) : int(t_peak_interval*fs/2+
np.ceil(0.9*fs))]
                interval_env = env[int(t_peak_interval*fs/2-np.floor(0.1*fs)) : int(t_peak_interval*fs/2+n
p.ceil(0.9*fs))]

            #---------------------------------------------------------------------------
            # Python version uses stockwell transform package from github https://github.com/claudiodsf/st
ockwell.git.  Verify with MATLAB version is within 1-2Hz
            STSignal = st.st(intervalST, 0, limit_fr)

            #************************
            #*****ADDED AN UPPER LIMIT TO INDICES SO WE WEREN'T ACCESSING THE
            #ST TRANSFORM OR ENVELOPE OUTSIDE ITS LENGTH
            #********************
            upper_index=len(interval_env)

            #------------------------------------------------------------------------------
            # analyze instantaneous power spectra
            true_HFO = 0 # counter for recognized HFOs

            for tcheck in np.arange(np.max([det_peak-det_start,]), np.min([det_peak+det_stop,upper_index])
):

                #check if the envelope is above half of the peak+threshold
                if (interval_env[tcheck] > (0.5*(Detections['peakAmplitude'][n] + THR))):

                    #for maximum upper start_f frequency
                    maxV = np.max(np.abs(STSignal[start_fr:,tcheck])) #HFO peak
                    maxF = np.argmax(np.abs(STSignal[start_fr:,tcheck]))
                    maxF = maxF + start_fr+1

                    #search for minimum before found maximum
                    minV = np.min(np.abs(STSignal[bound_min_peak:maxF, tcheck])) #the trough
                    minF = np.argmin(np.abs(STSignal[bound_min_peak:maxF, tcheck])) #the trough
                    minF = minF+bound_min_peak+1

                    #print(tcheck,minF,minV,maxF,maxV)

                    #check for sufficient difference
                    #set signal to look through to x so we can pull peaks out if it
                    x = STSignal[0:minF, tcheck]
                    if np.size(np.abs(x)) == 0:
                        peaks=[]
                    else:
                        peak_loc = signal.find_peaks(np.abs(x))[0] #this find the location of the peak, to
```

```
find the actual value you need x[peaks]
                        peaks = x[peak_loc]

                    if np.size(peaks) == 0:
                        fpeaks=np.floor(minF/2)
                        peaks = np.abs(STSignal[fpeaks, tcheck])
                        ratio_HFO=0
                        ratio_LowFr=0
                    else:
                        ratio_HFO = float(10*np.log10(maxV) - 10*np.log10(minV)) #ratio between HFO peak a
nd the trough
                        ratio_LowFr = float(10*np.log10(peaks[-1]) - 10*np.log10(minV)) #ratio between Low
Frequency peak and the trough


                    #check the difference and check for sufficient trough
                    if ((upper_index>0)&(ratio_HFO>(ratio_thr*ratio_LowFr))&(ratio_HFO>(min_trough*10*np.l
og10(maxV)))&(maxF<500)):
                        true_HFO=true_HFO+0
                    else:
                        true_HFO=true_HFO+1

            if ((upper_index > 0) & (true_HFO==0)): #all conditions are satisfied
                #search for peak
                tcheck = det_peak
                maxF = np.argmax(np.abs(STSignal[start_fr:,tcheck]))
                maxF = maxF + start_fr+1 #try to understand why change -1 to +1, maybe python indexing?

                #search for minimum before found maximum
                minF = np.argmin(np.abs(STSignal[bound_min_peak:maxF, tcheck])) #the trough
                minF = minF+bound_min_peak+1

                #check for sufficient difference
                #fpeaks=[]
                #fpeaks.append(signal.find_peaks_cwt(np.abs(STSignal[0:minF, tcheck]),np.arange(1, 2))[0])
#low frequency peak
                fpeaks = signal.find_peaks(np.abs(STSignal[0:minF, tcheck]))[0]

                nDetectionCounter = nDetectionCounter + 1

                #times are translates to seconds
                PSvalidated['channel_name'][nDetectionCounter] = Detections['channel_name'][n]
                PSvalidated['start'][nDetectionCounter] = Detections['start'][n]/fs
                PSvalidated['stop'][nDetectionCounter] = Detections['stop'][n]/fs
                PSvalidated['peak'][nDetectionCounter] = Detections['peak'][n]/fs
                PSvalidated['peakHFOFrequency'][nDetectionCounter] = maxF
                PSvalidated['troughFrequency'][nDetectionCounter] = minF

                if (np.size(fpeaks) != 0):
                    PSvalidated['peakLowFrequency'][nDetectionCounter] = fpeaks[-1]
                else:
                    PSvalidated['peakLowFrequency'][nDetectionCounter] = 0

                PSvalidated['peakAmplitude'][nDetectionCounter] = Detections['peakAmplitude'][n]


        if (nDetectionCounter < 0):
            PSvalidated['channel_name'][0] = channel_name
            PSvalidated['start'][0] = -1
            PSvalidated['stop'][0] = -1
            PSvalidated['peak'][0] = -1
            PSvalidated['peakHFOFrequency'][0] = 0
            PSvalidated['troughFrequency'][0] = 0
            PSvalidated['peakLowFrequency'][0] = 0
            PSvalidated['peakAmplitude'][0] = 0

    return PSvalidated;
```

```
#function used to test out channel aggregating idea to speed up processing time
def channel_mean(dataframe):
    df_mean = pd.DataFrame(columns = ['D','C','A','B','E','F'])

    D = ['D1','D2','D3','D4','D5','D6','D7','D8','D9','D10','D11','D12','D13','D14','D15','D16','D17','D18
','D19','D20','D21','D22','D23','D24','D25','D26','D27','D28',
     'D29','D30','D31','D32']
    C = ['C1','C2','C3','C4','C5','C6','C7','C8','C9','C10','C11','C12','C13','C14','C15','C16','C17','C18
','C19','C20','C21','C22','C23','C24','C25','C26','C27','C28',
     'C29','C30','C31','C32']
    A = ['A1','A2','A3','A4']
    B = ['B1','B2','B3','B4']
    E = ['E1','E2','E3','E4','E5','E6']
    F = ['F1','F2','F3','F4','F5','F6','F7','F8']

    df_mean['D']=dataframe[D].mean(axis=1)
    df_mean['C']=dataframe[C].mean(axis=1)
    df_mean['A']=dataframe[A].mean(axis=1)
    df_mean['B']=dataframe[B].mean(axis=1)
    df_mean['E']=dataframe[E].mean(axis=1)
    df_mean['F']=dataframe[F].mean(axis=1)

    return df_mean;
```

9 References

(2020). Retrieved from Johns Hopkins Medicine: https://www.hopkinsmedicine.org/health/conditions-and-diseases/epilepsy/epilepsy-causes

(2020). Retrieved from Open Notes: https://www.opennotes.org/about/

(2020). Retrieved from DASK: https://dask.org/

Brunos, S., Hilfiker , P., Sürücü, O., Scholkmann, F., Krayenbühl, N., & et, a. (2014). Human Intracranial High Frequency Oscillations (HFOs) Detected by Automatic Time-Frequency Analysis. *PLoS ONE*. doi:doi:10.1371/journal.pone.0094381

Buzsaki, G., Horvath, Z., Urioste, R., & et, a. (1992). High-Frequency Network Oscillation in the Hippocampus. *Science, 256*(5059), 1025-1027.

Cardenas, R., & et, a. (2017). Mendel,MD: A user-friendly open-source web tool for analyzing WES and WGS in the diagnosis of patients with Mendelian disorders. *PLOS Computational Biology*.

Cimbàlnìk, J., Hewitt, A., Worrell, G., & Stead, M. (2018). The CS algorithm: A novel method for high frequency oscillation detection in EEG. *J Neurosci Methods.*, 6-16.

Clay, R. P. (2011, October 11). *An Open Source Approach to Medical Research*. Retrieved from Stanford Social Innovation Review: https://ssir.org/articles/entry/interview_an_open_source_approach_to_medical_research

Elahian, B., Yeasin, M., Mudigoudar, B., Wheless, J., & Babajani-Feremi, A. (2017). Identifying seizure onset zone from electrocorticographic recordings: A machine learning approach based on phase locking value. *Seizure, 51*, 35-42. Retrieved from https://doi.org/10.1016/j.seizure.2017.07.010.

Fisher, R. S. (2014, April 15). *A Revised Definition of Epilepsy*. Retrieved from www.epilepsy.com: https://www.epilepsy.com/article/2014/4/revised-definition-epilepsy

Islam, R. (2015). Human Intracranial High Frequency Oscillation Detection Using Time Frequency Analysis and Its Relation to the. Masters Theses.

Kneller, A. (2016, January 18). *How to address the culture gap between academia and industry in biomedicine*. Retrieved from Novartis: https://www.novartis.com/stories/from-our-labs/how-address-culture-gap-between-academia-and-industry-biomedicine

Malladi, R., Kalamangalam, G., Tandon, N., & Aazhang, B. (2016). Identifying Seizure Onset Zone from the Causal Connectivity Inferred Using Directed Information. *IEEE*.

Navarrete, M., Alvarado-Rojas, C., Quyen, M., & Valderrama, M. (2016). RIPPLELAB: A Comprehensive Application for the Detection, Analysis and Classification of High Frequency Oscillations in Electroencephalographic Signals. *PLOS One*, 27.

Noachtar, S., & Rèmi, J. (2009). The role of EEG in epilepsy: A critical review. *Epilepsy & Behavior*, 22-33.

Price, L. (2018, August 1). *The Digital Health Hype Cycle 2018*. Retrieved from healthcare.digital: https://www.healthcare.digital/single-post/2018/02/20/The-Digital-Health-Hype-Cycle-2018

Scheuer, M., & WIlson, S. (2014). Data analysis for continuous EEG monitoring in the ICU: seeing the forest and the trees. *Journal of Clinical Neurophysiology*, 353-78.

Sirven, J. I., & Shafer O. Patricia, R. M. (2014, January 21). *What is Epilepsy?* Retrieved from www.epilepsy.com: https://www.epilepsy.com/learn/about-epilepsy-basics/what-epilepsy

Siwicki, B. (2018, July 23). *How one medical group uses AI, machine learning to improve value-based care.* Retrieved from Health IT News: https://www.healthcareitnews.com/news/how-one-medical-group-uses-ai-machine-learning-improve-value-based-care

The Economist. (2018, February 1). *A revolution in health care is coming*. Retrieved from The Economist: https://www.economist.com/leaders/2018/02/01/a-revolution-in-health-care-is-coming

Treatments for Epilepsy. (2019). Retrieved from www.spectrumhealth.org: https://www.spectrumhealth.org/patient-care/neurosciences/epilepsy-and-seizures/epilepsy/epilepsy-treatment