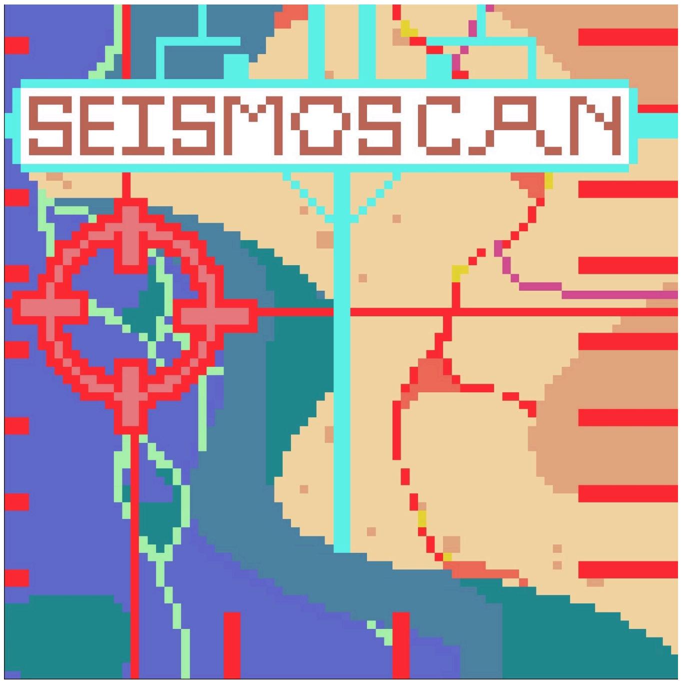
Tech Feasibility

SeismoScan

Sponsors: Dr. Donna Shillington, Dr. James Gaherty, Dr. Christine Regalla



Mentor: Scott LaRocca

Alyssa Sombrero

Brady Wisniewski

Noah Carges

Thomas Rotchford

10.17.2025

2. TABLE OF CONTENTS

3. INTRODUCTION

Machine Learning has been proven to be more efficient in areas of study like artificial intelligence, advertising, games, and research . But, what if we could push the concept of machine learning into fields involving geology and seismology; somewhere where it would be extremely beneficial by professionals. We, as SeismoScan, present our project of using Machine Learning tools to map Faults through Bathymetry Data.

Today geologists use tools such as GeoMapApp and ArcGIS, to hand interpret faults one at a time which is a time-consuming process. Our clients Dr. Donna Shillington, Dr. James Gaherty, and Dr. Christine Regalla worked with a graduate student researcher who spent 80 hours in two weeks of work to map an area around 40,000 square kilometers large by hand. As more bathymetry data is collected and available, there is an opportunity to use some level of automation to make analysis more efficient and facilitate more science.

In order to design our product, we are researching various machine learning methods in order to find one appropriate for analyzing digital map data. We are also working closely with our clients to make sure we understand the ins and outs of analyzing faults by hand. Our goal is to create a tool which takes in bathymetry data and runs it through a trained machine learning model in order to identify faults in subduction zones across the globe.

By creating this tool, we can allow geological researchers to spend much less time identifying and mapping faults, and instead focus more of their time using those identified faults to get a deeper understanding of geological processes. We are planning to initially develop this as a small-scale model, identifying faults in a small region, providing users with a file including latitude and longitude data (along with other information we feel might be helpful for the user) for each given fault. Once this is complete, we hope to scale this to cover a much larger area of bathymetry data, identifying faults at a much larger range.

Throughout this Technological Feasibility analysis, we will be analyzing the ways to use NetCDF data, different approaches to a machine learning model, different existing packages we can use to support our machine learning model as well as clean data, and then look at different languages that could support all these features. To conclude we will prove the feasibility of the chosen technologies to ensure each piece fits as part of the whole puzzle.

4. TECHNOLOGICAL CHALLENGES

A project of this nature brings with it an interesting set of challenges. The set of challenges will face are not trivial. Along with that within each challenge there are a vast number of ways to tackle them, each of which needs to be considered meticulously.

4.1 Data Format:

The data will be imported utilizing a NetCDF data format per our client's instruction. We must decide if we are going to use NetCDF to process the data or convert to another data type. In order to make this decision we will look into the native libraries of a variety of datatypes as well as how compatible they are with other technological challenges we face, like the programming language and machine learning in general.

4.2 Machine Learning Model:

The most significant technological challenge in this project is developing and training a machine learning model capable of accurately identifying faults in bathymetry data. Because this data is large, noisy, and highly variable across regions, the model must recognize subtle spatial patterns while avoiding false detections. Another challenge lies in determining which type of model (or combination of models) best suits this task. We will need to evaluate several approaches, including CNNs, clustering algorithms, and classification models, to balance accuracy, interpretability, and efficiency.

4.3 Programming Languages:

When considering the languages we will use, there are a few solid options for languages: C, Python, and Julia. This choice will depend upon how well each package is made for each of our chosen languages. Based on how easily the language is to implement, we will go forward with the option that most team members are comfortable with.

4.4 Packages/Libraries:

Looking into the packages and libraries for our machine learning model, we have access to multiple different options of which we can compare. Looking specifically into pattern recognition, we will consider the strengths and limitations of scikit-learn, OpenCV, and PyTorch. For data analysis, we will turn to Generic Mapping Tools (GMT), Numpy, and Pandas. All of these are open source and, as a result, have their source code accessible to us in the event that we need to delve deeper into any particular choice.

4.5 Model Training Approach:

The most important decision we make will be the creation and training of the requested machine learning model. Depending on how we handle the input data, our approach may change. If the data is formattable in a way in which the machine learning model is able to treat the modeling of inputs as a pattern recognition problem, we will be able to leverage a wide range of already existing models. However, if we use the raw data, there is a higher chance we will have to build and train our own model in order to enable it to both understand and analyze the data in the particular way necessary.

5. TECHNOLOGICAL ANALYSIS

Throughout this analysis section we will compare and contrast a large number of choices as it pertains to the five Challenges that we have identified. To start we will look at data formatting, and the different types available to us. Then we will go into machine learning algorithms to see which will best fit the challenge we face. After that, we will look at the programming languages that are available to support these tools, and after that the libraries we can use to help with data preprocessing and with our machine learning methods within that language. Lastly we will look at specific models themselves to see if we may use a pretrained model or build our own.

5.1 Data Format Analysis

Data formats are many and varied, and choosing the right one is important not only for managing data but also for ensuring that the preprocessing and training stages of the machine learning process can be done efficiently.

5.1.1 Desired Characteristics

Our ideal data type must be **scalable**, **reliable**, and **compatible** with a wide range of scientific computation tools, as well as machine learning techniques. It must be **scalable** since our project will deal with large geospatial data sets, this means it will need to handle large multidimensional datasets efficiently since a given data unit may contain tens of pieces of data inside of it which means a multidimensional array of these points could contain millions of variables . Because of this, It would be very useful to be able to access a portion of a large data set without having to load all of it. Also certain metadata aspects may be more useful than others.

In this context **reliability** means a few different things. Some of the data formats we will look at in the next section have been updated for over 40 years and bake backwards compatibility into the design of their format. This is important to us since we see our project having a long lifespan which stretches for many years after we are done with this class. Another factor that goes into the creation of reliable data is how the format's adjacent library supports the ability to check for errors in the data. These errors could come from a myriad of places but the most likely is from one of the programmers. If the format library can check for data mismatch before using the data, that would be very helpful.

Compatibility may be by far the most important aspect. This is simply a measure of how well a given data format interacts with surrounding technologies. For example, If we chose to program in Julia we would want to make sure that the format's native library would allow the data to be interacted with in Julia. However there may be a case where more important technologies do not support the format for one reason or another; In the scenario we must weigh the importance of the technologies with the time it would take to add our own support. Thus, the compatibility of the format and its native library must be carefully considered as the wrong choice could require time and effort we do not have.

With respect to machine learning the aforementioned concepts of **scalability** and **reliability** apply in similar ways. Since a machine learning technique may read and write large files many times the data format must be error-resistant and have a native library which can feed the data to the machine learning algorithm in small parts. Since we are looking at scientific data, we will likely be using an array-based format since they are more efficient for large complicated data sets. The scalability is a bit more complicated. While the amount of data we are working with is large it is still able to fit on a local machine. One area that is 40,000 square kilometers takes up around 20 gigabytes of storage space, and this includes low resolution data which we cannot use because of its lack of precision. We estimate there are about 2 million square kilometers of fault zones on earth, and thus about 2.5 terabytes (or 2500 gigabytes) worth of data by the previous measurements. However, the majority of this data has not been collected and even if it had been it would not be difficult to store 2.5 terabytes of data on one machine, keeping in mind that this number is likely an overestimate. This leads to the conclusion that large scale scalability is likely not a priority for the data type given the relatively small amount of raw data we even have to use for training, the fact that the majority of the data still needs to be collected, and the fact that the maximum amount of data could be no more than a few terabytes. As far as **compatibility** goes the main concern here will be with the programming language since that will determine certain properties that will affect processing speeds and which packages can be used with it.

5.1.2 Alternatives

There are a few different data formats for geospatial data all with different strengths and weaknesses. The ones we will look at here are NetCDF, HDF5, and GeoTiff.

NetCDF or Network Common Data Form was developed by Unidata in 1989 and has since become a staple of formatting for natural sciences based pursuits like oceanography since it was built specifically to handle climate and geospatial data. It is a self describing, array oriented format. It is backwards compatible, and has some very powerful native libraries due to the community support which makes it very cohesive with relevant tools like GMT, xarray, and NumPy. The most updated version takes the hierarchical data format from HDF5. Historically it has been used by larges labs like NOAA and NASA for storing climate and geospatial data.

HDF5 aka “Hierarchical Data Format, version 5” was developed by the HDF group in the late 90s. Its most prominent feature is the way it uses a hierarchical structure to organize data. This makes it like a filesystem condensed into a file. This makes it remarkably good at handling large data sets. It has support across many languages, however it lacks conventions which specifically relate to geospatial data. The widespread support of this file type means it is also broadly cohesive. This format is used wherever there are large detailed amounts of data, from online forums to labs.

GeoTIFF is a Tagged Image File Format that includes geospatial metadata. It was developed in the 90s by the OGC and it is by far the most specialized file here. It represents a 2d grid in a raster image file. It is used by mapping professionals like civil engineers and labs for storing elevation data.

5.1.3 Analysis

Now we will consider the specific strengths and weaknesses of each file format with regard to their scalability, reliability, and widespread cohesion.

GeoTIFF’s visualization skills come at the cost of containing much more data than the other types of files since it stores data points as pixels. GeoTIFF also does not support lazy loading which allows data to be taken from files as needed instead of loading the whole thing at once. This means that the data would have to be broken down into smaller pieces with our own code and that would be an unnecessary use of our time. With regard to reliability GeoTIFF data is at risk for corruption at greater file sizes, however this is mitigated by its native library(GDAL) which actively checks for corruption during writes. GeoTIFF is also the least broadly compatible file type here since it is most commonly used with mapping software. It is compatible with python but most data analysis libraries don't support it and the data format needs to be altered for it to be analyzed. It is also not commonly used with machine learning since there are better data formats for image related learning.

HDF5 is extremely scalable with chunking and lazy loading allowing for lower memory consumption. It is also extremely reliable with many features to support data integrity such as atomic writes which prevent corruption during a write. Since the native libraries support chunking the size of a write can be greatly reduced which lowers the risk of corruption further. However it must be noted that at large data manipulations there is still a risk of corruption, but this would not be an issue if managed properly. HDF5 is broadly supported across multiple programming languages and tools like GMT. A potential downside is the dependence on old c code which is written without consideration for parallel computing. Despite this, HDF5 is commonly used for machine learning applications where data is accessed from a local machine and offline. Since HDF5 is so old, It is very widely supported. It is also typical to turn HDT5 into a custom dataset class for it to be read, native libraries can help with this.

Finally, now we will look at NetCDF which, as previously mentioned, draws on HDF5's approach to formatting and thus has all of the same benefits of reliability and scalability that HDF5 has. In addition to this NetCDF is very broadly supported and can be ready directly by libraries like xarray and tools like GMT. NetCDF also is CF-compliant which means it has standardized metadata with respect to many things like spatial and temporal data. NetCDF is not as common for machine learning applications but that is simply because there are less machine learning projects using this type of information. Despite this the native library can assist in the creation of a wrapper class for use with machine learning packages

5.1.4 Chosen Approach

By far our best choice is NetCDF. It has all the reliability and scalability of HDF5 with broader support within the scientific field. GeoTIFF is simply not viable with its very poor scaling and lack of modern data loading features.
In the graph below each of these three file types they will each be assessed on a scale of 1-5 for their scalability, reliability, and widespread cohesion.

1. Scalability
2. Reliability
3. Compatibility

	GeoTIFF	HDF5	netCDF
1	★★ Poor scalability	★★★★★ Scales very large	★★★★★ Based on HDF5
2	★★★ Rudimentary measures for data protection	★★★★★ Has robust measures to protect data integrity	★★★★★ Based on HDT5
3	★★ Cohesive with mostly specialized libraries and tools	★★★ Cohesive a good amount of general libraries and tools	★★★★★ Cohesive with a number of useful libraries and tools

5.1.5 Proving Feasibility

In order to prove the feasibility of these choices, we will perform some basic calculations using NetCDF's native python library and additional tools and libraries. We will load the data and leverage the chunking capabilities of NetCDF's native library, then preprocess the data so that only that which is relevant will be passed on to the machine(high rez data) learning process, and we will wrap the data into a custom class. If all of this goes well with this we will consider the feasibility proven. In the meantime we will look for memory leaks, data corruption, and consider the speed of processing.

5.2 Machine Learning Model Analysis

The machine learning model is the core of our project, as it is responsible for identifying and mapping faults within the bathymetry data. Choosing the correct model will determine the accuracy and efficiency of information analysis. Because this is the most technically demanding component, we must consider several different approaches to determine which will best fit the needs of the project.

5.2.1 Desired Characteristics

The desired characteristics of the Machine Learning model revolve around the ability to identify and classify faults from large-scale bathymetry data. The model must also be able to recognize patterns in the terrain and detect linear fault structures amongst large amounts of noise. Since bathymetric data can vary drastically from region to region, the model must also be generalizable so it can perform well outside of its training data.

Interpretability is another very important characteristic for our model. We want to be able to understand *why* the model identifies a given fault. In addition to this, the model will need to be compatible with the tools and libraries that we plan to use, such as Scikit-learn and PyGMT, while still being feasible to train with the resources available.

5.2.2 Alternatives

Several different types of models could help us achieve our goal. Each model has its own unique strengths and weaknesses depending on the type and amount of data available to us. The model must be able to recognize linear/edge-like features in a large, two-dimensional dataset:

1. **Convolutional Neural Networks (CNNs)**
    a. These are models that work best with spatial and image-based data. They can recognize edges and linear patterns in two-dimensional data, without much manual work, making them a strong candidate for detecting fault lines in bathymetry data.
    b. These models require large, labeled datasets in order to train effectively, which could be intensive and may pose some challenge in earlier testing.
2. **Support Vector Machines (SVMs)**
    a. These models are efficient and accurate for smaller datasets/clear when boundary conditions are well-defined (faults vs non-faults). They are able to accurately perform with limited training data and are supposedly efficient to train.
    b. These models, however, do not naturally handle spatial data, so it is possible for it to miss the context of continuous fault structures.
3. **Random Forests**
    a. This is an ensemble model that uses decision trees in order to make predictions. These models are good at handling noisy data and can look at both numerical and categorical features. In our project, we could use this model to take its slope,

change in elevation, or information from other models in order to get a second look at whether or not a given region is actually a part of a fault.
    b. These models lack built-in spatial awareness, meaning that they would have to look at each point independently.

4. **Clustering Algorithms**
    a. K-Means and DBSCAN are common clustering algorithms that can work well in cases where labeled data is limited or completely unavailable. These models are able to identify patterns based on similarity alone, meaning that they can be used to group certain areas with similar slopes or elevation, potentially highlighting fault zones without needing training.
    b. These models still have the risk of producing inconsistent results if data has a lot of noise of variation.
5. **Hybrid/Pipelined Models**
    a. This method is probably the most effective route, combining other models to yield better results. For instance, combining a CNN model for identifying fault regions based on spatial patterns, using a clustering algorithm in order to group nearby detections into continuous lines, and then a random forest or SVM which can then classify those segments as faults or non-faults based on additional information.
    b. This pipelined method would allow us to utilize each model's strengths and result in a higher level of accuracy and interpretability.

5.2.3 Analysis

When we look at analyzing our different approaches, it becomes clear that each model has its own strengths and weaknesses. Our challenge is to balance accuracy, efficiency, and interpretability while still ensuring that the approach can handle a large geospatial dataset.

Looking at the **Convolutional Neural Networks** (CNNS), we can see that this type of model shows the most potential for our goal since it excels at spatial pattern recognition. Fault lines often appear as linear discontinuities or ridges, which are patterns that a CNN is designed to detect. CNNS can also automatically learn complex spatial patterns in the data without any manual interference or tuning. However, labeled data is required for training a CNN model which could be time-consuming to generate and it also demands a greater computing power meaning that we would have to start small and scale up as development progresses (which is already something that we had planned to take place).

Support Vector Machines, or SVMs, along with **Random Forests** Models allow for a more accessible starting point for our project. Both of these models work well with limited data and are much lighter in terms of computational power. Random Forests are useful for integrating additional features, such as the slope, curvature, roughness, and more which can provide us with a clear and easily-interpreted decision structure. However, neither of these methods have an inherent understanding of spatial context, so they are a bit less effective by themselves when we are dealing with a continuous, line-based geological pattern.

Moving onto the **clustering** algorithms, such as K-Means or DBSCAN, we could use this strategy in order to easily group together regions that share similar traits. This could help highlight continuous fault zones or reduce noise from CNN outputs. The only downside to the clustering algorithm is that it still cannot *fully* distinguish between true faults and natural variations in the seafloor, meaning that it would need a 'sister-model' to help with distinguishing.

A **hybrid pipelined approach** offers the most robust and flexible solution given the analysis of the other models. Using a CNN as a feature detector, responsible for identifying potential fault structures would be a good start. Combining the CNN with a clustering algorithm would allow us to refine the detections into more coherent segments. Layering this with a Random Forest or SVM model would allow us another level of confirmation of possible faults, reducing the possibility of false positives/negatives. This layered approach would maximize the strength of each type of model while continuing to maintain a level of interpretation and control that suits our goals for the project.

5.2.4 Chosen Approach

As mentioned above, we are choosing to implement a hybrid pipelined approach centered around a Convolutional Neural Network as the primary fault detection tool. Using this approach would enable us to integrate multiple machine learning models and techniques in a sequential manner with each stage contributing to a specific part of the fault identification process.

Beginning with a CNN trained on labeled bathymetry data to detect spatial discontinuities and edge-like patterns suggesting faults, the CNN would output a map highlighting regions where faults are most likely to be located. Using a clustering algorithm next, the model would group neighboring detected faults into a continuous fault segment, reducing the noise and identifying clear fault structures. This information would then be passed to the Random Forest model, which would then evaluate the additional features of the faults, such as the slope, to determine which segments are most likely geological faults.

Using this hybrid approach will leverage the strength of the CNN model by providing highly accurate spatial feature detection, clustering algorithms to ensure continuity/noise reduction, and the Random Forests to ensure interpretability and another level of accuracy. Pipelining our project with different models will also give our team more flexibility, allowing us to test and improve each component individually before integrating it into a single automated system. This method also ensures that the final product is scalable, transparent, and compatible with tools such as PyTorch and Scikit-learn.

1. Scalability
2. Reliability
3. Cohesion

	CNN	SVM	Random Forest	Clustering	Hybrid
1	⭐⭐⭐ (High computational demand, scales gradually)	⭐⭐⭐⭐ (Lightweight, scales decently)	⭐⭐⭐⭐ (Handles large data fairly well)	⭐⭐⭐ (Scales moderately with dataset size)	⭐⭐⭐⭐⭐ (Very scalable through a modular design)
2	⭐⭐⭐ (Accurate with enough data, depends on training quality)	⭐⭐⭐ (Stable, but limited spatial context)	⭐⭐⭐⭐ (Reliable and interpretable)	⭐⭐ (Dependent on parameter tuning)	⭐⭐⭐⭐⭐ (Combines strengths for reliable outputs)
3	⭐⭐⭐ (Integrate well with deep learning frameworks)	⭐⭐⭐ (Works well with structured data, less spatially)	⭐⭐⭐⭐ (Works well with many libraries and tools)	⭐⭐⭐ (Not much standalone cohesion)	⭐⭐⭐⭐⭐ (Multi-model, flexible, and interpretable)

(This is an early idea of the model used for the project and may end up changing later in the development process)

5.2.5 Proving Feasibility

To prove the feasibility of our proposed hybrid approach, we will design and test a series of experiments that evaluate the performance of multiple candidate models on a representative subset of bathymetry data. The goal is to confirm that our chosen pipeline (CNN → Clustering → Random Forest) performs best in terms of accuracy, interpretability, and efficiency.

We will begin by building independent prototypes for each of the model types discussed earlier. Each model will be tested on a small 10km x 10km section of bathymetric data to evaluate how well it identifies fault-like linear discontinuities. Each model's predictions will be tested and compared against a small set of manually labeled data provided by our sponsors. Quantitative performance metrics, such as precision and accuracy, will be recorded. On the other hand, qualitative visual checks will be used to assess interpretability.

Once baseline results are collected, we will perform an analysis on the models and see which ones yield the best trade-offs between accuracy and computational cost. After testing each component individually, we will integrate the best-performing elements into a final hybrid pipelined model, re-evaluate its performance, and assess the scalability to a large dataset. This staged testing strategy will not only verify feasibility, but also demonstrate the benefits of using a hybrid model compared to single-model approaches, confirming that our hybrid approach is our most effective solution.

5.3 Programming Language Analysis

We understand that this type of programming language is very crucial for communication between the user and the application at hand, the machine learning mode. We need to establish which programming language fits the best to not only serve as communication, but to also tell the machine learning algorithm on how to "observe", "learn", and manipulate the inputted NetCDF files. The possible challenges that may be encountered will be limitations on packages to help alleviate programming challenges, lower flexibility and abilities with the data and its data type it is provided with. In the final product, we hope to see the machine model successfully understand the given NetCDF file type and its contents to be analyzed and converted into a readable file for the sponsors to analyze.

5.3.1 Desired Characteristics

With the possible challenges in mind, we want to strive for programming languages that have a lot more potential and flexibility in manipulating the data type we may be working with via NetCDF data type through the various packages and libraries that is needed have the machine learning model to understand the data its working with through mathematical functions to calculate slopes and valleys. It needs to know this so it can classify whether the potential locations contain a similar pattern of other faults it learned. Since this project is fairly new, we are aware it may be passed down to another group, whether that is programming professionals or computer science students, it is mandatory for the language to be high-level, where it's more human readable. Additionally it will need a language widely known on the internet, so if there's any bugs or errors that occur in the program, it's lenient on other users to understand rather than to figure it out on their own. Lastly and foremost, we need a programming language that is very compatible with machine learning attributes. That is with handling complex math computations,

5.3.2 Alternatives

We have selected three possible programming languages that may be utilized for the project presented.

Python is a widely used high-level programming language for machine learning and artificial intelligence and it has been implemented in most discord bots. It was created in December 1989 by Guido Van Rossum. Python is a dynamically typed language, meaning variable types are determined on runtime. Therefore allows users to write code more quickly rather than explicitly declaring which variable is what data type. Plus it has simple syntaxes to make the code more clear and understandable to the users. We have been recommended by our sponsors to use it, since it does have access to libraries and community libraries relating to any functions that'll help develop netCDF file operations. Not only that, but python is also used for database related handlings like Structure Query Language (SQL).

C is a foundational programming language developed in the 1970s by Dennis Ritchie. It's a very popular programming language used to understand the computer system, since it is a middle-level programming language; hence it is able to directly communicate to the computer's kernels and operating system. It was used when developing the UNIX operating system. Unlike with Python, it is a weakly typed programming language, so the user must manually enter the type of variable to be used. Though, it does require some background knowledge on computer memory management.C is very flexible in transitioning between hardware and software, like General Purpose Input / Output (GPIO) and games like Touhou Project that's been built off C code. This programming language was something we were all too familiar with. This language will serve as our back up incase any programming language doesn’t go up to our expectations

Julia is a dynamic, high-level programming language that is similar to Python. In fact it was supposed to be in competition with Python, aiming at the elements Python doesn’t necessarily have. Julia has many built in math operations to aid in machine learning elements, simulations, reinforcement learning, and much more. Julia is one of a more recent languages created in 2012, with the creators of Jeff Bezanson, Alan, Edelman, Stefan Karpinski, and Viral B, Shah. Overall, it was designed to have a higher performance than Python. Overall it has a big emphasis on statistical computation and data analysis, which may be very helpful in contributing towards our project.

5.3.3 Analysis

The optimal programming language we can use for the machine learning project is Python; it follows the desired characteristics of being able to maintain, debug, and read due to its human-friendly syntax. Due to being a high-level language, Python allows any user to understand and modify the code, which is extremely beneficial if the project is passed down to another professional group for further improvements. It also offers a strong machine learning workflow compatibility and is able to handle large sums of datasets in an efficient manner; ensuring the system can process data quickly. However, the most notable above all, is the fact it can handle NetCDF file and its data type through the use of PyGMT and other available NetCDF tools. Through the libraries, we are able to do manipulation, visualization, and data analysis, thus making the development more faster and efficient. Overall, python stands out as a reliable, efficient, and future-proof language in a machine learning environment.

The C programming language serves as a basic building block to understand computer memory and architecture concepts. However, it requires a deeper understanding of system operations since it is a middle-level language; sitting above Assembly Language. Whilst C can be used for machine learning applications, it needs a more complex and technical approach that demands extensive programming knowledge and manual memory management. Something we do concern over, if the project will be handed down to another batch of students who are unaware about Programming. Though, a NetCDF library is offered within C, but comparing it to Python's NetCDF tools. It doesn't support visualization, and needs more effort to implement any complex analysis and operations. Although it may have some drawbacks, it may serve as an alternation or an addition to the main program.

The last option is Julia. Julia is a programming language that's high-level like Python, meaning it's human-readable when it comes to creating the programs. It does contain a netCDF, and it seems to offer more data manipulation and analysis compared to that of C. Plus it is widely used in machine learning aspects since it does offer a mathematical environment for simulation, learning, and more (#needs more thinking on)

5.3.4 Chosen Approach

With the available programming options, each language presents its own strengths and weaknesses. However Python stands out to us for its extensive community support, abundant libraries, and human-readable syntax, making it ideal for data-oriented projects and user-friendly applications; in which it very much applies to the main components of our machine learning project. In contrast, C offers speed and low-level control but lacks the rich library environment needed for handling NetCDF files efficiently. In terms of human readability, the user must have extensive knowledge on computer memory management in order to understand and apply changes to the code. While Julia offers perfect speed and strong mathematical capabilities, it has a smaller user community and fewer resources for troubleshooting and learning. Therefore Python's libraries (PyGMT and the NetCDF4 library) provide powerful tools that directly support our main project's components.. Henceforth, our decision is to use Python as our primary programming language. Though, this choice does not exclude the possibility of integrating other languages like C or Julia in the future if performance optimization or specific functionality is needed.

Comparison Table:

1.) Machine Learning compatibility
2.) Human Readable
3.) Packages and Libraries Capabilities for NetCDF

	Python	C	Julia
1	Python is widely used for Artificial Intelligence and machine Learning through its package and libraries. Plus, it	C provides a good level of control over Machine Learning code. Though there's a few tutorials on using C for Machine learning	Julia is a great environment, commonly used in Machine Learning aspects like Deep Q Learning and its

	has a larger user base for help, forums, ⭐⭐⭐⭐⭐	⭐⭐⭐	calculations behind it. ⭐⭐⭐⭐
2	Python has human readable syntax that allows maintenance and debugging to be easily managed. ⭐⭐⭐⭐⭐	C code is a middle-language, so it would need more background knowledge on how to navigate in a C environment. ⭐⭐	Julia is a high-level language, so it's as high-level as python. It's able to have a mathematical environment, great for ⭐⭐⭐⭐
3	Python contains the libraries and packages for NetCDF: PyGMT, netCDF library ⭐⭐⭐⭐⭐	C does contain a library through netCDF-C, netCDF Fortran, ⭐⭐⭐⭐	Julia does contain a package for netCDF: juliaGeo. ⭐⭐⭐⭐

5.3.5 Proving Feasibility

Due to the excellence in machine learning capabilities, flexibility with data manipulation, and human readability, we decided to go forwards with the programming language, Python. Python has been proven to be widely used for machine learning and artificial intelligence through the packages and libraries it offers, as well as the countless forums relating to machine learning, in case we come across any problems within the machine learning application. However, if any problem arises with Python, we will resort to C since it does offer similar computational aspects of Python, but it will be much of a challenge when it needs to deal with NetCDF files.

5.4 Package and Library Analysis

The packages and libraries that we decide to use will greatly affect our project. Our choices will impact the systematic work we do with the bathymetry data to prepare for analysis with our machine learning model. It will also define how we connect with our machine learning model and what actions we will have at our disposal to tune the model.

5.4.1 Desired Characteristics

For the data work the most important characteristic would be the ability to work with our NetCDF data type, or with a way the data could otherwise be formatted. Another necessary feature is access to a large library of mathematical functions, as an example, the ability to take the derivative of the data will be vital, because the slope of the seafloor is a great identifier of faults per the client. Another aspect is efficiency as we are dealing with massive data sets at a time, so it will be vital that we can comb through it quickly, to not only allow for a better user experience but also improve our ability to create a functioning product.

As for the connection with a machine learning model it will be important to find one that coincides well with our chosen approach of Convolutional Neural Networks, in order to increase cohesion between all parts of the project. A beginner friendly option would be best suited as none of the group members have much experience in the machine learning space. One of the most important characteristics is having a large chunk of features, as the ability to smoothly pivot, compare and contrast different strategies and how they perform will be important to deliver the best possible software to the client. As a reminder the strategies we plan to use are a Convolutional Neural Network, clustering, and a Random Forest model.

5.4.2 Alternatives

To start, for data processing, cleaning, and preparation our options are PyGMT, NumpPy, and Pandas. Along with that our machine learning options include OpenCV, PyTorch, and Scikit-learn, which all provide a different avenue of attack. Now to run through them all:

PyGMT was recommended to us by the client as something they have worked with on a smaller scale in the past. This library started development in 2017 by Leonardo Uieda and Paul Wessel, and is a wrapper for the Generic Mapping Tools which is a command line program which is used very widely in the earth sciences.

NumPy was found through online research as a multi-dimensional array specialist. It was built by Travis Oliphant in 2005 with the goal of combining Numarray into Numeric which NumpPy was built off of. It is commonly used in Machine Learning, as well as in data science, and linear algebra due to its specialization with matrices.

Pandas was started in 2008 by Wes McKinney with the goal of giving quantitative analysis on data sets. It was also found in general online research while looking for potential tools to use. Pandas was initially used with financial data sets, but has grown for general purpose with large structured data sets. It is commonly used to clean, manipulate, and analyze data.

Scikit-learn was designed to provide a wide range of machine learning algorithms in python. Originally released by David Cournapeau in 2010 and then taken over by Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, and Vincent Michel. We found this doing research on different machine learning algorithms and offers the ability to use outside models as well as develop machine learning models itself. It takes advantage of algorithms such as clustering, regression, classification, and dimensional reduction.

PyTorch is commonly used by large brands such as Meta, Tesla, OpenAI, and NVIDIA to help with their research. Its most common use cases are deep learning research, computer vision, and natural language processing. Originally it was developed by Meta AI and the initial release was in 2016.

OpenCV is specifically designed for Computer Vision applications and was originally developed by Intel released in 2000. It has common uses for facial recognition, video analysis, and 3d reconstruction. OpenCV has the ability for object detection, and training machine learning models on clustering and classification.

5.4.3 Analysis

In order to Analyze these options we will be looking into all of our desired characteristics as it pertains to each library and package. Some things that will be kept in mind through each section are the extent of its mathematical capabilities, if it has any cohesion with the NetCDF data type. Also we will look into how it handles larger data sets and the data structures it has available to do so, along with the data cleaning tools as this is a vital part of the preprocessing cycle.

Starting with PyGMT which was built specifically to use geospatial data, which is exactly what we will be using. When looking at its mathematical capabilities it has the ability to find common information such as mean, median, mode, and distance. It also provides more advanced information such as derivatives and triangulation. It can also output this data as gradients and histograms. PyGMT also has direct loading capabilities with a .nc file, with the pygmt.load_dataarray function which will return a dataArray ready for use with no other work required. However, as we work with larger areas of land it may struggle with a dataset over RAM due to wrapping around the GMTc package. Though it is able to use multiple cores when processing the data. PyGMT has a dependency on both NumPy and Pandas which it relies on heavily in for the data structures they provide, and does not provide any of its own data cleaning attributes but can invoke those of NumPy and PyGMT. Lastly, and most importantly, our clients have used PyGMT in the past when working on smaller projects or just to get use of its mathematical functions and have recommended it to us as an option to gain access to important tools in identifying faults such as taking the derivative of the sea floor, and creating visualizations of the data.

Next up is NumPy which leaves a lot to be desired when it comes to the mathematical functionality. It does provide access to standard arithmetic functions on the data sets but does not have the ability to do any calculus or other higher level operations on a set of data. It also does not provide any immediate connection between NumPy and NetCDF data. Meaning it does need outside tools to correctly input data and additional setup with the metadata from the NetCDF file is required to input the data correctly. Where NumPy shines is in its data management, using the memmap class it can put data into storage directly and then call that data back into memory when it needs it later allowing it to work with much larger datasets. It also provides a specialized array data structure called the ndarray which can create an array of n dimensions large, however to make its storage capabilities it has a fixed size at creation and each entry must be of the same datatype. When it comes to cleaning the data NumPy has the ability to ignore missing data when operating on the data set and fill missing data but is unable to remove it from its structures.

Last up is Panas which similar to NumPy does not provide any higher level mathematical functions, but it does have access to some statistical analysis based operations including options like the standard deviation of a data set or generating quartiles of that data. Since it is built off NumPy and is a more statistics based library it does not have a methods to correctly import NetCDF data, and in the process using outside tools Pandas will also lose some of the vital metadata that comes with the NetCDF data. A major shortcoming exists in the realm of performance as well as it has the same issue as GMT where it cannot exceed its available RAM but also does not make use of more than one core meaning it will likely run very slow, especially in these very large data sets. Pandas does have useful data structures however, including the series and dataframes. A series is an array that can be indexed in any way the programmer would like, and Dataframe is set up like a python object but is very similar to an sql table in its usages as it can access different columns and rows. Most impressively it also provides a slew of options for data cleaning, including but not limited to filling empty data, dropping duplicate data, removing whitespace, replacing information, merging data sets, and standardizing entries.

Now, to analyze our options when it comes to our machine learning libraries, OpenCV, PyTorch, and Scikit-learn. Throughout this process the features that will be taken into consideration are the tools to train models, the types of machine learning it supports, and its friendliness to beginners.

Starting with OpenCV its training tools include image processing, the ability to split a dataset to have one be the training set and the other be the test set, and includes built in evaluation features to check how the model did. OpenCV specifically specializes in image based machine learning, which allows it to be used in places such as facial recognition, image reconstruction, predictive analysis, and object identification models. However, when looking at the documentation it is a bit underwhelming, much of the documentation is non-explanatory and is mostly syntax based which is not particularly beginner friendly.

Next up is PyTorch which focuses much more or LLM’s (Large Language Models) and Deep Learning. PyTroch has some impressive tools when it comes to model training, along with simple training upload options, there are also features to normalize a data set, and ways to nudge a model without uploading data through their optimization features. There is also documentation for a training loop that includes testing the accuracy of the model. PyTorch mainly works with Neural Networks which Deep Learning models are fundamentally based on, as well as optimization algorithms. As for PyTorch’s documentation there is a very well made and comprehensive documentation available online which would allow us to learn and use throughout development.

Finally Scikit-learn, which similarly to the other two options, has a very robust training ability with options of training by hand, self-training, along with multiple features to create test environments quickly. It also has by far the most robust choices for machine learning methods, including things such as classification, regression, data clustering, and dimensionality reduction which would all be potential approaches to solving our problem. To go along with that it is widely considered to be a beginner friendly tool with a very robust documentation, and due to its popularity has many available resources to go along with what is considered official.

5.4.4 Chosen Approach:

1: Data Preprocessing) For our project we plan to use PyGMT as our main character when it comes to data work, which by relation will require us to involve NumPy and pandas as they are both dependencies of PyGMT but they will be backup tools in case GMT is missing something necessary to complete this project. This decision came down to the strength of the clients recommendation, and in a field where we are the newcomers using something they are familiar with can be a great asset, but in terms of project planning the power of taking a derivative cannot be understated. Due to the dependencies of NumPy and Pandas should we need, we will have access to the NumPy data structures and the Pandas cleaning tools.

n terms of grading these are the main categories we considered along with their star rating on a scale of 1 through 5:

1. Extent of Mathematical Library
2. Cohesion with the NetCDF data type
3. Efficiency with large data sets
4. Available data structures
5. Data Cleaning Tools

	GMT THE GENERIC MAPPING TOOLS	NumPy	pandas
1	Great options such as derivatives and access to visualizations ⭐⭐⭐⭐⭐	Has standard Arithmetic Functionality ⭐⭐	Includes standard operations along with some statistical analysis ⭐⭐⭐
2	Has direct loading capabilities with NetCDF ⭐⭐⭐⭐⭐	Outside tools required ⭐⭐	Requires outside help and will not retain all important metadata ⭐
3	Struggles with a dataset larger than the RAM available ⭐⭐	Capable of temporarily using storage and calling data to RAM when needed ⭐⭐⭐⭐	Struggles with exceeding RAM and only makes use of a single core. ⭐
4	Uses provided tools from NumPy and Pandas ⭐⭐⭐⭐	Has a specialized array data structure for n dimensional data ⭐⭐⭐⭐	Provides specialized one low dimensional structures ⭐⭐⭐
5	No native capabilities ⭐	Ability to ignore and replace missing values ⭐⭐⭐⭐	Provides a slew of options most importantly the ability to create new values based on data set ⭐⭐⭐⭐⭐

2: Machine Learning Libraries) As for Machine Learning libraries, choosing to work with Scikit-learn and PyTorch is our best option as they are by far the most modular options and having the support for many different approaches across both libraries which will be a valuable asset over the course of this project. It also cannot be understated how important it is that it is easy to use as the team will benefit from not facing a large learning curve throughout the process.

For the grading we will again scale from 1 to 5 on these specific categories based on the desired characteristics:

1. Training tools
2. Algorithm Availability
3. Ease of Use

	PyTorch	OpenCV	scikit learn
1	Includes important tools to train a model and also format the data within ⭐⭐⭐⭐⭐	Provides image processing tools for training and grading tools ⭐⭐⭐⭐	Supports both training via uploading data and self training ⭐⭐⭐⭐⭐
2	Uses Neural Networks, and optimization algorithms. ⭐⭐⭐	Specializes in Image based machine learning ⭐⭐	Provides different approaches including Classification, Clustering, and Dimensionality Reduction ⭐⭐⭐⭐⭐
3	Sizable Documentation with help videos available ⭐⭐⭐⭐⭐	Documentation not very well written out but is very comprehensive ⭐⭐	Comprehensive Documentation and very intuitive API ⭐⭐⭐⭐⭐

Overall, these choices seem fairly obvious. When it comes to PyGMT the specific Mapping related tools are too useful to ignore and will be vital in the completion of this project. There is definitely some concern with the struggles in large datasets, but that can be worked around by splitting up the input into multiple parts and in practice running the program multiple times with no issues.

As for scikit-learn and PyTorch it became very obvious its well rounded nature was important as beginners, and they also compliment each other very well. They will allow us to experiment easier and will give us the best chance to be successful in this project.

5.4.5 Proving Feasibility

In order to prove the feasibility of these choices, we will first work with a small data set and ensure we can do some simple calculations using the tools that PyGMT provides to us. Then separately we will confirm connectivity between Scikit-learn and our model with a small preconstructed data set with the goal to find one outlier in a simple dataset. From there we will connect the two and begin to scale up and train our model.

5.5 Model Training Analysis

When it comes to creating a Machine Learning model there are two major choices we have. Either we can construct a model from scratch using the foundational tools available to us in the libraries we choose, or we can find a pretrained model that can be fine tuned based on our training data with less effort required.

5.5.1 Desired Characteristics

The training approach must support the hybrid architecture we previously discussed, which includes a Convolutional Neural Network (CNN) for spatial detection, clustering algorithms for grouping, and a Random Forest model for classification. It should also integrate smoothly with Python-based libraries, such as PyTorch and Scikit-learn, for model control, retraining, and evaluation within a pipeline. If we use a pre-trained model, it should already possess an understanding of geological/geophysical data structures (netCDF inputs) and allow for fine-tuning to optimize it for seafloor fault recognition.

5.5.2 Alternatives

We have two main options that we will consider for model training. One is a pretrained model called the Geophysical Foundation Model (GFM), and the other is building and training our own model by using the tools in libraries such as scikit-learn and PyTorch.

Geophysical Foundation Model was recommended to us by our mentor which was found on hugging face, which is a platform that is a hub for machine learning and artificial intelligence. This specific model was developed by Think Onward and officially released in 2024 and was built off of the Seismic

Foundation Model released in 2023. It was made with the intent of being flexible in order to be used for a variety of different tasks.

Building our Own Model would no doubt be the more challenging approach but no doubt has its own level of viability. Building our own model would allow us to have a higher level of flexibility. This has been an option we have been considering from day one and to do so we would make use of the tools made available to us in the Scikit-learn and PyTorch packages we have chosen to use.

5.5.3 Analysis

When we compare a pretrained model to building a custom model, there are many factors:

1. The GFM has already been trained on large 3D seismic datasets, which share spatial and geological similarities with bathymetric data. This understanding would give it a major advantage for transfer learning. On the other hand, a model trained from scratch would have no geological context and would need to learn all spatial relationships independently, which would be time-intensive.
2. The GFM's Vision Transformer with Masked Autoencoding (ViT-MAE) architecture is designed to learn about spatial relationships by masking and reconstructing portions of seismic images. This is well-suited for fault detection because it focuses on missing or discontinuous structures. Fine-tuning this pretrained model would require significantly less computational power than building our CNN from scratch since only the final layer would need to be adjusted.
3. Even though a custom-built model would give us full architectural control, a pretrained GFM would provide us with a proven framework. It can serve as the CNN backbone of our hybrid system, using its feature maps to feed directly into the clustering and Random Forest models. Since it is implemented in PyTorch, integration within our system should be straightforward through Scikit-learn.
4. The GFM uses transformer-based encoders, which means that the intermediate features are less interpretable than traditional CNNs because they use global self-attention rather than local convolutional filters. This causes representations more powerful for modeling but harder for humans to visualize. The non-seismic data warning also indicated that we may need to carefully preprocess our bathymetry data to match the structure that the model expects.
5. Choosing to build our own model will allow for full control of all design choices. This would guarantee there is strong understanding between our

data input and the model itself. This would allow us to spend less time focusing on the data preprocessing as we can train our model specifically to handle data in whichever way we choose to input it.

6. Building our own model from scratch would require a much larger data set to be available for training than using the GFM. At this point in time the training data sets that are currently built have not been provided to us therefore it is impossible to know if we have enough training data available to accurately train a model from the ground up.

5.5.4 Chosen Approach

We have chosen to begin with the Geophysical Foundation Model (GFM) as our foundation model. Being pretrained on seismic data and the compatibility with Pytorch make the GFM ideal for transfer learning on our bathymetric dataset. We will fine-tune the model to identify fault-like discontinuities and evaluate its ability to generalize to seafloor data. If the GFM cannot adapt effectively to bathymetric data, we will have to construct a smaller, custom CNN to replicate the behavior within our hybrid pipeline model.

Comparison Table:

1.) Compatibility with chosen algorithms
2.) Flexibility
3.) Smooth integration with Python-based libraries
4.) Suitable with NETCDF or Geospatial file inputs
5.) Training Requirements

	thinkonward	SEISMOSCAN
1	Uses Regression and classification algorithms, and specifically is in progress on segmentation of faults ⭐⭐⭐⭐	With a custom-built model training, it would take a large amount of time to build one from scratch to be compatible with the chosen algorithms ⭐⭐
2	Was designed in order to handle vast amount of seismic data, to identify many unlabeled features in	Building our own model would provide maximum flexibility as it is up to us to make the decisions on

	that data ⭐⭐⭐	what exactly is implemented ⭐⭐⭐⭐⭐
3	Designed to work specifically with PyTorch, which is something we plan to use ⭐⭐⭐⭐	Would allow us to use both of the libraries we have decided on in the way that we see fit ⭐⭐⭐⭐⭐
4	Not directly compatible with NetCDF data, however bathymetry data is in a sense derived from seismic data ⭐⭐	The model would have to specifically trained to understand and interpret what is going on with the data provided ⭐⭐⭐
5	With the documentation provided from the Geophysical Foundation Model, it has a clear outline on what and the number of data that was given to the pre-trained model. Thus making the model more experienced ⭐⭐⭐⭐⭐	Since there is no model implemented, there is no data given to the custom model. Thus, it will take a large sum of time to train the model and there is no clear outline on what data it needs ⭐

5.5.5 Proving Feasibility

To prove the feasibility of the Geophysical Foundation Model (GFM), we will analyze in detail with the model and install it, along with the required packages needed to make it work. Once it begins working, we will modify the model to be able to identify faults within the bathymetric data given to it. It will start off with the data already computed by the graduate student's fault placements. If it detects the faults that the graduate student placed and other faults, it will show it successfully understands what to look for. Otherwise, we will fine tune the model. However, if GFM is unable to meet the requirements it was used for, then we will resort to using our own model.

6. TECHNOLOGY INTEGRATION

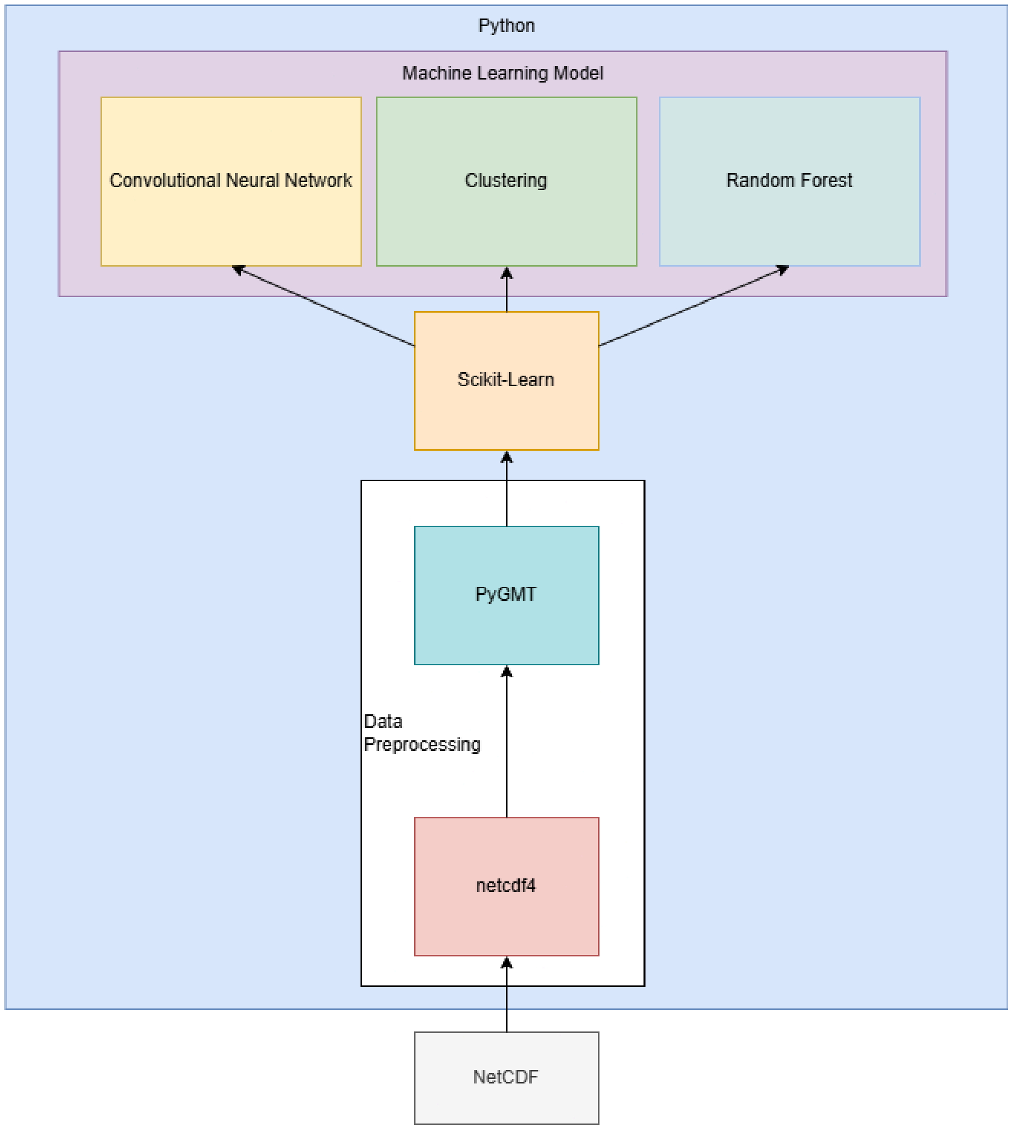
Our team has been chosen as a rather open-ended project, and as such there are many advantages and disadvantages that come with formulating a plan. Above we discussed data formats, Machine Learning models, languages, and libraries, and chose specific ones which we feel will best coalesce together to create a system for analyzing the data sets and knowledge provided by our clients. Our program flow diagram is rather simple since our project is simply a lightweight set of tools which will be able to be used offline by users and does not require a UI, server, or general high level infrastructure.

The workflow begins with data we have downloaded from GeoMapApp as a NetCDF file with a GRD extension for compatibility with python based libraries we are using like Generic Mapping Tools. The data will then interact with the native library for NetCDF, netdf4 python, which will enable the data to be loaded efficiently into Generic Mapping Tools. We will also generate anisotropic data which finds the slope of each point in the four cardinal directions to be used later.

Now GMT will run several scripts to trim down our search area in a broad manner by identifying underwater slopes since any area with one could possibly be a fault. It will also eliminate low resolution data from the map since this data is not detailed enough to come to a definite conclusion about faults underwater. We will probably create more routines for preprocessing in the future, but these two are the most general filters and probably the most important. Once the data has gone through preprocessing it will be handed off to the machine learning model which will search for potential faults.

The Machine learning model will first use a Convolutional Neural Network to search for spatial discontinuities along the ocean floor. It will then pipe the processed data into a clustering algorithm to reduce noise and differentiate continuous faults from neighboring faults. Finally, It would pipe data to a Random forest model to make a final determination on which features are faults with slope analysis from the data we generated earlier.

Here is an example of what the data transfer throughout this project might look like:



7. CONCLUSION

Our clients hope to have a program to successfully map faults based on bathymetry data using machine learning, that way they can use the data in order to research important geological processes. Processes such as the deep water cycle and earthquakes are believed to be greatly affected by the faults and the time saved by a piece of software will be vital to help geologists understand these processes.

In order to complete this project we will be using a variety of technologies. We have chosen the NetCDF data format since it integrates well with python, is relevant geospatial data, and has a very powerful and useful native library. For the model, we will be creating a hybrid machine learning model that utilizes a Convolutional Neural Network (CNN), a clustering algorithm, and a Random Tree, leveraging strengths from each component in order to ensure scalability, transparency, and compatibility. There are many potential ways to utilize other languages, but we have chosen to use Python as our main source of language for this project since there is a large community behind it, widely used for machine learning, containing simple syntax; utmost it is a language our team is the most comfortable with . Within Python we will use a variety of packages, for data processing PyGMT is our go to choice as it has the best tools to provide context of the sea floor, and to connect with our model Scikit-learn is the friendliest and best option to give us the widest variety of choices as it pertains to working with our model. We will move onwards with utilizing the Geophysical Foundation Model as our pretrained base model for the project, because its python compatible, able to handle the algorithms, Convolutional Neural Networks and Random Forest Model, and can understand the given geospatial data inputs. Should we run into problems with using this prebuilt model we have the option to pivot to creating our own. Using these technologies together will provide few hiccups and allow us to put our best foot forward as it pertains to completing this project.

We feel confident that after doing this research we will be able to complete this project successfully using the technologies we have found. Our next steps will be to create a simple tech demo to provide a template of how these technologies will work together to and build the project outwards using this demo.