Predicting Security Vulnerabilities using Machine Learning and Code Metrics

12/4/19

Group 8:
Baran Barut
Michael Harris
Curtis Helsel
Kyle Reid
Thomas Serrano

Sponsor: Dr. Paul Gazzillo

Contributor: Dr. Elaine Weyuker

Table of Contents

Executive Summary

Code is inherently insecure, with both security vulnerabilities and other faults in production systems decreasing the reliability and performance of code that runs the modern world. Our project connects a world-class front-end Application, driven by a machine learning backend that has been trained on publicly available metadata about the development cycle, in order to allow developers to make accurate predictions about where to find potential code faults within a software package. This will allow developers, both small and large, to be informed on where problematic areas in their software projects are likely to occur. Resolving issues before they are released using this tool will save time and money.

To wit, we utilize cutting-edge machine learning technology for our statistical model, expanding on existing work in the field by a contributor, Dr. Elaine Weyuker. Key to the project is comprehensive data acquisition methods and custom utilities to ensure our model can operate at maximum efficiency. Our product requires a robust front-end to interface with the end-user and communicate results in a meaningful way about how to improve the development cycle. Finally, a flexible backend infrastructure is necessary to reliably deliver the data required at each step, in addition to accommodating shifting requirements at an early stage in the project.

Goals and Objectives

Goals

- Make the software engineering field much more secure by providing software engineers and testers with better tools to find security issues in their code
- Raise awareness of security vulnerabilities that may not be as popular or as widely recognized to software engineers
- To lessen the number of vulnerabilities that exist in code by providing software engineers with all of the necessary data to prevent them from occurring
- Improve the software engineering field by making the development cycle more friendly to the humans who participate in writing code
- Perform research on publicly available information by turning APIs into a trained dataset backed by a robust statistical model
- Perform research on identifying the best network components and connections for this kind of classification to ensure that it is operating at peak efficiency

Objectives

- Predict security vulnerabilities within code using specific metrics gleaned from the code
- Provide users with a friendly interface informing them of where vulnerabilities may lie
- Provide users with informative analytics of information regarding their code
- Improve upon existing work in the field by utilizing:
    - Machine Learning statistical modeling
    - Data Acquisition using publicly available metadata

Example Scenario

Imagine you are part of a team that spends their time developing an open-source project hosted on GitHub. Each person has their task, and you all work well on your particular parts of the project. While each person may be mindful of security faults that they must mitigate and avoid in their part of the project, it is still possible to introduce new security faults when the different pieces all come together. Developers are also fallible human beings and can miss or unknowingly add security faults while they develop. Combing through every single line of code of every single file is not feasible and would be done better and quicker by a specialized tool. While some tools exist to find non-security-related faults, and others do exist to find security-related faults, you can never be too careful when it comes to the safety of your project. Therefore, it would be wise to incorporate the Predicting Security Vulnerabilities (PSV) tool into your project testing workflow and development lifecycle.

To begin using our tool, you would start by downloading and running the binary from our open-source repository located on GitHub. After the binary has started, you are presented with a user-friendly graphical interface to assist in scanning your open-source project. Begin the scanning process by inputting the *.git* URL to your remote GitHub repository and a local path where you'd like to clone the repository. You may also test the GitHub URL to verify that it exists and is reachable if need be. Click *Clone and Open Project* to clone the project locally and open it within the PSV tool. Proceed by selecting any scan settings you may find appropriate and then initiate scanning by clicking *Begin Scan*. You may view the progress of the scan through the progress bar located at the bottom of the program, and the scan results automatically begin to populate in the *View* tab. Also located in the *View* tab are the *Vulnerable File List* which displays all files deemed to contain a security fault, the *Vulnerability Table* which contains all vulnerabilities found within the project, and the *Vulnerability Graph* located under the *Graph* sub-tab, which displays the type and count of vulnerabilities that were discovered. With this information, you are made privy to which specific files may be affected by security vulnerabilities and can quickly and effectively go about remedying such faults.

Example of Vulnerability and Fault

Below we will display a security vulnerability, in the code of a SQL injection, and a code fault, in the form of a C program which utilizes an insecure method in a way that causes undefined behavior, and potentially a crash. This should make clear the difference between a vulnerability and a fault in the context of this project.

Security Vulnerability

Here, a string is provided by the user, which is meant to return a user record in the database.

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Below we see the danger of not sanitising input, since the equation “1=1” will always evaluate as true, which, if passed into the above code, can be used to return every user and their password from the database. This is clearly an exploit of the above security vulnerability.

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```

Code Fault

For this example, strcpy, a method that does not check the source destination to ensure it is large enough for the operation to complete successfully, is executed, causing potential undefined behavior when the dest string is printed to screen.

```
int main() {
    char src[] = "buffer overflow";
    char dest[2];
    strcpy(dest, src);
    printf("%s", dest);
    return 0;
}
```

1. Introduction

Personal Motivation

Baran Barut [Front-end Application]:

Security has always been a personal interest of mine, and it has been one that I have pursued for years. While I have had such an interest in security, I have not done much formally in the security field and I felt like this project was a great way to finally "do something" with my interests. Having a machine learning component made the project that much more exciting, as while I've been exposed to the ideas of machine learning, I've only barely touched on it. I believe that by forcing myself to have to be a part of a project that greatly involved machine learning, I can finally muster up the motivation to learn about a field that so many are saying is "the future of technology". I think that it'd also be great to note that Dr. Gazzillo seemed like a very understanding and relaxed person when presenting, which to me was a nice break from the rigor and strictness of general academia. While this project is no walk in the park, having someone who is there to work with you instead of telling you what to do is a very nice change of pace. Overall, nearly every single aspect of this project interested me, as it married topics that I love, topics that I've worked with, and topics that I've wanted to work with all into one nice idea. It seemed almost natural to want to choose this project, and I look forward to making the most out of it to help the world around me.

Curtis Helsel [Machine Learning]:

After learning more about what this project was really looking for, I was intrigued by the impact the work we will do will have on security in software development. Couple that with the machine learning aspect of the process and this became a project that I knew would enjoy working on. I have a large interest in machine learning. I intend to go to graduate school to pursue research in this field. The goal of this project is to write an algorithm that will predict when a security vulnerability is put into code. This is not just a quality assurance program that checks against already known vulnerabilities but rather a program that predicts where vulnerabilities lie based on different features of the codebase. To me, this is a really cool idea because we are going to be basing our predictions on features that may not previously been explored such as time specific code was pushed to version control, how long the code base is, where in the code base specific vulnerabilities occur. The number of features we could adapt are endless and the researcher in me is really motivated to find a best fit model since the outcome of the project could benefit all industries of programmers. If we know where we are most vulnerable, we could combat these vulnerabilities and produce better overall code.

Kyle Reid [Data Acquisition]:

My personal interest and motivation in doing this project is wanting to have a better understanding of computer security, I've been slowly working on security issues on my own free time so I figured this would help me out in some way. I also have a key interest in machine learning and learning big data and data analysis and would like to learn more about both of them while doing research and putting them to use in a real-world project. I don't have any specific language or area that I'm superbly strong at considering I only just heard about computer science 3 years ago but having narrowed down my areas of interest I'd like to gain more knowledge in them and master the languages and tools they use.

Michael Harris [Database/Server, Project Manager]:

My interest in this project comes from using cutting edge machine learning technology to detect flaws in software in the form of patterns which are difficult for humans to recognize, and reliably improve upon this, yielding a product which gives some highly useful information to software developers. The more useful information we can feed into a model, the more useful a tool like this can be at both predicting faults but also providing a deeper insight into why these events are happening, which can improve the development process and the quality of life of developers. Goals include identifying reliable data sources which can be mined for effective points of data that can be fed into the model, identifying the proper and effective model to use, and training it so that it continues to improve the more useful data it is provided with. Identifying a platform to host data and a backend, and also facilitate any GPU related learning/training tasks. Familiarizing myself with this technology and working effectively in a team atmosphere using collaboration tools to create an amazing product that we can all be proud of is also a goal and a main objective. The function of the project should be to provide feedback about code in some constructive and highly ordered way which leads to noticeable gains when this feedback is introduced back into the development process in a positive, constructive manner. As a teammate I strive to bring attention to detail, informed opinions and well executed platforms to the table and be ready to assist my teammates whenever I can to help ensure our project is always on track.

Thomas Serrano [Machine Learning]:

Cybersecurity and artificial intelligence are two areas of computer science that I've been interested in for a while now. Not only do I think that the material is interesting, I think that the two areas of study are very important, especially in our often times rapidly-changing field. The union of these

two fields is a no-brainer for me, and I think that a project like this is only scraping the surface of what can be accomplished when combining cybersecurity and machine learning. A big reason why I chose this project is because I don't actually have a lot of experience in those fields and I think this will be a great opportunity to dive in and get my hands dirty. I think it's a real privilege to be able to indulge in the subjects I care about and to have "free" (since this is still for a grade) time to learn about what I want to learn about and to learn from my peers. At the time of writing this, we've had a few meetings between ourselves and with our sponsor, Dr. Gazzillo, and after each one I find myself thinking "Wow, this project got even cooler!" As much as I was interested in the project from the beginning, I underestimated it - in a good way. This project has so much potential depth and I really can't wait to get this thing going and see where it leads us.

Constraints

Our results are constrained primarily by the quality and reliability of the data we can acquire, so it is essential to source useful data and perform any necessary cross-referencing to obtain integral data points which lead to acceptable results. The process involves having data collection roles communicate with those responsible for the machine learning, and also with the backend person who needs to store this data and make it accessible, so communication between roles is critical. Pulling dev cycle information from Git using APIs can yield a heap of data that is sanitized and ready for modeling, but we may need to be prepared to explore other methods of obtaining useful metrics if this does not prove to be enough for a learning-based approach to this problem.

Broader Impacts

In today's world, software code is everywhere. Security vulnerabilities and faults can affect our lives in countless ways, especially in real-time systems like those used in vehicles and other critical embedded platforms. With the internet of things becoming real, it is crucial to emphasize how important it is that the code that runs our lives should be safe and secure. To that end, we aim to design a portable solution that combines data modeling, machine learning, reinforcement training, and a robust interface to detect code faults or security holes, and provide useful, targeted information to the developer so that they can both produce more accurate code, and become aware of issues that may arise over time. The application improves the reliability and functionality of existing platforms, which have the net effect of making our lives safer and increasing the reliability of the technological solutions we can provide as a human race.

2. Technical Content

Requirements

Application

- Must be easy to use, should be able to get “up-and-running” with ease
- Must allow selection of directories/repositories/files for analysis
- Must show where in the code we predict there may be issues, as a heatmap
- Must provide analytics after running our analysis through graphs and charts
- Seamless continuous-integration through TravisCI
- A multitude of unit tests for the debugging of our application
- Must be able to run background ML scripts against directories/repositories/files

Data Acquisition

- Will be gleaned through public repositories
- Code directory meta-data (For example, number of collaborators or number of times committed)
- Stored into our database in an easily accessible format
- Must be able to be used to train our model
- Must be refined consistently to train our model further
- Custom scripts to grab data not available through APIs

Machine Learning

- Must be able to train a model efficiently and promptly using our data
- Must be able to run the model against code to predict the locations of security vulnerabilities
- Ability to detect useful data points to help refine data collection
- Refine collection methods by integrating model outputs
- Leading to a more correct, efficient model to increase prediction accuracy

Server

- Database for machine learning data
- Data collection tools hosted within a shell script
- Ability to quickly update the database by running data collection scripts
- Local SQL connection, endpoints if needed
- Push application updates
- Push model updates to the application
- Facilitate GPU cluster access for machine learning training roles
- LAMP Stack or equivalent for flexibility
- SQL Database is a critical component for machine learning data repository
- Can pivot to provide any data delivery mechanism, be it endpoints, or database CSV dump. Not tied to PHP. Current plan is for finalized data collection role to live on server, along with the model once it is approaching maturity.

Specifications

Due to the research-driven nature of this project, it is challenging to put quantitative specifications into place since they may constrain the future development of the project. However, we can glean specific objectives from the nature of the project's scope.

Machine learning takes lots of useful data, so acquiring this and storing it in a way that makes sense is a must. Utilizing publicly available development tools, like GitHub APIs and metadata, are necessary to identify projects that are good sources of data and yield the best results for our purposes. To facilitate this, we should utilize open source projects and make the data store easily accessible by all parties. Code metrics, or metadata, are critically important to making predictions and is what our model uses to find out which files contain security faults, and which do not. Further, we need a front-facing solution that is easy to use and able to provide users and development teams with reliable information that can be used to improve their development cycle.

Machine learning tools, like neural nets, are an excellent way to build a statistical model that can make predictions based on data it has seen before. Experimenting to find the right model using different training sets and architectures is essential to get valid information about what is possible to predict through public APIs and bug information. Once development the model is complete, it is imperative to test it upon unseen data to develop a sense of how it can work in the "real world." Evolving data fields and updating the network architecture are vital at the mid-stage of the project.

The following project outline diagram shows the project's explicit design summary, with the block diagram that follows it elaborating on each segment, and the UML diagram providing a high-level abstraction of how each party interacts with the project.

Figure 1: Project Diagram Outline

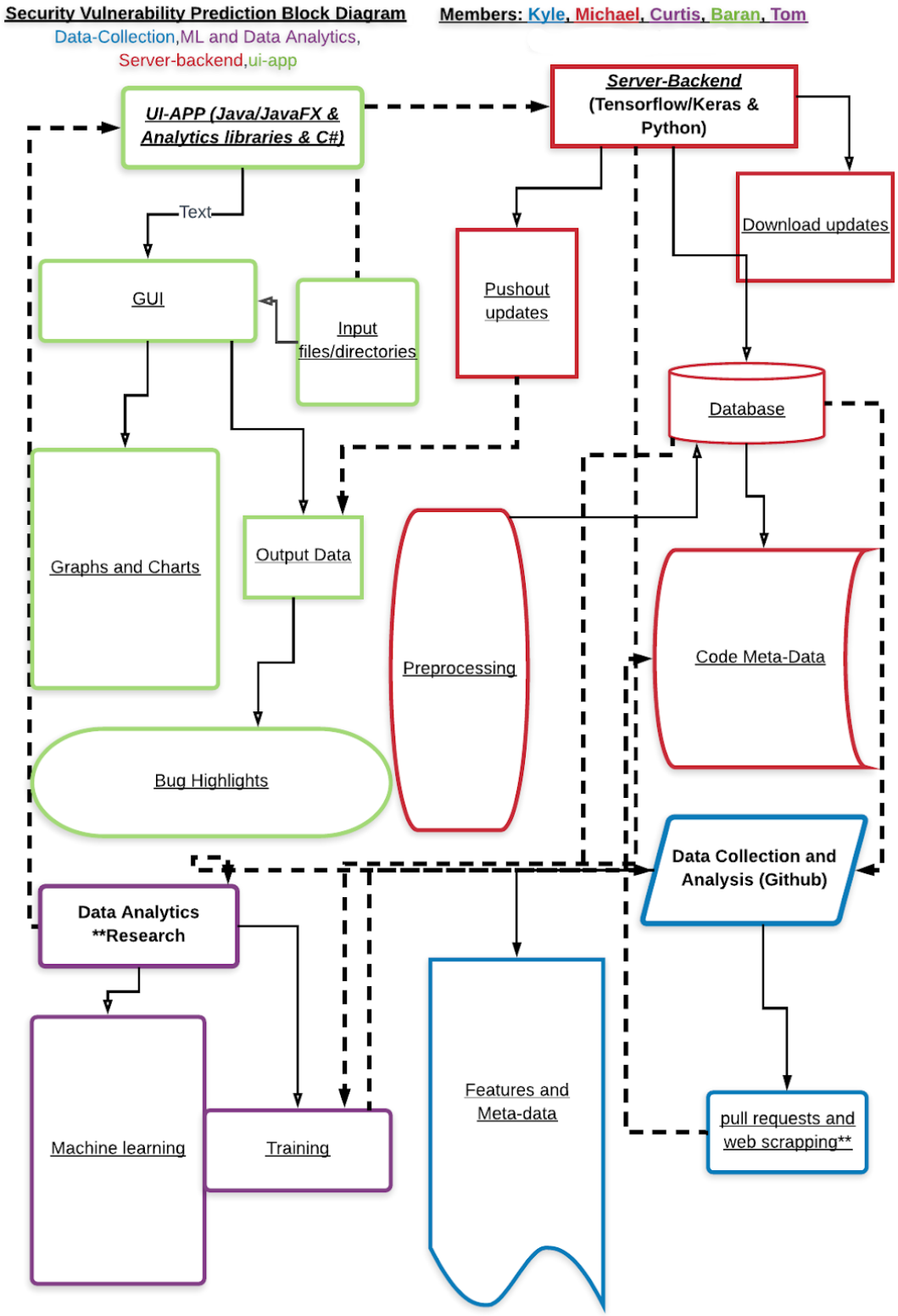


Figure 2: Project Block Diagram

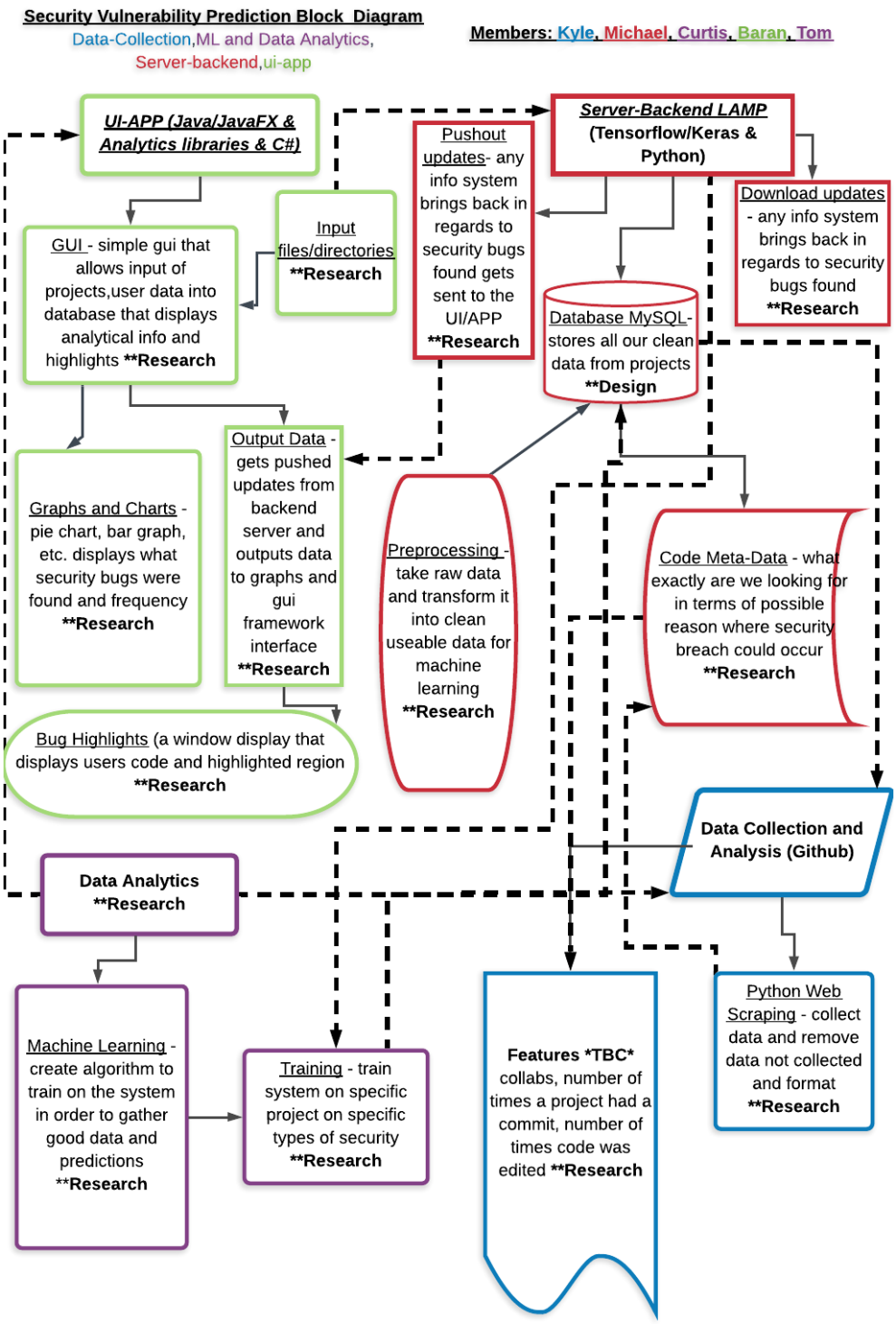


Figure 3: UML Diagram

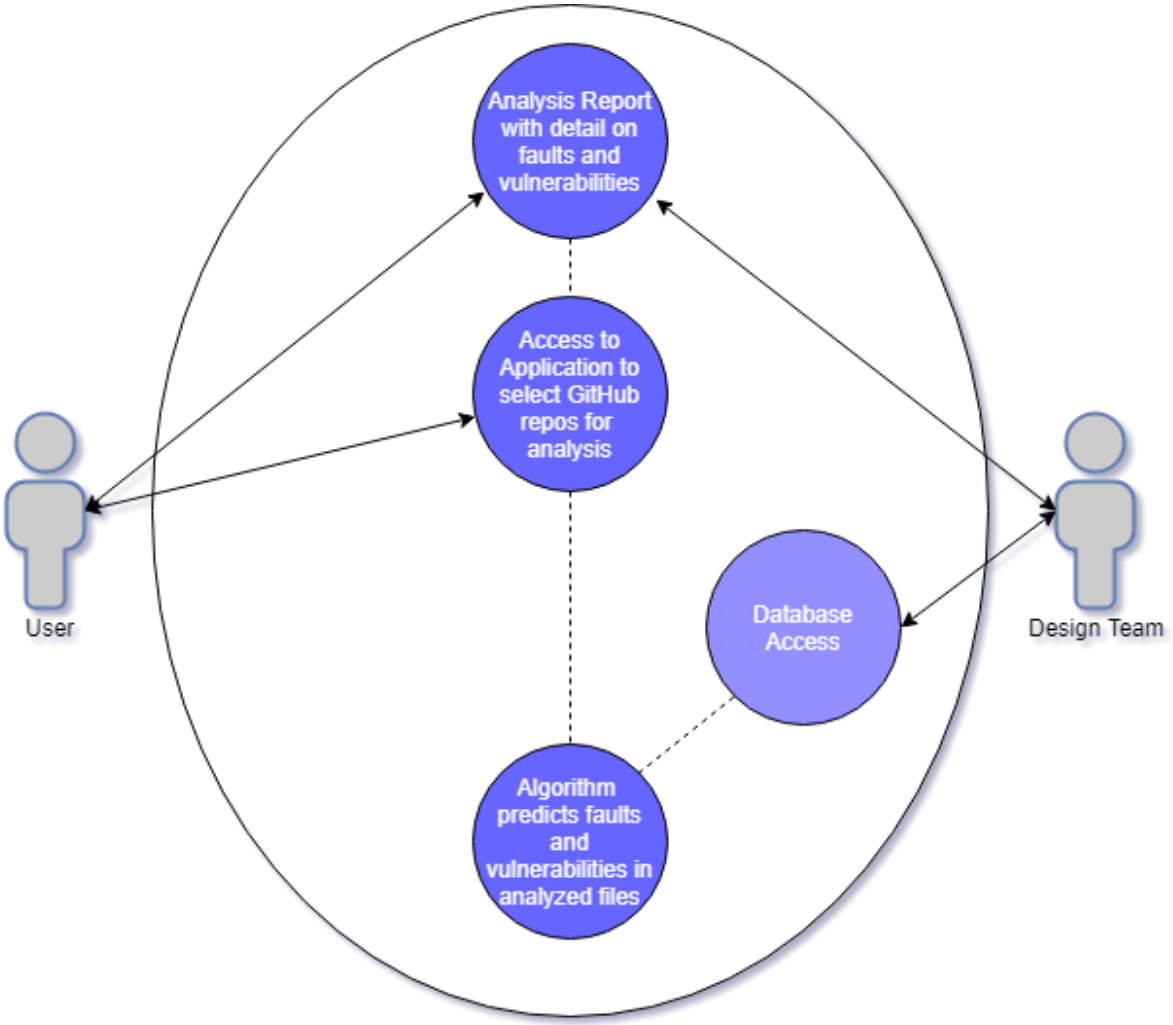
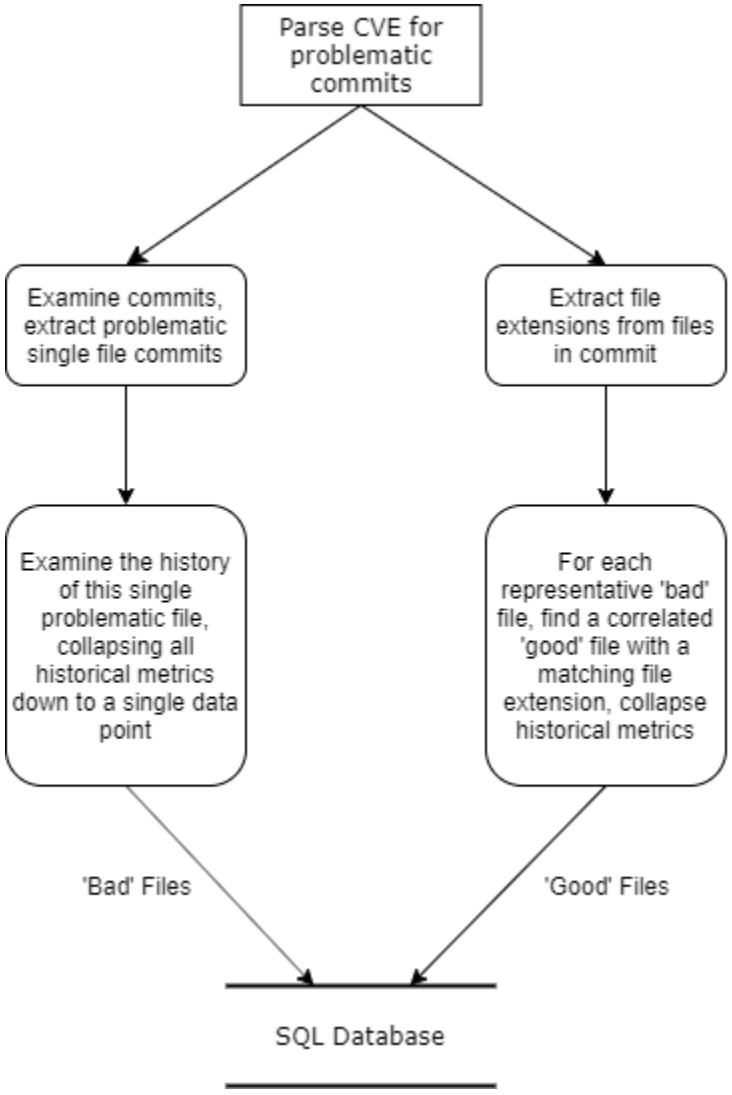


Figure 4: Dual path data intake Diagram

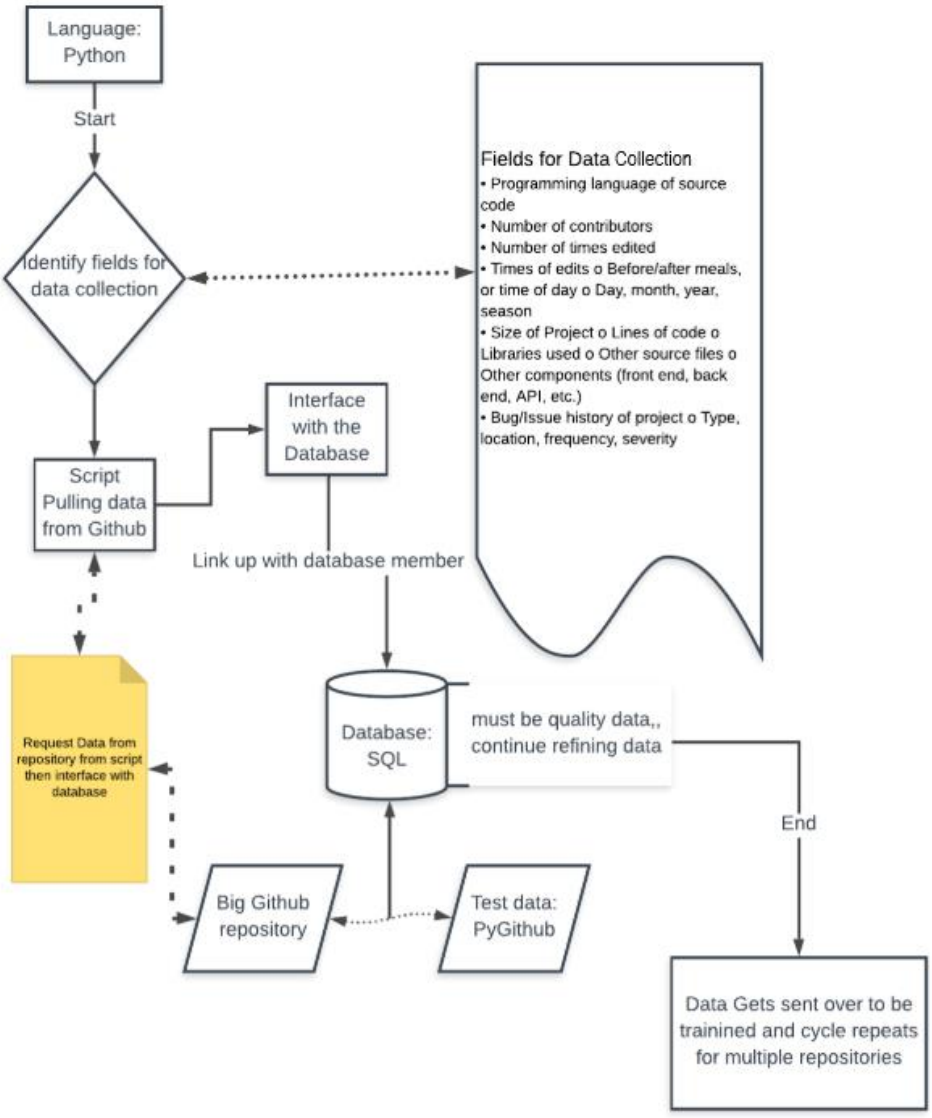


Entering clean, balanced data into the training set is important, as this rough look at the intake pipeline shows.

Acquiring Data Fields

Data collection sources: [1,2,3]
Programming Language: Python [4]
With the repository data fields being collected, the necessity arose to have more specific data on specific projects rather than the information provided to us by the use of the PyGithub [5] library that focused on repository data associated with GitHub such as number of subscribers, watchers, branches, commits, issues, collaborators, etc. [6,7]

Figure 5: Data Acquisition Process Diagram



Using Python to facilitate data acquisition, homing in on fields for data collection and storing the data to refine further are the critical processes for acquiring data. Once modules are complete, a

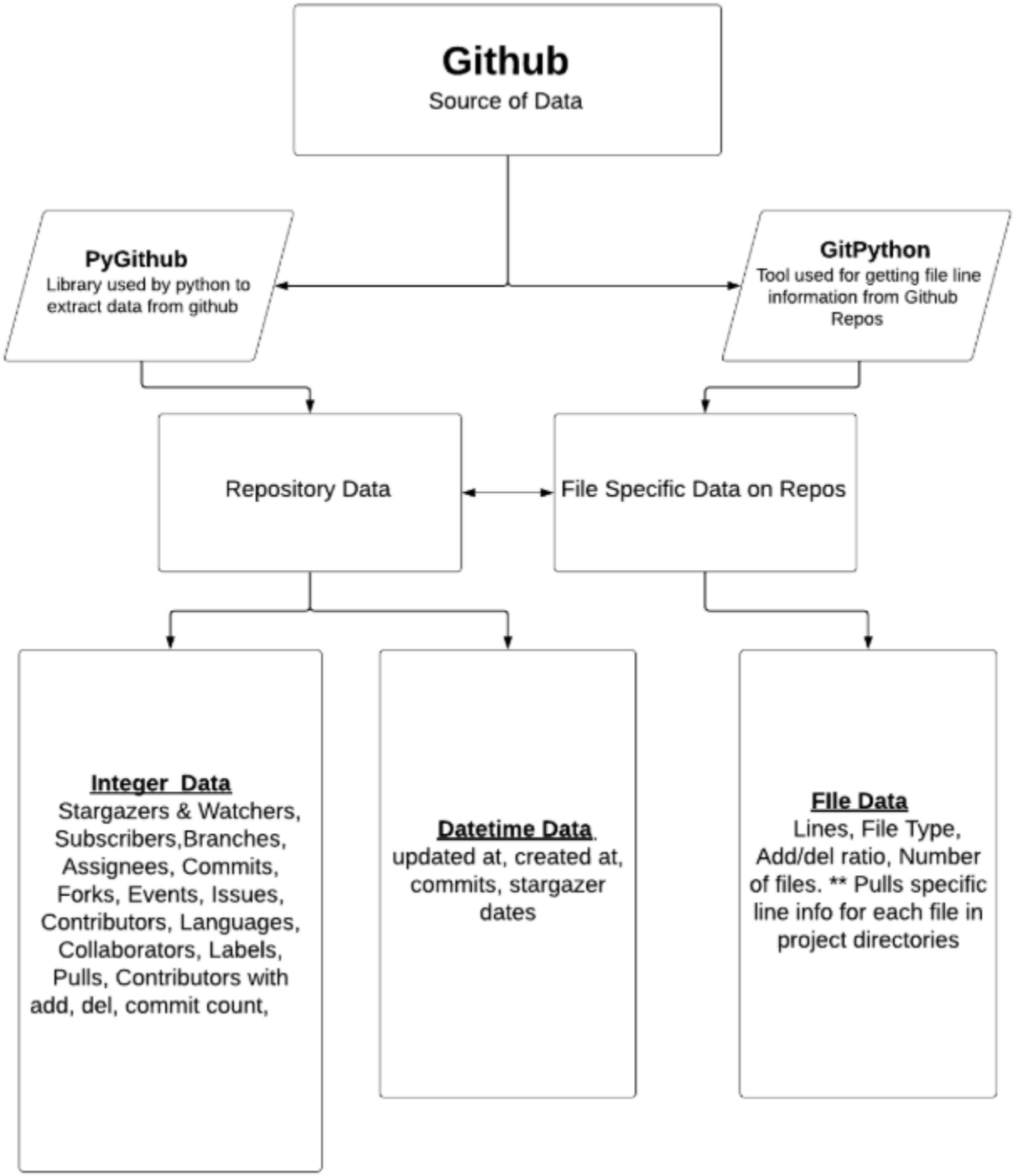
fully functional script can be written to acquire all meaningful information and store it in the database. At this point, data is available for modeling and iterative refinement.

GitPython, [8] another python library was used to find a way to collect data specific to the files of projects. The data collection was done using the commit log history which had detailed information on the number of files changed and messages related to security vulnerabilities that were needed. With the commit information, we can parse through the data given and pull more specific data points for use in our Machine Learning algorithm.

The data acquisition flow diagram shows the fields slated for collection, and how they interrelate. PyGithub is used to pull specific generic data from GitHub repositories, whereas GitPython focuses more on the file specific data over commits. Data is organized into integer data, date-time data, and file data.

Python's subprocess and os libraries were used to run Linux commands through a python module, which parses the string of information provided. The parse was used to acquire the number of lines changed and or a total number of lines in each file.

Figure 6: Data Acquisition Flow Diagram



Repository Data from Github

Ex: Type of data, Integer value amount, Dates & times

Stargazers & Watchers	Issues
Subscribers	Contributors
Branches	Languages
Assignees	collaborators
Commits	Labels
Forks	Pulls
Events	Contributors
referrers	size
has_issue	has_projects
has_downloads	has_wiki
milestone	Network
Updated at	Created at

File Specific Data on Repos

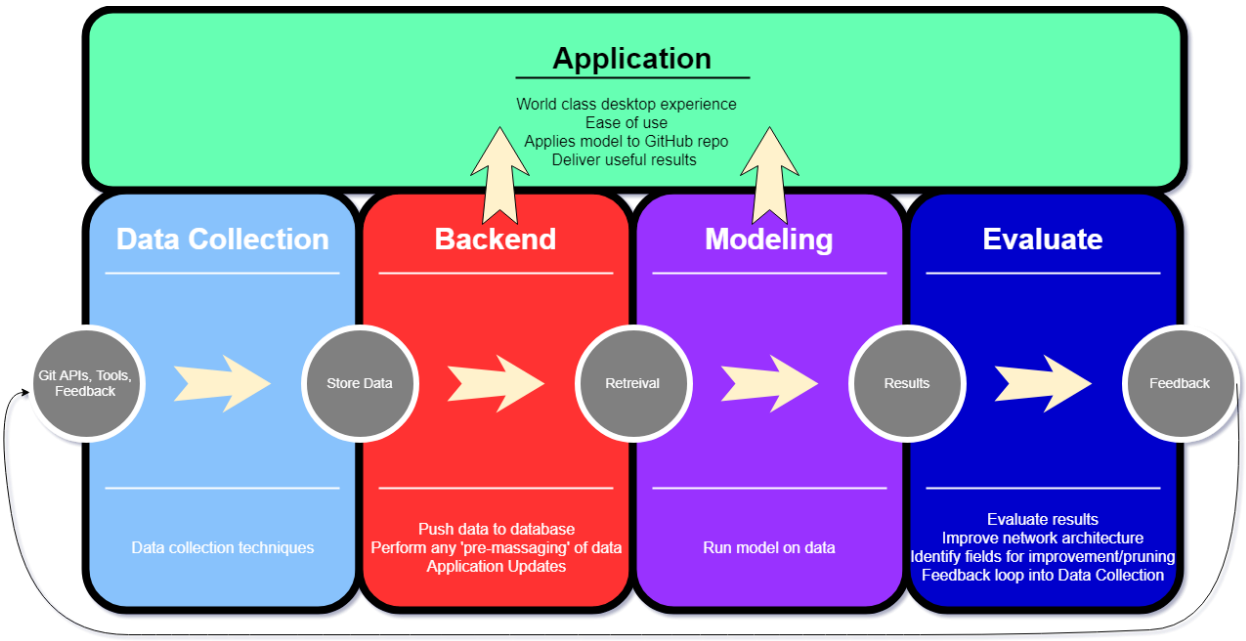
Ex: Number of lines, File types, number of files & type

Lines of code	Language type	commits /hr
Avg number of deletes	Avg number of adds	commits/day

Storing Data Fields

As shown, the SQL [9] database is a critical intermediary between the data gathering tools and the machine learning backend components which rely on this data. Below the databases' role is shown as it relates to connecting the other portions of the project.

Figure 7: Project Flow Diagram



Database logistics

The machine learning portion of the project relies upon this data to make accurate predictions, and as such, some fields likely need to be 'massaged' or turned into a floating point (or integer) representation of the data, whichever is consistent. The data includes all relevant trackable info about the file history and may include mapping a numerical representation to arbitrary data, such as averages over time. As such, a final design of the database includes mostly double or Boolean fields, so that the data is fully ready for the model and in as much of a streamlined form as possible.

The primary database currently utilizes "many to one" relationships between the database tables, but there are other relationship types. In a "one to one" relationship, each table can only have one record on either side. Each primary key value relates to only one, or no record in the corresponding table. If this kind of relationship looks like it may be viable, instead of considering just putting all values in the same table, it simplifies the logical layout of the system.

Similarly, in a "many-to-many" relationship, an intermediary table must be used to store these relationships. Whereas "one to one" is like being married, many-to-many is like having any number of siblings. These should also be avoided if possible due to how it logistically complicates

the layout and obfuscate what data is being held. The layout should be as simple and prepared as possible to facilitate the prediction model.

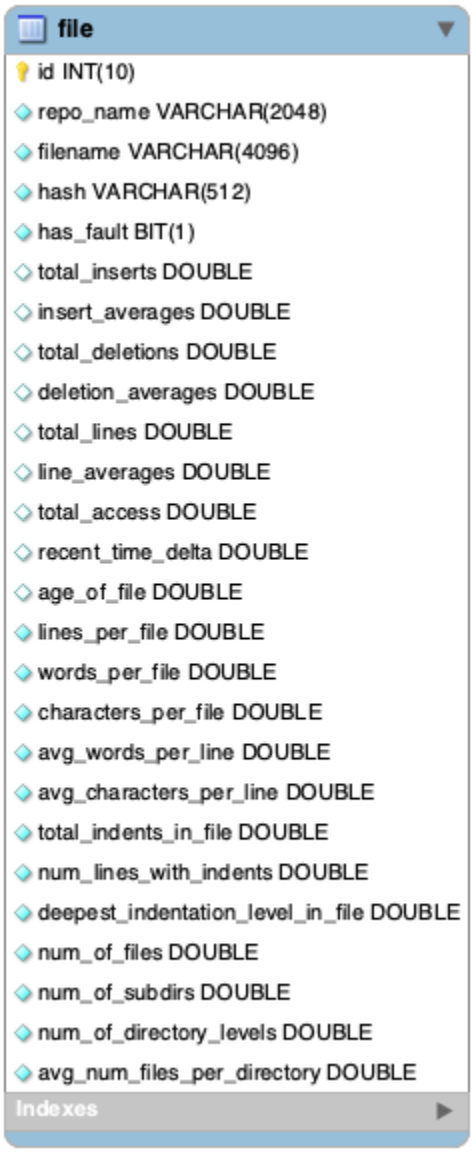
Database ERD

Here, the full database layouts are shown, with a simple PK/FK relationship to trace back to the originating commit/repo. The first database design included fields common to a repo, and this information was entered into the training database as being common to each file in the repo. However, training the dataset without these repo fields resulted in a higher accuracy, so only per-file metrics were considered at this point in the project.

At this point, a second database design was created to expand on these file specific metrics by adding new data points in the form of new file-specific features. Since this design does not include repo specific information, it is only a single table with all information about the problematic file, with the repo being stored in the form of a varchar field.

Figure 8.1: Database Entity Representation Diagram, with repo fields

Figure 8.2: Database Entity Representation Diagram, repo fields replaced with expanded file-based features



Database Structure

The ERD [10] above shows the logical layout structure for our database. The repository is the highest level of the hierarchy, and each ‘repo’ object holds ‘commit’ items in a one-to-many relationship. Likewise, the commit contains many instances of each file, holding file information in a one-to-many relationship. The ‘file’ is the finest grained object that our model looks at.

Primary keys and foreign keys are how to identify and track information within a relational database. When an entry is created in a table, it is assigned a unique key that is used by the system to index results and optimize searching. When creating records that are children, they also get a new, unique primary key for the database table they are in, but the foreign key also maps back to the original primary key. This way a chain can be formed where it is evident where the originating parent object is, for example, which repository a file belongs.

GitHub API accessible fields

Repo:

allow_rebase_merge	bool
allow_squash_merge	bool
archived	bool
created_at	datetime.datetime
default_branch	string
description	string
fork	bool
forks	integer
forks_count	integer
full_name	string
has_downloads	bool
has_issues	bool

has_projects	bool
has_wiki	bool
homepage	string
id	integer
language	string
name	string
network_count	integer
open_issues	integer
open_issues_count	integer
private	bool
pushed_at	datetime.datetime
size	integer
stargazers_count	integer
subscribers_count	integer
topics	list of strings
URL	string
watchers_count	integer

The fields for commit and file are much sparser and require some interpretation and creation of different fields to be sure useful information exists about each file which can be used to make accurate and precise predictions.

Commit:

author	github.NamedUser.NamedUser
comments_url	string
commit	github.GitCommit.GitCommit
committer	github.NamedUser.NamedUser
files	list of github.File.File
html_url	string
parents	list of github.Commit.Commit
sha	string
stats	github.CommitStats.CommitStats
url	string

File:

content	string
download_url	string
encoding	string

git_url	string
html_url	string
license	github.License.License
name	string
path	string
repository	github.Repository.Repository
sha	string
size	integer
type	string
url	string
text_matches	string

Database access

With the data fields mapped out, data collection is ready to begin putting information into the database. First, an account was created for remote access with a secure 64-character password that would be sufficiently secure for temporary remote database access. All permissions were granted for this user, then a few non-essential ones were disabled, such as dropping database tables. Then, to allow remote connections on this user account, it is necessary to modify his record's 'host' field from 'localhost' to %, the wild card character. The modification allows an incoming connection from the user account from any IP address, but will only be necessary for testing as the final package will run entirely on the backend server and connect to the database in a local mode.

To allow remote database connections, editing the file at /etc/mysql/mysql.conf.d/mysqld.cnf is necessary. Under the header [mysqld], changing the "bind-address" flag to hold the external IP of

the server. This change makes the SQL database look for incoming connections over the internet instead of only over the local area network.

Currently, the data collection utility is planned to be moved onto the server, where a local SQL connection is not prohibitive or a security hole. The modeling role may opt to use endpoints or some kind of CSV dump from the database, but the current access is sufficient for training purposes and endpoints can be configured at a moment's notice to facilitate access. A feature was planned where after locally scanning a repo, it could be submitted to the database as a training data point. However, due to the inability to verify claimed fault integrity, this feature was scrapped as it would potentially contaminate the database with incorrect fault information.

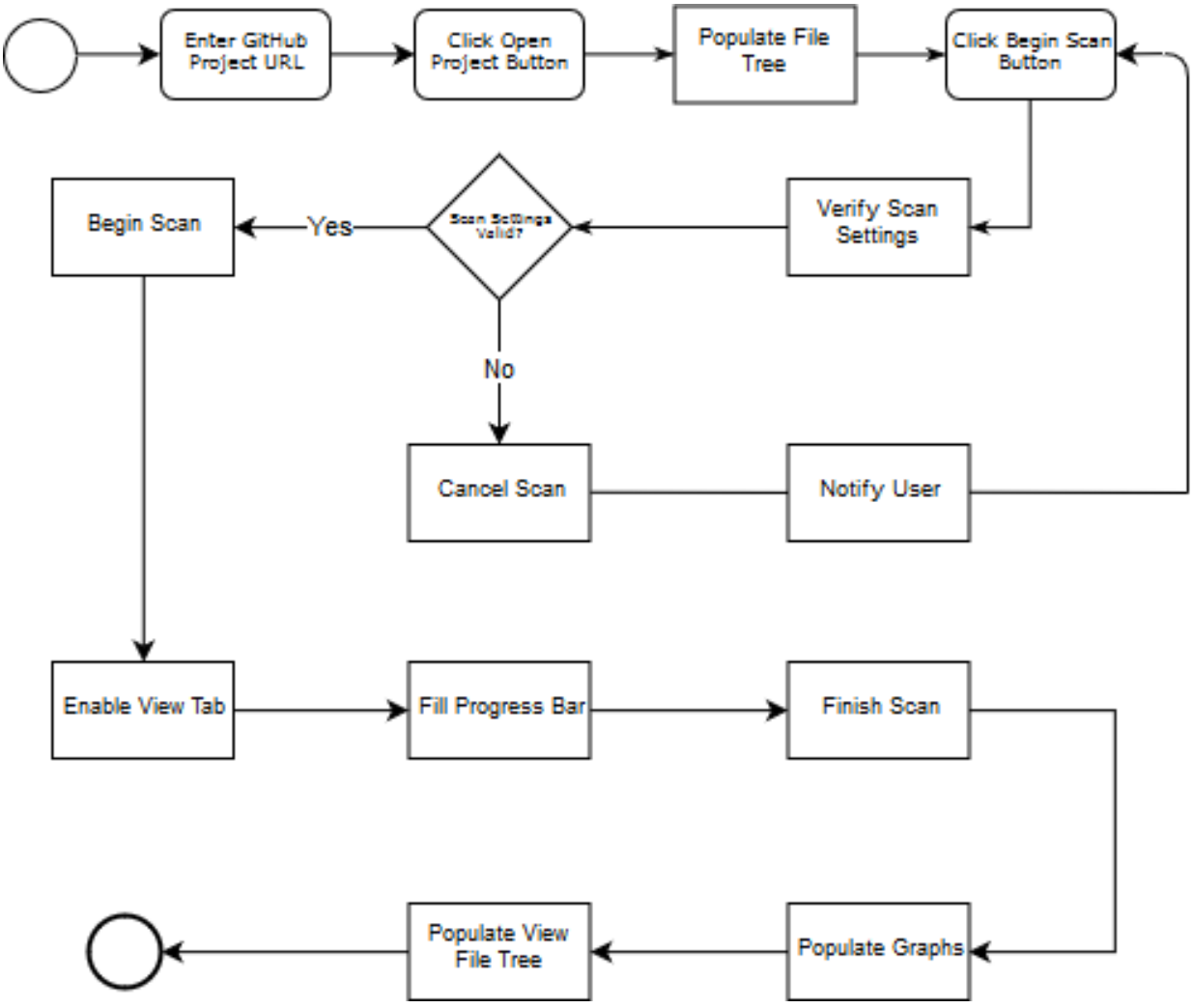
Design Status

This database design has reached a final stage where the fields are reasonable and representative, but improvement and iteration will still likely be required moving forward to identify further file-specific data points. The backend configuration is flexible and can pivot to facilitate any role or task as the need arises, which is required by a research driven project such as this where needs can change quickly as the scope of tasks is adjusted. If a design in the future is examined where information about each commit is stored, such as for use in an RNN, an additional table would likely need to be added as one-to-many for each file.

Application Design Prototyping

As the primary interface between our product and the user, prototyping, and planning of our application happened at an early stage and was refined upon with feedback from the team and a shared vision on what it should accomplish. Users select a GitHub project, populate the file tree, verify settings and initiate scanning. While scanning, the progress bar is filled to inform the user on the operation's status. After the scan is complete, problematic files and graphs showing statistical information are displayed to the user.

Figure 9: Application Process Diagram



Application Design Mockups, Current Status

The prototypes that were sketched up after the team reached consensus on an overall design were quickly turned into a working desktop application, built in C# as a universal application that should function on Windows, Mac, and Linux based platforms.

Figure 10.1: Scan Tab mockup

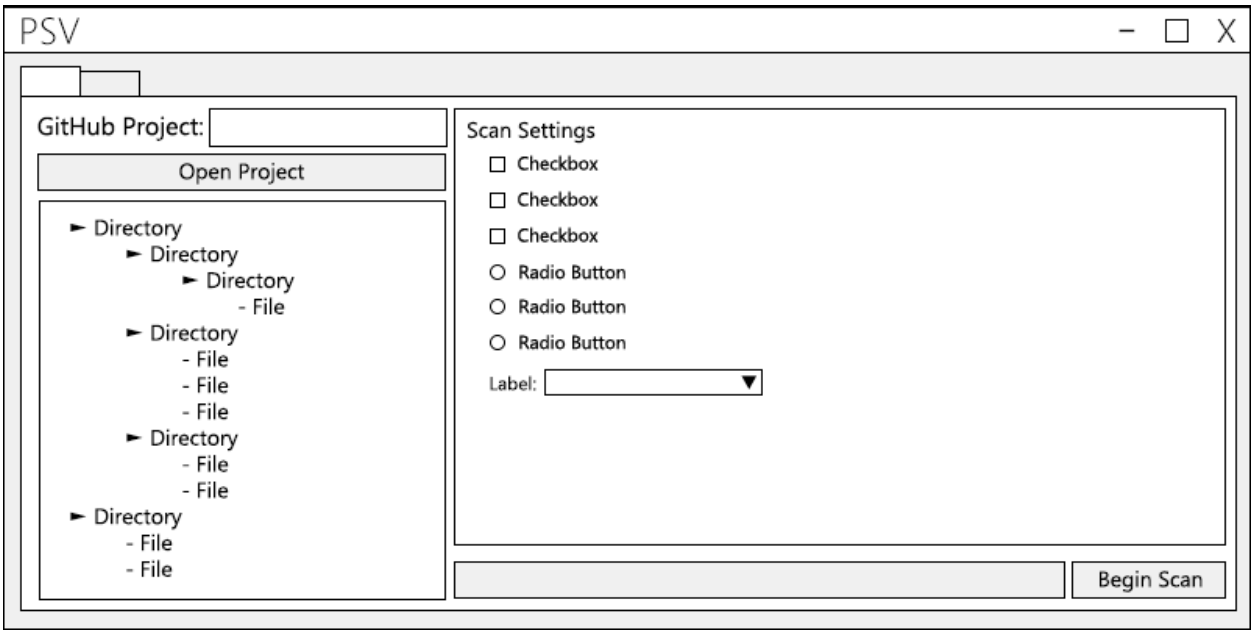


Figure 10.2: View Tab (Table) mockup

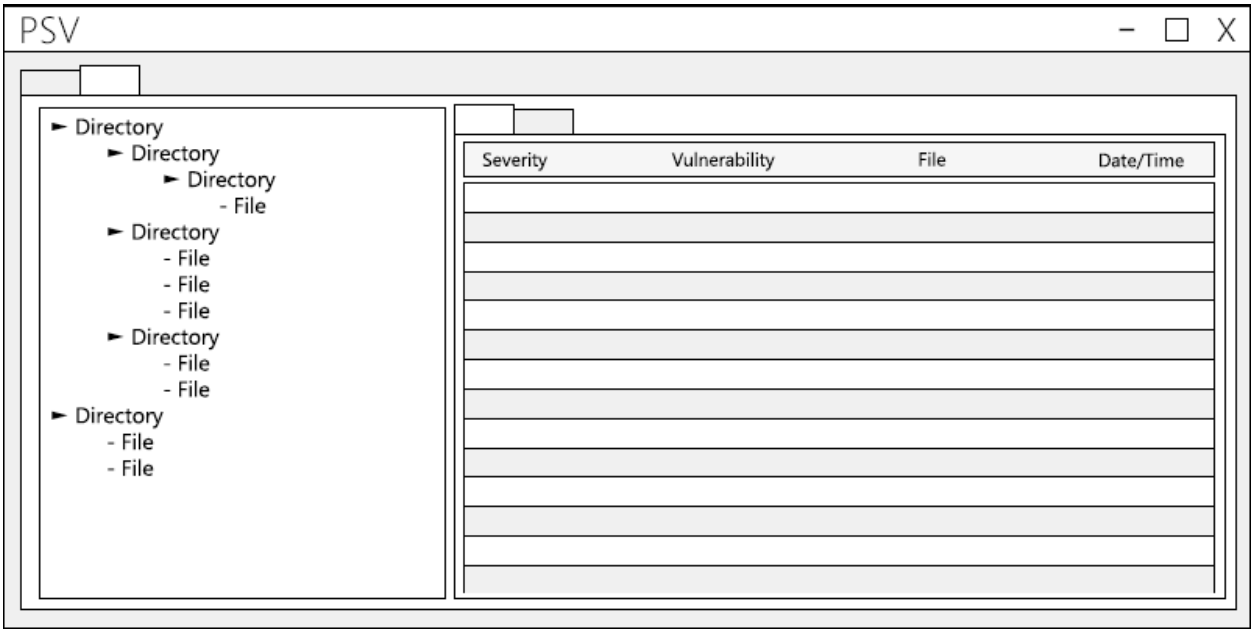


Figure 10.3: View Tab (Graph) mockup

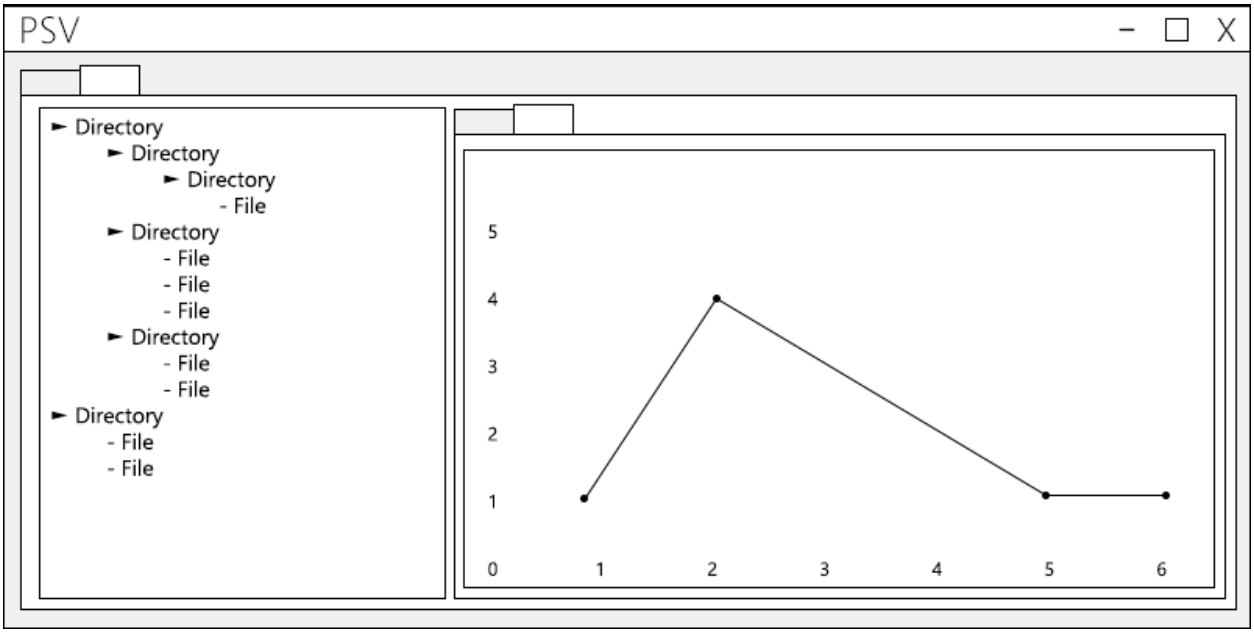


Figure 11.1: Scan Tab original design

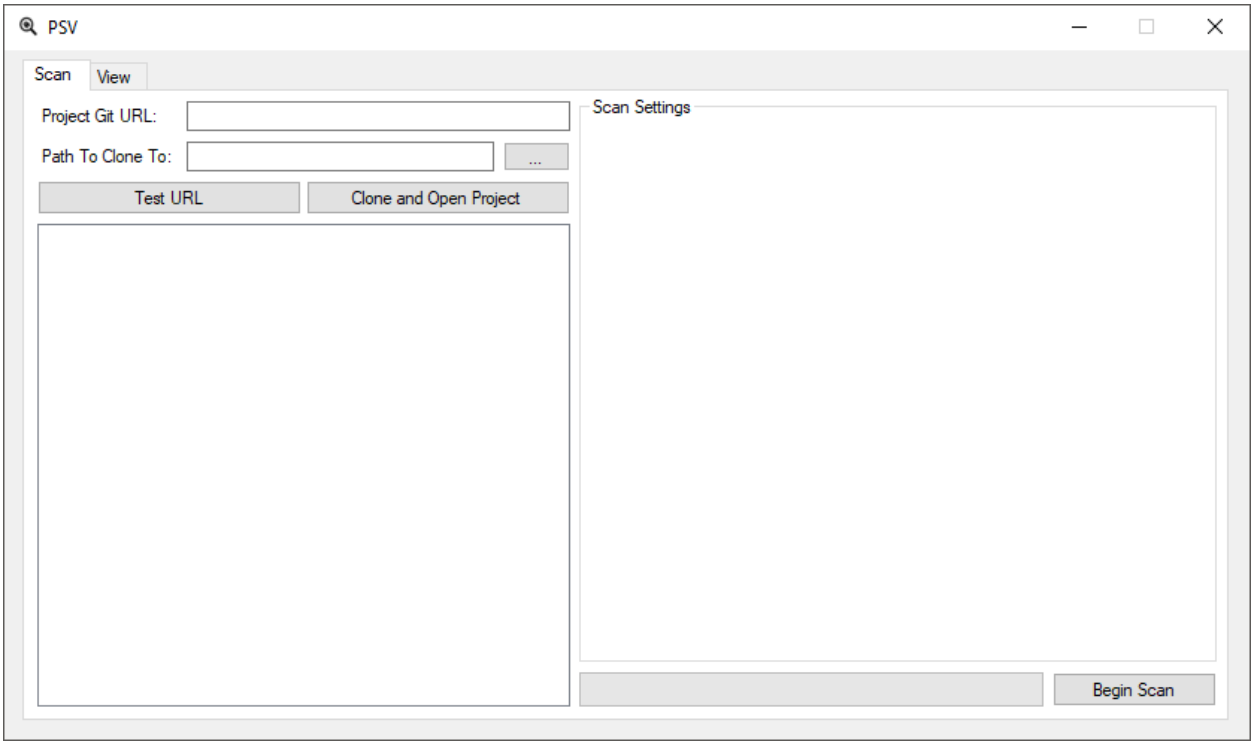


Figure 11.2: View Tab (Table) original design

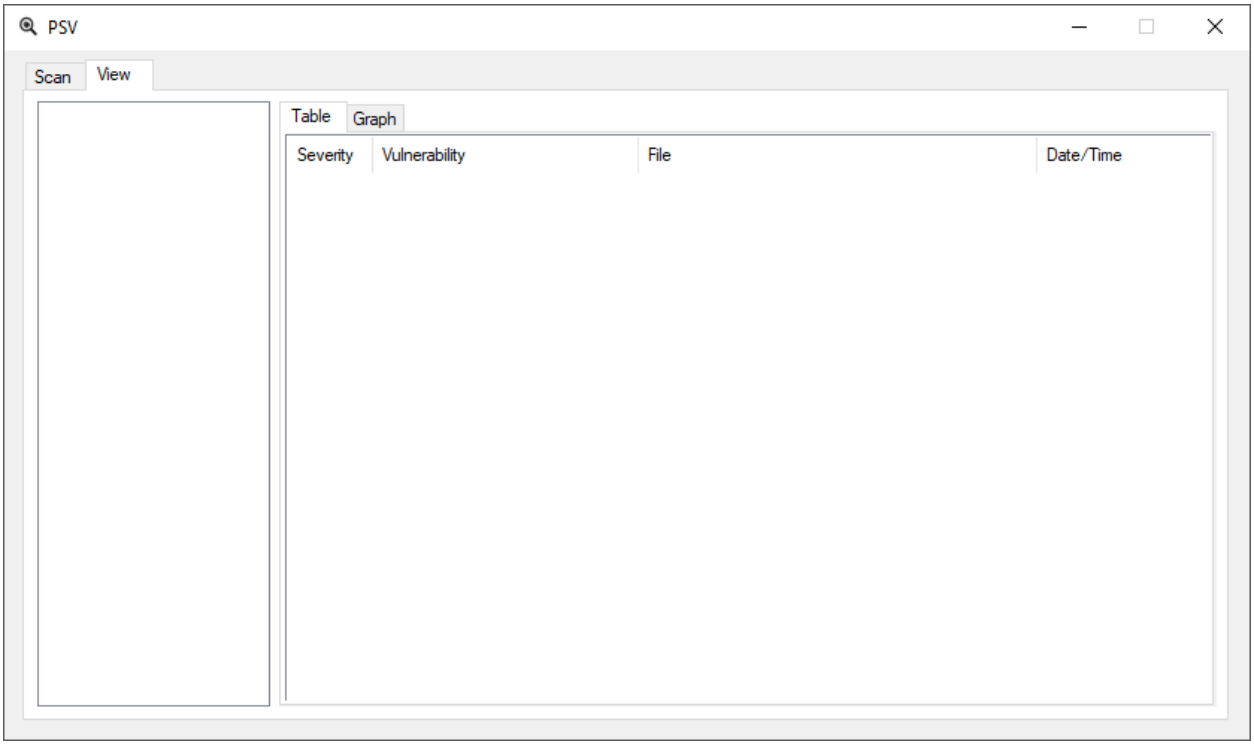
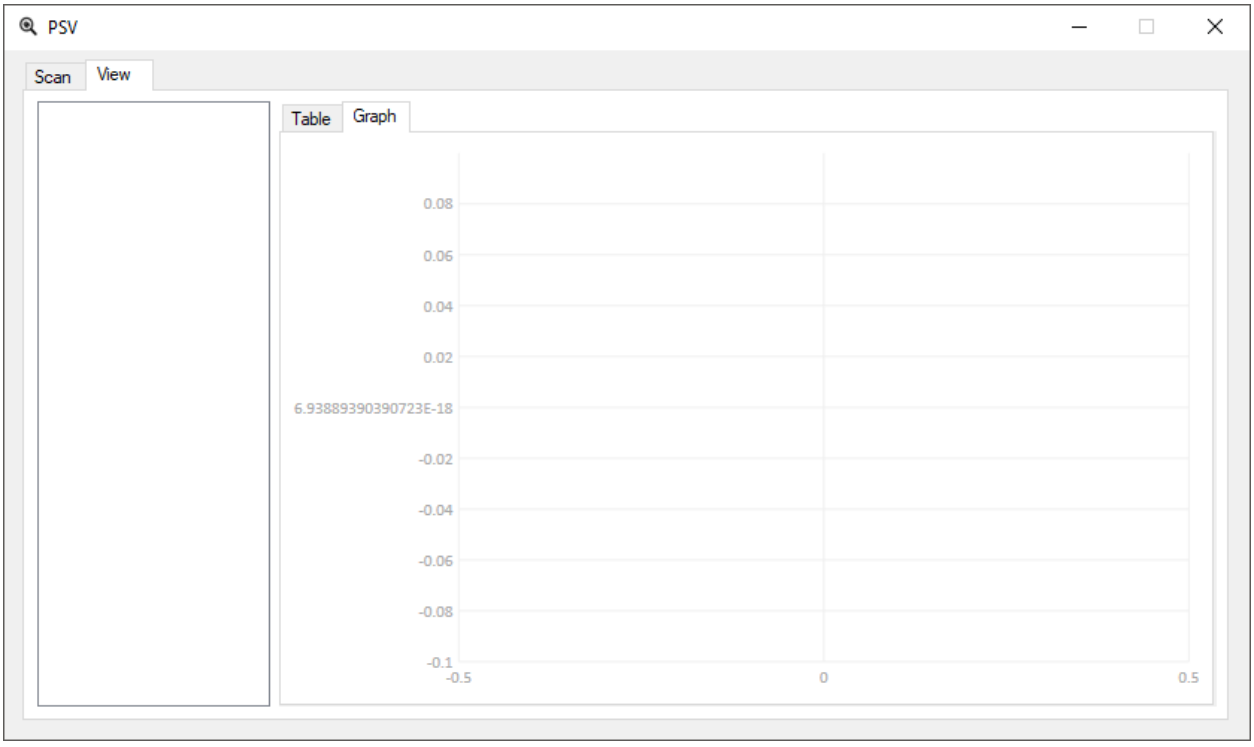


Figure 11.3: View Tab (Graph) original design



3. Research and Investigations

Ideas

- Data Aggregation:
  - Manually gather data from repositories
  - A simple script to scrape GitHub repos for data:
    - A script using an API that queries the GitHub API
    - A script using a library that condenses the API into easily accessible methods
- Data Cleaning and Format
  - Removal of erroneous or incomplete data from the overall dataset to ensure we can make an accurate model
    - Can be done with substituting missing values with dummy variables such as n/a for categorical or 0 for numerical values
    - Substitute missing numerical values with mean values of the datapoint
    - For categorical data, we can use the highest mode count.
  - Format the data in a way that is able to be loaded into our model.
  - Depending on the data in a category it could be beneficial to break data types into categories
  - Data normalization on variables that may overpower other variables
  - Change specific numerical variables into categorical values if it provides a more effective model
- Model Training:
  - The model is trained using the Python programming language. The main reason for this is that Python has numerous machine learning libraries and is currently a top language for machine learning research.
  - Regression Models:
    - Linear Regression - provides a linear relationship between a dependent variable and one or more independent variables (features) using a best fit straight line. $y = \beta_0 + \beta_1 x + \varepsilon$
      - Pros: Widely used and easy to establish
      - Cons: Not robust to outliers
    - Polynomial Regression - similar to linear regression but with a curved line. $Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_n x^n + \varepsilon$
      - Pros: sets up the model to have a lower error
      - Cons: can result in overfitting with higher degrees

        - Ridge Regression (Tikhonov regularization) - allows optimization of linear regression by minimizing the collinearity among the feature variables.
            - Pros: reduces the variance of the overall model
            - Cons: adds a small amount of bias
        - Negative Binomial Regression (Poisson Regression) - similar to linear regression but with the assumption that the response variable follows the Poisson distribution.
            - Pros: Model recommended for previous work on this type of project by Dr. Elaine J. Weyuker. Very useful for count data
- Machine Learning Algorithms
    - Continuous algorithms - These algorithms give us a continuous output which would be useful when trying to classify wherein a file the vulnerability could occur.
        - Linear Regression - This model is stated above under the regression models list. It is usually the first and most widely used machine learning technique to predict the outcomes of a dataset.
        - Neural Nets - Uses weighted points to make predictions between individual neurons of the net. Implements a black box of layers where the in-between models are not necessarily known.
            - Pros: Can train extremely complex models efficiently.
            - Cons: Resource and memory intensive.
    - Classifying/Discrete algorithms - Possibly not as useful as continuous methods but would be able to classify if a file or method contains a vulnerability.
        - Logistic regression - Performs a binary classification of the data. In our case, it could be “does this file contain a security vulnerability”. It takes linear combination features and applies a sigmoid function to it.
            - Pros: Provides excellent probabilistic interpretation and easily updatable.
        - Naive Bayes - Based on Bayes’ theorem used to classify outcomes.
            - Pros: Very easy to build and useful for vast datasets. Right choice if we have limitations on CPU and memory resources
            - Cons: Cannot learn interactions between features
        - Support Vector Machines - used for pattern recognition and classification problems.
            - Pros: High accuracy rate
            - Cons: memory intensive, difficult to interpret and tune

- Other Algorithms

    - Random Forest - Mixed bag algorithm as it can solve both regression and classification problems. It is a combination of decision trees to create an overall prediction
        - Pros: easily can handle feature interactions and are nonparametric so there is no concern for outliers or linear separability.
        - Cons: Does not support online learning which means we would need to retrain the entire dataset once new data comes in. May not be an issue at the start but would be time-consuming as the dataset grows.
- Feature Selection: At the beginning of the project, we decided to brainstorm as many metrics as we could – anything we thought that could contribute to the presence or lack of security vulnerabilities in a given program. Some of these metrics regard the program itself and others regard the programmer, and the list is in no particular order. Although these are all *possible* to use, we acknowledge that many are not necessarily *practical* to use. The point of this exercise was simply to get as many ideas, good or bad, out as possible and try to get as many data points for our machine learning algorithm as possible.
    - Age (of the programmer)
    - Age (of the program)
    - Gender
    - Native language – harder to program in English-like languages if you don't speak it
    - natively?
    - Fluency in other languages – does this help with programming?
    - Programming language of source code
    - Number of contributors
    - Number of times edited
    - Times of edits
    - Before/after meals, or time of day
    - Day, month, year, season
    - Restfulness/alertness/energy of programmer
    - Size of Project
    - Lines of code
    - Libraries used
    - Other source files
    - Other components (front end, back end, or API)
    - Bug/Issue history of the project
    - Type, location, frequency, severity
    - Education of programmer (degrees and certifications)
    - CS/IT related and not CS/IT related
    - Income/wealth of programmer

    - Job history of programmer
    - Current AND former jobs
    - Country/state/city of origin of programmer
    - Current country/state/city of residence of programmer
    - Programmer's software preferences (OS and IDE.)
    - Programmer's hardware preferences
    - Programmer's family (related to other programmers?)
    - Neurodivergence of the programmer (autism, OCD, ADD and dyslexia)
- Building the UI:
    - Using Java to build our application:
        - Search for Java libraries that provide graphing elements
    - Using C# to build our application:
        - Search for C# libraries that provides graphing elements
        - Look for cross-platform functionality
- Setting up the Server:
    - Use an AWS instance to handle our server needs (costly)
    - Use a personal server for our database (difficult to manage)
    - Paying for a separate VPS provider such as Digital Ocean or OVH
    - Using free credit provided to students to fund a VPS
- IDE plugin once the product is functional and mature
    - Streamline functionality
    - Results to indicate what line, region, method call is problematic
    - Submit data to backend, query model, return results
- Backend automation of data collection
    - Shell scripts can be configured to run the Git API commands or python scripts necessary to pull information about a new project or update the database about changes
- "Best practices" guide for developers once model matures
    - Coding archetypes that are known to be problematic or cause issues so that developers can adjust their coding style to be more resilient from the start
    - Part of the User Guide
- Assembly parser
    - Predictions about running code could be made into a sort of "anti-virus" daemon which can give the user real-time security notifications if a potential high-risk fault is detected
    - Could tie into DEP (Data Execution Prevention) on Windows platform

Efforts Underway

Data Acquisition

Starting in February PyCharm and Python's virtual environment pipenv [11, 12] were chosen to write our modules and scripts. This integrated development environment was chosen over Visual Studio Code which is a full-featured editor, but not ideal for debugging and doing what a lot of IDEs have to offer. A personal access token was required from GitHub to access APIs, and it needed to be stored somewhere safe as to keep its legitimacy. The current token exists under a python file called token_file.py and in a variable called access_token. Next, a log file was created to identify all the possible fields of interest to gather later once we begin collecting the data.

We began the process of creating our python script, researching libraries we need and set up the basic outline of the program. As we learned new things and wanted different things the script would continue to get updated, we also went ahead and created code to connect data to MySQL database, all it did at this point was connect to the database, if it didn't connect it would display an error as to why it didn't connect. During the entire month, we focused heavily on setting up the code we needed for our program to run and also finding more features that could be useful data. We came up with our process diagram which helped show us what we needed to do to get things done. We found a way to test the data we would collect on a database using the wampserver which allowed us to connect to the local database we created for testing purposes without messing with our actual database.

By mid-February, we started to get reliable data, so we thought at least, and we continued to pull this data from GitHub repositories. The primary data we currently had was integer data and DateTime data. During this time we made a README file which explained in detail how to use our program and all the libraries we needed. Pipenv is a python virtual environment that would allow us to contain all our dependencies in one location without having them downloaded for the entire system. By the end of February, the initial program was complete, and much of the focus was done on understanding databases and working with the wampserver to create dummy databases for testing purposes, all while updating code and the README file.

(2/11/19):

- Look into pipenv and PyCharm.
- Get a personal access token from GitHub to access its API.
- Get API documentation and search through Python libraries to use.

(2/3/19):

- Create a log file of possible features to use

- Update python script as needed to remove or add new features
- Create code for connecting data to MySQL database

(2/18/19):

- Find more features that could be useful data
- Find a way to make a test database to store the data on
- Come up with a process diagram

(2/20/19):

- Pull more data from a repository
- Integer value data over string data for now
- Create a readme file for py file
- Organize a project with pipenv

(2/25/19):

- Learn more on connecting to databases
- Create a proxy database for testing connections
- Clean up the readme file

(2/27/19)

- Use Wampserver to create a test database
- Figure out how to populate the database

In March, we hit a snag in the type of data we had collected. We had full access to the official database; however, after having several meetings and talking with our Machine Learning members, we decided we needed more data. The data had to be specific to individual files not just random data from GitHub repositories. Knowing this we spent several weeks looking for the type of data they would need this included number of lines, commit history on lines deleted and added as well as the number of files changed and type of language used for them.

We were able to get file data through a python GitHub library and found a website that would tell us about the individual files in a repository which included the language type and line count of each file. There is a way in the Linux terminal to get the line count from files; however, implementing that method in python was challenging and wouldn't work out in the end. We did a few test methods on the website to pull that information however that didn't return anything useful, so we decided hard coding said data would be an option and running the commands we needed separately from the python program.

By the end of March to the beginning of April, we started to get the data we needed and using another python library we were able to get file data through the commit history of any GitHub repository. We downloaded/cloned the repos to a local file and then created our program to run on those project folders. We could run directly through GitHub, however, as the projects get larger so does the time it takes to get the data from them. We updated the README files and separated the

methods we had into repository data, file data, and database connection; this was done to simplify our program and make it easy for debugging and testing purposes.

(3/1/19):

- Get Access to the official database
- Check new libraries for ways to get better data
- Get better data

(3/4/19):

- Found a way to get file data from GitHub repo
- Found website that also does it but in a bit more detail
- Find a way to implement and store data using python

(3/6/19):

- Test methods of pulling data from Linux terminal
- Test methods of pulling search results from the website
- If successful, find a way to store the data

(3/11/19):

- Create variables for hard coded data
- Get data from website

(3/20/19):

- Data mostly useless
- Group meeting, discussed data required
- Found a way to access new data
- Create modules to pull data through Git

(3/25/19):

- configure file collecting script
- Find a way to get file data
- Find a way to get projects

(3/27/19:

- Download multiple projects to determine the usefulness

(4/1/19):

- Update Design Document
- Pull more projects and check for CVE mentions in commits
- Connecting the scripts

Early to mid-April has seen much refining of the original program as we try to narrow down what data we have and what data we have left to not only pull but parse. We currently have 85% of all the data we need to set up and ready to be sent over to the database. The 15% left have to deal with parsing the DateTime data and still working to link the line count method and file count method so that they can work to pull each file to count from a project while reading through the file and storing the count value as it goes along. We currently can get a line count for a given file however the file count method produces lists of file names from each directory it searches.

(4/8/19):

- Clean up program file
- Get the line count method working

(4/12/19):

- Work on file count method
- Add TODO for assistance in method creation
- Prep files for addition to GitHub for team collab

Mid to the end of April we will be focused on finalizing any remaining methods that need to be created. We will also begin connecting data points to the database itself and finalize and finish up any TODOs that still needed to be done.

(4/15/19):

- Finish writing up TODOs
- Polish up readme file
- Push to GitHub page

(4/16/19):

- Work on doc and methods
- Read over program

(4/17/19):

- Test individual methods for correct results

(4/17/19):

- Download more projects related to cve for collection

Nearing the end of the Project, late August through early September we realized we needed better data, the current data we were collection wasn't specific enough towards showing which files might have security faults based on the meta-data. We drew up a new design that would focus more on gathering specific file data based on data parsed from a csv file containing github repositories that had files we know had security faults. With this new data we created new methods that would allow us to focus down the data we collected to be far more accurate.the CSV data we received included a hash that coincided with the file containing the security fault so with that information we setup our file database to so it would contain the one bad file we knew was bad from a repository and then a "good" file as well.

(10/21/19):

- Work with team to redesign backend methods for data collection
- Brainstorm new features
- Create branches to work on new methods

(11/06/19):

- Continue work on methods
- Test new methods for collecting averages of each file

(11/17/19):

- Test individual methods for correct results

(11/27/19):

- Run new full test on new file features using GitHub

repositories from csv file
detailing specific files with
security fauls.

Data Encoding

The data retrieved from the GitHub APIs needs to be encoded in a way that we can use for our model. Each of the data points retrieved is its designated feature in the neural network. The features being used are either numeric or categorical.

Numerical variables (or quantitative variables) are variables that can be measured or counted and have a corresponding number value to them. This data type can be broken down further into discrete or continuous variables. Discrete variables are ones that can be counted but not measured or broken down into smaller components and remain as part of the count. For example, the number of methods in a file, or the number of times a specific function is called. It would be impossible to break down this data into smaller components. On the other side, continuous variables are ones that can be measured and can not be counted. Examples of this type of data would include the time of day that a commit was added into the commit history. In our database, the discrete numerical variables are stored as integers, and the continuous numerical variables are stored as floats.

Categorical variables are variables that represent the characteristics of the data. This data type can also be broken down further into either nominal or ordinal data. Nominal variables are ones that represent individual values of the data but have no quantitative value to themselves. They have no order to them; they are just values that represent the data. In the case of our project, these types of values would include the programming language used or the author of the commit. Ordinal variables are similar to nominal variables in that they are a non-numeric way to represent a specific data type but have an ordering to them. For example, if the GitHub API stored information on the level of skill the programmer who created the commit, such as Senior, Junior, Entry-level, these would be considered ordinal variables.

For the project to build a successful model using a neural network, it needs to have all features encoded as numerical values (integers or floats). Encoding is a simple process from the data we receive from the GitHub API that is already numeric as we can store them in their predefined data type. For categorical data, it needs to be modified in a way that can be represented as numerical data. [13]

The process of changing categorical data into numerical data can be broken down into two different processes. The first one being integer encoding and the second called one-hot encoding.
Integer encoding is a straightforward process that mirrors the logic behind ordinal data. It involves assigning an integer value to the categorical value. The encoding is a straightforward process and is easily reversible as long as the values are consistent among multiple different features. For example, if we were to encode different levels of programming skill into the features, we could put label it as follows: [14]

0 - Trainee/Intern Programmer
1 - Junior Programmer
2 - Middle Programmer
3 - Senior Programmer

This type of encoding would be fine for the neural network because it justifies the average competence of the programmer that submits the commit. However, this is not going to be the best encoding process because our data does not contain any variables that can be ordered by significance. We would not want to use this encoding for something like the type of programming language used because we can not put a value on the order of the language. Personal preference aside, we can not objectively state that one programming language is better than another and therefore integer encoding is not the way to go with our categorical data.

The other process of encoding categorical data is called One-Hot Encoding. Since we don't want to include an ordinal pattern to categorical data, we can resolve this by doing a binary representation for each of our categorical data. This process is done in a matrix of the data type as the feature set where the number one (1) represents the value is active, and zero (0) represents that the value is not active.

Below is an example of a subset of possible programming languages:

Java	Python	C
1	0	0
0	1	0
0	0	1

In this table, we can map whether or not a language is being used but not putting a higher numbering system on languages higher in the list. Representing the data this way prevents the neural network from putting an incorrect emphasis on the language type that happens to be encoded as a higher integer as we would with integer encoding. [15, 16]

We have two sources for our data collection. One is from the GitHub API that pulls information regarding at the repository and commit level. Due to the unique problem we are faced with to determine the file at which the security vulnerability occurs, we need to create additional methods in which to gain features about the file.

Below are the proposed mappings for the variables retrieved from the GitHub API:

- Stargazers & Watchers(count): Discrete numerical variable - no change required
- Subscribers(count): Discrete numerical variable - no change required
- Branches(count): Discrete numerical variable - no change required
- Assignees(count): Discrete numerical variable - no change required
- Commits(date & time): Continuous numerical variables but converted if value comes in string format.
- Commits(count): Discrete numerical variable - no change required
- Forks(count): Discrete numerical variable - no change required
- Events(types): Categorical variable, encode using one-hot encoding for types of events
- Events(count): Discrete numerical variable - no change required
- Repo date created(date & time): Continuous numerical variables but converted if the value comes in string format.
- Date repository pushed(date & time): Continuous numerical variables but converted if the value comes in string format.
- Issues(count): Discrete numerical variable - no change required
- Contributors(count): Discrete numerical variable - no change required
- Languages (types): Categorical variable, encode using one-hot encoding for types of languages.
- Languages (count): Discrete numerical variable - no change required
- Collaborators(count): Discrete numerical variable - no change required
- Labels(count): Discrete numerical variable - no change required
- Pulls(count): Discrete numerical variable - no change required
- Network(count): Discrete numerical variable - no change required
- Open_issues (count): Discrete numerical variable - no change required
- Date repository updated(date & time): Continuous numerical variables but converted if the value comes in string format.

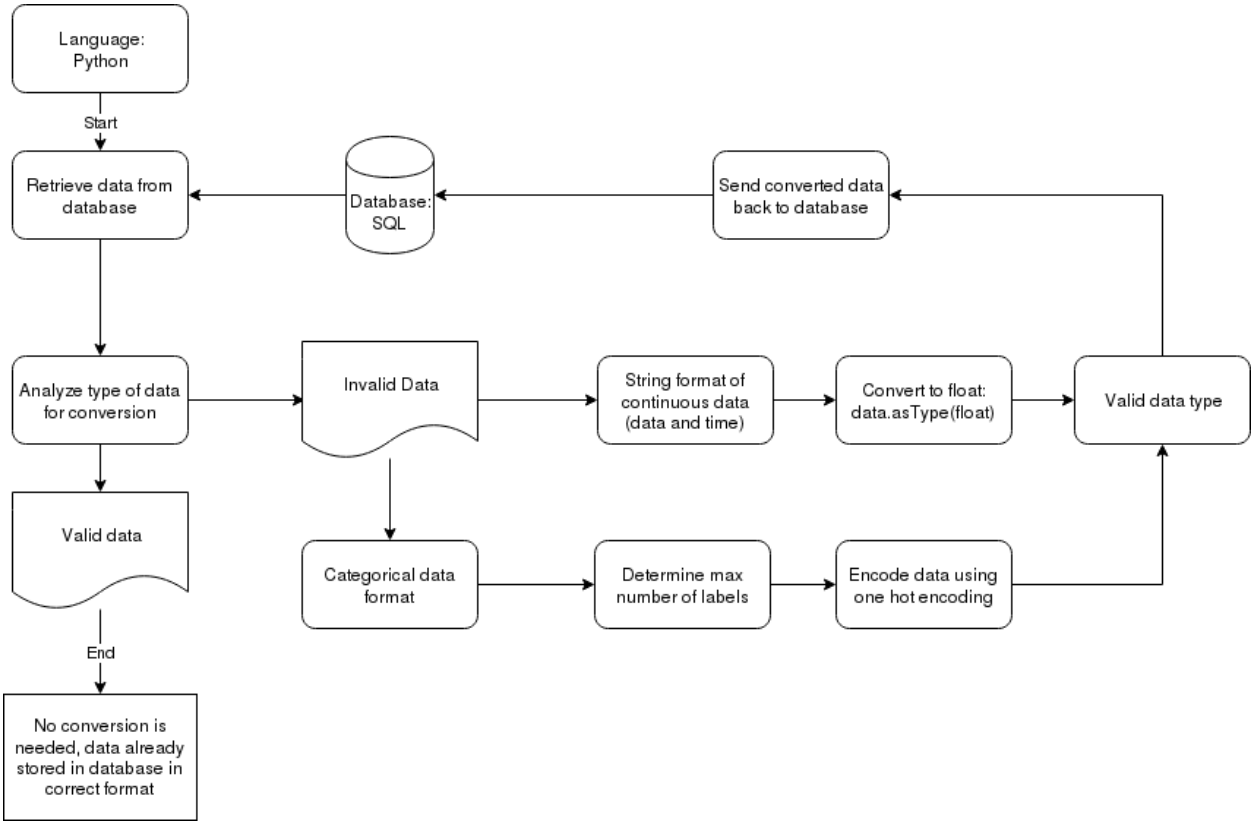
Below are the proposed mappings for the variables in our methods:

- Lines of code: Discrete numerical variable - no change required
- Language type: Categorical variable, encode using one-hot encoding for types of languages.
- Number of commits by the hour: Discrete numerical variable - no change required
- Number of commits by the day of week: Discrete numerical variable - no change required
- Number of commits by the day of year: Discrete numerical variable - no change required
- The average number of line adds per commit: Continuous numerical variable - no change required
- The average number of line deletes per commit: Continuous numerical variable - no change required

At this point, we are not sure if all of these data points are valuable to the model. The intention is to go through different architectures utilizing all of the features of above and find the one with the most accuracy.

Further, if the data is not in the correct format, it needs to be ‘massaged’ and stored appropriately so that the training can be performed straightforwardly. The diagram below shows this process.

Figure 12: Categorical Data Conversion Diagram



Analysis of feasible modeling options

Machine learning is a big field at the moment, and there are many different approaches to implement machine learning techniques with various languages, libraries, or models. With all the possibilities open to us, we decided to narrow our focus to the choices that were simple and reliable. As much as we would love to experiment with machine learning, we have a strict schedule to maintain, and we can't risk spending so much time figuring out the tools that we don't get around to building anything with them.

On the subject of programming languages, Prolog and Lisp have been used to develop and research artificial intelligence for decades, but using them for this project would be overkill, for both the power they're intended for, and the time it would take to learn them. R is an excellent language for processing large amounts of data, and some of us are familiar with it from our Programming Languages class. However, while it does have its share of machine learning libraries, we don't think that they're as powerful or as intuitive as Python libraries. Python is a language that is very easy to learn and understand, with a lot of well-developed (and, importantly, well-documented) machine learning libraries.

Python has many machine learning libraries. Our initial choice was scikit-learn, because it's intended to be used by beginners, and as we've said before we entered this project with no real knowledge of machine learning. However, we quickly realized that its simplicity was going to be very limiting for our project. Scikit-learn isn't very flexible in certain areas like neural networks, it doesn't scale well for large projects, and it doesn't support any GPU-based machine learning implementations. Additionally, TensorFlow was always the end goal, and although translating between the two is supposedly easy, cutting out that step entirely made more sense.

TensorFlow is easily the most popular and most well-documented machine learning library available for Python. The functionality it provides allows us to quickly and easily create and alter machine learning models. TensorFlow is also well-suited to neural networks through the manipulation of tensors, and it was initially designed for neural network development. Keras is an additional neural network library that can run on top of TensorFlow, among other libraries, and it makes the construction of neural networks even easier and faster for developers, with a high degree of modularity. Using both of these libraries in conjunction with each other is the best choice for using machine learning – and especially neural networks – with Python.

Initially, we intended for our primary machine learning model to be linear regression. The idea was born out of our desire to keep things simple, and linear regression is about as simple as machine learning models get. We intended for a neural network model to be more of a "stretch goal," but after further consideration, we decided to make it our primary model.

For one, after meeting with Dr. Weyuker, we learned that her team had used a linear regression model. Of course, she didn't use machine learning, and the calculations were done by hand, but we wanted to go beyond merely replicating her work. Additionally, though training a neural network takes more time than simple linear regression, the kind of robust pattern recognition that neural networks bring would be, in our opinions, well-suited to this project. If time permits, we may also investigate and use other machine learning models we initially considered and compare results to determine which of them, if any, are better suited for the project.

Why not static analysis:

Static analysis - analyzing a program without executing it, such as by analyzing the source code itself - is a common way to detect bugs and security vulnerabilities, and there have even been projects that used machine learning for static analysis. In this project, we intend to avoid using static analysis as much as possible, least of all because that is what our sponsor would prefer. Firstly, the solution we intend to implement is language-agnostic, whereas if it were based on the source code, we would potentially need to modify our model for each language, which would waste time and resources. Additionally, we believe by looking at the metrics of the code rather than the code itself; our models will in a sense be able to look at the actual development of the code rather than the end product. We believe this gives us a clearer picture of the code and increase our models' effectiveness by giving them more and better data points.

Two Models

The machine learning portion of this project has two components: the overall model and the granular model. The overall model essentially replicates Dr. Elaine Weyuker's work at AT&T. It reads the metrics of a particular file or group of files and their contributor or contributors and determines which of them, if any, have security vulnerabilities. The granular model does mostly the same thing on a smaller scale. It analyzes a single file and determines on which line or lines security vulnerabilities could be found.

We decided to split the machine learning portion for a few different reasons. Firstly, we do not think that the development of the overall model would require two people working on it, and we wanted to divide the workload.

The second reason is to compare and contrast the approaches. Dr. Weyuker's work is roughly equivalent to the overall model, and so we'd like to see if the granular approach could be just as good if not better than that.

Lastly, there's the possibility that we could combine both models to create an even better and more accurate system. The overall model could predict which files are likely to have security vulnerabilities and then the granular model could take those files and identify the precise problem areas in them. This integration would be especially helpful for users who are analyzing larger projects with many lines of code, saving them from wasting time having to analyze each line of code themselves.

Overall:

The overall model is used to determine if a particular file or group of files have any security vulnerabilities.

Granular:

The granular model was to operate within a single file and determine which sections of code, if any, have security vulnerabilities in them. Like the overall model, this would also classify any security vulnerabilities the model finds. This model would use most, if not all, of the same metrics of the overall model.

As mentioned previously, we intended to avoid static analysis, and so information within the files themselves - which the overall model does not consider either - can not be added as additional metrics. In order to identify where in the code the error is, the granular model would likely have to use a line number or similar metrics. Since we also wanted to identify not only the location but the *type* of security vulnerability, it may have been difficult if not impossible to do so without using some amount of static analysis since there simply might not be any way to correlate location and type without such information. For example, if a file has multiple vulnerabilities, how can the model tell where each one is without looking at the code? Training the model to classify on line number alone would not be a reliable statistic.

The granular model would use a feed-forward neural network rather than a feedback neural network. In a feedback network, save the output of a layer of the design and re-input it to a previous layer. This type of neural network can be powerful, but it is also computationally expensive, and the code can become complicated and hard to follow. In a feed-forward network, on the other hand, data flows directly from the input layer to the output layer without any looping whatsoever. Not only is this much simpler and more direct; this type of neural network is well-suited to mapping problems: determining how several inputs may affect the output.

Because this project is purely a mapping problem, a feed-forward network is the most obvious choice for our design.

Around late-August or early-September of this year, the granular model was abandoned. It was always intended to be a stretch goal, something that would be great to have but not necessary for the project to succeed, and around this time, the team was having problems processing input data, and so that task was prioritized. The project could survive without the granular, but without the pipeline working with the overall model, we would have nothing.

In retrospect, the granular model is also simply outside the scope of this project. The overall model was intended to only look at metadata about repositories, and for the granular model to look into the files themselves to extract enough information, it would require adding many new features, and given our pipeline, that would have multiplied the work required for everyone, from parsing that feature from a file to adding it to the database.

The granular model would also have represented a much more difficult problem than the overall, and it would've had far more unknowns due to its greater complexity. Analyzing a file in-depth instead of just looking at related statistics would require a much more complex algorithm and likely a more specialized machine learning model. Even choosing what kind of features to use in the machine learning model could be a challenge: analyzing indentation may be vital for predicting vulnerabilities in a language like Python, but could be useless in a language where indentation is irrelevant or non-existent. It's possible that, at least with the time and resources we had for this project, it wouldn't have been practical to create a generalized solution that would be as language-agnostic as possible.

In the end, the granular model was a necessary sacrifice and it proved to be a useful experience. With the overall model's pipeline complete, it should not be too difficult to modify it for a granular model if this project is continued. Lastly, working on data acquisition helped give me a better idea of what the granular model would need to be successful if I were to return to working on it.

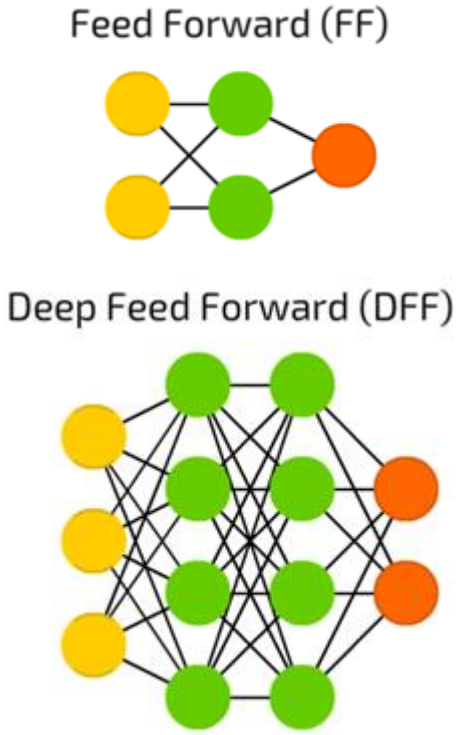
Types of Neural Networks

There are many different types of neural networks, each with advantages and tradeoffs that often cause them to perform better at training and classifying certain types of datasets over other types of datasets. Which choice of neural network the overall and granular models run on can make a huge difference, and so, while we wait for the data acquisition team to create our datasets, we are researching the different types and we are testing them on placeholder datasets to get a sense of which ones could work better, but we may need to reevaluate our choice after testing the models on our actual data.

Firstly, neural networks can be supervised, unsupervised, or semi-supervised. Supervised learning features fully labeled training data, unsupervised has entirely unlabeled training data, and semi-supervised has some labeled training data. Without labeled data to provide feedback, the machine learning model examines what the training sets have in common and what is different about them to learn to classify them.

Unsupervised learning requires less work on our part, but it tends to become a “black box”. Since this type of training learns entirely on its own, it’s often difficult if not impossible for the programmer to determine how exactly the model makes its decisions, which makes it harder to refine because you cannot be sure why it’s getting the prediction right or wrong. We could later experiment with other types of learning, but to begin with, we stick with supervised learning.

There are several specific types of neural networks that we are considering for this project:

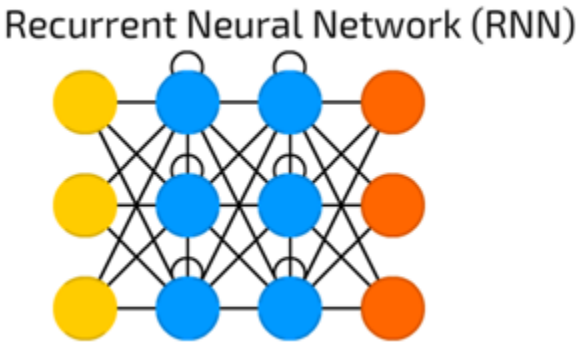


From Fjodor van Veen, asmiovinstitute.org

A feedforward neural network is a simple and common type of neural network. It’s called feedforward network because information flows forward, from the input nodes to the output nodes, and through any hidden nodes that there may be. Deep feedforward networks are just FF networks with multiple hidden layers.

One thing that makes deep feedforward networks so useful is the use of backpropagation. Like other parts of the design of neural networks, backpropagation is based on the behavior of the neurons within organisms. With backpropagation, the output layer generates an error which is then sent through the neural network's layers backward, adjusting the weights to make it more accurate. The Keras library performs backpropagation automatically, which makes the neural networks we can create with it even more powerful.

Additionally, backpropagation is where loss functions come in. A loss function is a function that takes some set of variables and maps it to a real number representing the cost associated with that set. In the case of neural networks, a loss function calculates the difference between the actual output and the expected output of the network. The Keras library provides many common loss functions that we can use with our neural networks. Some loss functions measure the magnitude of the error without regard to direction, which is to say whether or not the model has a positive or negative bias, and some are more suited to continuous or discrete variables. With loss functions, there aren't any right or wrong answers, but some options are more or less suited to the data and could have better or worse results, and choosing the best option is at least partly a product of experimenting once we have the datasets in place.



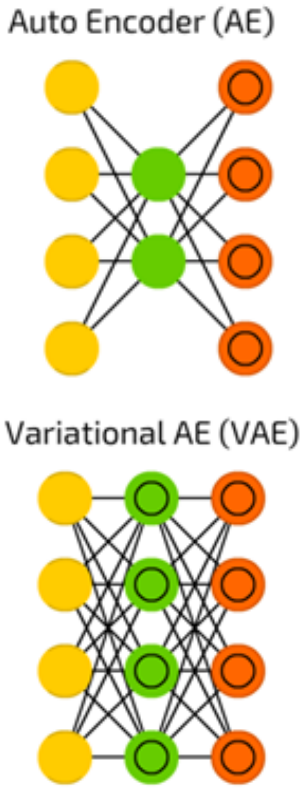
From Fjodor van Veen, asmiovinstitute.org

A recurrent neural network, unlike a feedforward network, can feed data backward as well as forwards. As you can see in the image above, the hidden nodes are receiving their outputs as an additional input. The intuition behind the recurrence is that in real life, humans base our understanding of current events is based upon previous events –to understand a sentence, you need to understand every word in the order they are in, not just the individual words themselves.

This type of neural network could be useful for our project, especially the granular model, as recurrent neural networks are better at analyzing context. While we haven't finalized our designs for the granular model yet, it could very well involve analyzing the source code itself and the context in which individual sections appear to determine where the security vulnerabilities are.

Indeed, in practice, recurrent neural networks are often used for tasks that involve processing and understanding languages.

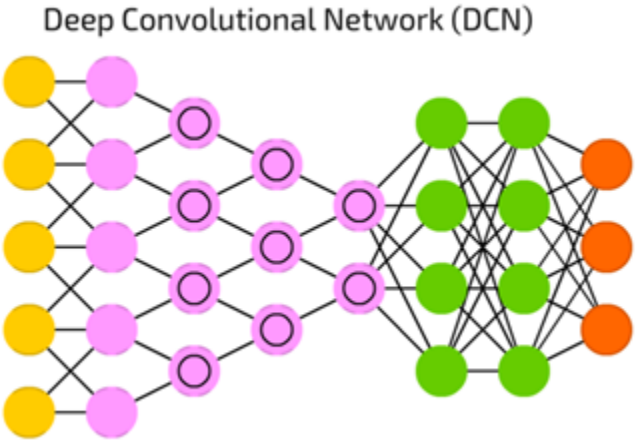
However, this type of neural network is much more complex and challenging to design, even with TensorFlow and Keras libraries doing most of the heavy lifting. Recurrent neural networks are also more resource intensive, though we have access to powerful machines in the Senior Design Lab. RNNs are an option that we should seriously consider, but in the end, they may be too difficult to implement for this project.



From Fjodor van Veen, asmiovinstitute.org

Auto-Encoders and Variational Auto-Encoders are two simple types of unsupervised neural networks. As we discussed above, we are unsure as to whether we will use unsupervised learning, but we do intend to at least experiment with the simpler models such as these if only to test every possible avenue of solving the problem.

Auto-Encoders are trained by having to compress the input on one end, and the decompress successfully on the other. The compression forces the network to learn to ignore the noise and focus on the parts of the input that matters the most. Auto-Encoders compress features, whereas Variational AEs compress probabilities. Other types of Auto-Encoders exist, but unless these two basic ones show promise, we will likely not explore them.



From Fjodor van Veen, asmiovinstitute.org

Convolutional Networks and especially Deep Convolutional Networks are a type of neural network that has proven to be very ubiquitous in machine learning research in recent years. The prevalence of this type of network is in large part because they are used to process image data, such as classifying images or identifying subjects in them. During the forward pass, the network's convolution layers learn which of their filters activate when some feature exists in the input. They are much better at processing images than feedforward neural networks.

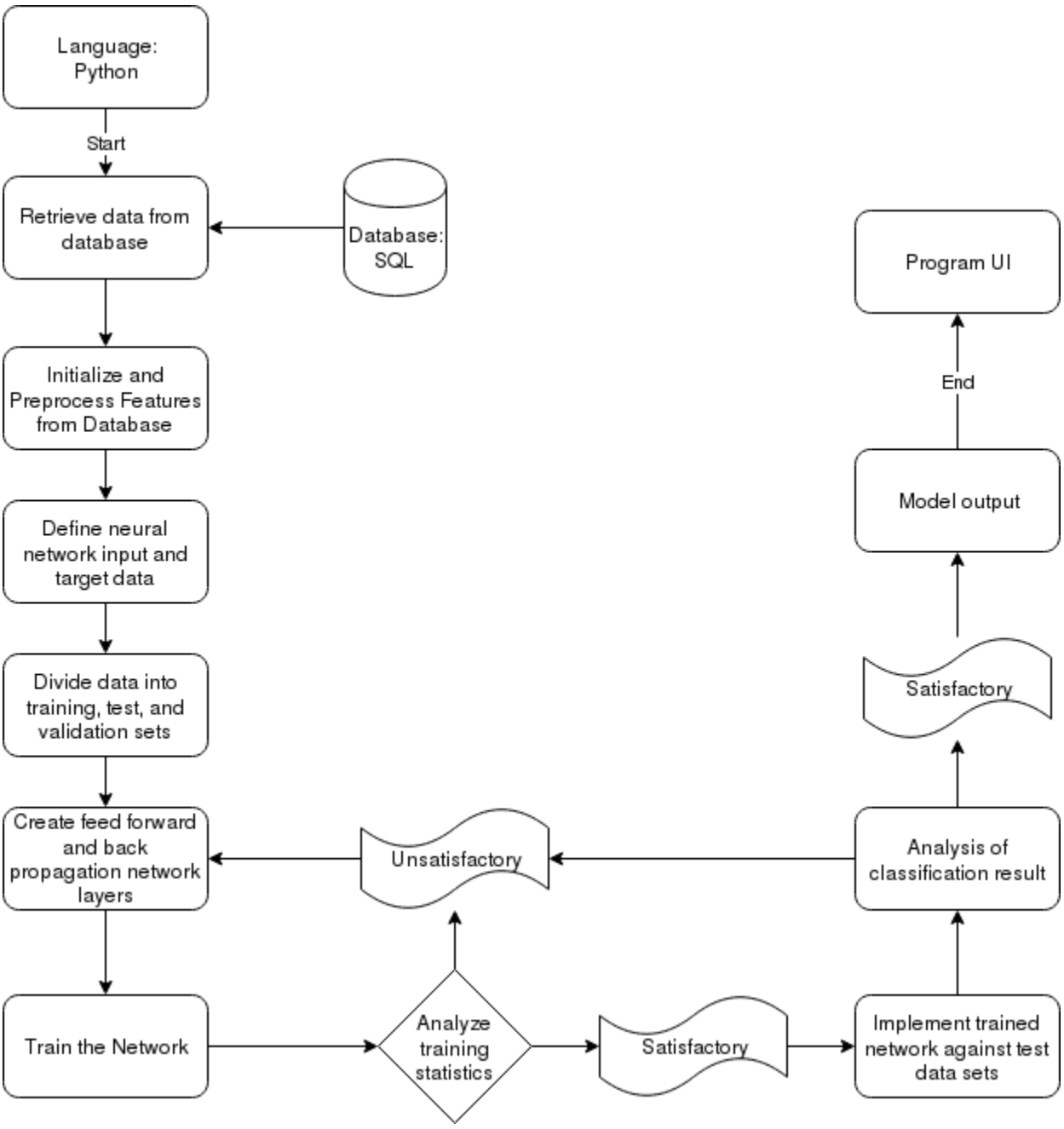
Like Recurrent Neural Networks, Convolutional Networks are sometimes used for natural language processing, as they can analyze the order of word spatially. This analysis means that a convolutional network could work well in the granular model if analyzing the code itself in some way, especially since CNNs lend themselves to classification problems such as identifying security vulnerabilities.

Work So Far:

To ensure that the machine learning portion is ready to be implemented once our datasets are in place, we've been spending time learning and experimenting with our machine learning libraries so that we know how to use them and can quickly iterate on the models once the time comes.

The coding section contains some segments of the code we've developed so far, some of which were derived from online tutorials. There we also explain what each section of code does, what we've learned from it, and how we can put those lessons into practice for the machine learning portion of this project. The below neural network flowchart shows what the design of the neural net supporting framework looks like in our project.

Figure 13: Neural Network Flowchart



Feature Scaling

Feature scaling (also known as normalization) [17] is the process of standardizing data so that more significant valued features do not overpower smaller valued ones. For example, if we look at the two features, the number of authors on a file and the number of lines in a file, obviously the number of lines would be considerably larger in value and would potentially weigh more in our model and skew the accuracy. For this reason, we need to set the magnitude of all features to be within the same level.

An additional bonus to feature scaling is that it speeds up the process of gradient descent. This speedup is essential to neural networks with many data points because the change in weights can happen quickly with smaller weights than it can with larger ones.

There are four major types of feature scaling: Min-Max Scaling, Standardization, Mean Normalization, and Unit Vector. [18]

Min-Max Scaling

Min-Max Scaling is the simplest method of scaling since you only need to have the minimum and maximum feature values to perform the operation. The value is normalized by subtracting the minimum feature value from the value being changed and dividing by the maximum feature value subtracted by the minimum feature value. The formula is as follows where X is the original value and X' is the updated value.

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

This scaling technique brings feature values into the range of [0,1]. [19]

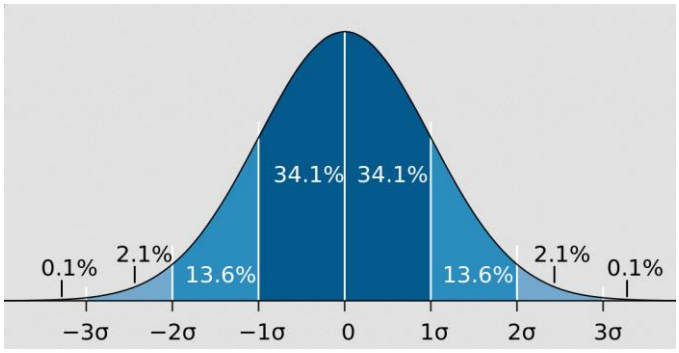
Standardization

Standardization takes the values and replaces them with their z-scores. This process is done by subtracting the mean of the feature from the value and dividing by the standard deviation for the feature. Standardization gives a normal distribution of the data about its mean value set at zero ($\mu = 0$) with a standard deviation of one ($\sigma = 1$). The formula for standardization is as follows:

$$z = \frac{x - \mu}{\sigma}$$

This scaling technique has the feature values between $(-\infty, \infty)$ with a bell curve around the mean of zero which means that 99% of the data

should be within three standard deviations of zero (−3,3). Below is an example of this distribution.



This process can be found in the sklearn.preprocessing library with the method called scale and is the method used for the neural network for training the overall model. [17, 20]

Mean Normalization

Mean normalization is similar to standardization but using the minimum feature value and maximum feature value in the denominator of the equation. The mean value is $\mu = 0$. The value is normalized by taking the original value and subtracting the mean value of the feature and dividing by the maximum feature value subtract the minimum feature value. The formula for mean normalization is as follows:

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

This scaling technique has the feature values between (-1, 1).

Unit Vector

Unit vector scaling is done by using the feature vector length to which the value is normalized. The normalized value is found by dividing the value by the Euclidean length of the vector. The formula for unit vector scaling is as follows:

$$x' = \frac{x}{||x||}$$

This scaling technique brings feature values into the range of [0,1]. [19]

Activation Methods

Activation functions of neurons in a neural network define the output of the neuron when given a set of inputs. This output is then used as the input to the next layer of neurons until the desired solution to the problem is found. Since each layer will have its own activation function, the activation functions are not required to be the same. In fact, a combination of activation functions is used in our model as they each have their own characteristics that provide a benefit to the overall network. Keras provides numerous activation functions available for use. In this section, the most popular activations will be briefly discussed and determined if they will be chosen for our models.[21][22][23]

Linear

The linear activation function is a straight line function where the activation is proportional to the input. This function is not beneficial to a neural network because the gradient has no relationship with the features since the gradient is constant during backpropagation. Due to the constant gradient, this function creates a chain of linear functions which in itself is also a linear function. By using this function in a neural network, we lose the ability to add stack layers.[24][25]

$$R(z, m) = \{ z * m \}$$

Sigmoid

The sigmoid activation takes a real value input and outputs a value between zero and one. This activation is nonlinear and has a smooth gradient which allows for it to be an excellent activation function for classifying data. A drawback for this function is that outputs that are close to either zero or one tend to respond very little to the changes in backpropagation with causes training to be very slow. In the overall model, the sigmoid activation will be the last layer of the network and will classify if a file contains or does not contain a security fault.[24][25]

$$S(z) = \frac{1}{1 + e^{-z}}$$

Tanh

The Tanh activation function is similar to sigmoid except that the outputs are in the range negative one and one. The values of the function are centered around zero, and the function is nonlinear

which means that it can be used to stack multiple layers on top of each other. The gradient in this function is also much stronger than in the sigmoid activation.[24][25]

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Relu

The Rectified Linear Units function gives the output of x if x is positive or zero otherwise. This activation function avoids and corrects the vanishing gradient problem and is less computationally expensive when compared to tanh and sigmoid. Relu has a limitation that it should only be used in the hidden layers and suffers from the dying ReLu problem. This issue is when activations in the region where x < 0, the gradient will be zero causing the neuron to no longer become activated. Even with the problem, this activation is one of the most popular in neural network training.[24][25]

$$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$$

ELU

The Exponential Linear Unit activation function is a function that tends to converge the loss to zero quickly and produce accurate results. Unlike other activation functions, ELU has an extra alpha constant that allows the function to become smooth slowly until the out is equal to negative alpha. This activation function is similar to the RELU activation but allows for negative outputs. [24]

$$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha.(e^z - 1) & z <= 0 \end{Bmatrix}$$

Softmax

The softmax activation is a function that calculates the probabilities over n different classifiers. This function is generally used when multiclass activation is needed for determining which output is the result. In our model, we are only doing a binary classification, so this activation method is not needed. [24]

Optimization

Gradient Descent

Gradient descent is one of the most important and influential optimization algorithms. Gradient descent simply calculates how a small change to each individual weight would affect the loss function. Then, it adjusts the weights of the model according to their gradients, essentially a partial

derivative that measure change, connecting the weights and the loss function, telling us what changes to make. It then repeats the process.

The algorithm aims to get the loss function to be as low as possible, and it is programmed to stop after a certain number of iterations have passed without the loss function lowering, at which point the training is said to have "converged". One potential problem with gradient descent is that it could find a local minimum in the graph and confuse it for a global minimum, or it may not converge at a minimum at all. In order to overcome this, we use a *learning rate* variable to essentially scale the gradients and ensure the "steps" taken are neither too large nor too small. The learning rate can be difficult to fine-tune, especially since the same rate applies to every parameter equally.

Stochastic Gradient Descent (SGD)

Traditional gradient descent only updates once, which can make it unwieldy for larger datasets. Even for smaller datasets, the algorithm must process the entire dataset at once, which is time and memory-intensive and inefficient. Stochastic gradient descent, on the other hand, updates the parameters for each example in the training set. However, these frequent updates make convergence more complicated as it's easier for the algorithm to skip over minimums entirely. As with normal gradient descent, finding the precise learning rate can be difficult.

Adaptive Gradient (Adagrad)

Similarly to SGD, Adagrad calculates a learning rate for each parameter. Effectively, larger updates occur for rarer parameters and smaller updates occur for common parameters, which allows Adagrad to be more fine-tuned than SGD. Additionally, this is done automatically, meaning it requires less manual micromanagement. However, as the algorithm progresses, the learning rate steadily decreases, eventually becoming so small that the model actually stops learning entirely. This makes convergence and training time much slower compared to SGD despite greater specificity.

Root Mean Square Propagation (RMSprop)

RMSprop is a specialized version of Adagrad designed to fix the flaw of steadily decreasing learning rates. Like with Adagrad, learning rates are automatically calculated for each parameter. However, by using a moving average, the algorithm can keep track of recent gradients and adjust newer ones and prevent them from continually decreasing. Importantly, RMSprop adds *momentum*, which makes minimums easier to find by ensuring that the algorithm is travelling in the right direction along the graph.

Adam

Adam stands for “adaptive moment estimation”, and it is an optimizer designed specifically for training deep neural networks, and since it was first created in 2014 it has become a very popular choice for optimizer. Adam combines some of the features of the optimization methods previously described. Like RMSprop, it scales the learning rate without decreasing over time, and like an SGD with momentum it uses an average of a gradient rather than the gradient itself. This synthesis means that Adam lacks the problems of other optimizers, such as learning rate decay and slower convergence.

It is worth noting that while other optimizers often have their hyper-parameters tinkered with by programmers, the default values for Adam are often considered more than sufficient, which would simplify our jobs a fair amount. Adam is good for convolutional neural networks, which could be useful if we are forced to use static analysis in the granular model. However, no algorithm is perfect and while Adam can perform better with some training tasks, there are other tasks where it performs worse than even simple SGD. While we should certainly consider Adam, we should also take care not to fall into the “golden hammer” fallacy and ignore its potential shortcomings and its alternatives.

Overfitting

The process of training a neural network to make accurate predictions happens over many iterations, called epochs, where the weights are adjusted after each iteration to lessen the amount of loss in a model. A fundamental process is to train on one set of data and to test the trained model on a separate set of data. The training set and the test set are used to check how well your model predicts. However, there is an issue with this process where when you train over many epochs on the training set, the test set has lower accuracy. This issue is called overfitting and occurs because the training done on the training set is meant to lower the amount of loss, but sometimes the model becomes overly tuned to only this dataset and is not representative for the overall dataset.

The below images show a neural network that is overfitting to the data. In the first image, you can see that as the training accuracy increases, the test accuracy stagnates around 70% accuracy. In the second, we see the loss for the training set steadily decrease over time but the test set having an increase in the loss as the model trained.

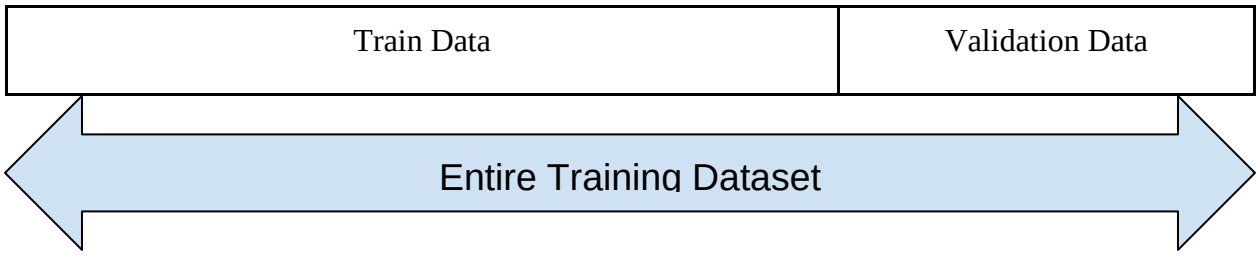
The following sections are methods in which we use to combat overfitting.

Cross-Validation Methods

A simple but effective method for limiting the amount of overfitting is by the use of a validation set. Instead of having a training set and a test set, we split the training set into two different datasets, training and validation. This technique allows us to train on the training set, validate our model on our validation set, and then do a final sanity check on our test set. By doing this, we get an unbiased evaluation of the model so we can fine-tune the parameters before checking the model against a test set. The accuracy of the validation set can be said to be the accuracy of the model as a whole. There are three methods of cross-validation that are being explored for this project: hold-out validation, k-fold validation, and iterated k-fold validation.

Hold-out Validation

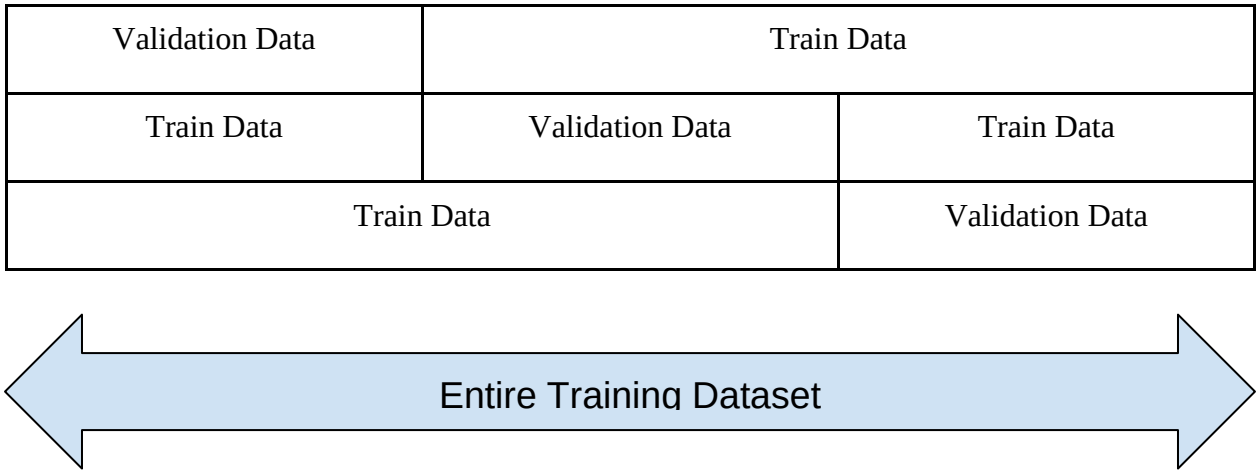
Hold-out validation is the simplest of all the cross-validation methods. This process consists of shuffling the training data and splitting the training data at an arbitrary point into two separate sets: training and validation. The neural network is trained on the training set and the model produced is evaluated on the validation set. The point of the split is arbitrary, but in standard practice, the resulting training data is larger than the validation data.



For the validation, our dataset is large enough that we are able to utilize this method for validation. If our final dataset was smaller we would need to employ one of the following methods for validation.

K-Fold Validation

K-fold validation takes the idea from the hold-out validation and expands it over multiple folds of the dataset. This method helps to offset the misleading results by averaging the accuracy results to the validation set. The process to shuffle the data then pick several folds k and divide the dataset k equal partitions. For each partition, you train on the remaining k-1 partitions. The average accuracy score is then taken over all the folds.



Iterated K-Fold Validation

This approach to validation is the same as the k-fold validation, but instead of shuffling once before the first division of data, the data is shuffled in between each fold. This technique is a suitable method if the dataset is not large. The drawback to this method is that it is possible that some data points are never part of the validation section where some points may be part of multiple validations across the folds.

Code in support of modeling options

Database Model Connection

The database contains the dataset that is used in the neural network training. The data has to be in correct form to be able to use it with the keras library. The features for a file will be stored in an index of a numpy array named *data* and labels stored in a separate numpy array named *label.* Below is an example of the connection between the database and the array being used in training.

```
import mysql.connector
import numpy as np

db = mysql.connector.connect(
        host="localhost",
        user=user,
        password=password,
        database="api"
        )
cursor = db.cursor(buffered = True)

cursor.execute("SELECT file.total_inserts, file.insert_averages, " \
               "file.total_deletions, file.deletion_averages, total_lines, " \
               "file.line_averages, " \
               "repo.assignees, repo.size, repo.commits, repo.events, " \
               "repo.forks, repo.branches, repo.contributors, repo.labels, " \
               "repo.language_count, repo.language_size, repo.milestones, " \
               "repo.issues, repo.refs, repo.stargazers, repo.subscribers, " \
               "repo.watchers, repo.network, repo.open_issues, repo.pulls, " \
               "repo.num_files, repo.commit_size, repo.commit_count," \
               "repo.insertions, repo.deletions, repo.lines_changed " \
               "FROM file INNER JOIN repo ON file.repoID = repo.id")

data = np.array(cursor.fetchall(), dtype=float)

cursor.execute("Select file.has_fault FROM file INNER JOIN repo on "\
        "file.repoID = repo.id")

labels = np.array(cursor.fetchall(), dtype=float)
cursor.close()
db.close()
```

Encoding

Categorical data needs to be modified into a representation in order to train our neural network. The method used in the project to convert from categorical data to numerical data is called one-hot encoding. [15, 16, 84] Since Keras does not provide a way to convert string data into one-hot representation, this process is done in two parts: integer encoding and then one-hot representation.

Integer Encoding

For integer encoding, we use the LabelEncoder method from sklearn.preprocessing. This method converts our string values into integer representations which the Keras to_categorical can receive as an argument.

```
fit_transform(y)
```

Arguments:

- y : array-like of shape (n_samples,)

Returns:

- y : array-like of shape [n_samples]

This is an example of how we would transform languages into integer representations.

```
from sklearn.preprocessing import LabelEncoder
import numpy as np

language = np.array(["python", "c", "java", "c", "java"])

language = LabelEncoder().fit_transform(language)

output: array([2, 0, 1, 0, 1)]
```

One-Hot Encoding

This method is part of the utils package in the Keras library. It takes in our integer encoding and returns a matrix of binary representation for the values.

```
keras.utils.to_categorical(y, num_classes=None, dtype='float32')
```

Arguments

- y: class vector to be converted into a matrix (integers from 0 to num_classes).
- num_classes: total number of classes.
- dtype: The data type expected by the input, as a string (float32, float64, int32...)

Returns:

- a binary matrix representation of the input. The classes axis is placed last.

Example:

```
from keras.utils import to_categorical

one_hot_language = to_categorical(language)

output:
array([[0., 0., 1.],
       [1., 0., 0.]
       [0., 1., 0.]
       [1., 0., 0.]
       [0., 1., 0.]], dtype=float32)
```

Feature Scaling

To ensure that the neural network has fairness for the weighting of the features we need to implement a scaling method that keeps all of the features within a certain range. This method allows smaller valued features to be treated the same as larger valued features and allows for faster training as gradient descent has less distance to move.

Overall Model Feature Scaling

In the overall model, the feature scaling is done with scale method from sklearn.preprocessing.

```
sklearn.preprocessing.scale(X, axis=0, with_mean=True, with_std=True, copy=True)
```

Arguments :

- X : The data to center and scale.
- axis : int (0 by default). axis used to compute the means and standard deviations along. If 0, independently standardize each feature, otherwise (if 1) standardize each sample.
- with_mean : boolean, True by default. If True, center the data before scaling.
- with_std : boolean, True by default. If True, scale the data to unit variance (or equivalently, unit standard deviation).
- copy : boolean, optional, default True. set to False to perform inplace row normalization and avoid a copy (if the input is already a numpy array or a scipy.sparse CSC matrix and if axis is 1).

Returns:

- X with values normalized based on input parameters.

Example:

```
from sklearn import preprocessing
import numpy as np

number_of_commits = np.array[5, 57, 2, 45, 100, 8, 1, 68, 45])

number_of_commits_scaled = preprocessing.scale(number_of_commits)

output: array([-0.96087917,  0.61146856, -1.05159154,  0.24861909,  1.91167919,
               -0.8701668 , -1.081829  ,  0.94408059,  0.24861909])
```

Validation

In the overall model, validation is done by partitioning the dataset into sections for training and validation. The model then learns based on the training set and the model is validated against the validation data. Over the interactions, the accuracy for the training model will increase as the network learns but the test of a good model is being able to use the model in the general case. We check this by ensuring that the accuracy increases in both training and validation data at a similar rate.

Example:

```
def hold_out_validation(data, data_labels, percentage):

    num_validation_samples = int(len(data) * (percentage / 100))

    train_data = data[num_validation_samples:]
    train_data_labels = data_labels[num_validation_samples:]

    validation_data = data[:num_validation_samples]
    validation_data_labels = data_labels[:num_validation_samples]

    return (train_data, train_data_labels), (validation_data,
validation_data_labels)
```

Optimizers

An optimizer changes the weights and biases of the model during training. During the training process, the machine learning model is attempting to optimize a particular loss function (discussed elsewhere in this document), which represents how accurate the model is by comparing the expected output to the actual output. The optimizer is an algorithm that adjusts the model's weights over multiple iterations to try to minimize this function as much as possible. Naturally, choosing a optimizer and the proper settings for it can radically change the performance of the neural network, making these choices very important.

The Keras library has multiple useful optimizers built in. Their settings can either be adjusted ahead of time or invoked with their default parameters.

For example, the code for modifying an optimizer is:

```
my_sgd = optimizers.SGD(lr = 0.01, decay = 1e6, momentum = 0.9, nesterov = False)
model.compile(loss = 'poisson', optimizer = my_sgd)
```

And the code for using it with default parameters is simply:

```
model.compile(loss = 'poisson', optimizer = 'sgd')
```

Neural Net

One important consideration when working with neural networks is that every input to the neural network must be the same size. If there are inputs that are smaller than others, they must be padded with a null value, such as a zero or empty string. In the case of this project, there could easily be gaps in the dataset if, for example, there is no comment with a commit or if the project doesn't have any Issues on GitHub.

The code below pads the matrix sequences to ensure it is of length 10,000. In this case, 1 is a "null" value.

```
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
       results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results
```

The next portion of the code creates the actual model itself. Keras provides two options for creating a model: the sequential model and the model class API. This example uses a sequential model.

```
model = models.Sequential()

# Input - Layer
model.add(layers.Dense(50, activation = "relu", input_shape=(10000, )))

# Hidden - Layers
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
```

```
# Output - Layer
model.add(layers.Dense(1, activation = "sigmoid"))
```

Every neural network organizes its nodes into three layers: the input layer, the hidden layer(s), and the output layer. The input layer simply passes information from outside the model to the hidden layer, which then performs the necessary computations before handing off the data to the output layer. The output layer performs some calculations and passes information to outside the model.

"Dense" layers are standard densely-connected neural network layers, with the integer argument above representing the number of output nodes. "Dropout" represents a percentage (as a float between 0 and 1) of neurons which are randomly ignored during a particular pass of the neural network. This float is the first argument in the dropout function above. Dropout is used to try to minimize overfitting of the model. When a model is overfitting, that means that, after training, it matches the dataset(s) it was trained on too close and won't give desirable results for other datasets. "Activation" refers to the activation function, which decides which neuron should fire and with what intensity, with RLU abbreviating "rectified linear unit".

The choices of activation functions, dropout rates, and number and type of hidden layers are an important part of designing a neural network. There aren't any right or wrong answers, and we can only experiment to see which choices we can fine-tune to improve our model, but in the end, small changes can have a large impact on the model.

Once the model is set up, it must be compiled. The optimizer changes the weights and biases during training, and the loss function evaluates how well the algorithm models the dataset. The higher the number, the worse the algorithm's predictions, and vice-versa. Metrics are used to judge the performance of the model, with "accuracy" simply giving a ratio of correct predictions to incorrect predictions.

```
model.summary()

# Compilation: configuring for training
model.compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = ["accuracy"]
)
```

This example uses binary classification, which is to say the model answers a yes (1) or no (0) question. This is why the example uses binary cross-entropy, which gives a probability between 0 or 1 as our loss function. Our project has a binary classification problem: “Does this code contain security vulnerabilities?” However, we would also like to classify what type of security vulnerability may be present, and so we have to use another type of loss function instead of or in addition to binary cross-entropy.

Now that the model is prepared, it undergoes training with the “fit” function. The variables train_x and train_y represent the data to train the model on, and test_x and test_y are used to evaluate the loss respectively. A single epoch represents a single forward and backward pass on the model. The batch size represents the number of inputs to be trained in one pass.

```
results = model.fit(
  train_x, train_y,
  epochs = 1000,
  batch_size = 500,
  validation_data = (test_x, test_y)
)
```

The batch size is an important variable because while larger batch sizes often improve performance, they can also cause the model to become overfit and require more memory. In our research, the recommended batch size is 32, which is also the default value for that variable in Keras. Altering the batch size and several epochs are important for refining the model and ensuring that it is neither underfitting or overfitting.

Once the neural network has been trained with acceptable results, the model will be saved to the server to be used in the application.

```
model.save('vulnerability_model.h5')
```

Backend

Analysis of Database Characteristics

Relational Database System features

- Logically presented information
- Data is logically accessible by the table, column and/or key
- Null values are treated differently from zero values
- Metadata is stored in the database
- Single language used (SQL)
- Views show updates of base tables and vice versa
- A single operation retrieves, inserts, updates, or deletes data
- End-user operations are separate from physical storage and access
- Batch and end-user ops can perform in-place database schema modifications
- Integrity information is stored and handled by the database itself
- Data manipulation language agnostic from physical data storage scheme
- Row processing ops obey same integrity constraints as set-processing ops

External, conceptual, internal views

- External Level
    - Defines how end users see the arrangement of data. This is unique to each person and can include varying user permissions, their role at the company, or their choice front end application.
- Conceptual Level
    - External views are composited into a consistent global view. Provides a synthesis of all external views. Out of scope of end users, more of interest to admins and devs.
- Internal Level
    - Internal organization of data inside the system. Concerned with cost, performance, scalability, other operational matters. Storage layout of the data, indexing, cache coherence, balancing external views' performance requirements are all handled under this level.
- Separating these three views from each other was a major driving factor for the relational models whose implementations mostly dominate in the current paradigm.

Types of DBMS

- Hierarchical
    - Data is stored in parent-child relationship nodes. Data is held along with meta information about these relationships. Organized in a tree-like structure. One parent that can have multiple children. Retrieval involves traversing the tree until the result is found. Origins with IBM in the 1960s.
- Network
    - Network structure used to create relationships between entities. Many to many relationships are utilized internally, and a network node can have multiple 'parent' relationships. In this style, children are called 'members' and parents are called 'occupier'.
- Relational
    - Relationships between data are relational, and data is stored in columns and rows. Each column in a table represents an attribute, and each row in a table represents a record. Fields provide representations of stored data values. Structured Query Language is used to query and perform insert, update, delete, and search operations on the database. A defining factor of RBDMS is a key which identifies each row uniquely and using these keys to form connections between data.
        - Values are atomic
        - Rows are singular entities
        - Column values always hold the same underlying representation
- Object-oriented
    - Extend the functionality of C++ and Java by adding database functionality. Integrates the application and database development into a consistent model and language environment. Applications require less code, perform more natural modeling while allowing code to be easier to maintain over time. Cyclical treatment of consistent data, like found in databases, and transient data, like found in running programs
    - Objects are used, containing state and behavior information. This allows reuse of objects across the system.
- Graph
    - This style of DBMS includes NoSQL databases where a graph structure is used for queries. Data is stored as nodes, edges, and properties. Nodes may represent an entity or instance, and is equivalent to a record, or row, in a traditional system. Edges are used to represent relationships that connect nodes. Properties are metadata added to the nodes. Oracle and SQL Server 2017 support this style of database.
- ER model

    - Entity relationship modeling. Each row represents one instance of an entity, and each field represents an attribute type. Relationship between entities is enforced by storing the primary key of one entity as a pointer in the table of another entity.
- Document
    - NoSQL databases that store data in the form of documents. Documents represent data, the relationship between other data elements, and attributes about these elements. A key value storage scheme is used for document databases. Mongo falls into this group of database type.

Code in Support of Backend Upkeep

Update advanced package tool

```
sudo apt-get update
```

Install mysql using apt

```
sudo apt-get install mysql-server
```

Digital Ocean’s mysql installer

```
Mysql_secure_installation
```

Edit file to allow remote database connections

```
sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
```

Restart the mysql server

```
/etc/init.d/mysql restart
```

Log in locally as root

```
mysql -u root -p <<password>>
```

Log in remotely as ‘user’

```
mysql -u user -p -h server.ip.here
```

Connect to the remote server via SSH

```
ssh root@server.ip.here
```

Restart the server

```
sudo shutdown -r now
```

Code to Create the Database

Database 1 final design

The repo is the “parent” in the current “parent-child” database design, and holds API fields about the repo, such as stargazers, which is the number of people who have favorited the repository, the number of branches in the repository, the number of people collaborating on the repository, and other kinds of assorted information.

The primary key is ‘id’, which is a unique id given to each repository once it has been entered into our system. Other fields in this table represent date and time data, or integer quantities.

```
CREATE TABLE repo
 (
 id INT unsigned NOT NULL AUTO_INCREMENT,
 repo_name VARCHAR(100) NOT NULL,
 assignees INT unsigned NOT NULL,
 size INT unsigned NOT NULL,
 commits INT unsigned NOT NULL,
 EVENTS  INT unsigned NOT NULL,
 forks INT unsigned NOT NULL,
 branches INT unsigned NOT NULL,
 contributors INT unsigned NOT NULL,
 labels INT unsigned NOT NULL,
 language_count INT unsigned NOT NULL,
 language_size INT unsigned NOT NULL,
 milestones INT unsigned NOT NULL,
 issues INT unsigned NOT NULL,
 refs INT unsigned NOT NULL,
 stargazers INT unsigned NOT NULL,
 subscribers  INT unsigned NOT NULL,
 watchers  INT unsigned NOT NULL,
 network INT unsigned NOT NULL,
 open_issues INT unsigned NOT NULL,
 pulls INT unsigned NOT NULL,
 num_files INT unsigned NOT NULL,
 commit_size INT unsigned NOT NULL,
 commit_count INT unsigned NOT NULL,
 insertions INT unsigned NOT NULL,
 deletions INT unsigned NOT NULL,
 lines_changed INT unsigned NOT NULL,
 PRIMARY KEY (id)
 );
```

The file is the “child” in the current “parent-child” database design, where the repo’s primary key is stored in the field record as the repoID foreign key. This means that each file is owned by only one repo, and a repo can own an unlimited number of files. Most fields are NULL, but will be switched over to NOT NULL when the data configuration utility is reliably grabbing this field from sources, and storing it in the database. This means that entering this information with a record is not optional, and this ensures all records in the database contribute to the overall quality of the model, and sparse records do not exist in our database.

Once again ‘id’ is the primary key, which is a strictly increasing integer used to identify records in this table as unique.

```
CREATE TABLE FILE
  (
  id INT unsigned NOT NULL AUTO_INCREMENT,
  repoID INT unsigned NOT NULL,
  filename VARCHAR(4096) NOT NULL,
  hash VARCHAR(512) NOT NULL,
  has_fault BIT(1) NOT NULL,
  total_inserts INT unsigned NOT NULL,
  insert_averages INT unsigned NOT NULL,
  total_deletions INT unsigned NOT NULL,
  deletion_averages INT unsigned NOT NULL,
  total_lines INT unsigned NOT NULL,
  line_averages INT unsigned NOT NULL,
  commit_size INT unsigned NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (repoID) REFERENCES repo(id)
  );
```

Database 2 final design

After determining the accuracy of the model improved without repo fields, additional file based features were developed which necessitated the need of a second database design. Here we see a lone file table which stores information about each problematic file scraped from the CVE database.

```
CREATE TABLE FILE
(
id INT unsigned NOT NULL AUTO_INCREMENT,
repo_name VARCHAR(2048) NOT NULL,
filename VARCHAR(4096) NOT NULL,
hash VARCHAR(512) NOT NULL,
has_fault BIT(1) NOT NULL,
total_inserts DOUBLE unsigned NULL,
insert_averages DOUBLE unsigned NULL,
total_deletions DOUBLE unsigned NULL,
deletion_averages DOUBLE unsigned NULL,
total_lines DOUBLE unsigned NULL,
line_averages DOUBLE unsigned NULL,
total_access DOUBLE unsigned NULL,
recent_time_delta DOUBLE unsigned NULL,
age_of_file DOUBLE unsigned NULL,
lines_per_file DOUBLE unsigned NOT NULL,
words_per_file DOUBLE unsigned NOT NULL,
characters_per_file DOUBLE unsigned NOT NULL,
avg_words_per_line DOUBLE unsigned NOT NULL,
avg_characters_per_line DOUBLE unsigned NOT NULL,
total_indents_in_file DOUBLE unsigned NOT NULL,
num_lines_with_indents DOUBLE unsigned NOT NULL,
deepest_indentation_level_in_file DOUBLE unsigned NOT NULL,
num_of_files DOUBLE unsigned NOT NULL,
num_of_subdirs DOUBLE unsigned NOT NULL,
num_of_directory_levels DOUBLE unsigned NOT NULL,
avg_num_files_per_directory DOUBLE unsigned NOT NULL,
PRIMARY KEY (id)
);
```

Data Acquisition

Code in Support of Data Acquisition

Connecting to the MySQL Database

Connecting with MySQL database was straightforward - the code required for connection can be located on the MySQL website after searching for a connection with python. However, to connect with our database, we needed to make sure the database was set up, and whatever username and password was used we needed that information as well and if there was a specific database we needed to input the name of that database as well. Once we had that information and the host IP address we were able to connect to the database through our python program fully. This script was done in a separate module. We also created a separate module that would update the indexes automatically whenever we would add new features to our database so we didn't have to update the script manually.

```
def make_query(table_name, cursor):
	execute = "SELECT * FROM " + table_name + " LIMIT 0"
	cursor.execute(execute)
	fields = [field[0] for field in cursor.description][1:]

	if table_name == "repo":
		query = "INSERT INTO repo ("
	else:
		query = "INSERT INTO file ("

	for field in fields:
		query += field + ", "

	query = query[:-2] + ") VALUES ("

	for field in fields:
		query += "%s, "

	query = query[:-2] + ")"

	return query

def update_db(update_files, github_name, repo_dir, repo):
	git = access_github()

	# grabs repo object for repo table data (collecting)
```

```
	try:
		github_repo = git.get_repo(github_name, lazy=False)
		repo_data = repository_data(github_repo, repo, github_name, repo_dir)
	except:
		print("Could not access github api for" + repo_dir)
		return

	try:
		conn = mysql.connector.connect(user=user, host=host,
password=password, database=database)
		cursor = conn.cursor(buffered=True)

		query = make_query("repo", cursor)
		cursor.execute(query, repo_data)
		repo_id = cursor.lastrowid

		query = make_query("file", cursor)
		for update in update_files:
			print("Adding " + update[0] + " to database.")
			file_update = file_data([repo_id] + update)
			cursor.execute(query, file_update)

		conn.commit()
		cursor.close()
		conn.close()
	except mysql.connector.Error as err:
		if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
			print("Something is wrong with your user name or password")
		elif err.errno == errorcode.ER_BAD_DB_ERROR:
			print("Database does not exist")
		else:
			print(err)
```

Getting Repository Data

Getting the repository data was easy once we established which library we were going to use for collecting the data and we had an access token to gain access to GitHub API. Once we had API access, all we need to do is type in the GitHub repository name, and the script pulls all the repository data from that repository. This data includes Integer data and boolean data for our database. The Machine learning team required more specific data, but we still have this data available for training if needed. It is imported as a module to use in the main file being ran.

```
Def access_github():
    """Gain access to GitHub API and allow for the gathering of repository data
     Return: access to the github api using access token
     """
    Return Github(git_access)



def repository_data(git_repo, repository, repository_name, repository_dir):
      """
      Accesses github api to pull info such as number of subscribers, issues,
commits and size of repo
      :param git_repo: repository object containing the entire repo
      :param repository: repository object
      :param repository_name: name of repo currently being scanned
      :param repository_dir: location of cloned repo
      :return:
      """
      lang_string = []
      lang_size = []
      language_size = 0
      files = []
      commit_size_sum = 0
      commit_files_count = 0
      commit_insertion_count = 0
      commit_deletion_count = 0
      commit_lines_changed_count = 0

      # collects data using the git_repo repository object
      assignees = git_repo.get_assignees().totalCount
      branches = git_repo.get_branches().totalCount
      contributors = git_repo.get_contributors().totalCount
      count_open_issues = git_repo.open_issues_count
      commits = git_repo.get_commits().totalCount
      events = git_repo.get_events().totalCount
      forks = git_repo.get_forks().totalCount
      issues = git_repo.get_issues().totalCount
      labels = git_repo.get_labels().totalCount
      language_count = len(git_repo.get_languages())
      for lang, count in git_repo.get_languages().items():
            lang_string.append(lang)
            lang_size.append(count)
      for lang in lang_size:
            language_size += lang
      milestone = git_repo.get_milestones().totalCount
```

```
	network_count = git_repo.network_count
	pulls = git_repo.get_pulls().totalCount
	refs = git_repo.get_git_refs().totalCount
	stargazer = git_repo.get_stargazers().totalCount
	subscribers = git_repo.get_subscribers().totalCount
	watchers = git_repo.watchers_count
	size = git_repo.size

	# runs through the complete repository directory to get number of files
	for (dirpath, dirnames, filenames) in walk(repository_dir):
		for filename in [f for f in filenames]:
			files.extend(filenames)
			break
	number_of_files_in_project = len(files)

	# parses all commits in commit log for repository
	commit_list = list(repository.iter_commits())[:]
	for commit in commit_list:
		commit_size_sum += commit.size
		# get data points from stats, since it's stored in a dictionary we had
to use a nested for loop to gather it
		commit_stats = commit.stats.files
		for file_path, value in commit_stats.items():
			commit_files_count += 1
			for type_of_change, change_value in value.items():
				if type_of_change == "insertions":
					commit_insertion_count += change_value
				if type_of_change == "deletions":
					commit_deletion_count += change_value
				if type_of_change == "lines":
					commit_lines_changed_count += change_value

	# used for passing to database and/or to training module
	repo_data = (repository_name, assignees, size, commits, events, forks,
branches, contributors,
			labels, language_count, language_size, milestone, issues, refs,
stargazer, subscribers, watchers,
			network_count, count_open_issues, pulls,
number_of_files_in_project, commit_size_sum,
			commit_files_count, commit_insertion_count,
			commit_deletion_count, commit_lines_changed_count)

	return repo_data
```

Getting File Specific Data

Retrieving the file data required for training has been the most challenging part of this whole process so far. There aren't a lot of libraries or tools that Python had that would allow us to access Github repos and pull file specific data from such as lines of code of each file and number of additions and deletions of lines from each file.

Additionally, finding which features we could use is only one part of the problem. Many pieces of data required further processing before they could become usable. For instance, PyGithub could give us the number of insertions and deletions, but the data would be in the form of nested dictionaries for each file, and the machine learning model needs its input data to be in the form of a single list for each file, and so these dictionaries essentially had to be disassembled and reassembled into the proper format.

Other potential features didn’t have any built-in library calls we could make, and so we had to create functions for them wholesale. These functions were not individually difficult, but they each required a fair bit of tinkering to get operating properly and there were a dozen of them.

Creating these new functions also caused new and unforeseen complications. For instance, creating when creating functions that navigated directories, we would sometimes create a function that worked perfectly on our machines, but did not work at all on someone else’s machine. Making these functions OS-agnostic required diving into Python libraries that we had never used before.

The functions that extracted features by looking inside files also had issues. First of all, they only work properly on text files, and in fact the machine learning model can only really analyze text files for security vulnerabilities. However, the files read in from a repo are not guaranteed to be text files, and so the program would, for instance, attempt to find the number of indents in a .png and subsequently crash.

The solution to this problem was to catch any exceptions that were raised during feature collection and deleting the offending file from the dataset entirely. This solution is a bit of a hack and we didn’t have time to fully test it. While testing there were some files that the program should have been able to read that it couldn’t, and some that it shouldn’t have read but did without crashing. These were eliminated after some refinement, but without better testing, we can’t know if this problem is entirely fixed. The accuracy of the predictions could be seriously affected if the program is ignoring integral files and instead generating data from junk files.

Getting this data was a process that took us well into the latter end of the year. However we were able to figure out ways to gather more file specific meta-data for a more accurate prediction.

Getting a lot of the features we received required several different methods each collecting a different piece of information from each file. Then, other functions were written which would aggregate each of these individual pieces of data into larger datasets until the final, completed dataset could be passed to the application.

```
def get_averages(file_list, commit_hash, repo):
	"""
	Creates a list containing the averages of all the commit history data
	:param file_list: a lst containing both dirty and grey files
	:param commit_hash: the hash of a given commit
	:param repo: the repository object
	:return: the filename, total inserts, insert averages, total deletions,
deletion averages, total line changed,
			lines changed averages
	"""
	commits = repo.iter_commits(commit_hash)
	file_totals = []

	"""
	DONT TOUCH ANYTHING IN THE CODE BELOW
	for each commit in the commit history we look at its filenames and for each
of those filenames we check to see
	if that filename is in the list [filenames] we mark our flag [check] as true
which means the file does
	exist in [file_totals] therefor we create and update its values else if the
file is not in [file_totals] we
	**DO SOMETHING I FORGET ASK CURTIS** and then we updated [file_totals] with
the averages of each file.
	"""
	count = 1

	for commit in commits:
		commit_files = parse_dic(commit.stats.files)

		for path in commit_files:
			if path[0] in file_list:
				check = True
				for phile in file_totals:
					if path[0] == phile[0]:
						adder = 1
						for index, vals in enumerate(path[1:]):
							phile[index + adder] += path[index +
1]

							phile[index + adder + 1] += 1
```

```
                                          adder += 1
                                    check = False
                                    break
                        if check:
                              file_info = []
                              file_info.append(path[0])
                              for vals in path[1:]:
                                    file_info.append(vals)
                                    if vals > 0:
                                          file_info.append(1)
                                    else:
                                          file_info.append(0)

                              file_totals.append(file_info)

      for phile_info in file_totals:
            for index, update in enumerate(phile_info[1:]):
                  if index % 2 == 0:
                        val = phile_info[index + 1]
                  else:
                        if phile_info[index + 1] != 0:
                              phile_info[index + 1] = val / phile_info[index +
1]

      return file_totals


# Takes a path to a directory and returns the total amount of subdirectories it
contains.
def num_subdirs(path):
      return max(0, len([dirName for _, dirName, _ in walk(path)]) - 1)


# Takes a path to a directory and returns the deepest level of subdirectories as an
int.
# Uses a recursive depth-first search. The current directory is considered to have
a depth of 0.
def max_subdirs(file_path, depth=0):
      max_depth = depth

      # Go through each thing in this directory and find it's depth.
      for cur in listdir(file_path):

            '''
            Concatenates the file path with what we're currently looking at to get
```

```
the full
            path of the object we're looking at (note that these are NOT strings).
            ex: 'C:/Users/Desktop/example/' + 'file1.txt' =
'C:/Users/Desktop/example/file1.txt'
            '''
            full_path = path.join(file_path, cur)

            # If this thing is a directory, parse through it with a recursive call
to max_subdirs,
            # incrementing depth as we're going another layer deep.
            if path.isdir(full_path):
                  max_depth = max(max_depth, max_subdirs(full_path, depth + 1))

      return max_depth


def fileLineCount(path):
      with open(path) as pathFile:
            return sum(1 for _ in pathFile)


def fileWordCount(path):
      with open(path) as pathFile:
            return len(pathFile.read().split())


def fileCharacterCount(path):
      with open(path) as pathFile:
            return sum(len(word) for word in pathFile.read().strip().strip())


def fileAvgWordsPerLine(path):
      return fileWordCount(path) / fileLineCount(path)


def fileAvgCharPerLine(path):
      return fileCharacterCount(path) / fileLineCount(path)


def indent_parser(file_path):
      """
      reads given file and grabs number of indentations (counts by spaces)
      @param path: file path
      @return indentation_count: total number of indentations from a file
      """
```

```
	indentation_count = 0

	# opens file and grabs the line number and the number of indentations for
each line and adds them up
	with open(file_path) as file:
		for mark, line in enumerate(file.readlines()):
			indentation_count += (len(re.findall(r"\t", line)))

	return indentation_count


def indented_lines(file_path):
	"""
	reads given file and number of lines that have indentations in them (goes by
spaces)
	@param path: file path
	@return deepest: deepest level of indentation
	"""
	lines_indented = 0
	count = 0

	with open(file_path) as file:
		for mark, line in enumerate(file.readlines()):
			count = (len(re.findall(r"\t", line)))
			if count > 0:
				lines_indented += 1

	return lines_indented


def indentation_depth(file_path):
	"""
	reads given file and returns deepest indentation level in file (as deepest
number of spaces)
	@param path: file path
	@return deepest: deepest level of indentation
	"""
	indentation_count = 0
	depths = []
	deepest = 0

	# opens file and grabs the line number and the number of indentations for
each line and adds them up
	with open(file_path) as file:
		for mark, line in enumerate(file.readlines()):
```

```
                    indentation_count = (len(re.findall(r"\t", line)))
                    depths.append(indentation_count / 4)

      deepest = max(depths)

      return deepest

def num_files(path):
      files = []
# runs through the complete repository directory to get number of files
      for (_, _, filenames) in walk(path):
            for filename in [f for f in filenames]:
                  files.extend(filenames)
                  break
      return max(0, len(files))

def avg_files_per_dir(path):
      return num_files(path) / num_subdirs(path)


def dirty_data(repository, commit_hash):
    """
    Dirty Data takes a given commit_hash and parses information from it which is
then stored in the database
    it also stores the filename in a list that is stored globally
    :param repository: full repository object
    :param commit_hash: a hash of a specific commit in a repository
    :return:
    """

    # get the data on a specific hash from the given repo
    commit = repository.commit(commit_hash)

    black_list = list(commit.stats.files.keys())
    if len(black_list) != 1:
        print("Current file_list has " + str(len(black_list)) + " files.")
        return black_list, None, None

    # use totals to find averages
    commit_size = commit.size
    averages = get_averages(black_list, commit_hash, repository)

    for fault_file in averages:
        fault_file.insert(1, 1)
        fault_file.insert(1, commit_hash)
```

```
        fault_file += [commit.size]

    return None, black_list, averages


def clean_data(repository, commit_hash, black, grey):
    """
    Clean data looks through a repository history up until the given commit hash to
find another file
    with the same extension. It strips the filename for the extension then checks
to see it another filename
    matches that extension in the commit history. once found parse that commit to
database as clean data
    :param repository:
    :param commit_hash:
    :return:
    """

    clean_files = []
    dirty_ext = []

    commits = repository.iter_commits(commit_hash)

    # grab the extension of the dirty file
    for path in black:
        dirty_ext.append(path.split(".")[-1])

    dup_path = list(set(black + grey))

    # go through commit history up to dirty file commit and store all files with
the same extension as the
    # dirty_ext into a list
    for commit in commits:
        for path in commit.stats.files:
            if path.split(".")[-1] in dirty_ext:
                if path not in dup_path:
                    clean_file = get_averages([path],commit,repository)[0]
                    clean_file.insert(1, 0)
                    clean_file.insert(1, commit.hexsha)
                    clean_file += [commit.size]
                    clean_files.append(clean_file)

                    dup_path.append(path)
                    dirty_ext.remove(path.split(".")[-1])
                    if len(dirty_ext) == 0:
```

```
                        return clean_files



if __name__ == "__main__":
    try:
        start_time = time.time()

        with open("./valid.repo.data", "rb") as f:
            temp = pickle.load(f)

        #temp = temp[1:]
        for index, name in enumerate(temp):
            # Pop the first item on the list to leave only hashes
            github_name = name.pop(0)
            print("Starting process for " + github_name)

            if os.path.exists(os.path.basename(github_name)):
                shutil.rmtree(os.path.basename(github_name))

            # Clones repo to current directory and gets the github data
            repo_dir = clone_repo(github_name)
            repo = Repo(repo_dir)

            # lists containing the files we know that have faults and those that
may or may not have them
            black_files = []
            grey_files = []
            clean_files = []
            db_update = []

            for hash_commits in name:
                print('processing dirty data for hash: ' + hash_commits)
                grey, black, update = dirty_data(repo, hash_commits)
                print('finished dirty data for hash: ' + hash_commits)

                if grey is not None:
                    grey_files += grey

                if black is not None:
                    black_files += black
                    db_update += update


            print('finished updating black and grey lists')
```

```
            # once we get the complete black and grey lists we parse them to find
the clean data
            if len(black_files) != 0:
                for hash_commits in name:
                    print('processing clean data')
                    clean_file = clean_data(repo, hash_commits, black_files,
grey_files)
                    if clean_file is not None:
                        clean_files += clean_file
                        db_update += clean_files
                        if len(clean_files) == len(black_files):
                            break

                print('finished clean data')

            if len(db_update) > 0:
                print("Updating db")
                update_db(db_update, github_name, repo_dir, repo)
                print()
            else:
                print(github_name + " has no valid files for database")
                print()

            # Removes the cloned repo from current directory
            shutil.rmtree(repo_dir)

            print(temp[index+1][0])

            if index + 1 < len(temp):
                with open("./valid.repo.data", "wb") as f:
                    pickle.dump(temp[index+1:],f)

        print('Script Complete|Runtime: {} Seconds'.format(time.time() -
start_time))
        print()



    except Exception as e:
        print(e)
```

Application Interface

Once the tools to assemble the datasets were in place, we next had to write code to connect the desktop application with these tools. The application supports local repositories, public remote repositories, and private remote repositories, each of which required slightly different and sometimes mutually exclusive actions, and so proper flow control was very important for the interface.

Once the flow control was down, the next and most important part was interfacing with three parts of the project - the desktop application, the data processing functions, and the machine learning model. Interfacing with the application was a simple matter of reading in command-line arguments (the amount of which was then used for flow control) and printing the results of the prediction to the standard output. The command-line arguments were then used to gather the information the data acquisition layer needed in order to assemble all of the repository and file features.

Once this data had been gathered and returned it needed minor modifications before it was in the proper form for that predictor. This part of the interface was hard to debug because we had to ensure that every individual step of the process modified the data correctly before passing it to the next, and solving these bugs sometimes involved fixing code in other files. It also didn't help that the machine learning libraries gave almost impenetrable error messages that were difficult to parse let alone rectify.

Once the interface was completed, the various parts of the project finally converged into one coherent pipeline and the whole of the project could be tested in earnest.

```
def app_interface():

    #Get number of command-line arguments
    argc = len(sys.argv)

    '''
    print("Starting interface...")
    print(argc)
    for a in sys.argv:
        print(a)
    '''

    #Must have 1, 2, or 4 arguments (program name is the first argument)
```

```
    if argc not in [2, 3, 5]:
        print("\n\tERROR: INCORRECT NUMBER OF ARGUMENTS PASSED")
        exit()

    #Used for flow control for different types of repos
    repo_remote = False
    repo_private = False

    #The first argument is the repo directory. This is required for all repos, and
is the only
    #argument for local repos.
    repo_dir = sys.argv[1]
    repo = Repo(repo_dir)

    #Repo name is required for remote & public repos.
    if argc > 2:
        repo_remote = True
        repo_name = sys.argv[2]

        #Login information is required for remote & private repos.
        if argc > 3:
               repo_private = True
               username = sys.argv[3]
               password = sys.argv[4]

        #Get Github access. Use login if applicable.
        if repo_private:
               git = Github(username, password)
        else:
               git = Github(None, None)

        #Grabs repo object for data collection
        github_repo = git.get_repo(repo_name, lazy=False)

        #Retrieve repository data
        repo_data = list(repository_data(github_repo, repo, repo_name, repo_dir))

    #Retrieve all filenames for all directories
    files_path = []
    get_filenames(repo_dir, repo_dir, files_path)

    #Get a list of data points for each file
    features = get_file_features(repo_dir, files_path, repo)

    #Separate file names from avgs
```

```
    #file_names = []
    features_final = []
    for f in features:
        #file_names.append(a[0])
        features_final.append(f[1:])

    #Collect data together into a single list
    final_data = []

    #If this is a remote repo, we need to add the repo data, otherwise use averages
    if repo_remote:
        for ff in features_final:
               final_data.append(ff + repo_data[1:])
    else:
        final_data = features_final

    #Predict and print results data to std out
    results = predict(final_data)

    #Final results output
    for i, f in enumerate(files_path):
        print(f + "," + str(results[i][0]))
```

Testing in Support of Data Acquisition

Here are some notes listing some of the testing we did on the data collection in the early stages of development, this includes some of the processes we went through, different repositories we tested on and what other processes we went through in the Fall, Spring and Summer.

- Running tests on different Github Repositories using the python script. The script pulls repository information using the PyGithub library, and it pulls file specific information through the commits using the GitPython library. The main focus of March was to locate a reasonable project with good metadata. We did this by searching through Github commit to search for Common Vulnerabilities and Exposures (CVE) and Common Weakness Enumeration (CWE) commits.

Testing the projects with the most commits related to either of the two searches to find out which ones have the most commits on the topic in their commit history. The was done using GitPython which allowed me direct access to a projects version history and repository information.

- Tests have found only OpenSSL with 11 commits in its history, which is far lower than what we had hoped. Other projects that have been looked at are

OpenSSL: The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, fully featured, and Open Source toolkit implementing the Transport Layer Security (TLS) protocols (including SSLv3) as well as a full-strength general purpose cryptographic library.
[https://github.com/openssl/openssl]

Sonic: A fast, lightweight and schema-less search backend. It ingests search texts and identifier tuples that can then be queried against in a microsecond's time.
[https://github.com/valeriansaliou/sonic]

Pyright: Pyright was created to address gaps in existing Python type checkers like mypy.
[https://github.com/Microsoft/pyright]

Exiv2: Exiv2 is a Cross-platform C++ library and a command line utility to manage image metadata. It provides fast and easy read and write access to the Exif, IPTC and XMP metadata *and* the ICC Profile embedded within digital images in various formats.
[https://github.com/Exiv2/exiv2]

- There have been a few more projects that have potential for containing CVE or CWE mentions in the commit history, these include.

Linux: a family of free and open-source software operating systems based on the Linux kernel,an operating system kernel first released on September 17, 1991 by Linus Torvalds.Linux is typically packaged in a Linux distribution (or *distro* for short).
[https://github.com/torvalds/linux]

Visual Studio Code: VS Code is a type of tool that combines the simplicity of a code editor with what developers need for their core edit-build-debug cycle. It provides comprehensive editing and debugging support, an extensibility model, and lightweight integration with existing tools.
[https://github.com/Microsoft/vscode]

- Running new tests on an updated script has yielded completely different results that are extremely promising. Querying cve on githubs website under specific repos returns a value of number of commits that menton cve. That number is within range of the value produced when i run my module against the cloned repo.

OpenSSL:

Total mentions of CVE found in project 150
Total commits 23594

Poky:

Total mentions of CVE found in project 741
Total commits 53729

- Downloading multiple projects from GitHub that mention cve in the commits multiple times. Search for projects with 500 or more commits that mention cve in the message/summary.

Pyfiscan: Pyfiscan is free web-application vulnerability and version scanner and can be used to locate outdated versions of common web-applications in Linux-servers. Example use case is hosting-providers keeping an eye on their users installations to keep up with security-updates. Fingerprints are easy to create and modify as user can write those in YAML-syntax. Pyfiscan also contains tool to create email alerts using templates.
[https://github.com/fgeek/pyfiscan]

Quix: a purely functional package manager, and associated free software distribution, for the GNU system. In addition to standard package management features, Guix supports transactional

upgrades and roll-backs, unprivileged package management, per-user profiles, and garbage collection.

It provides Guile Scheme APIs, including a high-level embedded domain-specific languages (EDSLs) to describe how packages are to be built and composed. [https://github.com/meiyopeng/guix]

Pyfiscan:

Total mentions of CVE found in project 505
Total commits 2255

Quix:

Total mentions of CVE found in project 1112
Total commits 42105

Notes:

(Spring)

- Using http://line-count.herokuapp.com/ would allow the extraction of line data from each file in the repository. It gives the number of specific file types, lines of code and total count. It's not specific on each file, however. Will possibly need to run a query on the main page of http://line-count.herokuapp.com/ to see the results and use the data it has stored in it.

- Hardcoded the data from http://line-count.herokuapp.com/ and also ran the Linux command on the project to compare the information. Tried pulling data from a website with no success. On the actual website, all the rows with data are there, but when trying to pull the data using beautifulsoup, it doesn't show up.

- Tried using a Linux command: git ls-files | xargs wc -l On a given repository, but would need to be able to clone the file in python and then set the Linux command on the folder. The Linux command won't run properly because of some unknown error that changes with each attempt made. Will probably have to clone git projects to the home computer and run commands then, and hard code results in to be used for data.

- Decided to change the type of data we wanted to gather, the current data we have was not good enough especially if we wanted to get to the file level. GitPython will be used to find lines of code, commit changes and more specific file data to push to the database for training. Gathering this type of data will get us better quality data.

- Spent the week looking through projects and testing them on my python script to see which ones produced the right results we need for further testing. The original test produced poor

results, but after making some changes to the script, the results have proved to be close to what GitHub showed as their results. Continued testing will be done to determine the validity of my results.

Summer..

(Fall)

- Redesign Database to capture more specific data. We want to collect a good 50/50 mix of “good” and “bad” data. We will use csv files containing github repositories with commit hashing containing files we know are bad. Then we find a “good” file from the same repository to add to the database

- An Algorithm that grabbed the averages meta-data from each file both “good” and “bad” by going through the commit log history for that repository.

- New data collection methods were added to the backend development folder. Found new features pertaining to file specific meta-data. This includes number of lines in a file, the number of words in a file, number of indentations in a file and number of subdirectories in a repo.

- After all parts of the pipeline were functional, an effort was made to develop further file-based features which necessitated the a new database, based solely on file fields and dropping the repo fields.

Tools to Support Data Acquisition

Several tools were used for dealing with the data, some more than others but using these tools helped make this whole process much easier. We focused on PyCharm as the main IDE of choice, with built-in debugging and testing tools with a pre-installed virtual environment to create local installations of the libraries we used. We used GitHub as the main source of data collection and the wampserver as a testing tool for database creation and connection on my local computer.

PyCharm IDE
Developers: JetBrains
Main use: Code development, Debugging, Testing
Features:

- Intelligent Python Assistance
- Scientific Tools
- Cross-technology Development
- Remote Development Capabilities
- Built-in Developer Tools

We decided PyCharm was the best choice for editing and debugging our code most because it was up to date and was designed around python as the main language to use, and since we are mainly using python as our only language for this program, it was a simple decision to make.

Anaconda
Developers: Anaconda Inc.
Main use: Data Analysis, Python
Features:

- Data Analysis
- Documentation
- Community support
- Training

Anaconda was initially thought of a useful way to edit and run out code, through the use of jupyter notebooks for its live coding ability. However, using a simple IDE which had all the tools we needed gave us better results and was significantly easier to use especially when collaborating with people who don't own or use Anaconda and its extended tools.

Visual Studio Code
Developers: Microsoft Corporation

Main Use: Testing and Code Editing
Features:

- IntelliSense
- Debugging
- Built-in Git
- Extensions

Visual Studio Code was also considered for use as the main editor for this program; however, it was designed for simple programs and wasn't specifically designed for one specific language. Therefore, it had a lot of addons and extensions that made it confusing to know what was being used. It is still a great editor that we highly recommend.

Github
Founders: Tom Preston-Werner, Chris Wanstrath, Scott Chacon, P. J. Hyett
Main Use: Data Collection
Features:

- Github repository
- Projects
- Large collection of data

Github was the main source for our data collection. It had a well-organized API, and a lot of companies and users use this as a place to store their projects. Therefore, it gives us endless possibilities for data.

Wampserver
Author: Romain Bourdon
Main Use: Testing database connection and population
Features:

- phpMyAdmin
- Apache web server
- OpenSSL
- MySQL database
- PHP programming language

Wampserver was a tool used to help with understanding databases, and it allowed us to create our local MySQL databases to run tests on with the actual program to make sure we were able to connect properly and to make sure we were able to link the data up to the database itself as well.

Dropbox

Developers: Dropbox, Inc.
Main Use: Store Files and Testing git connections
Features:

- Cloud storage
- Online backup Service

The Dropbox is merely used for easy storage and access across multiple computers, it allows us to access all the files we want locally from any computer as long as we have access to the dropbox account and or if it is already downloaded on the local computer. We thought this was better than having important files in a single desktop folder on one local computer.

Libraries

Pipenv - It automatically creates and manages a virtualenv for your projects, as well as adds/removes packages from your Pipfile as you install/uninstall packages. It also generates the ever-important Pipfile.lock, which is used to produce deterministic builds.

Commands:

Shell - terminal shell with virtual environment
Run - run given command with any arguments forwarded
Check - checks for security vulnerabilities
Graph - shows dependency graph of installed dependencies
Help - run “-h” or “--help” for a complete list in terminal window

Module links:

https://pipenv.readthedocs.io/en/latest/ - python virtual environment
https://github.com/pypa/pipenv - python virtual environment

Installation & use:

Check links for complete documentation on the use of pipenv. To install pipenv in a directory go to project directory and run

Terminal :: pipenv install

This will create a virtual environment in that directory and upload a pipfile and pipfile.lock to the same directory. To download modules to the specific project run

Terminal :: pipenv install {module name}

This will install module and update pipfiles accordingly

PyGitHub - Get access to Github APIv3. It enables management of GitHub data such as repositories, user profile information and organization in PyAppliation.

Module links:

https://media.readthedocs.org/pdf/pygithub/stable/pygithub.pdf - python git data collector
https://github.com/PyGithub/PyGithub - python git data collector

Installation & use:

Terminal :: pipenv install PyGithub
Program :: from github import Github

```
# create github instance
Git = github("access_token") # use access token instead of user & pass

# Play with github objects
For repo in g.get_user().get_repo()
    print(repo.name)
```

Types of Data:

* See GitHub objects in Doc

1) Repository
   - Stargazers,issues,labels, top refers, top contents, clones, views/breakdown
2) Branch
   - Branch list, protection status
3) Commits
   - Get commit dates
4) Pull Requests
   - Get pull requests by query
5) Milestones/Webhooks
   - Get milestone and create and listen to webhooks

MySQL - an open source relational database management system.

Module links:

https://www.w3schools.com/python/python_mysql_getstarted.asp - MySQL
https://www.mysql.com/ - MySQL

Installation & use:

Terminal :: pipenv install mysql
Program :: import mysql.connector #import mysql connection

```
# connect to database
mydb = mysql.connector.connect(
        host="localhost",
        user="yourusername",
        password="yourpassword"
        )

print(mydb)
```

GitPython- GitPython is a python library used to interact with git repositories, high-level like git-porcelain, or low-level like git-plumbing.

Module links:

https://gitpython.readthedocs.io/en/stable/intro.html - *GitPython*

Installation & use:

Terminal :: pipenv install gitpython
Program :: from git import Repo

```
# rorepo is a Repo instance pointing to the git-python repository.
# For all you know, the first argument to Repo is a path to the repository
# you want to work with
repo = Repo(self.rorepo.working_tree_dir)
assert not repo.bare
```

Requests - allows you to send *organic*, *grass-fed* HTTP/1.1 requests, without the need for manual labor. There's no need to manually add query strings to your URLs, or to form-encode your POST data.

We thought this library would have been useful in the begin as we thought we would be scraping through websites for data but as we progressed we realized scraping websites wasn't something we needed as a lot of the data we needed could be pulled through an API.

The libraries listed below were all possible ways of connecting with Github's API however we realized, for the most part, all the libraries were about the same, and it didn't make a whole lot of difference in terms of which one we used. We assumed it was more of preference at that point as to which one you were comfortable using and implementing.

Libsaas - A library to take the pain out of using SaaS APIs.
It provides an abstraction layer on top of various APIs, taking care of constructing the URLs, serializing parameters and authentication. You just call Python methods and receive Python objects. It's like an ORM for SaaS!
Libsaas is built by Ducksboard and distributed under the MIT license. You can file bugs in the issue tracker, browse the documentation or help out by contributing support for new services.

Github3- github3.py is wrapper for the GitHub API written in python. The design of github3.py is centered around having a logical organization of the methods needed to interact with the API.

agithub- agithub is a REST API client with transparent syntax which facilitates rapid prototyping — on *any* REST API

The most striking quality of AGitHub is how closely its syntax emulates HTTP. In fact, you might find it even more convenient than HTTP, and almost as general (as far as REST APIs go, anyway). The examples below tend to use the GitHub API as a reference point, but it is no less easy to use agithub with, say, the Facebook Graph.

OctoHub - OctoHub is a Python package that provides a low level interface to the full GitHub v3 API:

OctoHub was developed out of a need to have a one-to-one interface to the GitHub API based on the excellent online documentation, with the least amount of abstraction.

OctoHub does do its part by parsing raw json responses into Pythonic attribute dictionaries, as well providing an optional iterative Pager for handling pagination.

Also included is a command line interface for quick interaction with GitHub's API.

Application Development

Tools to Support Application Development

Visual Studio 2019
Developers: Microsoft
Main use: Code development, debugging, testing
Features:

- IntelliSense
- Unit testing tools (for TravisCI)
- Cross-technology development (using Mono and .NET Core)
- VCS (GitHub) integration

JetBrains Rider
Developers: JetBrains s.r.o.
Main use: Code analysis, code editing, unit testing, debugging
Features:

- Runs cross-platform (for Linux and Mac OSX development)
- Very powerful refactoring
- Unit testing tools and runners (for TravisCI)
- Cross-technology development (using Mono and .NET Core)
- VCS (GitHub) integration

GitHub
Founders: Tom Preston-Werner, Chris Wanstrath, Scott Chacon, P. J. Hyett
Main Use: Remote source versioning
Features:

- Group Version Control System (VCS)
- KanBan “goal” Board
- Integration with TravisCI

GitHub Desktop
Developers: GitHub, Inc.
Main Use: Ease-of-Use desktop application for GitHub collaboration
Features:

- Commit attribution with collaborators
- Easy branch checkouts, pulling/pushing to origin, and diff checking
- Full CI status checks (for use with Travis)

Discord
Developers: Discord Inc.
Main use: Voice and text communication
Features:

- Voice chat using voice channels
- Text chat through text channels
- Important date and message pinning (for meeting dates and deliverable due dates)
- GitHub webhook integration (for repository updates)

Travis CI
Developers: Travis CI community
Main use: Continuous Integration service
Features:

- On-Demand code builds
- On-Demand code testing
- Direct integration from GitHub (automatic building)
  Private repository access for builds and tests

Libraries

LiveCharts - "A .NET library for C# WPF and WinForms that allows simple, elegant, and interactive live charts within desktop applications." Allows for data manipulation and real-time data tracking.

Important links:

https://lvcharts.net/ - Live Charts project homepage
https://github.com/Live-Charts/Live-Charts - Live Charts GitHub repository (source)
https://lvcharts.net/App/examples/v1/wf/Install - Live Charts API documentation

Installation:

Using Visual Studio "NuGet Package Manager", you can install the library using the following command (for WinForms): **Install-Package LiveCharts.WinForms**

Then proceed to right-click on your project solution and click "Manage NuGet Packages…" and make sure that all of the Live Chart packages are installed through NuGet.

Add the Live Chart namespaces to your project using the following code:

```
using LiveCharts; //Core of the library
using LiveCharts.Wpf; //The WPF controls
using LiveCharts.WinForms //The WinForm wrappers
```

Build your project.

Right-Click on your project toolbox and add the Live Charts components by selecting the .dll files created after the project build.

Use:

After adding a Live Charts component to your project, you can interact with it within the code or within the component editor in the [Design] menu. An example pie chart can be made using the following sample code:

```
public partial class PieChartExample : Form
    {
        public PieChartExample()
        {
            InitializeComponent();

            Func<ChartPoint, string> labelPoint = chartPoint =>
                string.Format("{0} ({1:P})", chartPoint.Y,
chartPoint.Participation);

            pieChart1.Series = new SeriesCollection
            {
                new PieSeries
                {
                    Title = "Maria",
                    Values = new ChartValues<double> {3},
                    PushOut = 15,
                    DataLabels = true,
                    LabelPoint = labelPoint
                },
                new PieSeries
                {
                    Title = "Charles",
                    Values = new ChartValues<double> {4},
                    DataLabels = true,
                    LabelPoint = labelPoint
                }
            };
```

```
            pieChart1.LegendLocation = LegendLocation.Bottom;
        }
    }
```

OctoKit - "A client library targeting .NET 4.5 and above that provides an easy way to interact with the GitHub API." OctoKit is the de-facto GitHub API wrapper for C# and .NET as a whole. The library is made and fully supported by GitHub, Inc.

Important links:

https://github.com/octokit/octokit.net - OctoKit GitHub repository (source)
https://octokitnet.readthedocs.io/en/latest/ - OctoKit Read-the-Docs (API)

Installation:

Install the OctoKit package and apply it to your project by entering **Install-Package Octokit** into the NuGet Package Manager.

Use:

After the OctoKit package has been installed, you can create and modify objects with ease, here is some sample code:

```
async Task Main(string[] args)
{
       var owner = string.Empty;
       var reponame = string.Empty;

       owner = "octokit";
       reponame = "octokit.net";

       var client = new GitHubClient(new
Octokit.ProductHeaderValue("octokit.samples"));

       var repository = await client.Repository.Get(owner, reponame);

       Console.WriteLine(String.Format("Octokit.net can be found at {0}\n",
repository.HtmlUrl));

       Console.WriteLine("It currently has {0} watchers and {1} forks\n",
              repository.StargazersCount,
           repository.ForksCount);

       Console.WriteLine("It has {0} open issues\n",
```

```
repository.OpenIssuesCount);

      Console.WriteLine("And GitHub thinks it is a {0} project",
repository.Language);
}
```

LibGit2Sharp - "LibGit2Sharp brings all the might and speed of libgit2, a native Git implementation, to the managed world of .NET and Mono.". LibGit2Sharp allows C# developers to interact directly with git and .git files, with the ability to clone and manage entire repositories.

Important links:

https://github.com/libgit2/libgit2sharp - LibGit2Sharp repository (source)
https://github.com/libgit2/libgit2sharp/wiki - LibGit2Sharp wiki (documentation)

Installation:

Install the LibGit2Sharp package and apply it to your project by entering **Install-Package LibGit2Sharp** into the NuGet Package Manager.

Use:

After the LibGit2Sharp package has been installed, you can clone a repository by doing the following:

```
Repository.Clone("https://github.com/libgit2/libgit2sharp.git",
"path/to/repo");
```

Code to Support Application Development

TravisCI Build Configuration:

```
before_install: cd Application

language: csharp
mono: none
sudo: required
dist: xenial
dotnet: 2.2

solution: PSV.sln

script:
 - dotnet restore
 - dotnet test
```

Application-to-Backend Interfacing Example Code:

```
// Setup the base for our python environment and scripts
string pythonInterpreter = "python";
string pythonScript = @"..\..\..\..\Backend\app_interface.py";
ProcessStartInfo pythonStartInfo;

// For local repositories
// Supply the local repo path
pythonStartInfo = new ProcessStartInfo
{
    FileName = pythonInterpreter,
    // Use the localPathTextbox here because the repo is already cloned
    Arguments = string.Format("{0} {1}", pythonScript, localPathTextbox.Text),
    UseShellExecute = false,
    RedirectStandardOutput = true
};

// For remote, public repositories
// Supply the path where it is cloned to and the name of the repo
pythonStartInfo = new ProcessStartInfo
```

```
{
    FileName = pythonInterpreter,
    Arguments = string.Format("{0} {1} {2}", pythonScript, combinedPath, repoName),
    UseShellExecute = false,
    RedirectStandardOutput = true
};

// For remote, private repositories
// Supply the path, the name, the user's GitHub username & password for
authentication
pythonStartInfo = new ProcessStartInfo
{
    FileName = pythonInterpreter,
    Arguments = string.Format("{0} {1} {2} {3} {4}", pythonScript, combinedPath,
repoName, githubUsername, githubPassword),
    UseShellExecute = false,
    RedirectStandardOutput = true
};

// Start the process and redirect its output stream
Process pythonProcess = Process.Start(pythonStartInfo);
StreamReader pythonOutStream = pythonProcess.StandardOutput;

// Wait until the scan is finished
pythonProcess.WaitForExit();
```

Application Specifications and Design Choices

Application Specifications

Built using .NET Framework 4.6.1
WinForm Windows Application
Able to run on .NET Core 2.0+ (cross-platform capabilities such a GNU/Linux and Mac OSX)
Maintainability Index: 71*
Cyclomatic Complexity: 68*
Depth of Inheritance: 7*
Class Coupling: 109*
Lines of Code: 1,423*

These specifications and metrics are up-to-date as of December 1st, 2019.
* Metrics obtained from Microsoft Visual Studio 2019's "Code Metrics" feature.

Application Design Choices and Discussion

Why A Desktop Application Over a Web Application

When thinking of possible solutions for the problem proposed (whether or not it is possible to predict security faults located within program solely given code metrics), one thing was certain, that a front-end user application of some sort was necessary. Regardless of what research we had done, it was absolutely necessary that a user could input data into some form of an application, and receive an output detailing the security faults within their projects. Our immediate thoughts were between a web application and a desktop application, two of the most common forms of software that exist in today's age. Originally, we had figured that a web application would be more viable. Web applications allow multi-platform access given that they run within a browser, can utilize bleeding-edge technologies to work quickly and effectively, and are simply the more conventional form of modern software. While mostly all of us had web application development experience, there were two crucial points that threw a wrench into the idea of making a web application and pushed us to create desktop application instead.

The first point being privacy and the second being efficiency. Regarding privacy, a web application has many measures to ensure the privacy of their users and the privacy of the data of their users. However, what were to happen if the project being examined were classified, or otherwise private in some way? Surely no confidential entity would allow secret or private data to pass through a third-party website? Our desktop application solution would allow anyone to simply download the application and communicate between themselves and their project, and *only* between them and their project. Without a third-party involved the user is free to do what they wish and not have to

worry about privacy or authentication. No credentials, confidential code, or other secrets passing through us or our servers. The second point, again, is efficiency. Given that the application must clone and comb through a repository for information, it is much simpler to house the data on the user's computer instead of our servers. The privacy argument proposed above can also apply to this. Instead of having to repeatedly clone, comb through, and delete hundreds of thousands of files queried by our users, we can simply leave it up to the user to do what they'd like with their data. While some may avoid developing a desktop application at all costs in favor of a more modern approach such as a web application, our thoughts regarding users having access and control over their data come first. It is simply more feasible and makes more sense to develop a desktop application in regards to a research project such as this given the constraints and requirements.

Plans Regarding Updating the Model

Machine learning models become smarter and smarter as they are trained more and more as time goes on. Our model is no different. Given that the desktop application is standalone, we must provide our users with some way to use the most up-to-date model to ensure that they receive the highest quality vulnerability reports possible. We plan on including a section within the application, easily accessible by the end-user, that allows the user to check the version of their currently downloaded model, check for updates for newer models, and download a new model if one exists. The model will be hosted at a static endpoint on our server (an example being something such as "http://<domain>.com/newestmodel") and the application will (with the user's approval) download the most up-to-date model. The connection to our server will be made using HTTPS (over TLS) and the application will have checksum checks to verify that the downloaded model is indeed a legitimate copy. An example walkthrough would be that you have just downloaded the program, you open up the program, and are notified that a new version of the model is available for download. After clicking the "update" button, you ask the server for the latest model over a secure connection. The server processes your request and sends the model to you, which then the application processes and loads. As the application loads the model, it verifies that the information sent by the server is legitimate and that no tampering has been done by a third-party (to the best of its abilities). If everything looks to be correct, the new model is loaded into the program and you are free to begin a new scan. The application will ship with the latest model when downloaded in an effort to help the user stay up-to-date.

Known Unknowns

There exist some specifics regarding the scanning process that we have labeled as "known unknowns", or specific details that we will not have knowledge regarding their effectiveness until thorough testing and training. One such topic is scanning settings. In the application design

mockups and prototypes provided in previous sections, one may notice that the “scan settings” section of the program is left blank. This is by no accident. The truth is that given how some extremely esoteric features in a machine learning model can correlate to positive results, it is not possible to entirely determine what settings we can allows the end-user to tweak when scanning their application. Some proposed settings could be whether to scan files of all types or specific types, to ignore specific files, or to use a specific metric that isn’t included in the scan by default. Another unknown is how the application will be able to communicate with the models in regards to exact specifics such as whether to scan a file in its entirety using the general model, or to look for line-specific faults using the granular model. These unknowns are not critical, but we believe it is good to acknowledge what topics we are still looking into and which topics we believe may give us more trouble than others.

Known Unknowns Post-Mortem

After having spent the better part of a year on this project, both of the aforementioned “known unknowns” ended up becoming “known”. Regarding the scanning settings, we decided to keep the settings minimal, allowing for the excluding of hidden items in directories, specific files, specific folders, and specific file extensions. With these four settings users are able to narrow down exactly what they want scanned and what they don't want scanned fairly easily and effectively. As for the application communication, this was simply done by starting a Python process and sending to the backend scripts any information that was necessary in the form of command line arguments. Once the backend has finished its scan, it prints its output to console, and the application parses that output and displays it to the user. The granular model and line-specific fault finding mentioned above was not completed due to time restraints.

Non-Goals

It is important to provide a broad list of non-goals that the application does not aim to accomplish in an effort to be as transparent as possible with what the team has in mind for the project and application. Some non-goals that this project and application will *not* aim to accomplish (due to being out of scope or not a requirement) are as follows:

- This application will not use static or dynamic analysis, or specific code snippets to predict the existence of security faults within code.
- This application will not serve to fix or otherwise provide exact information on how to fix (other than perhaps a reference) any vulnerabilities found within any projects.
- This application does not serve to replace any existing software security testing suite or application, only to be run supplementary with existing testing suites.

Project Testing Plan

Testing Overview

A key element to testing will be in training the model and refining it upon feedback from the architecture design process and ensuring that data fields are yielding good results when viewed holistically in the context of the modeling process. To this effect, training sets will be separate from testing sets, so that testing is not influenced by the training that has built the model. Further, one modeling milestone is the ability to run the model on a previously unseen GitHub repository and get accurate predictions about where faults lie.

The data collection role can test the effectiveness and reliability of their utility by running the tool on a small demo repository, then verifying that the data stored in the database matches the expected values that should be seen for that repository.

Backend testing has begun by utilizing a program that injects many test records into the database, showing that the database can accommodate a large volume of ongoing operations, and that the design of the database can efficiently store information. 10,000 records in the file table resulted in only 2.72 MB of disk usage. The administration of the droplet is handled by Digital Ocean's NYC1 datacenter, which provides uptime and availability guarantees to ensure our database is always available.

Application development has utilized TravisCI, which is continuous integration and testing software. The following section lays out detailed plans for unit and integration tests that will be able to quickly indicate the status of the software and verify that the functionality is unhampered by any bugs or faults.

Application Unit and Integration Tests

Test Case 1 (Unit Test) - IsGitURLValidTest

- Purpose:
  - To ensure that GitHub URL validity testing works
- Test Description:
  - Fills the projectURL textbox with an example (valid) GitHub URL and queries the IsGitURLValid method to test the validity.
  - IsGitURLValid() then attempts to list the remote references to the remote repository to verify that it indeed exists.
  - Also attempts to check the validity of a non-existent to verify that the test works both ways.
- Expected Results:
  - Will return True because the GitHub URL exists which, confirms that the check works.
  - Will assert a False for the non-existent URL, which is then expected to return False.

Current Test Code:

```
        [TestCategory("Unit")]
        [TestMethod()]
        public void IsGitURLValidTest()
        {
            PSVForm testForm = new PSVForm();
            testForm.projectURLTextBox.Text = "https://github.com/octocat/Hello-
World.git";
            Assert.IsTrue(testForm.IsGitURLValid());
            testForm.projectURLTextBox.Text = "https://github.com/octocat/Non-
Existent-Project.git";
            Assert.IsFalse(testForm.IsGitURLValid());
        }
```

Test Case 2 (Unit Test) - IsPathValidTest

- Purpose:
  - To ensure that path validity testing works
- Test Description:
  - Creates a new test directory called CloneTest and sets the pathToClone textbox text to the newly created local path.
  - IsPathValid() then attempts to verify that the directory exists and is empty.
  - The newly created path is then deleted as it is no longer used.
- Expected Results:

- Will return True due to the new path existing and being empty.
- Will throw a DirectoryNotFoundException after the directory is deleted.

Current Test Code:

```
        [TestCategory("Unit")]
        [TestMethod()]
        public void IsPathValidTest()
        {
            PSVForm testForm = new PSVForm();

            // Create a new path to test
            // Test that the path is valid (exists and non-empty)
            string testPath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Desktop),
"CloneTest\\");
            Debug.WriteLine(testPath);
            DirectoryInfo testDirectory = Directory.CreateDirectory(testPath);
            testForm.pathToCloneTextBox.Text = testDirectory.FullName;
            Assert.IsTrue(testForm.IsPathValid());

            // Delete the directory
            // Test that the non-existent directory is indeed non-existent
            testDirectory.Delete(true);
            _ = Assert.ThrowsException<DirectoryNotFoundException>(() =>
testForm.IsPathValid());
}
```

Test Case 3 (Unit Test) - AreScanSettingsValidTest

- Purpose:
  - To ensure that the scan settings are valid and do not conflict
- Test Description:
  - Set specific, valid, scan settings for an example scan and attempts to start the scan, verifying that the settings are indeed valid.
  - Set specific, invalid, scan settings for an example scan and attempts to start the scan, verifying that the settings are not valid.
- Expected Results:
  - Will return True when the scan settings don't conflict with each other or the scan.
  - Will return False when the scan settings do conflict with each other or the scan.

Current Test Code:

```
      [TestCategory("Unit")]
```

```
	[TestMethod()]
	public void AreScanSettingsValidTest()
	{
		PSVForm testForm = new PSVForm();

		// Test valid settings
		testForm.fileNamesBox.Text = "file1.txt, file2.c, file3.java";
		testForm.fileExtensionsBox.Text = ".cpp, .h, .bat";
		testForm.foldersBox.Text = "folder1, folder2, folder3";

		// Set the local path
		testForm.localRadio.Checked = true;
		testForm.localPathTextbox.Text =
Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
		Debug.WriteLine("Path: " + testForm.localPathTextbox.Text);

		testForm.UpdateScanSettings();

		// Verify settings
		Assert.IsTrue(string.Join(",", testForm.fileNameExclusions).Length ==
testForm.fileNamesBox.Text.Length);
		Assert.IsTrue(string.Join(",",
testForm.fileExtensionExclusions).Length ==
testForm.fileExtensionsBox.Text.Length);
		Assert.IsTrue(string.Join(",", testForm.folderExclusions).Length ==
testForm.foldersBox.Text.Length);
	}
```

Test Case 4 (Integration Test) - BeginScanTest

- Purpose:
  - To ensure that the entire scanning process works and that the previous three tests are testing properly.
- Test Description:
  - Runs the previous three tests in-tandem and attempts to simulate the entire scanning process.
- Expected Results:
  - All tests “pass” in their respective ways and the program successfully scans an example project.

Current Test Code:

```
[TestCategory("Integration")]
[TestMethod()]
public void BeginScanTest()
{
	IsGitURLValidTest();
	IsPathValidTest();
	AreScanSettingsValidTest();
}
```

Test Case 5 (Unit Test) - VulnerableFileTreePopulationTest

- Purpose:
  - To ensure that the scan correctly discovered vulnerable files and that they are populated within the “Vulnerable File Tree” properly.
- Test Description:
  - Ensure that BeginScanTest runs properly and continue the scanning process simulation.
  - For each marked “vulnerable” file in the scan, add the file to the Vulnerable File Tree.
- Expected Results:
  - Fills the Vulnerable File Tree with specifically marked files that are known to be “vulnerable”.

Current Test Code:

This unit test was not included due to the “Vulnerable File Tree” UI component being removed

Test Case 6 (Unit Test) - VulnerableTableTest

- Purpose:
  - To ensure that the Vulnerable Table is populated correctly from the files marked as “vulnerable” in the Vulnerable File Tree. Including the severity, vulnerability, file, and date/time.
- Test Description:
  - Take the vulnerable file information from the scanner to correctly mark and label files in the Vulnerable Table.

    - Populate the Vulnerable Table with all “vulnerable” files and include information regarding their severity, vulnerability, file name, date/time.
- Expected Results:
    - Populates the Vulnerable Table with specifically marked files that are known to be “vulnerable”.

Current Test Code:

```
      [TestCategory("Unit")]
      [TestMethod()]
      public void VulnerableTableTest()
      {
             // Perform the full process for scanning
             PSVForm testForm = new PSVForm();
             testForm.remoteRadio.Checked = true;
             testForm.projectURLTextBox.Text =
"https://github.com/Bonfire/Legionary.git";
             testForm.pathToCloneTextBox.Text =
Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
             testForm.cloneProject();
             testForm.loadProject();
             testForm.beginScan();

             // Arbitrary file and prediction
             Assert.IsTrue(testForm.faultListView.Items[2].Text == "Pipfile");

      Assert.IsTrue(Convert.ToDouble(testForm.faultListView.Items[2].SubItems[1].T
ext) == 0.24834655);
      }
```

Test Case 7 (Unit Test) - VulnerableGraphTest

- Purpose:
    - To ensure that the Vulnerable Graph is populated correctly from the files marked as “vulnerable” in the Vulnerable File Tree. Including vulnerability type and count.
- Test Description:
    - Take the vulnerable file information from the scanner to correctly add and count vulnerabilities in the Vulnerable Graph.
    - Populate the Vulnerable Graph with all vulnerabilities and include information regarding the type of vulnerability and the count.

- Expected Results:
    - Populates the Vulnerable Graph with specifically marked vulnerabilities and their counts.

Current Test Code:

```
      [TestCategory("Unit")]
      [TestMethod()]
      public void VulnerableGraphTest()
      {
            // Perform the full process for scanning
            PSVForm testForm = new PSVForm();
            testForm.remoteRadio.Checked = true;
            testForm.projectURLTextBox.Text =
"https://github.com/Bonfire/Legionary.git";
            testForm.pathToCloneTextBox.Text =
Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
            testForm.cloneProject();
            testForm.loadProject();
            testForm.beginScan();

            // Assert that there were 4 files scanned and populated
            Assert.IsTrue(testForm.faultChart.Series[0].Values.Count == 4);
      }
```

Test Case 8 (Integration Test) - ScanAndOutputTest

- Purpose:
    - To ensure that the entire scanning and output process works and that the previous three tests are testing properly. This test will also ensure that the previous integration test (BeginScanTest) is testing properly
- Test Description:
    - Runs the previous three tests in-tandem and attempts to simulate the entire scanning process as well as the output generated from the scan.
- Expected Results:
    - All tests “pass” in their respective ways and the program successfully scans an example project and provides correct output for that specific scan.

Current Test Code:

```
	[TestCategory("Integration")]
	[TestMethod()]
	public void ScanAndOutputTest()
	{
		VulnerableTableTest();
		MessageBox.Show("Delete path", "Notice", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
		VulnerableGraphTest();
	}
```

Modeling

Libraries to Support Modeling

TensorFlow - An open-source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

Important links:

https://www.tensorflow.org/ - TensorFlow Website
https://www.tensorflow.org/guide - TensorFlow Tutorial Guide
https://www.tensorflow.org/api_docs - TensorFlow Api Documentation
https://github.com/tensorflow/ - TensorFlow GitHub Repository (source)

Installation:

Linux installation through Python package manager Pip
1. Install the Python development environment on the system.

```
sudo apt update
sudo apt install python3-dev python3-pip
```

2. Install the TensorFlow pip package.

```
$ pip3 install --user --upgrade tensorflow # install in $HOME
```

Windows installation through Python package manager Pip
1. Install the *Microsoft Visual C++ 2015 Redistributable Update 3*. This comes with *Visual Studio 2015* but can be installed separately:

1. Go to the Visual Studio downloads,
2. Select *Redistributables and Build Tools*,
3. Download and install the *Microsoft Visual C++ 2015 Redistributable Update 3*.

Make sure long paths are enabled on Windows.
Install the *64-bit* Python 3 release for Windows (select pip as an optional feature).

2. Install the TensorFlow pip package.

```
$ pip3 install --user --upgrade tensorflow # install in $HOME
```

Use:

In the python file being used for modeling, you will import the tensorflow package.

```
import tensorflow as tf
```

From here you can use the tensorflow library to build, optimize, and test your model.

Numpy - A fundamental package for scientific computing with Python. Numpy has the ability to create powerful N-dimensional array objects and use linear algebra to manipulate data. This is important for vectorization within the machine learning modeling.

Important links:

https://www.numpy.org/ - NumPy Website
https://docs.scipy.org/doc/ - Numpy Documentation
https://github.com/numpy - Numpy GitHub Repository (source)

Installation:

Using Python package manager Pip

```
python -m pip install --user numpy
```

Install system-wide via a Linux package manager

```
sudo apt-get install python-numpy
```

Use:

In the python file being used for modeling, you will import the numpy package.

```
import numpy as np
```

Scikit-learn - Scikit-learn is an open source software machine learning library for the Python programming language. It provides simple and efficient tools for data mining and data analysis. Scikit-learn is built on NumPy, SciPy, and matplotlib and will be useful preprocessing data by encoding categorical data into numerical representation.

Important links:

https://scikit-learn.org/stable/index.html - Scikit-Learn Website
https://scikit-learn.org/stable/documentation.html - Scikit-Learn Documentation
https://github.com/scikit-learn/scikit-learn - Scikit-Learn GitHub Repository (source)

Installation:

Using Python package manager Pip

```
pip install -U scikit-learn
```

Using conda:

```
conda install scikit-learn
```

Use:

In the python file being used for modeling, you will import the preprocessing library from scikit-learn package.

```
from sklearn import preprocessing
```

Keras - Keras is an open-source neural network API for Python that can run on top of TensorFlow. It's designed to be user-friendly and modular. The tools Keras provides make implementing neural networks with TensorFlow much faster and easier.

Important links:

https://keras.io/ - Keras Website & Documentation
https://blog.keras.io/category/tutorials.html - Keras Tutorials
https://github.com/keras-team/keras/ - Keras GitHub Repository (source)

Installation:

Using Python package manager Pip:

```
pip install -U keras
```

Use:

Using Keras with TensorFlow:

```
import tensorflow as tf
from tensorflow.keras import layers
```

Code to Generate Example Data for Model Training

```
connection = mysql.connector.connect(host=config.host, database=config.database,
user=config.username, password=config.password)

for record in range(numNewRecords):
	repoID = random.randint(1, 5)
	filename = ''.join(random.choices(string.ascii_letters + string.digits,
k=64))
	hash = uuid.uuid4().hex
	trainingFlag = random.randint(0, 1)
	hasFault = random.randint(0, 1)
	committedDateTime = fake.date_time().isoformat().replace("T", " ")
	commitCount = random.randint(1, 999999999) / 100
	commitSize = random.randint(1, 999999999) / 100
	insertions = random.randint(1, 999999999) / 100
	deletions = random.randint(1, 999999999) / 100
	linesChanged = insertions + deletions
	language = random.randint(1, 5000)
	numMethods = random.randint(1, 999999999) / 100
	numVariables = random.randint(1, 999999999) / 100
```

The above script was created to generate thousands of rows of example data that can then be inserted into the database located on our server. With this “garbage” data - “garbage” as in not obtained from a genuine source - the machine learning team can begin their work creating a model without having to wait for the entirety of the data acquisition to be worked out. This enables all of the roles within the group to be working simultaneously instead of having to wait for each role to finish their task. The generated data is in accordance with the `files` table located in the database, and utilizes the “faker” and “uuid” libraries for random data generation, as well as the “mysql.connector” library for MySQL connection to the database.

Branding the Product

Marketing and branding are important in today's product and company driven culture. We understand the importance of choosing a name that is instantly recognizable by customers, yet resonates at a deep level about what our product really does. The team has brainstormed on ideas for names, and utilized a name generator at one point to come up with some exploratory ideas.

Moving forward with a product would require multiple names: a name for any company that would be formed to provide the product, naming the product itself, providing names to any banner features within the product so that customers can easily differentiate it from any competitors in the field, and even providing internal code names so that development can easily refer to different branches or feature sets internally.

Currently, our front-facing Application which serves as the main interface for the end-user is simply branded "PSV" for Predicting Security Vulnerabilities, which was the original name of the project. Since the modeling will be performing two evaluations at each file level, the overall model is proposed to be called "Automated Review" with the more granular results being called "Second Look." Names for the company that are popular among the team include AIsecure, Security Whale, and the team trend seems to prefer toward traditional, less "web 2.0" names.

Figure 14: Current Application Icon produced by Baran Barut, used with permission.

Names considered

- Integrify (Integrity + ify)
- Securify
- codeTegrity
- Second Look
- Automated Review
- Predicting Security Vulnerabilities
- Security.ly
- Sysafe
- Clience
- Visign
- Safetch
- Securityly
- HeySecurity
- Stacrypt
- Plexrisk
- Maxisecurity
- Flexrisk
- Zcloud
- SecurityLynx
- Extgenic
- Fasttalk
- FastD
- Strust
- SecurityVolt
- Virust
- SecurityVine
- SecureAD
- HyperAbility
- Safetyle
- Plexrise
- SleekSecurity
- Hipthint
- Ssight
- securityVue
- Virusty
- Sbase
- Cruisign
- Predic
- Cloudase
- Strusty
- SYSafety
- SecureW
- Enchfuge
- Hipthine
- Sysafevu
- Ecodeost
- Cloudast
- Codeost
- Viruse
- Ymentcub
- Winguard
- SecurityTap
- AISecure
- FastM
- SwipeSecurity
- Winguard
- Dealth
- SecurityBloom
- Suremint
- Zcload
- GiantSecurity
- Mmapline
- Econorts
- Codel
- Security Whale
- StripMetrics
- Metric Secure
- Security Whale

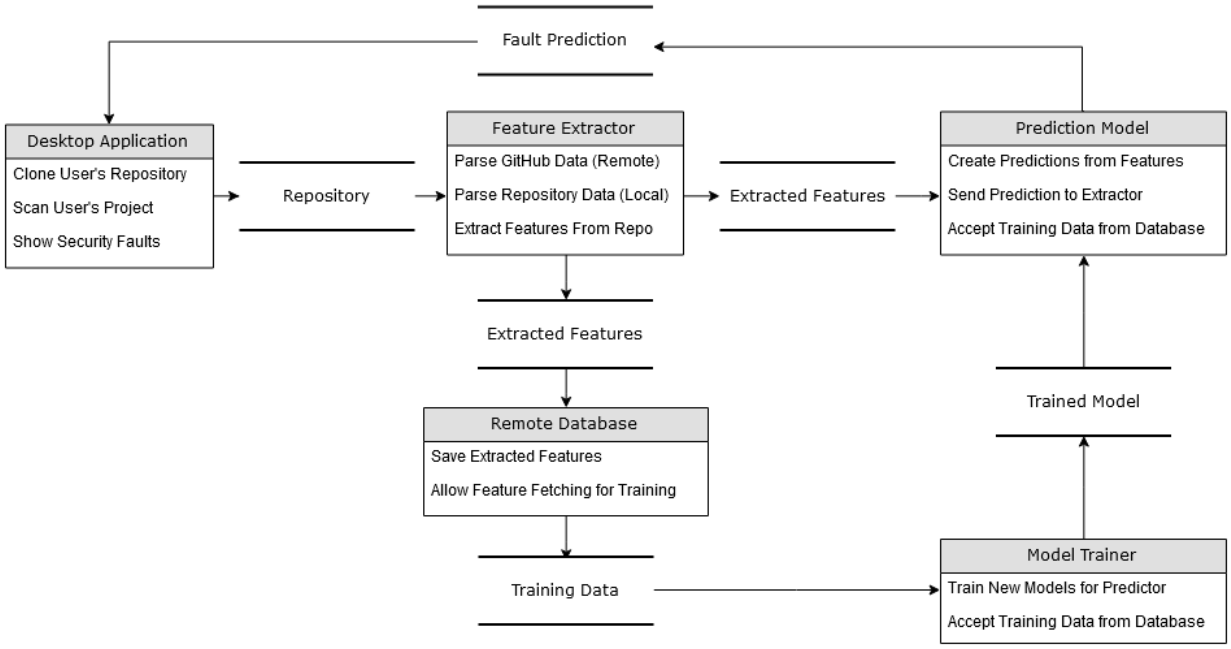
Final Candidate:

- SecurityWhale

4. Design Summary

The high-level diagram pictured below describes our project in a simple and understandable manner, with each component labeled, and each relationship between each component explicitly labeled. Following the flow of our project, the user would begin by opening the desktop application and input either a local repository, or remotely hosted repository on GitHub. They would then input their settings and load the project, then proceed to scan. The feature extractor takes any features it can from the repository and sends those features to the prediction model. After the model receives the entire project's features, it can make an accurate prediction on the data and send its predictions back to the desktop application. The user only sees the desktop application, and does not have to worry about anything other than inputting their repository and clicking "Begin Scan".

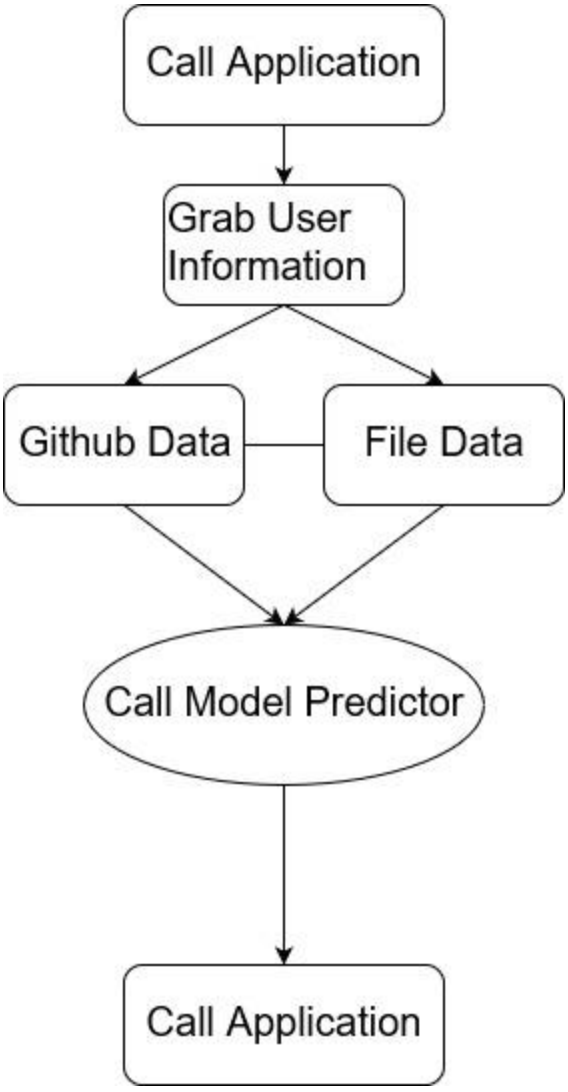
Figure 15: High-Level Diagram



Python Back-End Diagram

Figure 16 shows the flow of how the Back-end handles the information received from the user using our application. Our script calls the application gathering all the information from the user. Once the info is received the script shall collect both github data such as number of contributors and file data such as number of lines in the individual files. After all the data is collected and it is sent to the function that predicts based on the data the results of the prediction is returned back to the application for the user to see.

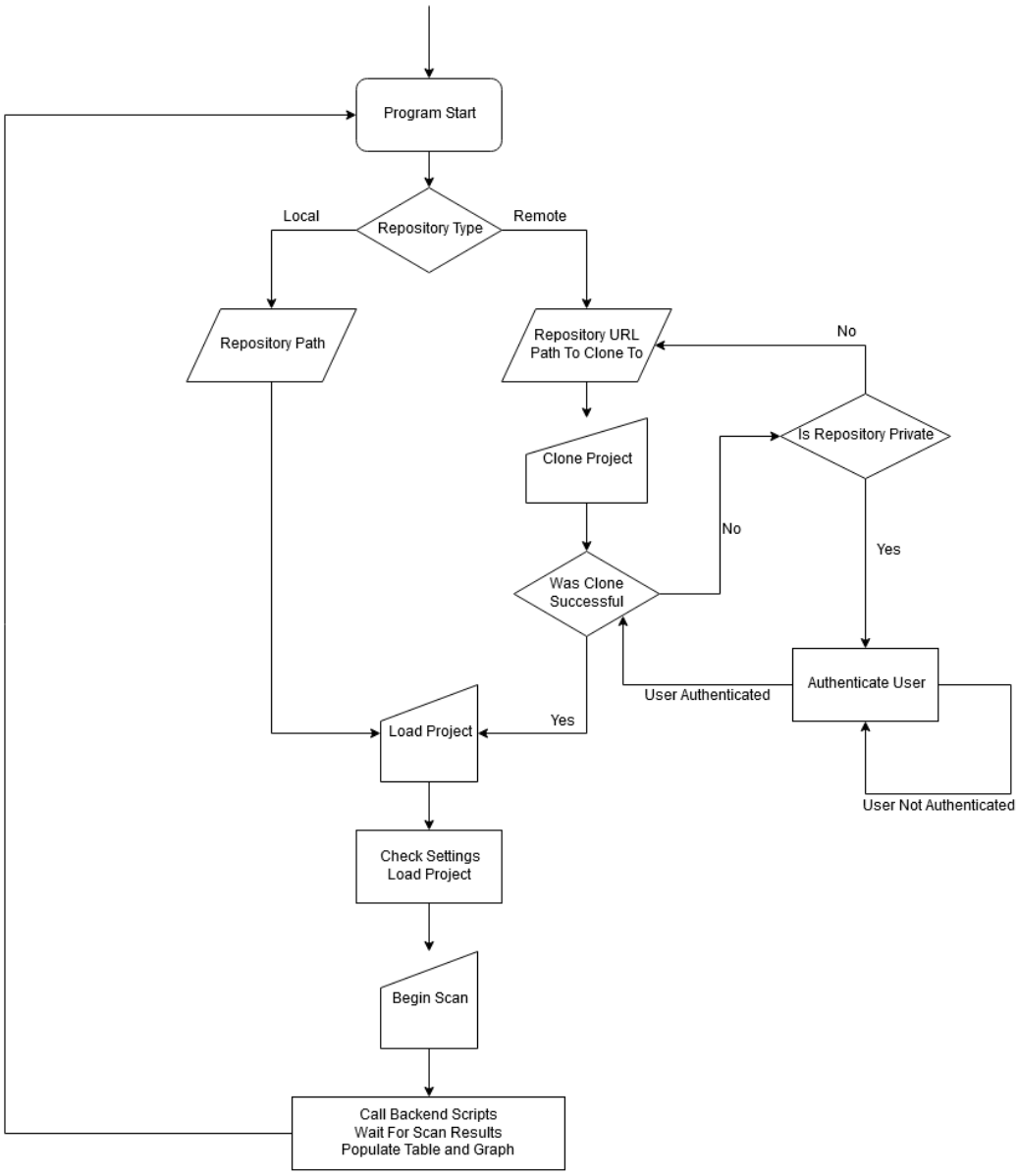
Figure 16: Python Back-End Diagram



Front-End Application Diagram

Figure 17 shows the front-end application diagram, describing in detail the control flow for the desktop application. The user is presented with the option of a local or remote repository, and then goes through the necessary steps to load the project into the program. Once the project can be loaded, the program will check their settings and load the project. After the user clicks “Begin Scan”, the program will call the backend Python scripts to begin the scanning process and waits for the scan results. When the backend provides the scan results to the program, it will take that information and populate the table and graph in the “View” tab with any relevant information.

Figure 17: Front-End Application Diagram

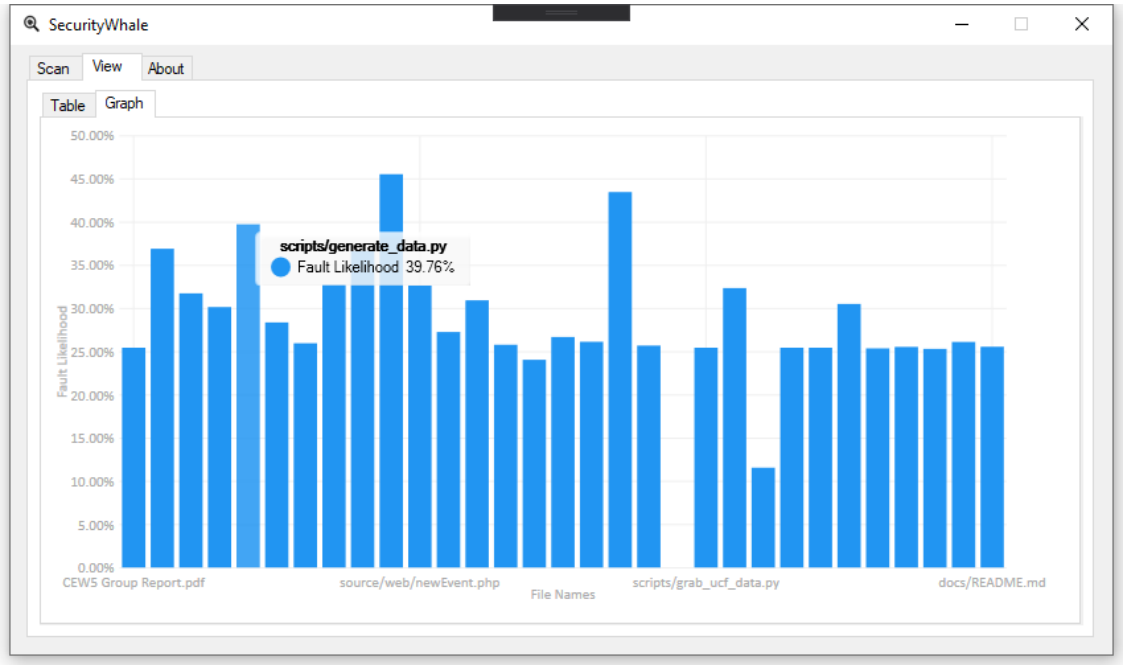


5. Results

We were able to make predictions as to where security faults may lie in projects with a 60% accuracy. On a per-file basis, our machine learning model with data pipeline backend is able to correctly predict that a file contains a security fault six-out-of-ten times. This is better than a coin flip and can be used by QA testers or software engineers to obtain a general overview as to where they should focus their security vulnerability finding efforts. We argue that our project should be used in a manner that works in-tandem with current programs and processes that aid in finding vulnerabilities in code and should not act as a replacement. By pairing our program with current vulnerability finding processes, developers can glean more information regarding the overall security posture of their projects. This increase in the amount of information at the developer's disposal is only beneficial and we hope that it helps developers in securing their projects.

Below, Figure 18 is demonstrating the output that a user would see after scanning their project with our application. As you can see, the application provides the user with a general overview, showing which files may be more problematic and which files may need to be looked at first. Due to the accuracy being "better than a coin flip", this is beneficial. We see software engineers then taking this information and aiming other security vulnerability scanners at those problematic files to dive deeper into what may actually be causing those percentages

Figure 18: Sample Output from Our Program Showing File Names and Security Fault Likelihoods



6. Conclusions

The most important – and most difficult – portion of this project was collecting features from repositories for use in the machine learning. The greatest barrier to any machine learning project is good data, and unfortunately, we didn't have enough time to further investigate these features, either by gathering new ones or discarding ones that weren't as useful. Additionally, much of the time spent working on this project was spent setting up algorithms to process and format these features and assembling the pipeline that connects each individual part of the project. However, once these tools were assembled, refinement proceeded at a much quicker rate. Being able to test every piece of the project in order greatly accelerated development, but it also created new difficulties due to having to debug a much larger program. Given even an additional month of development, many of these issues could have been ironed out and the program could likely give much more accurate predictions. We're all still very proud of our work and how the project ended up, but there is definitely room improving and even building on top of our work. This being said, ultimately, we did end up answering our question of "is it possible to make accurate predictions for security fault likelihoods using code metrics and machine learning?". We found that yes, you can make accurate predictions as to where security faults lie, and that in using our program, you can glean a high-level directional overview as to where fault may lie. Further, we discovered that file-specific data points were far more useful than ones common to the repo, since this clouded the dataset and made finding problematic files more difficult. If the aim was to detect *repos* which were likely to contain a large number of faults, considering this data would be more useful in accomplishing this objective.

7. Administrative Content

Budget

Software

5 x JetBrains Professional License - Free through the JetBrains Student Program
Digital Ocean $50 platform credit - Free through the GitHub Student Developer Pack
Private hosting of application code - Free through GitHub
Application graphing libraries - Free and Open-Source through other GitHub projects
SSL Certificate through Let's Encrypt – Free through Linux Foundation

Hardware

Use of a High-Powered GPU - Free through the Senior Design Lab
Bandwidth usage for Digital Ocean - Free up to 2000GB, $0.01/GB after per account
24/7 Droplet hosting through Digital Ocean - $10/month, for 5 months

Documentation

Printing Design Document with UCF discount - $17.22 x 2 semesters

Total Project Cost: $84.44

Milestone Dates

Senior Design I

100%: Feb 1, 2019 - Teams formed
100%: Feb 1, 2019 - Contact sponsor
100%: Feb 13, 2019 - Divide primary roles, begin extensive research
100%: Feb 20, 2019 - Operational backend through a LAMP droplet setup through Digital Ocean
100%: Feb 20, 2019 - Database up and running, data fields finalized
100%: Feb 20, 2019 - Have a basic prototype of the Application's GUI designed
100%: Feb 18, 2019 - TA check in, 8-10 pages each
100%: April 9, 2019 - TA check in, 20-25 pages each
100%: April 19, 2019 - Design Document finalized
100%: April 21, 2019 - Final Design Document printed and bound
100%: April 24, 2019 - Final Design Document due in person (HEC 433)

Senior Design II

100%: Summer - Facilitate model training
100%: August 26, 2019 - Senior Design II starts, begin research implementation
100%: September 23, 2019 - Run the model on a GitHub repo
100%: September 30, 2019 - Data collection utility complete, can push repository info to database
100%: September 30, 2019 - Application front-end working, features becoming integrated over time
100%: September 30, 2019 - Fully operational backend, connected to each module
100%: September 30, 2019 - Backend can push model, application updates to application
100%: October 1, 2019 - Train on data (epoch 1)
100%: October 1, 2019 - Refine network architecture, improve data fields (epoch 1)
100%: October 1, 2019 - Refine data from training, update collection utility (epoch 1)
100%: October 1, 2019 - Update database schema based on feedback (epoch 1), start next epoch in 2 weeks
100%: October 1, 2019 - Fully operational statistical model
100%: October 20-30, 2019 - Application unit tests written, fully-featured, bug testing can begin
100%: November 1-30, 2019 - Testing, refinement, fine tune parameters
100%: November 7, 2019 - Test model on unseen codebase
100%: November 15, 2019 - Application unit tests, bug testing complete
100%: November 25, 2019 - Finish additional file features, revise database
100%: December 3, 2019 - Project finished, practice presentation

8. Related Work

This project was inspired in part by previous work by Thomas Ostrand, Robert Bell, and Elaine J. Weyuker's "Where the Bugs Are," which was an attempt at predicting faults in code based on metadata and information about the development cycle. [105] Theirs used a binomial regression model, and was able to predict correctly, on average, 83% of the files that contained a fault. The main focus in the research was identifying *what* code in a software project should be tested, as opposed to *how* to test code.

However, this work was accomplished through examining a large scale inventory system at AT&T, a massive company, and required a large collective effort to gather the data required, which would either not be available from most software developers due to the scale of their systems, or not available due to lack of manpower to put heavy analysis into extracting the data. Switching over to using public APIs, available through websites like GitHub that allow software development lifecycle information to be available publicly, would bring the scope of this earlier work into the modern era by allowing for useful information to be provided to these developers which can be used to improve the quality of their product, and the software field overall.

In addition to acquiring data, the other main challenge of the project will be approximating the binomial model used in earlier work into a machine learning context. Machine learning tools are now commonplace, and easy to use, with most of the challenge in designing a network being in connecting the discrete elements of an effective architecture. Iterating upon the architecture with the data collection role will be key to improving the model and allowing it to advance. Further, utilizing these tools allow our model to improve over time and learn to make even better predictions about code that it has not yet seen, allowing it to become even more useful and correct as time progresses.

It is important to use the existing work as a guiding influence, but we must be careful to not model the previous work too closely due to the fundamental difference in the nature of the research. In the past, the research has moved from identifying fields that can be used for the greatest good in identifying faults, to identifying those faults using large systems which have been analyzed by many engineers. However, since our project is using publicly available sources of data, and a different sort of modeling where we must choose the best option, we must be careful to not follow the work too closely, else it constrain our design choices and potentially inhibit the project. Ours will be more of a shallow project, where we gather more shallow data about many projects as opposed to putting a large amount of research into a few projects, as in the past.

Dr. Weyuker has been a valuable resource when discussing her previous work in this area and what the guiding ideas were that drove each decision point. Her input on collaborating with the

team has been priceless and we look forward to her feedback in the future as to how the project can grow and improve from the perspective of an experienced professional with firsthand experience in the field.

9. Legal, Ethical, and Privacy Issues

Since our front end Application can run locally without an internet connection, this avoids any potential legal pitfalls by preventing any storage of code or code derivatives in the form of generalized metrics. It is impossible for a server to be breached or source code to leak as a result of running our Application. Information stored in the database is about specific public repos which have publicly disclosed security vulnerabilities, so the database or model do not store anything which could be viewed as sensitive.

Ethically, the Application will only allow further examination into potential problematic files, so no dilemmas should be posed by the software as it exists currently. Like Dr. Weyuker's work, there is no mechanism by which to assign responsibility for any issue, or potential issues, to a certain party. A high percentage ranking that a file may contain a fault only suggests that metrics may warrant a further look at this file. The only problem that could be posed is the thought that a "bad guy" would now have the ability to scan repos that it is looking to target with the aim of finding security vulnerabilities. We argue that any sophisticated attacker would already have tools like this at their disposal and that this only benefits the overall security posture of software developers. By giving the "good guys" tooling that may be used to attack their code, they can use such tooling to protect their code from such attacks.

In an early phase of the project, a feature was planned where a repo scanned by the Application could be submitted to the database for future training iterations, which would necessitate some sort of privacy agreement with the user, but due to the inability to guarantee that claimed bugs were actually problematic files, the threat of contaminating the dataset with false positives was deemed the highest priority, so this feature was deemed unfeasible for this iteration of the program.

10. References

[1] GitHub. Retrieved April 14, 2019 from https://github.com/

[2] GitHub. Retrieved April 14, 2019 from https://api.github.com/

[3] GitHub. GitHub API v3. Retrieved April 14, 2019 from https://developer.github.com/v3/

[4] Vincent Jacques. PyGithub Documentation, Release 1.43.6. Retrieved April 14, 2019 from https://media.readthedocs.org/pdf/pygithub/stable/pygithub.pdf

[5] PyGithub. 2019. PyGithub/PyGithub. (April 2019). Retrieved April 14, 2019 from https://github.com/PyGithub/PyGithub

[6] Scrapy. Scrapy 1.6 documentation. Retrieved April 14, 2019 from https://docs.scrapy.org/en/latest/

[7] Beautifulsoup. beautifulsoup4. Retrieved April 14, 2019 from https://pypi.org/project/beautifulsoup4/

[8] GitPython. Overview / Install. Retrieved April 14, 2019 from https://gitpython.readthedocs.io/en/stable/intro.html

[9] Anon. 2019. MySQL. (April 2019). Retrieved April 14, 2019 from https://en.wikipedia.org/wiki/MySQL

[10] Anon. MySQL Workbench. Retrieved April 14, 2019 from https://www.mysql.com/products/workbench/

[11] Anon. Python Dev Workflow for Humans. Retrieved April 14, 2019 from https://pipenv.readthedocs.io/en/latest

[12] Pypa. 2019. pypa/pipenv. (April 2019). Retrieved April 14, 2019 from https://github.com/pypa/pipenv

[13] Niklas Donges and Niklas Donges. 2018. Data Types in Statistics. (March 2018). Retrieved April 18, 2019 from https://towardsdatascience.com/data-types-in-statistics-347e152e8bee

[14] Altexsoft. Software Engineer Qualification Levels: Junior, Middle, and Senior. Retrieved April 18, 2019 from https://www.altexsoft.com/blog/business/software-engineer-qualification-levels-junior-middle-and-senior/

[15] Jason Brownlee. 2018. Why One-Hot Encode Data in Machine Learning? (May 2018). Retrieved April 13, 2019 from https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

[16] Rakshith Vasudev. 2017. What is One Hot Encoding? Why And When do you have to use it? (August 2017). Retrieved April 13, 2019 from https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

[17] Sebastian Raschka. 2014. About Feature Scaling and Normalization. (July 2014). Retrieved April 13, 2019 from https://sebastianraschka.com/Articles/2014_about_feature_scaling.html

[18] Sudharsan Asaithambi. 2017. Why, How and When to Scale your Features. (December 2017). Retrieved April 13, 2019 from https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e

[19] Anon. 2018. Feature scaling. (November 2018). Retrieved April 13, 2019 from https://en.wikipedia.org/wiki/Feature_scaling

[20] Anon. 2019. Standard deviation. (April 2019). Retrieved April 13, 2019 from https://en.wikipedia.org/wiki/Standard_deviation

[21] Anon.Retrieved April 19, 2019 from https://keras.io/activations/

[22] 10155233989138523. 2018. Activation Functions in Neural Networks. (December 2018). Retrieved April 19, 2019 from https://towardsdatascience.com/activation-functions-in-neural-networks-83ff7f46a6bd

[23] Anon. 2019. Activation function. (March 2019). Retrieved April 19, 2019 from https://en.wikipedia.org/wiki/Activation_function

[24] Anon. Activation Functions. Retrieved April 19, 2019 from https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html#softmax

[25] Avinash Sharma V and Avinash Sharma V. 2017. Understanding Activation Functions in Neural Networks. (March 2017). Retrieved April 19, 2019 from https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[26] Group 8. Bonfire/PSV. Retrieved April 14, 2019 from https://github.com/Bonfire/PSV

[27] JFree. Retrieved April 14, 2019 from http://www.jfree.org/jfreechart/api/javadoc/index.html

[28] JFree. Retrieved April 14, 2019 from http://www.jfree.org/jfreechart/

[29] Jfree. 2019. jfree/jfreechart. (February 2019). Retrieved April 14, 2019 from https://github.com/jfree/jfreechart

[30] Knowm. 2019. knowm/XChart. (February 2019). Retrieved April 14, 2019 from https://github.com/knowm/XChart

[31] libgit2. libgit2/libgit2sharp. Retrieved April 14, 2019 from https://github.com/libgit2/libgit2sharp/wiki

[32] libgit2. 2019. libgit2/libgit2sharp. (February 2019). Retrieved April 14, 2019 from https://github.com/libgit2/libgit2sharp

[33] Anon. Live Charts. Retrieved April 14, 2019 from https://lvcharts.net/App/examples/v1/wf/Install

[34] Anon. Live Charts. Retrieved April 14, 2019 from https://lvcharts.net/

[35] Live-Charts. 2018. Live-Charts/Live-Charts. (September 2018). Retrieved April 14, 2019 from https://github.com/Live-Charts/Live-Charts

[36] Octokit. 2019. octokit/octokit.net. (March 2019). Retrieved April 14, 2019 from https://github.com/octokit/octokit.net

[37] Anon. Octokit.net. Retrieved April 14, 2019 from https://octokitnet.readthedocs.io/en/latest/

[38] Anon. OxyPlot. Retrieved April 14, 2019 from http://www.oxyplot.org/

[39] Oxyplot. 2019. oxyplot/oxyplot. (January 2019). Retrieved April 14, 2019 from https://github.com/oxyplot/oxyplot

[40] Swharden. swharden/ScottPlot. Retrieved April 14, 2019 from https://github.com/swharden/ScottPlot/tree/master/doc

[41] Swharden. 2019. swharden/ScottPlot. (February 2019). Retrieved April 14, 2019 from https://github.com/swharden/ScottPlot

[42] Anon. Welcome to OxyPlot's documentation! Retrieved April 14, 2019 from http://docs.oxyplot.org/en/latest/

[43] Anon. XChart. Retrieved April 14, 2019 from https://knowm.org/open-source/xchart/

[44] Anon. 2019. XChart Parent 3.5.4 API. (February 2019). Retrieved April 14, 2019 from https://knowm.org/javadocs/xchart/index.html

[45] David Both. How to use cron in Linux. Retrieved April 14, 2019 from https://opensource.com/article/17/11/how-use-cron-linux

[46] Documentation Group. Essentials. Retrieved April 14, 2019 from https://httpd.apache.org/

[47] Karsten Eckhardt. 2018. Turn your ML model into a web service in under 10 minutes with AWS CodeStar. (December 2018). Retrieved April 14, 2019 from https://towardsdatascience.com/tourn-your-ml-model-into-a-web-service-in-under-10-minutes-with-aws-codestar-d9e0f4ae24a6

[48] Anon. Conda. Retrieved April 14, 2019 from https://conda.io/en/latest/

[49] Anon. 2019. Systemd. (April 2019). Retrieved April 14, 2019 from https://en.wikipedia.org/wiki/Systemd

[50] Anon. rsync. Retrieved April 14, 2019 from https://rsync.samba.org/

[51] DigitalOcean. LAMP Stack Tutorials. Retrieved April 14, 2019 from https://www.digitalocean.com/community/tags/lamp-stack?type=tutorials

[52] Canonical. Scale out with Ubuntu Server. Retrieved April 14, 2019 from https://www.ubuntu.com/server

[53] Squirrel. 2019. Squirrel/Squirrel.Windows. (April 2019). Retrieved April 14, 2019 from https://github.com/Squirrel/Squirrel.Windows

[54] Canonical. AI / ML from workstation to production. Retrieved April 14, 2019 from https://www.ubuntu.com/ai

[55] Anon. 2019. Getting Started with Kubeflow. (April 2019). Retrieved April 14, 2019 from https://www.kubeflow.org/docs/started/getting-started/

[56] Anon. Kubernetes on DigitalOcean. Retrieved April 14, 2019 from https://www.digitalocean.com/products/kubernetes/

[57] Canonical. Artificial intelligence, machine learning and Ubuntu. Retrieved April 14, 2019 from https://www.ubuntu.com/engage/ai-ml-ubuntu

[58] Canonical. Kubernetes documentation. Retrieved April 14, 2019 from https://www.ubuntu.com/kubernetes/docs

[59] Anon. Retrieving git metadata. Retrieved April 14, 2019 from https://chaoss.github.io/grimoirelab-tutorial/perceval/git.html

[60] Anon. Train fast on TPU, serve flexibly on GPU: switch your ML infrastructure to suit your needs | Google Cloud Blog. Retrieved April 14, 2019 from https://cloud.google.com/blog/products/ai-machine-learning/train-fast-on-tpu-serve-flexibly-on-gpu-switch-your-ml-infrastructure-to-suit-your-needs

[61] Anon. HTTP for Humans™. Retrieved April 14, 2019 from http://docs.python-requests.org/en/master/

[62] Pwaller. 2019. pwaller/pyfiglet. (February 2019). Retrieved April 14, 2019 from https://github.com/pwaller/pyfiglet

[63] CodingEntrepreneurs. 2014. Coding With Python :: Learn API Basics to Grab Data with Python. (February 2014). Retrieved April 14, 2019 from https://www.youtube.com/watch?v=pxofwuWTs7c

[64] Anon. Retrieved April 14, 2019 from https://www.w3schools.com/python/python_mysql_getstarted.asp

[65] Anon. Retrieved April 14, 2019 from https://www.mysql.com/

[66] Mozilla. 2019. mozilla/agithub. (March 2019). Retrieved April 14, 2019 from https://github.com/mozilla/agithub

[67] Turnkeylinux. turnkeylinux/octohub. Retrieved April 14, 2019 from https://github.com/turnkeylinux/octohub

[68] Mozilla. 2019. mozilla/agithub. (March 2019). Retrieved April 14, 2019 from https://github.com/mozilla/agithub

[69] Anon. github3.py: A Library for Using GitHub's REST API. Retrieved April 14, 2019 from https://github3.readthedocs.io/en/latest/index.html#more-examples

[70] Ducksboard. 2017. ducksboard/libsaas. (December 2017). Retrieved April 14, 2019 from https://github.com/ducksboard/libsaas

[71] Anon. Beautiful Soup Documentation. Retrieved April 14, 2019 from https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[72] Anon. First Steps with GitPython. Retrieved April 14, 2019 from https://www.fullstackpython.com/blog/first-steps-gitpython.html

[73] Anon. GitHub project Line Counter. Retrieved April 14, 2019 from http://line-count.herokuapp.com/

[74] Rrwick. rrwick/LinesOfCodeCounter. Retrieved April 14, 2019 from https://github.com/rrwick/LinesOfCodeCounter

[75] Anon. Retrieved April 14, 2019 from https://www.ling.upenn.edu/advice/url-primer.html

[76] Cgag. 2019. cgag/loc. (February 2019). Retrieved April 14, 2019 from https://github.com/cgag/loc

[77] Anon. 2013. Python Execute Unix / Linux Command Examples. (December 2013). Retrieved April 14, 2019 from https://www.cyberciti.biz/faq/python-execute-unix-linux-command-examples/

[78] Anon. Using Python to ask a web page to run a search. Retrieved April 14, 2019 from https://stackoverflow.com/questions/13962006/using-python-to-ask-a-web-page-to-run-a-search

[79] Anon.Retrieved April 14, 2019 from https://git-scm.com/

[80] Anon. TensorFlow. Retrieved April 14, 2019 from https://www.tensorflow.org/

[81] Anon. TensorFlow Guide | TensorFlow Core | TensorFlow. Retrieved April 14, 2019 from https://www.tensorflow.org/guide

[82] Anon. API Documentation | TensorFlow. Retrieved April 14, 2019 from https://www.tensorflow.org/api_docs

[83] Anon. tensorflow. Retrieved April 14, 2019 from https://github.com/tensorflow/

[84] Anon. NumPy. Retrieved April 14, 2019 from https://www.numpy.org/

[85] Anon. NumPy. Retrieved April 14, 2019 from https://github.com/numpy

[86] Anon. Scikit. Retrieved April 14, 2019 from https://scikit-learn.org/stable/index.html

[87] Anon. Documentation of scikit-learn 0.20.3. Retrieved April 14, 2019 from https://scikit-learn.org/stable/documentation.html

[88] Anon. 2019. scikit-learn/scikit-learn. (April 2019). Retrieved April 14, 2019 from https://github.com/scikit-learn/scikit-learn

[89] Anon. Keras: The Python Deep Learning library. Retrieved April 14, 2019 from https://keras.io/

[90] Adrian Rosebrock. The Keras Blog. Retrieved April 14, 2019 from https://blog.keras.io/category/tutorials.html

[91] Keras Team. 2019. keras-team/keras. (April 2019). Retrieved April 14, 2019 from https://github.com/keras-team/keras/

[92] Anon. MySQL Connector/Python Developer Guide :: 5.1 Connecting to MySQL Using Connector/Python. Retrieved April 13, 2019 from https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html

[93] Niklas Donges and Niklas Donges. 2018. Data Types in Statistics. (March 2018). Retrieved April 13, 2019 from https://towardsdatascience.com/data-types-in-statistics-347e152e8bee

[94] Anon. Software Engineer Qualification Levels: Junior, Middle, and Senior. Retrieved April 13, 2019 from https://www.altexsoft.com/blog/business/software-engineer-qualification-levels-junior-middle-and-senior/

[95] Anon. 2017. How to One Hot Encode Sequence Data in Python. (July 2017). Retrieved April 13, 2019 from https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/

[96] Anon. sklearn.preprocessing.LabelEncoder. Retrieved April 13, 2019 from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#sklearn.preprocessing.LabelEncoder.fit_transform

[97] Anon.Retrieved April 13, 2019 from https://keras.io/utils/

[98] Anon. Importance of Feature Scaling. Retrieved April 13, 2019 from https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

[99] Anon. 4.3. Preprocessing data. Retrieved April 13, 2019 from https://scikit-learn.org/stable/modules/preprocessing.html

[100] Anon. sklearn.preprocessing.scale. Retrieved April 13, 2019 from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html#sklearn.preprocessing.scale

[101] Andrew Tch. 2017. The mostly complete chart of Neural Networks, explained. (August 2017). Retrieved April 13, 2019 from https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

[102] Anon. sklearn.model_selection.KFold. Retrieved April 13, 2019 from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

[103] Anon. 2019. Cross-validation (statistics). (April 2019). Retrieved April 13, 2019 from https://en.wikipedia.org/wiki/Cross-validation_(statistics)

[104] Anon. 2019. Training, validation, and test sets. (March 2019). Retrieved April 13, 2019 from https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets

[105] Thomas Ostrand, Elaine Weyuker, and Robert Bell. Where the Bugs Are. https://people.eecs.ku.edu/~hossein/Teaching/Fa07/814/Resources/where-bugs-are.pdf

Appendix

Figure Descriptions

Figure 1: Project Diagram Outline - provides an overview of the individual project components and how they connect

Figure 2: Project Block Diagram - similar to Fig. 1, but provides a more detailed view at each project component

Figure 3: UML Diagram - shows a high-level abstraction of how the project's parts will connect

Figure 4: Dual path data intake Diagram - shows how a balanced dataset is built out of problematic and non-problematic files

Figure 5: Data Acquisition Process Diagram - describes how data collection connects to the database and what information is examined for storage

Figure 6: Data Acquisition Flow Diagram - Python libraries used for the data collection utility can be used to acquire different kinds of fields from GitHub APIs

Figure 7: Project Flow Diagram - demonstrates each role and how they interconnect

Figure 8: Database Entity Representation Diagram - show the logical database schema and the fields being used to store information at a repository and file level

Figure 9: Application Process Diagram - this shows the projected flow of the Application

Figure 10: Application Mockups - showing the planned design for the Application

Figure 11: Application Design - showing the current design status of the Application

Figure 12: Categorical Data Conversion Diagram - if data needs to be prepared for the modeling role, this diagram shows the process this will take

Figure 13: Neural Network Flowchart - the modeling portion's connections to the database, program, and internal processes are shown here