Visualizing the Effects of Android App Similarity on Android Malware Detection

Gabriella Xiong and Michael Cao

Fig. 1: Overview of visualization system to explore the effects of training on different subsets of Android applications. Left: Data Set Visualization; Right: Model Performance Visualization.

Abstract— Many approaches for Android malware detection have been proposed to combat against the rise in mobile malware–most of which have relied on machine learning [3, 10, 17]. Such approaches extract features from benign and malware applications, and train a classifier to distinguish between the two classes. One important fact is that the selection of training data can heavily influence the machine learning model in various aspects. This paper describes a visualization tool, aimed towards Android malware researchers, to explore the trade-offs of selecting different training application sets on the resulting model. The tool allows the researcher to input a pool of benign and malware applications, view temporal and feature statistics, and select subsets used for training through an interactive user interface. Once the researcher is satisfied with the selection of applications, a model is trained and the researcher is provided with information for comprehending the performance and attackability of the model.

Index Terms—Android, Machine Learning, Malware Detection, Model Performance, Model Attackability

1 Introduction

Mobile smart phones have become increasingly popular in the past decade. As a matter of fact, the number of smart phone users are expected to nearly double from 2016 to 2021 [22]. While smart phones bring many benefits to a user's life, their adoption has also greatly stimulated the growth of mobile malware. In particular, the Android market is a vulnerable target for malicious users due to its open sourced nature and large user market.

A number of approaches have recently emerged to support Android malware detection [1, 3, 10, 17]. Most of the proposed approaches rely on extracting application features and training a machine learning classifier to distinguish between the benign and malicious applications. Benign applications are easily collected and can be considered as an unlimited source. On the other hand, malware applications are often limited and difficult to find in practice. When training a classifier, practitioners often collect as many malware applications as possible and randomly sample from the unlimited benign pool. Training on such data is likely to create a highly separable classifier model.

A highly separable model clearly distinguishes between the benign and malware classes—often providing high performance results when detecting applications from the same distribution as training. The model will often learn highly distinguishing features that are associated with the benign or malware class. While a highly separable model provides good detection performance, it trades off reliability. Malware can easily exhibit the highly distinguishing benign features by adopting similar functionalities from the benign applications to disrupt the model's detection rate. In fact, recent studies have shown that simply injecting a small amount of benign features into malware can substantially reduce the malware detection rate of a highly separable model from 94% to 67% [5].

We hypothesize that a classifier trained using benign applications similar to the malware will reduce the number of highly distinguishing benign features, and hence produce a less separable model. While less separable models are more resilient to the benign feature injection attack, we suspect the models' overall ability to distinguish between the benign and malware classes to decrease. Varying the similarity between benign and malware training applications could possibly tune the trade-off between performance and reliability of the model. We built a visualization tool to effectively explore and confirm this hypothesis. In particular, we visualized the similarities of training applications and characteristics of different training data sets to study the relationship between data selection, the reliability of a model, and the detection performance of a model.

• *Gabriella Xiong and Michael Cao are M.A.Sc students at the University of British Columbia. Their emails are gxpeiyu, michaelcao @ ece.ubc.ca.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

2 BACKGROUND

In this section, we provide background related to Android application development. We then discuss a well-known Android malware detection tool, DREBIN [3], and introduce the notion of attackability for an Android malware detection model.

2.1 Android Application Development

Android is currently the most used mobile operating system. The output file used to represent an application is a zipped archive known as an android package (APK) file. Two key components of the APK file are the Android manifest and Java bytecode. In addition, the APK file contains various resources in the form of extensible markup language (XML) code, which represent the application layouts and predefined resource strings. Lastly, the application may contain additional native C++ binaries or assets that are dynamically loaded and used by the application.

2.2 Android Malware Detection: Drebin

DREBIN [3] is one of the first approaches for Android malware detection. It uses lightweight static analysis to extract eight different features falling under the Android manifest and Java bytecode. Features from the Android manifest include requested permissions (ex., accessing internet), intent filters (ex., events triggering on device boot), hardware components (ex., defining camera use), and application components (ex., activites and services). Features from the Java bytecode include application program interfaces (API) with associated permissions from the Android manifest (ex., getCurrentLocation with location permission requested); APIs that require permissions but are not requested within the Android manifest (ex., getActiveNetworkInfo without requested network permissions); suspicious APIs associated with sensitive data (ex., sendTextMessage); and network addresses. DREBIN denotes an application's features as a binary vector of zero and ones, where zero indicates not containing the feature and one indicates containing the feature. All eight categories of features are embedded into a joint vector space, and DREBIN trains a linear Support Vector Machine (SVM) [6] to distinguish between the benign and malware class.

2.3 Evaluating Attackability of Malware Detection Model

Cyber criminals which produce malware can perform actions to increase their chances of avoiding detection from machine learning malware detection models. One effective and lightweight approach to avoid detection is a mimicry attack [4,5]. The typical workflow for a mimicry attack first starts with the cyber criminal collecting a surrogate set of benign applications. After this reference set of benign applications is studied, the cyber criminal denotes common benign functionalities and injects them into the malware. An approximation to the mimicry attack is to extract DREBIN features from the surrogate set of benign applications, and then to inject the features into malicious samples based on their frequency. The effect of a mimicry attack can be measured based on the drop in malware detection rate after a given number of features are injected into the original classified malware applications.

3 DATA AND TASK ABSTRACTION

In this section, we describe our task abstractions in the context of terminologies introduced in Visualization Design and Analysis.

3.1 Data

We collected malware applications based on snapshots provided by VirusTotal Academic [23]. Six snapshots were provided between the years 2016 to 2019, consisting of 5K applications. We also collected 10K benign applications from AndroZoo [2], a large repository that continuously scrapes applications from a variety of markets between a similar time range. All applications will be converted into feature vectors using DREBIN [3]. From our previous experiences of using DREBIN, the feature space grows super-linearly as we linearly introduce applications into the data set; a training data set of 5K applications leads to 22K features, while a training data set of 10K applications leads to 108K features.

Table 1: What-Why-How-Analysis-Table

What: Data	Tabular, binary-valued attributes and sparse
What: Derived	• Dimension reduced application distribution. • Aggregated feature usage values. • Model performance results. • Testing application prediction & attack results. • Feature weight in trained model.
Why: Tasks	• Explore training data set. • Select and visually inspect characteristics of subsets of data. • Train model on selected data. • Evaluate the detection rate and attackability trained model. • Interpret model results.
How: Encode	Scatter plots, Bar charts, Line charts, Table
How: Facet	Multiform, overview-detail
How: Reduce	Filtering on the applications, filtering on the features
How: Embed	Pop-up window with details (e.g. application detail)
How: Manipulate	Zoom, pan, select, sort
Scale	Approx. 15,000 items (5,000 malware applications and 10,000 benign applications)

3.2 Tasks

At a high level, we would like to provide a tool that helps researchers analyze the relationship between the characteristics of the training data and the properties of the resulting model. The tool can assist the researcher in achieving the following tasks:

- Visualize the overall training data distribution in 2D and visually inspect possible clusters
- Evaluate similarities among all applications in the training data, based on selected feature categories of interest, and locate similar or distinguishable applications. This could help the researcher fine tune the similarities among benign and malware applications in the training dataset
- Visualize feature distributions over a selected set of benign and malware applications. This could assist the researcher in identifying key features that contribute to similarities or dissimilarities
- Construct a training data set from the total application pool to train the model for further analysis

After the researcher selects a set of training applications, our system will train a SVM model and test it on a set of malware and benign testing applications. Based on the results, the researcher can investigate the reasons behind the model's performance and attackability by:

- Visualizing the overall training and testing data distribution in 2D to compare the distribution of training and testing applications
- Identifying the misclassified applications, and locating similar training applications to reason about model prediction results
- Identifying the effort (i.e., number of benign features injected) required for malicious applications to evade detection
- Analyzing the weight assigned by the trained model for each feature to explain the prediction on applications

4 SOLUTION

In this section, we describe our proposed visualization system which can assist researchers in performing the previously defined tasks.

Task 1: Visualize the training data set distribution in 2D.

In order to create an overview of the applications in 2D, we performed popular dimension reduction techniques, namely t-SNE [16]

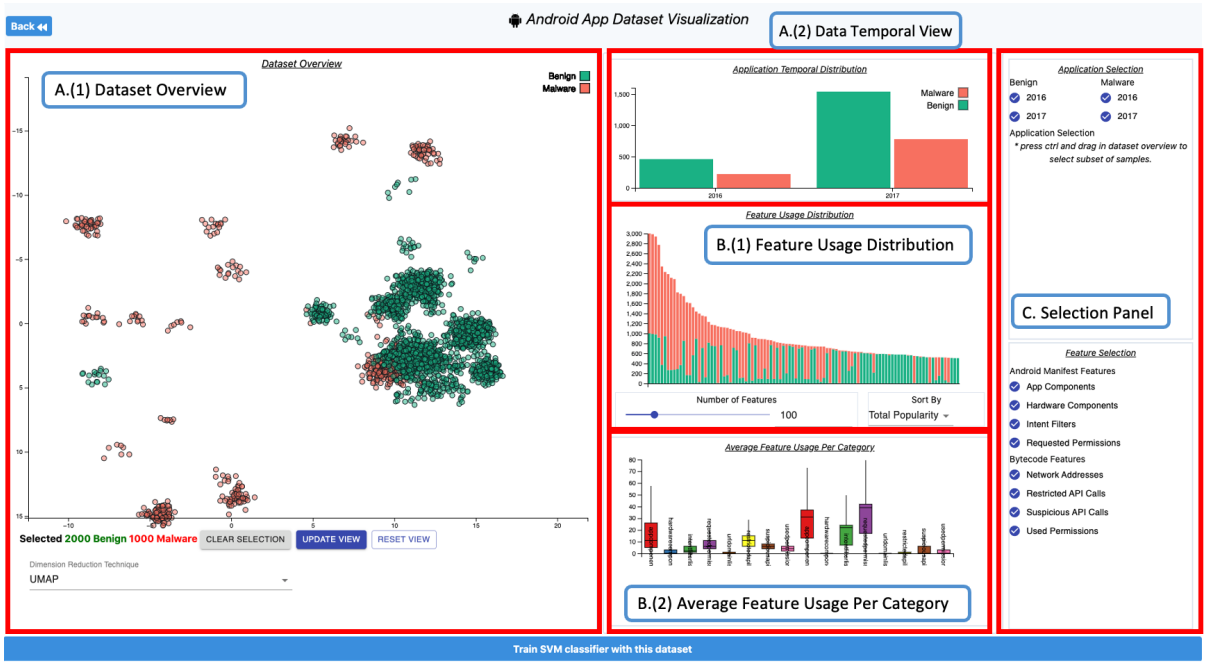


Fig. 2: Data Set Visualization Page

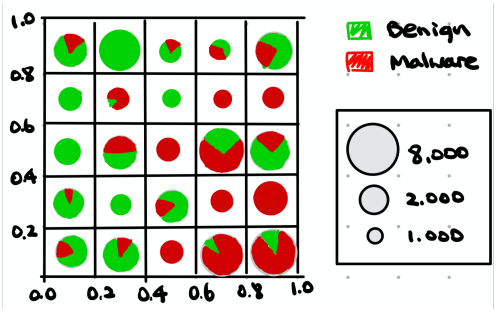


Fig. 3: Alternative application distribution view

and UMAP [18], on the application feature vectors. The implemented system uses UMAP by default, as it provide faster computation on large dimensions compared to t-SNE. However, the researcher can change the dimension reduction technique to t-SNE by using the drop down menu at the bottom of section "A.(1) Dataset Overview" in Figure 2. To visualize the results from dimension reduction, we considered using a scatter plot. However, we noticed that scatter plots have issues with scalability — overlapped applications could obscure the density information. To reduce overlap, we applied jittering by adding a small fraction of noise to each application's location, such that they are less likely to be on the same spot. However, it is important to note that overly applying jittering could disturb initial patterns in the data.

We also considered an alternative visualization where we divide the 2D space into square blocks, and use pie charts to display the benign and malware distribution over each block. This alternative mock up is shown in Figure 3. The size of each circle encodes the total number of applications within each block, and the angles inside the circle encode the proportion of benign and malware. This approach preserves the density information; however, breaking the continuous space into discrete blocks can potentially distort continuous clusters.

From our initial testing with 7,500 applications, we found jittering reduces overlap and produces little disturbance on the application distribution. Thus we decided to use a scatter plot with jittering within our final system.

Task 2: Identify similar and dissimilar applications.

Dimension reduction techniques project applications from a high dimensional feature space to 2D while preserving the relative distances among applications. Since the distances within a 2D scatter plot reflects the similarity among applications, the researcher can identify similar applications by locating clusters within the 2D plot.

Task 3: View trend and characteristics over a set of applications.

Normally, the characteristics for a set of applications can be abstracted through their temporal distribution and feature usage distribution.

To view the temporal distribution over applications, we used a grouped bar chart. The x-axis was used to indicate the year an application was published, and the y-axis to indicate the number of applications within a given year. In each year, there are two bars that encode the total number of benign and malware applications, respectively (shown in Figure 2, "A.(2) Data Temporal View').

For visualizing the feature distribution over a set of applications, we provided two charts at different levels of granularity. As DREBIN features only contain binary values, we can aggregate the feature usage information over a particular set of features, and visualize them using a stacked bar chart (shown in Figure 2, "B.(1) Feature Usage Distribution"); the x-axis corresponds to features, and the y-axis corresponds to the feature usage among benign and malware applications.

Due to the limit of space, we pre-set the view to show 100 features, but the researcher can increase the number of features in the chart by adjusting the slider. Rather than display a large number of features, we chose to display a set number of important features based on sorting heuristics often used in feature selection algorithms. Currently, we allow the researcher to view up to 500 features based on two sorting mechanisms, including: (1) sorting by the total popularity of features, and (2) sorting by the popularity difference between benign and malware applications. The sorting options are configurable based on a drop down box. Tool tips were implemented so that a researcher can hover over features in the bar chart to view more information, including the corresponding feature name and the number of benign or malware applications that contain the feature.

In addition to trends from overall feature usage, there could also be interesting patterns in how different categories of features are used among benign and malware applications (ex., malware may contain more features from the requested permissions than benign). Hence, we also designated a view on the average usage of different feature

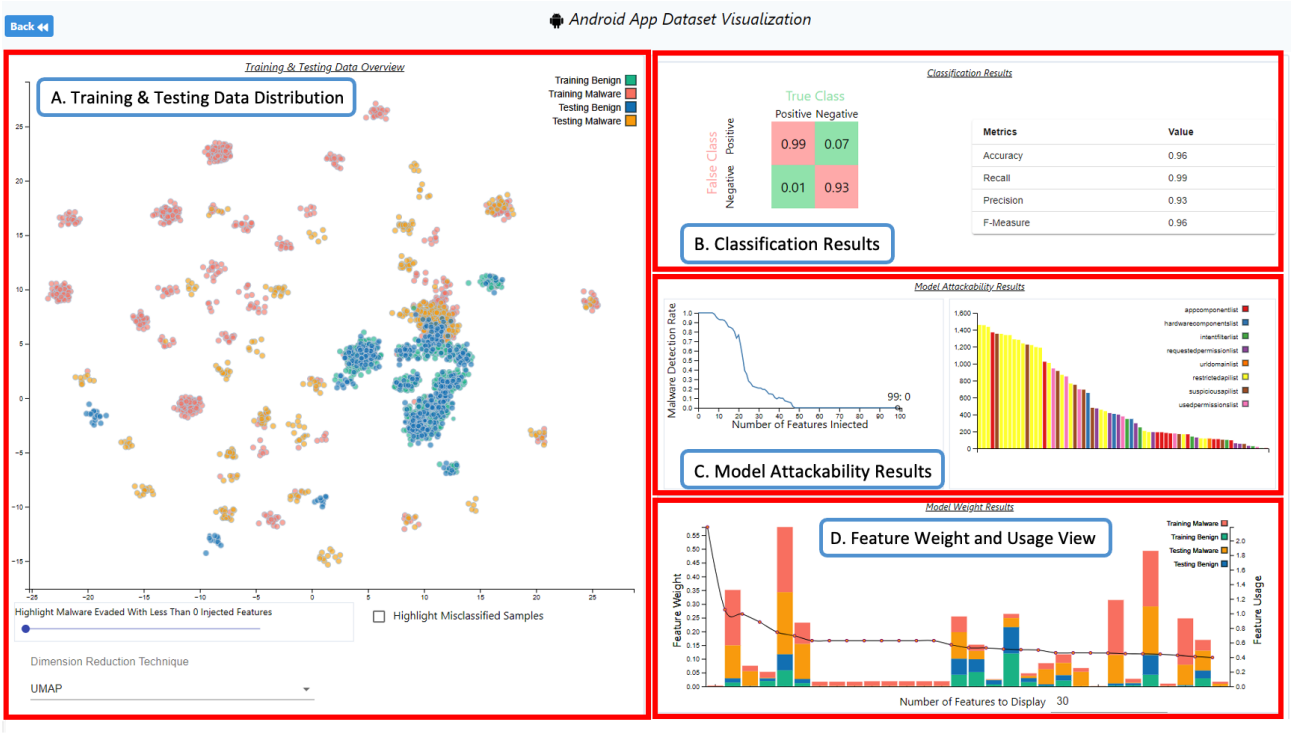


Fig. 4: Model Performance Visualization Page

categories over benign and malware applications. We used box plots to show the feature usage distribution for each category among benign and malware applications, as box plots can encode more distributional information over a set of applications rather than average values (shown in Figure 2, "B.(2) Average Feature Usage Per Category"). In the chart, the x-axis corresponds to the feature category usage in either benign or malware applications, and the y-axis encodes the number of features from a category present in applications. There are 16 boxes shown in the image, where the 8 left-most boxes correspond to benign applications, and 8 right-most boxes correspond to malware applications. The feature categories are encoded using colors selected from ColorBrewer [13]. To facilitate comparisons between benign and malware, we applied the same color for both benign and malware feature categories.

Task 4: Perform analysis on subset of data.

To see how characteristics vary over different data sets, we offer the researcher options to filter applications by time, or to select the applications directly by brushing over the application distribution view (shown in Figure 2, "C.Selection Panel"). By default, all applications uploaded from the researcher will be selected in the beginning, and the researcher can use the "Clear Selection" button to de-select all applications. Once the researcher is satified with their collection of applications, they can select the "Update View" button to refresh detail views, including: the Dataset Overview, Data Temporal View, Feature Usage Distribution View, and Average Feature Usage Per Category View. When the researcher wants to investigate another collection of applications, they can select the "Reset View" button to show and re-select all applications by default.

Task 5: Perform analysis using different features.

If the researcher wants to ignore features from certain categories, they can use the feature selection panel to restrict the features to only categories of interest (shown in Figure 2, "C. Selection Panel"). Once the feature category selection is updated, the researcher can once again refresh the detail views using the "Update View" button.

Task 6: Evaluate model performance.

After the researcher selects a desired training data set (the selected applications will be highlighted in the "Data set Overview" in Figure 2), the system will train and evaluate a Support Vector Machine (SVM) model on a pre-defined testing data set for both performance and attackability. The results provided regarding the performance of the model include: (1) a confusion matrix (i.e., True Positive, True Negative, False Positive, and False Negative), (2) accuracy, precision, recall, and f1-measure on malware detection (shown in Figure 4, "B. Classification Results"). The attackability of the model is shown through the malware detection rate under the mimicry attack, with an increasing number of features injected into the testing malware applications (shown in Figure 4, "C. Model Attackability Results").

Task 7: Understand model performance.

To help the researcher interpret the prediction results from the model, we visualize the application distributions over both training and testing data. Similar to the procedure used to visualize training data distributions, the same dimension reduction technique will be applied to generate the application distribution in 2D using both training and testing applications. The results are also shown using a 2D scatter plot. To differentiate training and testing data, we used different colors to encode training benign, training malware, testing benign, and testing malware (shown in 4, "A. Training & Testing Data Overview").

Based on the detailed performance results of the model on testing applications, we display the derived information on the applications used for testing to facilitate an in-depth interpretation of the model performance. In particular, we:

- Highlight misclassified applications in the training/testing data overview, controlled through a checkbox.
- Highlight the malware testing applications that can be manipulated to evade model detection with up to X number of features injected, controlled through a slider. The value from the slider indicate the upper bound of features which can be injected to each previously correctly classified malware testing applications.

By displaying the derived information on the application distribution view, the researchers can use the testing applications' surrounding training applications to explain the model's predictions.

To identify the features that are most exploited to attack within the resulting model, we created a linked visualization on the detection rate degradation line chart and the bar chart that includes the most injected features (shown in Figure 4, "C. Model Attackability Results"). Each

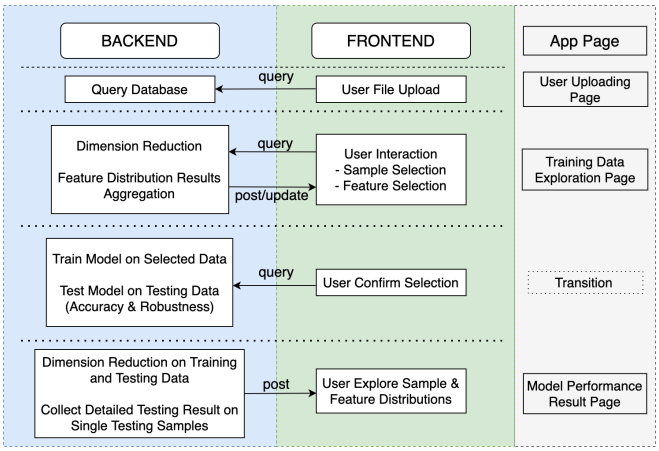


Fig. 5: Implemented Functionalities in Web Application

point on the detection rate degradation line chart denotes the maximum number of features injected into malware testing applications, and the corresponding detection rate after injection. When the user selects a point on the line chart with the maximum injected feature count being X, the bar chart on the right will display the most popular features injected when injecting up to X features; the height of a bar indicates the number of applications being injected with the given feature. The bars are color coded by the corresponding feature's category, using the same color scheme applied in "Average Feature Usage Per Category" view in order to better reveal trends in exploitable features.

Another approach to reason about the attackability of the model is the feature weight and usage chart (shown in Figure 4, "D. Feature Weight and Usage View"). In this chart, a line chart is superimposed on top of a stacked bar chart. The line chart encodes the absolute feature weights assigned by the model, while the stacked bar chart indicates the feature usage among training benign, training malware, testing benign, and testing malware applications. The bars are also color coded using the same color scheme as in the "A. Training & Testing Data Overview". The features shown in the chart are sorted by the absolute value of the feature's weight from the resulting model. In other words, features that are deemed more important will be displayed in the beginning by default. The researcher can also increase the number of features shown in the chart by using the input at the bottom of the section: "Number of Features to Display: __". The chart can be used to verify how aligned the feature usage is with respect to the importance assigned by the model. Commonly, a reliable model would assign higher importance to a feature that is shared by a larger amount of training and testing applications. The line chart of the feature weight alone can also be used to reason about the model attackability, as suggested by Demontis et al. [9], that a model is less attackable if the model's feature weights are evenly distributed.

To summarize, the design decisions we described above are listed in Table 1 by following the What-Why-How analysis approach introduced in Visualization Design and Analysis [19].

5 Implementation

We developed the visualization system as a web application. It is composed of a back-end and a front-end that communicates through hyper text transfer protocol (HTTP) responses. We decompose the implemented functionalities in Figure 5.

The back-end was implemented with Python and Django framework. The main responsibility of the back-end was to perform tasks that were computationally demanding, such as computing the dimension reduction results on selected samples, collecting updated feature distribution information, training an SVM model with selected samples, and testing the trained model on pre-defined testing dataset. Dimension reduction and training an SVM model were performed using off-the-shelf libraries from Python using the Scikit-Learn library. Other tasks like feature aggregation or model attackability testing were all implemented from scratch.

The front-end was implemented with the React framework and with d3 as the visualization library, serving as the interface to display information and update based on the researcher's interactions. Basic interactions were directly implemented in the front-end, such as changing the number of features shown in the display, selecting the sorting heuristics in feature distribution charts, and highlighting samples that satisfy certain criteria.

We chose React framework for the front-end due to the framework's stateful nature. The framework aids in linking interactions between different views, and the hierarchical component definition in React allowed for easy in-parallel development among teammates.

In this project, both authors had a chance to implement functionalities in both the back-end and front-end of the system. A detailed work break down among the two authors is included in the table included in the appendix.

6 Results

In this section, we walk through a typical scenario the researcher would perform using the visualization system, and demonstrate how defined tasks are fulfilled at each step. Unfortunately, we encountered scalability issues in terms of application responsiveness when constantly computing dimensional reduction beyond 5,000 applications; this was due to the non-linear relationship between the number of applications and the given feature space. We thus demonstrate the system scenario using a smaller application size of 3,000 applications, and aim to achieve the tasks listed in section 3. For the convenience of the reader, we list the tasks below:

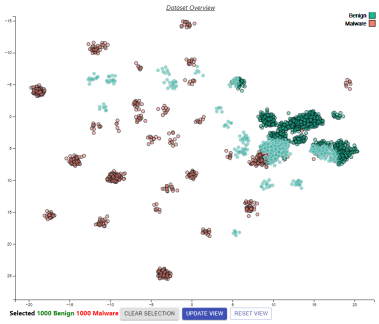
- Task 1. Visualize training data distribution in 2D
- Task 2. Identify similar and dissimilar applications
- Task 3. View trend and characteristics over a set of applications
- Task 4. Perform analysis on subset of data
- Task 5. Perform analysis using different features
- Task 6. Evaluate model performance
- Task 7. Understand model performance

6.1 Scenario: Exploring App Similarity

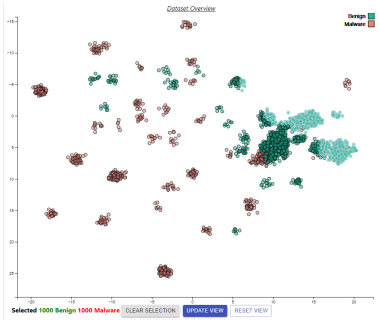
The researcher is interested in exploring the effects of application similarity on the resulting model's performance and attackability. To test this scenario, the researcher runs two experiment instances in parallel. In the first instance, the researcher selects and trains a model based on benign applications that are more similar to the input malware set. In the second instance, the researcher selects and trains a model based on benign applications that are less similar to the input malware set.

The researcher first opens two instances of the visualization system side by side. Both systems request a comma-separated values (CSV) file containing the set of benign and malware applications, and the researcher inputs the same CSV in both instances. Note that the CSV file contains a larger amount of benign applications, as the researcher wants to select a subset to perform their experiments. After uploading the file, the researcher is brought to the data visualization page (see Figure 2). All applications are selected by default on starting the data visualization page. For both system instances, the researcher first selects the "Clear Selection" button from the "Dataset Overview" component to deselect all applications. The researcher then selects all malware applications by checking every malware year under the "Application Selection" view. Note that the "Dataset Overview" component projects the benign and malware application distributions onto a 2D space (T1).

In the first instance, the researcher aims to select a set of benign applications that are more similar to the malware. The researcher can identify the similarity of the applications based on the distance found in the "Dataset Overview" component—applications that are closer in the scatter plot are more similar to one another (T2). The researcher then selects benign applications by holding control and brushing on



(a) Dissimilar Benign Selection



(b) Similar Benign Selection

Fig. 6: Selecting Benign Applications For Two Different Instances.

the "Dataset Overview". An equal amount of benign applications are selected which are closest to the malware (T4). The resulting selection for similar applications can be shown in Figure 6b.

In contrast, the researcher aims to select a set of benign applications that are more dissimilar to the malware for the second instance. The researcher follows the same approach, by clearing the default selected applications, re-selecting all malware, and selecting all benign applications that are farthest away from the malware. The resulting selection for dissimilar applications can be shown in Figure 6a.

Once the selections are created, the "Update View" button is selected under the "Dataset Overview" component for both visualization system instances. Visualizations related to the temporal distributions of the selected applications, features, and feature category usage are updated. The researcher briefly reviews the application statistics (T3) and clicks the "Train SVM classifier with this data set" button. Two models are trained as a result, one for each visualization system instance, and the user is brought to a model performance visualization page (see Figure 4).

On the model performance visualization page, the researcher is presented with information related to the model performance, attackability, and features. The researcher first examines the model performance results by reviewing information found under the "Classification Results" section of each visualization system instance (T6). We provide a comparison of the classification result components for the two experiments under Figure 7a and Figure 7b. Upon investigating the performance, the researcher would first review the confusion matrix. Comparing the experiment using dissimilar to similar benign applications, the researcher would identify a decrease in true positives (0.99 to 0.89), decrease in false positives (0.07 to 0.01), increase in false negatives (0.01 to 0.11), and increase in true negatives (0.93 to 0.99). Furthermore, the researcher would compare the performance metrics, and notice a decrease in accuracy (0.96 to 0.94), decrease in recall (0.99 to 0.89), increase in precision (0.93 to 0.99), and decrease in f-measure (0.96 to 0.94). From these results, the researcher might conclude that the dissimilar model is able to detect the testing malware better than the model using similar benign applications. It is interesting to also note that the model produced with similar benign applications trades off high malware detection rate for higher accuracy in recognizing benign applications.

After evaluating the performance results of the two models, the researcher is interested in investigating the attackability results. The researcher would first review information found under the "Model Attackability Results" view for each visualization system instance, and results of the two instances are then compared. It is interesting to note that the attackability of the models are substantially different from one another. On the one hand, the dissimilar model's malware detection rate drops from 100 to 0 percent after injecting 50 features. On the other hand, the similar model's malware detection rate can still detect 50% and 32% of malware after injecting 50 and 100 features, respectively. The researcher concludes that training on similar benign applications substantially reduces the attackability of the model. On the right hand side of the "Model Attackability Results" view, the researcher looks into features that were injected into the testing malware applications for evaluating the model attackability. The researcher first looks briefly at the overall color scheme of the bar chart, and notes that the categories for features injected are very similar. Afterwards, the researcher hovers over the bars to identify the exact feature injected. The researcher can then understand that the features injected into most applications were related to Android media player APIs.

After evaluating the model's performance and attackability, the researcher aims to understand the reasons behind model performance. The researcher first looks at the "Model Weight Results" view to observe how the most impactful features used for detection are distributed against training or testing benign and malware applications (T7). Within the stacked bar chart, the researcher identifies the amount of testing benign applications based on a dark blue encoding, and the amount of testing malware based on a yellow encoding. From the model trained with dissimilar benign applications, the researcher notes that many of the heavily weighted features are used in many testing malware, but little testing benign. In contrast, the most heavily weighted features in the model trained with similar benign are more evenly used than the dissimilar model. The researcher may conclude that the dissimilar model can detect malware better because the impactful features used for detection are able to better distinguish between the classes.

Lastly, the researcher may review the "Training & Testing Data Distribution" view to further understand reasons behind model performance (T7). The researcher highlights misclassified applications by clicking the "Highlight Misclassified Samples" checkbox, and compares misclassified applications in the scatter plot to training or testing benign and malware applications. Figure 8a and Figure 8b display the training and testing data distributions for the dissimilar and similar models, respectively. The researcher first notes that misclassifications are mainly benign applications in the dissimilar model, and malware applications in the similar model. Additionally, the researcher also observes that the training benign applications, while both models are close to a large cluster of malware applications, is more compact within the similar model comparatively to the dissimilar model. The researcher may conclude that the dissimilar model produces a decision boundary that is closer to the benign applications, leading to a higher misclassification rate of benign testing applications. Likewise, the researcher may conclude that the similar model produces a decision boundary that is closer to the malware applications, leading to a higher misclassification rate of malware testing applications. Afterwards, the researcher may repeat multiple iterations of the scenario using similar settings to confirm their findings.

7 Discussion And Future Work

Due to the time limit of the course, we were not able to perform a systematic evaluation with Android malware detection researchers. However, we briefly discussed with Android malware researchers from our lab group. When bringing up the idea of this system to Android malware researchers, they imagined this tool to be helpful in terms of organizing and debugging Android malware detection experiments. The system functionalities include the overall workflow of investigating and selecting training applications, training a model with a selected subset, and collecting performance information on the trained model, while

(a) Dissimilar Benign Selection

(b) Similar Benign Selection

Fig. 7: Model Results Based on Performance, Attackability, and Features.

offering the researcher the flexibility to interact within the process. In addition, the system provides visualizations on both the sample distribution and feature distribution of an Android malware detection data set, which collectively helps to extract semantically meaningful characteristics from the data set.

7.1 Limitations

There are a few limitations in the implemented system that we believe could be improved.

The first limitation is the scalability of the system. It is important to note that the responsiveness of user interactions is one of the key strengths that this system can bring to the investigation process. However, this fact is limited in the current implementation; the computational time required to perform dimension reduction bottlenecks the responsive interactions based on a custom set of selected applications. Due to the high dimensional application feature space, even with UMAP, the computation is too time consuming for real time interactions. The current back-end also does not support CUDA, which could substantially reduce the computation time of T-SNE or UMAP. In the current system, with 3,000 samples in the view, it takes roughly 30 seconds to 1 minute to update the dimension reduced sample distribution results. One possible way to improve on this is to host the back-end on servers with more computing power. Another improvement could involve performing feature reduction to remove noisy features, while retaining features with the most variance in the original data set before performing dimension reduction.

A second limitation includes the limited functionalities we provide to explore the feature space of the data set. The current system only provides a visualization on the top features. For instance, the "Feature Distribution View" in Figure 2 only displays up to the top 500 features. To provide a well-rounded view on the features used in the data set, one alternative approach is to use a visualization with context and focus idioms as shown in Figure 9. In this alternative approach, the lower view provides an overview on the whole feature space, and the upper view shows more detailed information on a currently selected set of features. The researcher can use brush to select a window from the lower view, and the features within the window will be displayed in the upper view. With this approach, the researcher can have a better idea of the feature space and additional flexibility to investigate the features.

The third limitation is that the current system lacks a mechanism for researchers to perform cross-experiment comparisons. One possible future direction would be to create a storage system that can save the configuration and the corresponding results from different experiments. Additionally, a new page could be created to allow the researcher to load and compare the experimental results from multiple experiments using a juxtaposed views.

Furthermore, as this project is limited to the scope of analyzing DREBIN Android malware detection tool, the implemented feature aggregation approaches are only applicable to Drebin-like features (e.g. string-based and integer count based features). New visual encoding and interaction mechanisms are required if researchers want to investigate other feature patterns, such as API call sequences, as features.

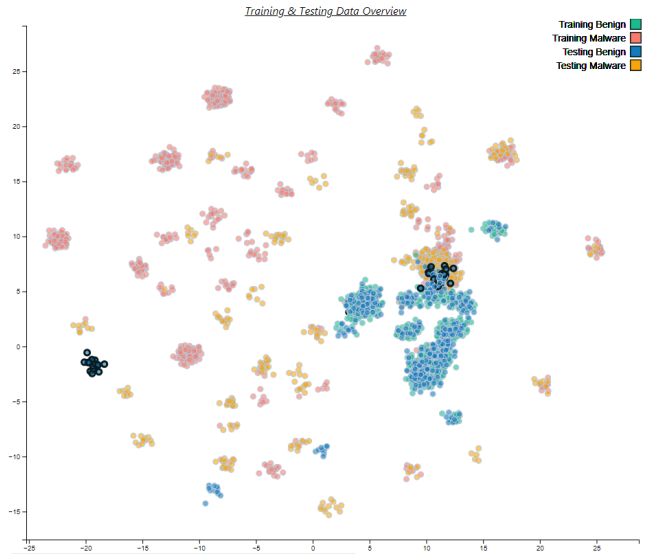
7.2 Lesson Learned

In this project, we applied our knowledge from visualization design and analysis to build a functional system, aimed to improve an Android malware researcher's experience in exploring trade-offs of selecting different training data on the resulting model. Through this project, one key lesson we learned is to discuss with more people during the design process. It is easy to form tunnel vision when focusing on the project. Constantly seeking feedback from Android malware researchers or peers can help to identify more efficient or unreasonable designs in an earlier stage.

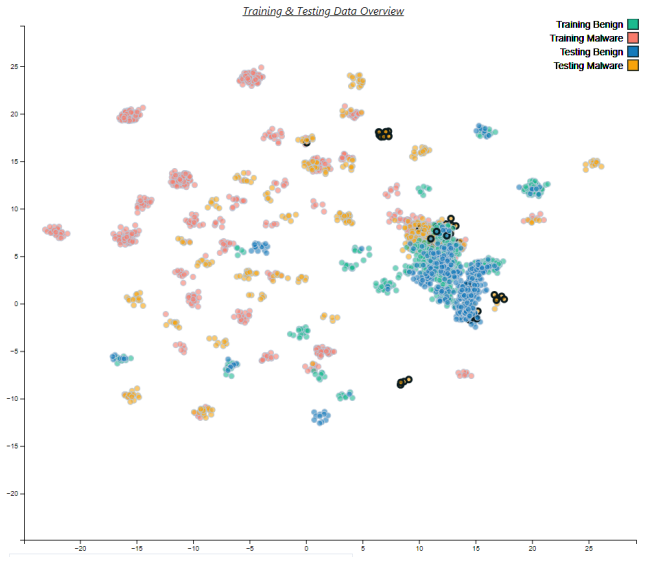
Another key lesson is to start building prototype early. We realized that some proposed visual encodings could not effectively display the information as desired only after we fully implemented the components and tried them in action. For some of these cases, we were able to change the implementation immediately, and some have been left as future work. However, we believe the system could be further refined if we were able to go through additional prototyping cycles.

7.3 Future Work

Both authors are interested in continuing the development and refining the current visualization system. We first plan to change the visual encoding for feature distribution views to improve the flexibility of a researcher's interactions with features. We then plan to experiment with approaches to improve the overall responsiveness of the system, such as reducing the time needed to perform dimension reduction on the set of selected applications. Lastly, we plan to include mechanisms to store and compare results from multiple experiments.



(a) Dissimilar Benign Selection



(b) Similar Benign Selection

Fig. 8: Training and Testing Distributions.

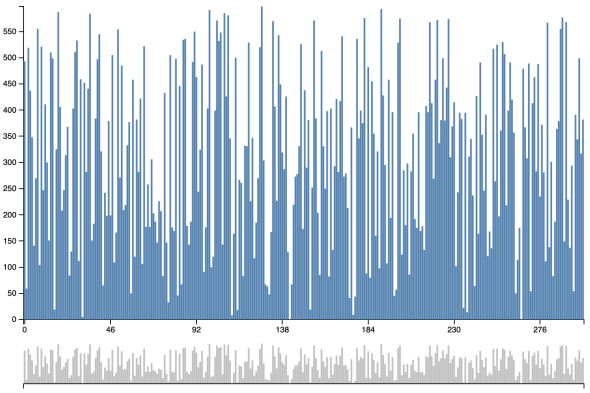


Fig. 9: Alternative feature view with context and focus styled barchart

8 Related Work

8.1 Android Application Visualizations

Existing visualization tools for Android applications focus on Android malware. These works are divided into two categories: (1) visualizations that help security analysts locate malicious behaviors [11, 21, 25], and (2) visualizations of Android malware families [11].

Yan et al. [25] presented a visualization framework to assist malware analysts in statically locating the malicious code. The proposed tool first prunes an app's function call graph into sub graphs that contain suspicious behaviors, and visualizes the graphs using a force-directed scheme. A set of interactions are provided for malware analysts to verify and further dissect the code block related to suspicious behaviors. Ganesh et al. also presented a visualization toolbox [21] designed to efficiently display program artifacts in manual malware analysis. They encoded the hierarchical class structure using containment marks, and allowed the user to abstract call sequences to a higher level by changing the visualization scope. To help researchers locate malicious behaviors while running the app, De Lorenzo et al. [8] presented a ML-based framework trained to classify blocks of execution traces related to malicious behaviors. After deployment, it monitors and visualizes the maliciousness of execution traces through a temporal bar chart.

Clustering Android malware into different families based on their malicious behaviors can also help malware analysts prevent attacks from malware variants. Gonzales et al. [11] applied various dimension reduction techniques to project high dimensional traffic data of malware applications to 2D, and visualized the results using scatter plots. By comparing the results from different projection functions, they identified differentiating traffic patterns among malware families.

To the best of our knowledge, only Rory et al. [7] have presented a tool to study similarities between benign and malware applications. They measured the distance among applications in the feature space and used a circular dendrogram to visualize the hierarchical relationships among applications. Malware analysts can easily infer the distance between applications by locating them on the dendrogram, and understand the classification results from classifiers trained on the same applications.

8.2 Visualizing Machine Learning Data

Over the years, an abundance of work has been proposed to incorporate visualization into the machine learning process. Most have focused on providing visualizations to help users explore correlations between features, or provide an interaction interface to keep users in the loop and improve the reliability of the model. Only a few tools [12, 15, 20, 24] have been proposed to help users investigate training data set characteristics, or to examine relationships between the training data and resulting model properties [14].

Patel et al. presented a visualization framework [20] to understand properties of the training data. Their tool helps users identify noteworthy examples by exploiting the classification results based on multiple trained model instances. The Profiler [15] system proposed by Kandel et al. automatically flags problematic data using data mining techniques. The authors present results of their flagging process by using coordinated summary visualizations that support interactions and links to assess detected anomalies. Wongsuphasawat et al. [24] presented a tool to help analysts in discovery by offering recommendations of potentially interesting visualizations on particular features. The tool also generates coordinated views given a user's partial specifications. In addition, Google launched the web interface FACETS [12] for users to visualize their high-dimensional machine learning data. It offers two visualizations: "Facets Overview" and "Facets Dive". "Facets Overview" produces summary statistics over each feature, and compares the distributions over training and testing data. "Facet Dive" provides an interactive interface for users to sort and examine individual applications in the data.

Hohman et al. [14] pointed out that understanding the data should be of equal importance as understanding the model. They describe that it is crucial to evaluate the quality of training data and monitor the performance of the model. They presented a visualization interface,

"CHAMELEON", which integrated multiple coordinated views to help researchers compare the trained model over different data sets.

9 Conclusions

In this paper, we presented a visualization system aimed to facilitate Android malware researchers in exploring the relationship between training samples and the resulting detection model through interactive visualization. The system uses multiple views based on aggregated information to help the researcher explore and discover trends and outliers in Android malware detection. Using provided interfaces, the researcher can select a subset of samples with particular characteristics for in-depth analysis. Additionally, with the process of training and testing Android malware detection model integrated into the system, the researcher can easily assess the trade-offs between performance and attackability of the trained model. Visualizations of the derived data from the trained model are provided for researchers to comprehend and verify the model's learning results. We hope, with the interactive visualization on training data and model performance results provided in this system, Android malware researchers can distill the relationship between applications and the trained model more effectively.

Acknowledgments

The authors wish to thank the README project group and professor Munzner for their thoughtful feedback.

References

[1] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, R. State, and Y. Le Traon. Empirical Assessment of Machine Learning-Based Malware Detectors for Android. *Empirical Software Engineering (ESE)*, 21(1):183–211, 2016.

[2] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proc. International Conf. on Mining Software Repositories (MSR)*, pp. 14–15, 2016.

[3] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proc. Network and Distributed System Security Symp. (NDSS)*, pp. 23–26, 2014.

[4] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli. Evasion Attacks Against Machine Learning at Test Time. In *Proc. European Conf. on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, pp. 387–402, 2013.

[5] M. Cao, S. Badihi, K. Ahmed, P. Xiong, and J. Rubin. On benign features in malware detection. In *Proc. International Conf. on Automated Software Engineering (ASE)*, pp. 1234–1239. ACM, 2020.

[6] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[7] R. Coulter, L. Pan, J. Zhang, and Y. Xiang. A visualization-based analysis on classifying android malware. In *Proc. International Conf. on Machine Learning for Cyber Security*, pp. 304–319. Springer, 2019.

[8] A. De Lorenzo, F. Martinelli, E. Medvet, F. Mercaldo, and A. Santone. Visualizing the outcome of dynamic analysis of android malware with vizmal. *Journal on Information Security and Applications*, 50:102423, 2020.

[9] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli. Yes, Machine Learning Can Be More Secure! a Case Study on Android Malware Detection. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 16(4):711–724, 2017.

[10] J. Garcia, M. Hammad, and S. Malek. Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 26(3), 2018.

[11] A. González, Á. Herrero, and E. Corchado. Neural Visualization of Android Malware Families. In *International Joint Conf. SOCO-CISIS-ICEUTE*, pp. 574–583. Springer, 2016.

[12] Google. Facets - know you data. `https://pair-code.github.io/facets/`.

[13] M. Harrower and C. A. Brewer. ColorBrewer. org: An Online Tool for Selecting Colour Schemes for Maps. *The Cartographic Journal*, 40(1):27–37, 2003.

[14] F. Hohman, K. Wongsuphasawat, M. B. Kery, and K. Patel. Understanding and Visualizing Data Iteration in Machine Learning. In *Proc. Conference on Human Factors in Computing Systems (CHI)*, pp. 1–13, 2020.

[15] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment. In *Proc. of the International Working Conference on Advanced Visual Interfaces(IWCAVI)*, pp. 547–554, 2012.

[16] L. v. d. Maaten and G. Hinton. Visualizing Data Using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[17] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, pp. 1–12, 2017.

[18] L. McInnes, J. Healy, and J. Melville. Umap: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[19] T. Munzner. *Visualization Analysis and Design*. CRC press, 2014.

[20] K. Patel, S. M. Drucker, J. Fogarty, A. Kapoor, and D. S. Tan. Using Multiple Models to Understand Data. In *International Joint Conf. on Artificial Intelligence (IJCAI)*. Citeseer, 2011.

[21] G. R. Santhanam, B. Holland, S. Kothari, and J. Mathews. Interactive Visualization Toolbox to Detect Sophisticated Android Malware. In *IEEE Symp. on Visualization for Cyber Security (VizSec)*, pp. 1–8. IEEE, 2017.

[22] Statista. Number of smartphone users worldwide 2014-2020. `https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/`.

[23] VirusTotal. VirusTotal. `https://www.virustotal.com/home`, 2020. last accessed August 2020.

[24] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *Proc. of Conference on Human Factors in Computing Systems (CHI)*, pp. 2648–2659, 2017.

[25] Y. Zhang, G. Peng, L. Yang, Y. Wang, M. Tian, J. Hu, L. Wang, and C. Song. Visual Analysis of Android Malware Behavior Profile Based on PMCGdroid: A Pruned Lightweight APP Call Graph. In *International Conf. on Security and Privacy in Communication Systems (ICSPCS)*, pp. 449–468. Springer, 2017.

A Appendix

The table below breaks down the implementation tasks performed by each of the group members.

Tasks	Author	Apr. Hours
Data Collection	Michael	4 hours
Back End		
Framework Initialization	Michael	2 hours
Database Initialization	Gabby	6 hours
Dimension Reduction	Gabby	6 hours
Sample Feature Extraction and Feature Aggregation	Gabby	5 hours
Model Training On Selected Data	Gabby	3 hours
Model Evaluation (Accuracy Metrics)	Gabby	2 hours
Model Evaluation (Attackability)	Michael	4 hours
Collecting Detailed Testing Results on Single Testing Samples	Gabby	4 hours
Front End		
Dataset Overview	Gabby	10 hours
Temporal Distribution View	Michael	8 hours
Feature Usage Distribution View	Michael	10 hours
Average Feature Usage Per Category View	Michael	8 hours
Selection Panel	Michael	6 hours
Training & Testing Data Overview	Gabby	8 hours
Classification Result View	Michael	4 hours
Model Robustness Result View	Michael	12 hours
Model Weight Result View	Gabby	6 hours

Fig. A.1: Detailed implementation tasks breakdown