Generation of Malware Samples Using Deep Autoencoders

Project Workbook

By

Aaron Choi
Albert Giang
Sajit Jumani
David Luong

January 2023

Advisor: Fabio Di Troia, Ph.D.

Workbook Contributions

Section	Description	Person
1	Literature Survey and SOTA Summary (1pg)	Sajit
1	References	Sajit
2	Project Justification (1pg)	Albert/David
3	Project Requirements	Aaron
4	Dependencies and Deliverables	Aaron
5	Architecture (1pg) and Description (1pg)	David
8	Implementation Plan and Progress	Albert/David
9	Project Schedule	Albert

Chapter 1. Literature Search, State of the Art

Literature Search

Malware attacks represent a significant and evolving threat to information security. Malware continues to increase in sophistication and can exploit an ever expanding attack landscape as digitization and connectivity permeates more facets of society. Malicious software, or malware, is an intrusive software designed to harm a computer, system, or network (Cisco, 2022). There are several types of malware, the most common being viruses, worms, trojans, spyware, ransomware, adware, fileless malware, bots, rootkits, mobile malware, and wiper malware (Baker, 2022). These can cause disruptions to businesses, supply chains, infrastructure, government systems, financial systems, personal computers and mobility systems, autonomous systems, and more.

In response to these threats, defense against malware attacks is also improving through research that applies machine learning to classify and detect malware (Ucci et al., 2019; Aslan & Samet, 2020), as well as machine and deep learning approaches to generate synthetic malware for augmenting training datasets (Trehan & Troia, 2021). Integral to all of these research fields is the extraction and representation of malware features, which serve as inputs to various models. There are several feature extraction options, including using n-grams, metadata, entropy, opcodes, as well as byte codes that can be represented as images. Oftentimes, combinations of these approaches complement each other to provide a comprehensive view of the malware signature and behavior (Ahmadi et al., 2016).

To train the latest malware classifiers, large datasets are required. Unfortunately, finding large malware datasets is challenging so there is active research in producing synthetic data to augment (or boost) the datasets. A popular approach is to represent malware as images and use various machine learning and deep learning generators to augment those datasets. Burks et al. (2019) has shown that both Generative Adversarial Networks (GAN) and Variational Auto-Encoders (VAE) can generate high quality synthetic benign and malware images, validated by single digit percentage improvements in the accuracy of deep learning classifiers when differentiating between benign and malicious files. Lu and Li (2019) achieved similar results when using a deep convolutional GAN (DCGAN) to boost the training datasets.

An alternative representation of malware is to replace images with operation codes, or opcodes, which are mnemonic representations of machine code that symbolize assembly instructions. The opcodes serve as inputs to machine and deep learning based malware generators, such as Hidden Markov Models (HMM) and Generative Adversarial Networks (GAN) as demonstrated by Trehan and Troia (2021). Their results were insightful; both HMM and Wasserstein GAN (WGAN) were unable to produce considerably realistic synthetic malware opcode sequences, however WGAN with Gradient Penalty (WGAN-GP) showed promise by reducing the average classifier accuracy by approximately 20% compared to HMM and WGAN (Trehan & Troia, 2021).

The use of malware classifiers deserves further explanation because they are measuring different, but related, properties of the synthetically generated data. There are two common implementations: first, where a classifier is differentiating between benign versus malicious software, and second, where the classifier is differentiating between real versus synthetic malware. In the first scenario, an improvement in classifier accuracy is desired because it indicates that the synthetic data that augmented the training dataset adequately represented both benign and malicious software. However, in the second scenario a decrease in the classifier accuracy is desired because it indicates that the synthetic malware is harder to differentiate from the real malware. The second scenario is only looking at malware, while the first scenario is looking at both benign and malicious software. Said another way, the creation of synthetic malware may be different enough from synthetically generated benign software for a classifier to identify, however, that does not imply that the synthetic malware was indistinguishable from real malware. Implementing classifiers from the second scenario more directly evaluates the quality of the synthetic malware compared to real malware.

Our research attempts to improve upon the performance of WGAN-GP by using deep variational auto-encoders (DVAE) to generate synthetic malware opcodes. Somewhat similar work was done by Bae and Lee (2021) where a relatively shallow VAE was used to augment the training dataset of opcodes (both benign and malicious files). Their VAE had a 128-dimensional embedding layer followed by the encoder and decoder which each contained one hidden layer and one gated recurrent unit (GRU) layer. This resulted in an improvement in accuracy of 0.98% for their long-short term memory (LSTM) classifier. This points toward their VAE being able to produce adequately realistic benign opcode and malware opcode sequences. However, it does not address the potential performance improvement of adding depth to the VAE, nor does it quantify how realistic the synthetic malware was compared to real malware.

There are several machine learning and deep learning classifiers available when classifying synthetic versus real malware opcode sequences. Trehan and Troia (2021) used four machine learning classifiers: support vector machines (SVM), k-nearest neighbor (kNN), random forest (RF), and Naive Bayes (NB). Of the four, SVM and RF had the best classification performance.SVM has been proven to be an effective classifier when classifying malware family types on deep learning models per Agarap (2017). RF also has been proven to be effective when classifying malware images to malware

families per Garcia and Muga (2016). When considering deep learning classifiers for opcodes, the trend appears to be using recurrent neural networks (RNN), most specifically an LSTM. Bae and Lee (2021) implemented LSTM in their work, and Maniriho et al. (2022) provided multiple examples in their survey paper where RNNs and LSTMs were paired with opcode sequences.

State-of-the-Art Summary

Synthetic malware generation is a growing field that uses both machine learning and deep learning techniques to augment datasets used to train malware classification and detection algorithms. The primary driver behind selecting the appropriate generator is based on which malware features will be extracted and how they will be represented. API call sequences, images, opcode sequences, assembly code and dynamic-link library import, control flow graph, portable executable, network activities, printable strings, malicious domain name server, contents of registers, registry changes, mutexes, running processes and threads, and browsing history are all examples of features that can be extracted from malware (Maniriho et al., 2022). After extracting the desired features there are several options for how to represent the data. One popular example is converting the binary malware file into an 8-bit grayscale image which can then be replicated three times to create an RGB corollary (Burks et al., 2019). Another approach is to use the opcode sequences and convert them to an integer vector, n-grams, or apply Word2Vec embedding (Trehan & Troia, 2021). For our research we will work with opcodes and implement Word2Vec embedding.

A common machine learning approach for sequential or time-series datasets is the HMM. Trehan and Troia (2021) employed HMMs to generate synthetic malware opcode sequences with good results, matching the performance of WGAN. However, to generate even better synthetic malware, deep learning approaches need to be considered. Most often researchers use GANs, VAEs, and regular auto-encoders (AE) when generating malware in either image format or opcode sequences (Bae & Lee, 2021; Burks et al., 2019; Trehan & Troia, 2021; Lu & Li, 2019). When working with opcode sequences, WGAN-GP appears to produce some of the most realistic synthetic malware, performing better than HMM, regular GAN, and WGAN (Trehan & Troia, 2021). While VAEs have been used to generate synthetic malware opcode sequences, the VAEs were relatively shallow networks and the malware produced was not evaluated against real malware (Bae & Lee, 2021). Comparing the ability of a deep VAE against WGAN-GP for generating synthetic malware opcode sequences has not yet been explored and forms the basis of our research.

References

1. Agarap, A. F. (2017). Towards building an intelligent anti-malware system: a deep learning approach using support vector machine (SVM) for malware classification. *arXiv preprint arXiv:1801.00318.*

   The paper focuses on using deep learning models for anti-malware detection systems. The deep learning models are trained to classify different malware family types using SVM classifiers.

2. Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., & Giacinto, G. (2016, March). Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy* (pp. 183-194).

   Categorizing malware variants into their family groups by focusing on extraction, selection of features, and representation of more than 20,000 malware samples.

3. Aslan, Ö. A., & Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE Access*, *8*, 6249-6271.

   Summary of various malware detection approaches, highlighting their strengths, weaknesses, applications, and areas for future improvement.

4. Bae, J., & Lee, C. (2021, January). Easy Data Augmentation for Improved Malware Detection: A Comparative Study. In *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)* (pp. 214-218). IEEE.

   Generated synthetic malware opcodes using easy data augmentation (EDA) and VAE, resulting in a 1.76% and 0.98% improvement of classification accuracy, respectively, by an LSTM.

5. Baker, K. (2022, August 11). *12 Types of Malware + Examples That You Should Know*. CrowdStrike. Retrieved January 14, 2023, from https://www.crowdstrike.com/cybersecurity-101/malware/types-of-malware/

   CrowdStrike article describing 12 common types of malware.

6. Burks, R., Islam, K. A., Lu, Y., & Li, J. (2019, October). Data augmentation with generative models for improved malware detection: A comparative study. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)* (pp. 0660-0665). IEEE.

VAE and GAN generated synthetic malware to train a ResNet-18 classifier, resulting in 2% and 6% improvements in malware detection, respectively.

7. Cisco. (2022, June 6). *What is malware? - definition and examples*. Cisco. Retrieved January 14, 2023, from https://www.cisco.com/c/en/us/products/security/advanced-malware-protection/what-is-malware.html

   Cisco article explaining what malware is, how it works, and how to defend against malware attacks.

8. Garcia, F. C. C., & Muga II, F. P. (2016). Random forest for malware classification. *arXiv preprint arXiv:1609.07770*.

   The paper focuses on converting malware binary into an image. To classify the different malware families, the paper focuses on Random Forest and shows effective Random Forest as an effective method for malware detection.

9. Lu, Y., & Li, J. (2019, December). Generative adversarial network for improving deep learning based malware classification. In *2019 Winter Simulation Conference (WSC)* (pp. 584-593). IEEE.

   Generated synthetic malware images using a deep convolutional GAN resulting in a 6% improvement in accuracy of a deep RNN classifier.

10. Maniriho, P., Mahmood, A. N., & Chowdhury, M. J. M. (2022). A Survey of Recent Advances in Deep Learning Models for Detecting Malware in Desktop and Mobile Platforms. *arXiv preprint arXiv:2209.03622*.

    Survey of the latest deep learning malware detection models across Windows, Linux, and Android platforms with emphasis on algorithms, network optimization, regularization, loss functions, activation functions, as well as feature extraction approaches.

11. Trehan, H., & Troia, F. D. (2021, December). Fake Malware Generation Using HMM and GAN. In *Silicon Valley Cybersecurity Conference* (pp. 3-21). Springer, Cham.

    Compares the synthetic malware (as opcode) generation capability of HMM, GAN, WGAN, and WGAN-GP across 4 machine learning classifiers and 3 feature representation transformations.

12. Ucci, D., Aniello, L., & Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. *Computers & Security*, *81*, 123-147.

Survey of machine learning techniques for malware analysis focusing on dataset features, and algorithms that process the inputs and outputs.

Chapter 2. Project Justification

The evolution of malware attacks in the past decade has grown and evolved to dangerous levels. Cybersecurity Ventures, an online cybersecurity publication, predicts that cybercrime will cost the world $10.5 trillion annually by the year 2025. Research into advanced malware detection algorithms is needed now more than ever with machine learning techniques emerging as an effective tool to counter many typologies of attacks in the cybersecurity domain.

Large quantities of malicious samples are required to develop effective machine learning and deep learning cybersecurity solutions. Improving the efficacy of detecting malicious files is significantly influenced by the quality of the training dataset, specifically sample size and authenticity. The lack of high-quality training data is one of the greatest challenges to achieving widespread adoption of malware detection by trained machine learning and deep learning models. In response, generative techniques have demonstrated initial success in producing synthetic malware samples (by recreating a list of opcodes) that are, in some cases, indistinguishable from real malware by machine learning detection models.

A trivial method to generate more malware training samples is simply modifying actual malware. Taking opcodes from benign code and inserting and replacing opcodes in malicious code is intuitive. However, while straightforward in producing opcodes resembling actual malware, the generated malware may not be representative of the actual malware and may actually cause malware detectors to learn the wrong signatures. Furthermore, the true malware parameters are never learned.

This project proposes to generate malware samples as opcode sequences by implementing a deep variational autoencoder with multiple original malware families as input. We attempt to differentiate generated malicious code from actual malicious code samples using machine learning and deep learning techniques as validation methods. The goal is to produce imitation malware data that can effectively train deep learning models in combination with the real samples, ultimately enhancing the model's efficacy through enlarged training datasets.

Chapter 3. Project Requirements

Requirements

User Story:

User story is an informal, high-level description of a feature in a software system. User stories are used in software development and product management.

- As a user I can generate samples of opcode sequences using real opcode sequences.
- As a user I can generate multiple malware family types.
- As a user I can see the results of malware detection classifiers.

Requirement List:

The requirements have been listed as per necessity in Table-1. The requirements are divided into three categories:

1. Essential: Essential requirements are required or else the software will not be acceptable.

2. Desired: Desired requirements would enhance the software product, but not make it unacceptable if the software product did not fulfill the desired requirements.

3. Optional: Optional requirements would not be necessary for the software product, but present an opportunity to improve the software product.

Requirement ID	**Description**
Essential	
REQ_1	The proposed solution shall generate sample opcode sequences using real opcode sequences.
REQ_2	Real opcode sequences shall be provided to the opcode sequence generator.
REQ_3	The proposed solution shall generate opcode sequences of multiple different malware family types.

REQ_4	The malware detection classifiers shall use both real and generated opcode sequences.
REQ 5	The malware detection classifiers shall use machine learning techniques to distinguish between real and generated opcode sequences.
REQ_6	The malware detection classifiers shall provide detection accuracy, precision, and recall results.
Desired	
REQ_7	The malware detection classifiers shall use deep learning techniques to distinguish between real and generated opcode sequences.
Optional	
None	None

Chapter 4. Dependencies and Deliverables

Dependencies

GPU access for training – when training the WGAN-GP and VAEs, there could be limited resources available in terms of GPU availability. This could limit the training time and number of epochs that will be implemented on the training.

Deliverables

Project code– Project code using VAE and GAN libraries to generate malware opcodes and machine learning and deep learning classifiers to differentiate between fake and real samples.

Results – Results of the machine learning and deep learning classifiers. The results will show classification metrics from Support Vector Machines (SVM), Random Forest (RF) and a deep neural network classifier.

Chapter 5. Project Architecture

The project architecture is shown in Figure 1. Real malware opcodes serve as inputs to Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) such as Wasserstein GAN with Gradient Penalty (WGAN-GP). VAEs and GANs generate fake malware opcodes that are ideally indistinguishable from the real malware. In other words, the generated malware resembles the real malware and cannot be identified as generated. Machine learning and deep learning classifiers are employed to identify the real and fake malware. The classifiers include Support Vector Machines (SVM), Random Forest, and deep learning neural networks. The results of the VAE and GAN generated malware are compared to determine which method generates malware that best fools the classifiers.

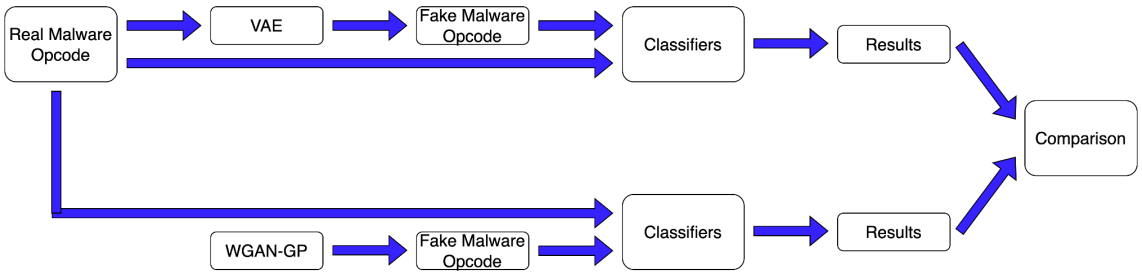


Figure 1. Project Architecture

In the VAE method, a lower-dimensional latent space is learned through malware encoding and the new malware is generated by reconstructing the input through sampling the latent space. The VAE architecture is shown in Figure 2. The real malware opcodes feed an encoding stage that learns the statistical latent space followed by passing through a decoding stage which generates fake malware. The encoder learns the mapping from the real opcode data to a lower-dimensional latent space while the decoder reconstructs the opcode. In contrast with traditional autoencoders, the VAE latent space is a probabilistic distribution parameterized by a mean and standard deviation. The sampling from this distribution allows the generation of new sample malware by perturbing the latent variables. By manipulating the latent variables so they are uncorrelated, one or more latent variables can be varied while keeping the others fixed. The perturbation of independent latent variables can uncover hidden latent variables, allowing the generation of unbiased and representative malware.

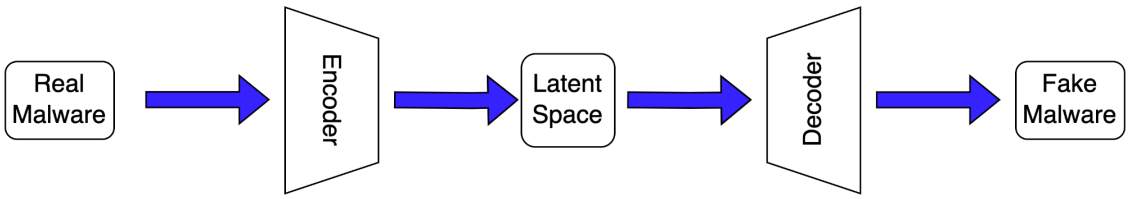


Figure 2. Variational Autoencoder Architecture

Alternatively, malware can be generated by simply sampling from noise and learning a transformation to the malware opcodes distribution. In a competing fashion, GANs have generators turn noise into fake malware that is then sent to a discriminator which attempts to correctly identify the fake from the real. The network is shown in Figure 3. During training of these adversarial objectives, the optimum point is reached when the generator learns the true malware distribution and the discriminator can no longer tell the difference between fake and real malware. After training, the generator can then create new malware from the learned transformation.

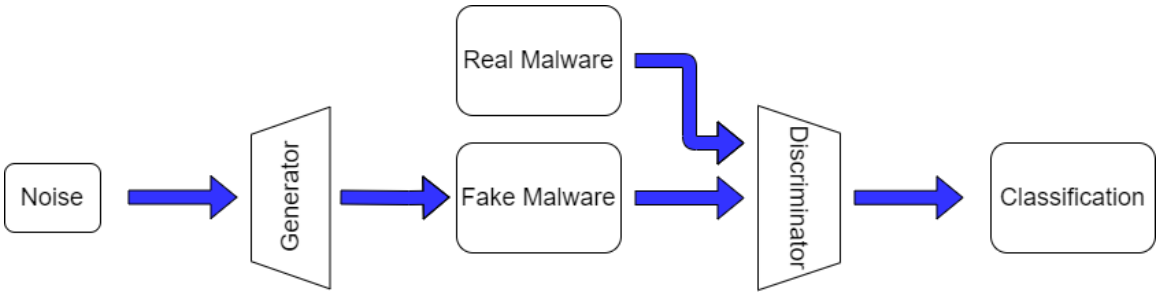


Figure 3. GAN Architecture

Chapter 6. Project Design

Workbook 2 Assignment

Chapter 7. QA, Performance, Deployment Plan

Workbook 2 Assignment

Chapter 8. Implementation Plan and Progress

This section outlines the implementation plan for our group's goal of using a deep variational autoencoder that can generate synthetic malware opcode sequences that are indistinguishable from real malware. There are many steps at the start of the project that are tackled in parallel by different members of the team. The output of each team member's work is then combined together to produce working opcode generators and classifiers.

Summary of Development Tools:

- Code Development: Jupyter file formats developed in Google Collab
- Deep learning framework: TensorFlow, Keras
- Programing language: Python
- Model training computational resource: Google Collab
- Graph visualizations: Google Charts, Diagram.net
- File Hosting: Google Drive
- Project Management Tools: TeamGantt

Implementation Plan:

1. Dataset Setup:

Our team will train deep-learning models with 6 different families of malware. Winwebsec, Zbot, and Zeroaccess families of malware come from a dataset compiled by the Malicia Project and each family has more than 1000 samples each. VirusShare also compiled the Renos, VBInject, and OnLineGames family of malware which also has thousands of samples to train with. Since the malware provided in these datasets are executables, our group decompiled the programs into Assembly code for opcode extraction. Unnecessary lines of code such as registers, labels, and addresses were removed to obtain the opcode sequences that are used in model training.

2. VAE Implementation

The team will use Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks for the encoder and decoder parts of the VAE. LSTMs and GRUs overcome the short term memory problem when learning the latent feature space from long opcode sequences.

3. WGAN-GP Implementation

For WGAN-GP, our group will reproduce the model parameters from Trehan, H., & Troia[6]. The team will enforce a gradient penalty on the discriminator's cost function and optimize its performance. The model architecture is written in Python using TensorFlow and Keras as the backend. An Adam Optimizer will be used for our stochastic gradient descent function. The loss function will be the Wasserstein distance which calculates the realness or fakeness of a given opcode. Our group will train the WGAN-GP model for 100,000 epochs.

Progress:

1. Dataset Setup:

Dr. Troia has provided 6 families of malware to our team to train our deep learning models with. Our group is currently looking through each dataset to understand each of the malware families while preparing the Python preprocessing scripts to generate the opcodes that will go into our generation models. The team is also evaluating Word2Vec and n-grams to transform the opcode feature sets into a list that the deep learning algorithms can ingest.

2. VAE Implementation

Our team is currently reviewing research papers describing how to implement deep VAEs

3. WGAN-GP Implementation

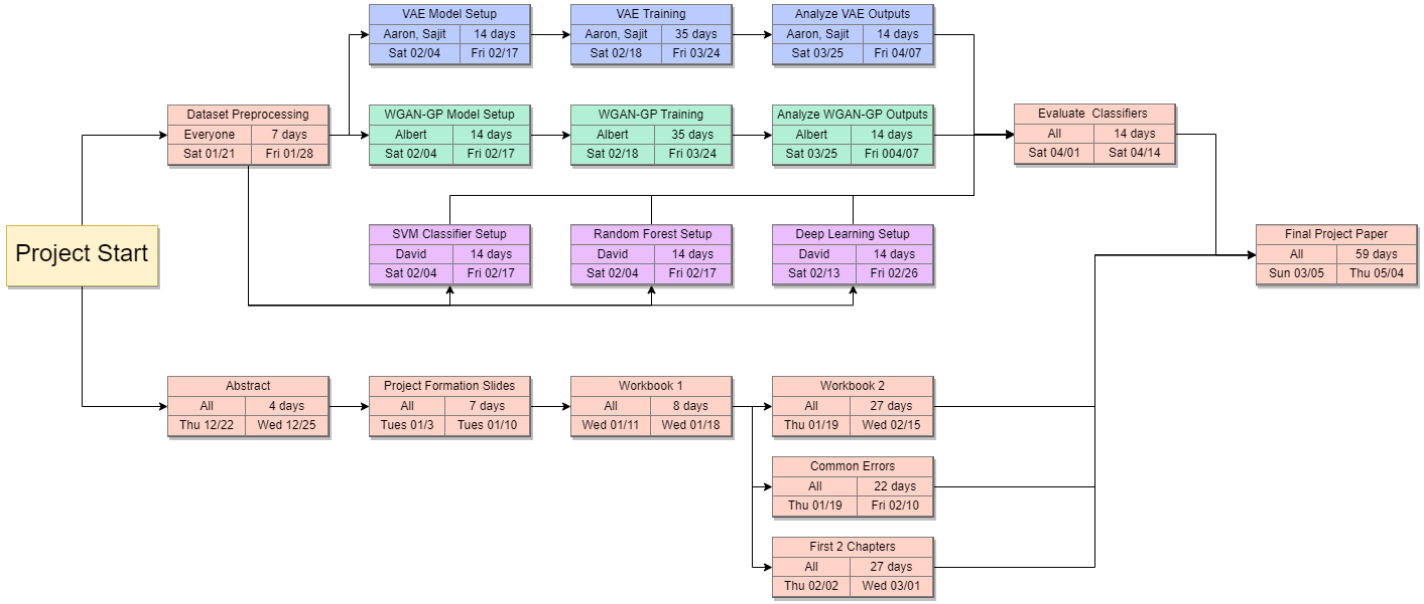
The team is studying the previous work by Harshit Trehan and how he implemented WGAN-GP. We plan to reproduce his approach and results as a baseline.

4. Classifiers

We are currently reviewing the documentation and past work of the two machine learning classifiers implemented by Harshit Trehan: Support Vector Machines and Random Forest. Our group is also going through a literature review to select the best deep neural network classifier. Our research suggests that a Recurrent Neural Network such as a Long Short-Term Memory (LSTM) network could work well for our classification task.

Chapter 9. Project Schedule

Project schedule and progress are tracked using a Gantt chart hosted on the online collaborative platform, TeamGantt. A PERT chart was created from the Gantt chart schedule to help visualize the task timelines and dependencies.



teamgantt

Generation of Malware Sam...	start	end	0h	6%
Codebase	**01/13/23**	**04/14/23**	**0h**	**3%**
Dataset	**01/13/23**	**02/03/23**	**0h**	**9%**
Verify Data Contents	01/13	01/20	0	25%
Dataset Preprocessing	01/21	01/27	0	0%
Finalize Dataset	01/28	02/03	0	0%
Variational AutoEncoder (VAE)	**01/28/23**	**04/07/23**	**0h**	**3%**
Review VAE Documentation	01/28	02/03	0	25%
Implement Initial Model	02/04	02/17	0	0%
Train and Refine Model	02/18	03/24	0	0%
Analyze and Visualize Outputs	03/25	04/07	0	0%
Wasserstein GAN (WGAN-GP)	**01/28/23**	**04/07/23**	**0h**	**3%**
Review Prior Work	01/28	02/03	0	25%
Implement Initial Model	02/04	02/17	0	0%
Train and Refine Model	02/18	03/24	0	0%
Analyze and Visualize Outputs	03/25	04/07	0	0%
Malware Classifiers	**01/28/23**	**03/10/23**	**0h**	**3%**
Review Classifier Documentation	01/28	02/03	0	25%
Set Up Support Vector Machines	02/04	02/17	0	0%
Set Up Random Forest	02/04	02/17	0	0%
Set Up Deep Learning Classifier	02/13	02/26	0	0%
Finalize All 3 Classifiers	02/27	03/10	0	0%
Final Steps	-	-	**0h**	**0%**
Evaluate Classification Metrics	04/01	04/14	0	0%
Writing Assignments	**12/22/22**	**05/11/23**	**0h**	**9%**
Abstract	12/22	12/25	0	100%
Project Formation Slides	01/03	01/10	0	100%
Workbook 1	01/11	01/18	0	50%
Workbook 2	01/19	02/15	0	0%
Common Errors Assignment	01/19	02/10	0	0%
Project Report First 2 Chapters	02/02	03/01	0	0%
Final Draft For Review	04/30	04/30	0	0%
Final Project Paper	03/06	05/04	0	0%
Presentation Slides	04/29	05/07	0	0%
Project Presentation	05/11	05/11	0	0%