Trojan AI Detection

Christopher Armstrong (cparmstr@ucsd.edu), **Daniel Hartley** (dhartley@ucsd.edu),
Spencer Hutton (sphutton@ucsd.edu), **Shirley Quach** (shquach@ucsd.edu)

Advisors: XinQiao Zhang, Dr. Tara Javidi, Dr. Farinaz Koushanfar

June 9, 2023

Table of Contents

1. Abstract

As machine learning (ML) gains prominence in the business world, the implementation of deep neural networks (DNN) has become more widespread. The security of DNN models has recently come under scrutiny as they are at risk of adversarial attacks such as backdoor Trojan attacks. These attacks depend on a trigger to activate malicious behavior. Due to the lack of transparency in DNNs, the effects of Trojans may remain undetected until activated by an attacker. This project demonstrates a significant reduction in the time and resources necessary to detect a poisoned model through the use of dimensionality reduction techniques. The detector utilizes Principal Component Analysis (PCA) and Independent Component Analysis (ICA) to reduce model weights that can then be used to train a classification model. This work builds on previous research, integrating reduction techniques to significantly reduce inference time while maintaining model accuracy at 85%. Are you protected from malicious AI?

2. Introduction

Data has taken a prominent role in the world and only continues to increase in importance. The demand for tools to analyze and model this data has continued to escalate. The complexity of these models has grown to a point where even large companies cannot train them “in-house” and must either outsource the training task to the cloud or rely on pre-trained models that they can then tune for their specific needs. Both of these options leave the company vulnerable to new security risks.

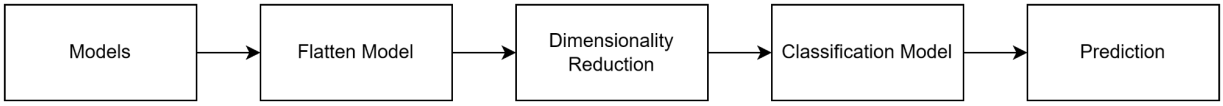
During the training process a malicious actor can step in, augment some of the training data to create a backdoor in the model. The scope of this attack can vary but the goal remains the same, to provide the end user with a model that is well trained for most inputs but causes misclassifications on a targeted class or degrades the accuracy of the model for inputs that contain the secret augmented property the attacker has chosen. Such inputs could cause models intended to diagnose diseases to misclassify either positively or negatively the disease or condition they were meant to identify.

Trojan attacks are nearly impossible for end-users to detect, let alone to be aware it is even a possibility. Even the data scientists working with the model may not know that the training data had been tampered with until the secret condition had already been met and the backdoor triggered. This risk is present even if the company

has taken a pre-trained model and adapted it to their needs. While most of the difficulty in training will be handled in this case, a trigger that existed in the original model will still affect its performance.

If the backdoor has already been triggered then it is too late to prevent the attack from occurring. While preventing a malicious party from accessing the model during training would be ideal, it is hard to control. However, a company has full control over whether or not they choose to deploy a trained model. Detecting a trojaned model before it is deployed is the best way a company can ensure an end user does not suffer from one of these attacks.

Although Round 1 of the NIST TrojAI leaderboards took place in June 2020, there was more we could expand and improve on. We hypothesize that trojan models may be detected by examining the model's weights. The method is fast, cheap and may be integrated into a more complex detection model.



3. Team Roles and Responsibilities

Christopher Armstrong - AWS Solutions Architect
Daniel Hartley - Data Visualization
Spencer Hutton - Data Scientist
Shirley Quach - Project Manager

4. Data Acquisition

Our dataset is the **image-classification-jun2020** dataset from Round 1 of the TrojAI competition. This dataset consists of 1000 trained, human classified image classification AI models using the following architectures (Inception-v3, DenseNet-121, and ResNet50). The models were trained on synthetically created image data of non-real traffic signs superimposed on road background scenes. Half (50%) of the models have been poisoned with an embedded trigger which causes misclassification of

the images when the trigger is present. Models in this dataset are expecting input tensors organized as NCHW. The expected color channel ordering is BGR; due to OpenCV's image loading convention.

NCHW stands for number of data samples **N** , image channels (where a RGB image will have three channels) **C**, depth **D**, image height **H**, image width **W**. This is one way for storing multidimensional arrays/data frames into memory into a 1-D array. .

Ground truth is included for every model in this dataset.

4.1. Data Sources

Dataset Name	Image Classification on Models (Round 1)
Source	https://data.nist.gov/od/id/mds2-2195
Destination	• AWS S3 bucket: s3://image-classification-jun2020/data/ • Locally
Acquisition Notebooks, Code, Documents	https://pages.nist.gov/trojai/docs/image-classification-jun2020.html#image-classification-jun2020 https://github.com/usnistgov/trojai-example
Data Size	150 GB compressed
Other Notes	The dataset has a software bug in the trigger embedding the code that caused 2 models trained for this dataset to have a ground truth value of 'poisoned' but did not contain any triggers embedded. Models without an embedded trigger: id-00000077 id-00000083

4.2. Data Collection, Environment and Pipeline

Our data is composed of 150GB of compressed models. Given that the total data set was 150GB, it was an unreasonable ask to have all of it downloaded locally. For our data preprocessing approach we decompressed the files using tar. Further preprocessing steps are not applicable in this case because our given data were models. A small portion of the data is stored locally because it is static and too large to download at the time of execution. The benefit of having AWS S3 available is to relieve us of storing everything locally. It was effective and more advantageous to run the detector remotely once we had a working prototype.

The data taken from the Google Drive mirror was uncompressed at the time of execution and saved directly to a S3 bucket in AWS using the AWS CLI. Locally, we had a subset of the uncompressed data for use cases where the entire dataset was not necessary in order to accelerate access and build a prototype. We leveraged our AWS resources to support the larger datasets and requirements for accelerated computing resources.

Once we loaded the model file, we extracted its weights and transformed these weights into a set of features. These features are extracted at the time of detection and not stored.

5. Data Preparation

Given our problem statement and dataset our method for data processing mostly involved copying and decompressing the files using tar and downloading the dataset that was stored in a google drive to our AWS S3 bucket. The dataset is ~150GB. Due to the size of the dataset it was determined impractical to download the dataset programmatically every time the detector is run. Since the project uses a static set of models, there is no additional need to refresh frequently. We needed to be aware of the dataset having a software bug in the trigger embedding code that caused 2 models trained for this dataset to have a ground truth value of ‘poisoned’ but did not contain any embedded triggers. The models without an embedded trigger that you may need to be aware of include: id-00000077, id-00000083. Additionally, we did not integrate more data outside of the NIST TrojAI source.

For initial exploration, a small set of the dataset was stored locally to replicate the results from the first round. To utilize data from the image classification rounds that followed Round 1, it will be necessary to utilize AWS with GPU resources such as CUDA capable with 32+ GB of RAM.

6. Analysis Methods

We started by collecting the results on the performance of different dimensionality reduction methods and classification models. By applying different dimensionality reduction methods we hope to eliminate noise, any redundant features, improve the model's accuracy and performance. Comparison of the shortcomings of each would drive the next step of development.

Dimensionality reduction methods we wanted to explore and compare included Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) to compare and look for insights between the performance between and unsupervised algorithm (PCA) and supervised algorithm that takes class labels into account (LDA).

For our trial run, we trained a random forest regression model to predict whether or not a given model is poisoned on a subset of data that was roughly ~10GB in size. Hyper parameter tuning was done to improve the model. Additional classifiers will be evaluated after our implementation is successfully migrated to AWS. As a result, we were able to successfully train and test a basic model utilizing our proposed architecture. There were challenges regarding code execution. We discovered that a small number of samples caused the dimensionality reduction to fail, and a large number of samples resulted in hitting a system memory constraint. Initial performance was particularly poor, because execution occurred locally on a small subset of our models. However, we believed this demonstrated that our approach was viable to continue to develop our process.

In order to overcome our memory limitations during model training we needed to migrate to AWS where we were able to test additional classification models and compare performance where we were able to successfully train and test a basic model utilizing our proposed architecture. We were also able to flatten and reduce the models in the training data set and save them as a .pkl file using Pythons pickle library. This allowed us to expedite the testing of different classifiers.

Our next steps were applying PCA on 30 components for the layers followed up by ICA against the stacked PCA features while experimenting with the number of ICA components. The two models that appeared to perform especially well on the subset of training data were Random Forest and SVC and we decided to continue to refine our findings with each iteration.

6.1. Analytic Approach

Our first goal was to investigate the different machine learning models available to us and compare the results. Some models we explored include: Random Forest, Gradient Boost, K Nearest Neighbor, Stochastic Gradient Descent, SVC, Gaussian Process and Xgboost. We were utilizing AWS for this and our setup included a 24vCPU instance which we found was not adequate to run the full dataset with less than 5% left to finish processing. It was necessary to use a 32vCPU instance in order to process all the data.

We added multi-layer feature reduction to get everything the same size and serendipitously sped up our process by 20 times. Viability testing only utilizes 6% of the training dataset and performance has improved through additional training.

6.2. Model Performance

Performance varied between the different classification types. The two models that appeared to perform the best on the subset of the training data were Random Forest and SVC and underperforming models were eliminated from the pool of candidates.

6.2.1. Random Forest

	precision	recall	f1-score	support
clean	0.43	0.75	0.55	4
trojan	0.86	0.60	0.71	10
accuracy			0.64	14
macro avg	0.64	0.68	0.63	14
weighted avg	0.73	0.64	0.66	14

6.2.2. Gradient Boost

	precision	recall	f1-score	support
clean	0.50	1.00	0.67	4
trojan	1.00	0.60	0.75	10
accuracy			0.71	14
macro avg	0.75	0.80	0.71	14
weighted avg	0.86	0.71	0.73	14

6.2.3. K Nearest Neighbor

	precision	recall	f1-score	support
clean	0.33	1.00	0.50	4
trojan	1.00	0.20	0.33	10
accuracy			0.43	14
macro avg	0.67	0.60	0.42	14
weighted avg	0.81	0.43	0.38	14

6.2.4. Stochastic Gradient Descent

	precision	recall	f1-score	support
clean	0.50	0.75	0.60	4
trojan	0.88	0.70	0.78	10
accuracy			0.71	14
macro avg	0.69	0.72	0.69	14
weighted avg	0.77	0.71	0.73	14

6.2.5. SVC

	precision	recall	f1-score	support
clean	0.30	0.75	0.43	4
trojan	0.75	0.30	0.43	10
accuracy			0.43	14
macro avg	0.53	0.53	0.43	14
weighted avg	0.62	0.43	0.43	14

6.2.6. Gaussian Process

	precision	recall	f1-score	support
clean	0.44	1.00	0.62	4
trojan	1.00	0.50	0.67	10
accuracy			0.64	14
macro avg	0.72	0.75	0.64	14
weighted avg	0.84	0.64	0.65	14

6.2.7. Xgboost

	precision	recall	f1-score	support
clean	0.40	0.50	0.44	4
trojan	0.78	0.70	0.74	10
accuracy			0.64	14
macro avg	0.59	0.60	0.59	14
weighted avg	0.67	0.64	0.65	14

6.3. Model Description

Different machine learning models available were investigated and results were compared. Some models we explored include: Random Forest, Gradient Boost, K Nearest Neighbor, Stochastic Gradient Descent, SVC, Gaussian Process and Xgboost. We utilized AWS for this and our setup included a 24vCPU instance which we found was not adequate to run the full dataset with less than 5% left to finish processing. It was later bumped up to a 32vCPU instance in order to run all the models.

7. Findings and Reportings

7.1. Model Accuracy

Our model performance has improved significantly since our initial run where it is currently at 84% accuracy at 2 sec/model compared to the top detectors on the TrojAi leaderboard where they are averaging at 400 sec/model. Speed is a key capability in our project which currently takes roughly 3 hours compared to 36 hours on the TrojAi leaderboard. We observed marginal performance (5-10%) improvement after implementing the ensemble.

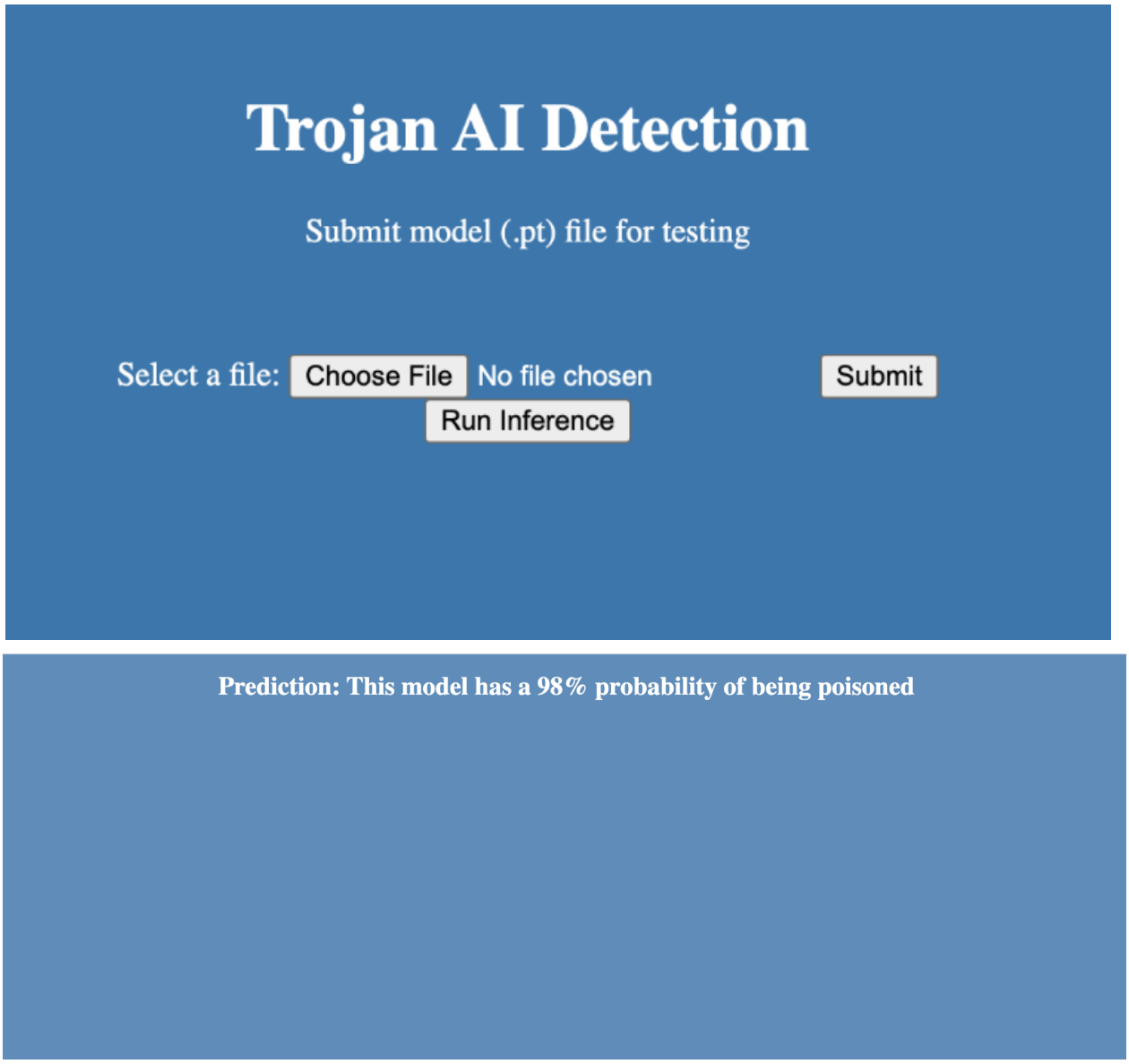
Our approach enables the detector to run in 2 seconds, indicating that this solution can be utilized at scale with minimal cost. We've added multi-layer feature

reduction. This includes PCA at each layer to get architecture dimensions on the dataset to be all the same size followed by ICA on the reduced dataset. Due to the many possible combinations at each step, a parameterized reduction cascade is introduced to enable component exploration in order to search for the optimal number of components. Our optimized reduction generation allows for a 20x speedup.

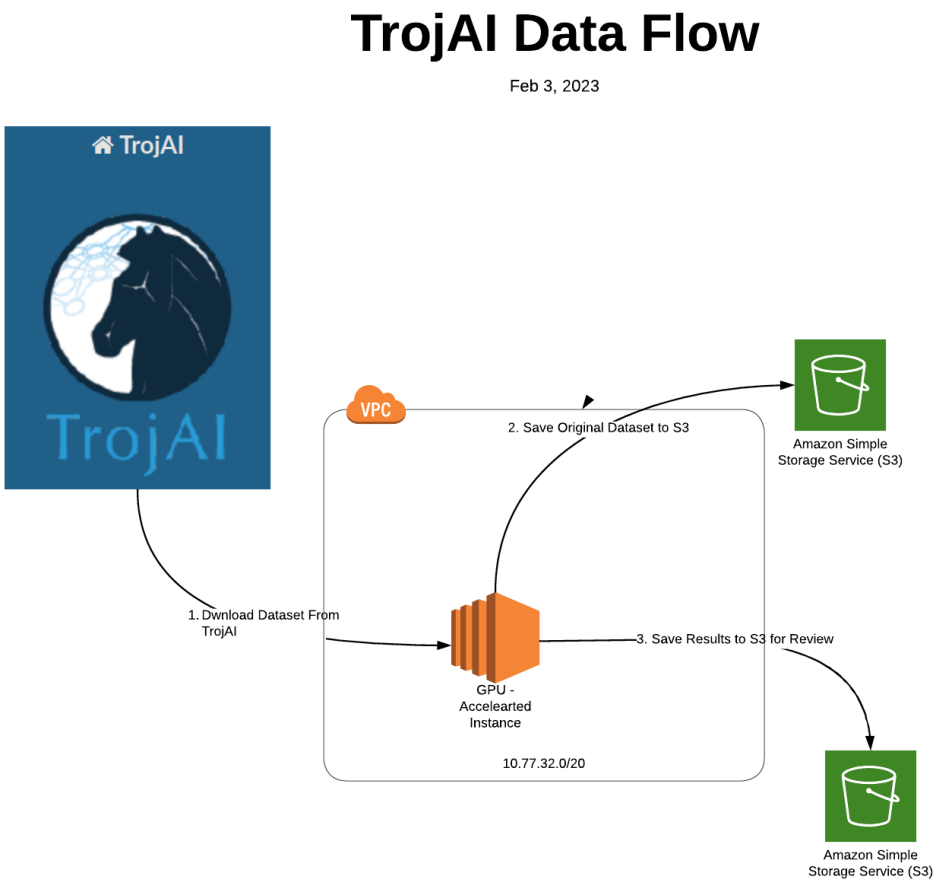
In our approach we implemented an automated model selection where we train the model for each dataset and throw it out if it does not meet the threshold. This allows us to be able to select the best performing model. Current implementation takes the probabilities of each model as features for an ensemble. Our optimization process involves the use of a sequential feature selector that minimizes the number of features that will be kept for the final detector to reduce the size and increase speed at inference. We store only the config/models to speed up the inference and to reduce the overall size.

7.2. Integrated Product

For our final product we have a basic user interface where a user is able to submit a model and in return receives a probability for whether their model is potentially poisoned or not.


Trojan AI Detection
Submit model (.pt) file for testing
Select a file:
Choose File
No file chosen
Submit
Run Inference
Prediction: This model has a 98% probability of being poisoned

8. Solution Architecture, Performance and Evaluation



8.1. Performance Metrics

Training the primary model currently requires having a system that can store all the available models in memory. So it performs poorly or even crashes on systems with less memory than is needed.

Accuracy performance is not very good for the primary classification method, but it is able to evaluate models quickly. Slower methods such as the Artificial Brain Stimulation (ABS) are able to consistently identify AI models that may act as trojans, but the time required to perform the evaluation makes it less useful in cases where compute resources are limited or where having highly responsive evaluation metrics are needed.

8.2. Cost Metrics

The dimensionality reduction method takes 1-2 seconds to evaluate a model. The ABS code takes 50-60 minutes to evaluate a single model on our development infrastructure with about 60-70% of the computational resources as the specially approved g5.2xlarge EC2 resources in AWS. We expect to see a decrease in time to run there. However resources using spot instances cost less than $0.50/hr so the cost to evaluate a single model is relatively low. This meant that the cost would increase dramatically if the time to evaluate a model must be decreased significantly.

In the TrojAI competition scenario, there are thousands of models to be evaluated. So any method that could improve the speed of the evaluations or that could help filter out the more easily classified AI models would potentially save hundreds of dollars per run.

8.3. Compute

We were initially leveraging a single node system to run code from the competition teams. Development system resources for running the ABS model include 32GB of RAM, A4500 GPU w/16GB RAM, and 1 CPU with 32 cores running in Arch Linux. AWS systems are single nodes running in AWS with 32GB RAM, A10 GPU w/24 GB RAM, and 8vCPU running in Amazon Linux 2. It was necessary to increase our maximum vCPU usage from 24 to 64 as we continued to fine tune our models.

We were allocated $1000 in AWS credit and have spent approximately $750 by the end of this capstone project. The majority of our costs went towards Amazon EC2 and S3.

8.4. Storage

AWS S3 Bucket: ~150GB for TrojAI round 1, and the data for rounds 2-4 have larger datasets

8.5. Scalability and Robustness

Our agnostic architecture approach makes our solution robust. This approach may also be effective for models trained for other tasks besides image classification. It includes a reduction fit search optimized for a 20x speedup. Implementation of a Sequential Feature Selector to minimize the reduction fits needed and model

hyperparameter tuning is embedded. Finally storing the fits/models for inference rather than reproduce during inference.

9. Technology Stack

The original code we were working off of was 2-3 years old since most of the TrojAI teams stopped updating it after the competition. The teams leveraged Python version 3.6 or 3.7 code and either PyTorch or Tensorflow machine learning platforms. The ABS code was using libraries that are no longer supported making it difficult to run on more modern AWS architecture which required updated drivers. We updated the code to leverage current versions of Tensorflow and CUDA as shown below.

Original NIST Provided Code Technology Stack (Primary Classification Method)

- Python 3.7
- Numpy
- Scikit-image
- Torch 1.6.0
- Torchvision 0.6.0
- Advertorch

Original ABS - Code from the winning TrojAI team (Rutgers/Purdue) Technology Stack

- Python 3.6
- Numpy
- Tensorflow 1.12.1
- CUDA 9
- CUDNN 7

Our Updated ABS Code Technology Stack

- Python 3.10
- Numpy
- Tensorflow 2.10
- CUDA 12
- CUDNN 8

- AWS

10. Conclusion

Our model's performance has improved significantly compared to the earlier stages of our work and compared to the top results on the TrojAi leaderboards. What originally took roughly 36 hours to complete, we were able to reduce the run time to 3 hours for 150GB of modeled data. With this improvement a single model can be processed in a couple of seconds and integrated into a business pipeline.

11. References

Backdoor Attack Detection in Computer Vision by Applying Matrix Factorization on the Weights of Deep Networks (2022)
https://arxiv.org/pdf/2212.08121v1.pdf

ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation
https://people.cs.rutgers.edu/~sm2283/papers/CCS19.pdf

Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks
https://sites.cs.ucsb.edu/~bolunwang/assets/docs/backdoor-sp19.pdf

An Embarrassingly Simple Approach for Trojan Attack in Deep Neural Networks
https://dl.acm.org/doi/pdf/10.1145/3394486.3403064

Michael Paul Majurski (2020), Trojan Detection Software Challenge - image-classification-jun2020-train, National Institute of Standards and Technology, https://doi.org/10.18434/M32195
https://data.nist.gov/od/id/mds2-2195

BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain
https://arxiv.org/pdf/1708.06733.pdf

https://www.nist.gov/itl/ssd/trojai

https://pages.nist.gov/trojai/docs/about.html

https://github.com/naiyeleo/ABS

12. Appendices

12.1. DSE MAS Knowledge Applied

Every skill learned as part of the DSE program was utilized throughout the capstone project. Beginning with python for data analysis as our foundation to

12.2. Library Link and Citation

Armstrong, Christopher; Hartley, Daniel; Hutton, Spencer; Quach, Shirley (2023). Trojan AI Detection. In Data Science & Engineering Master of Advanced Study (DSE MAS) Capstone Projects. UC San Diego Library Digital Collections. https://doi.org/10.6075/J0B56JX8

12.3. Github Link

https://github.com/shirleyquach/dse260