

**Midterm Project**

**AI and Cyber Security**

**DSCI6015**

**Simplified Midterm**

**PAVAN REDDY BOYAPALLY**

**00842630**

**University of New Haven**

**Dr. Vahid Behzadan**

**April 01, 2024**

:

# Summary:

This study details the effective use of AWS Sagemaker in the construction of a cloud-based PE (Portable Executable) static malware detection API. The API uses a Random Forest binary classifier to determine if Portable Executable (PE) files are malicious or benign. The classifier was trained using labelled datasets of binary feature vectors.

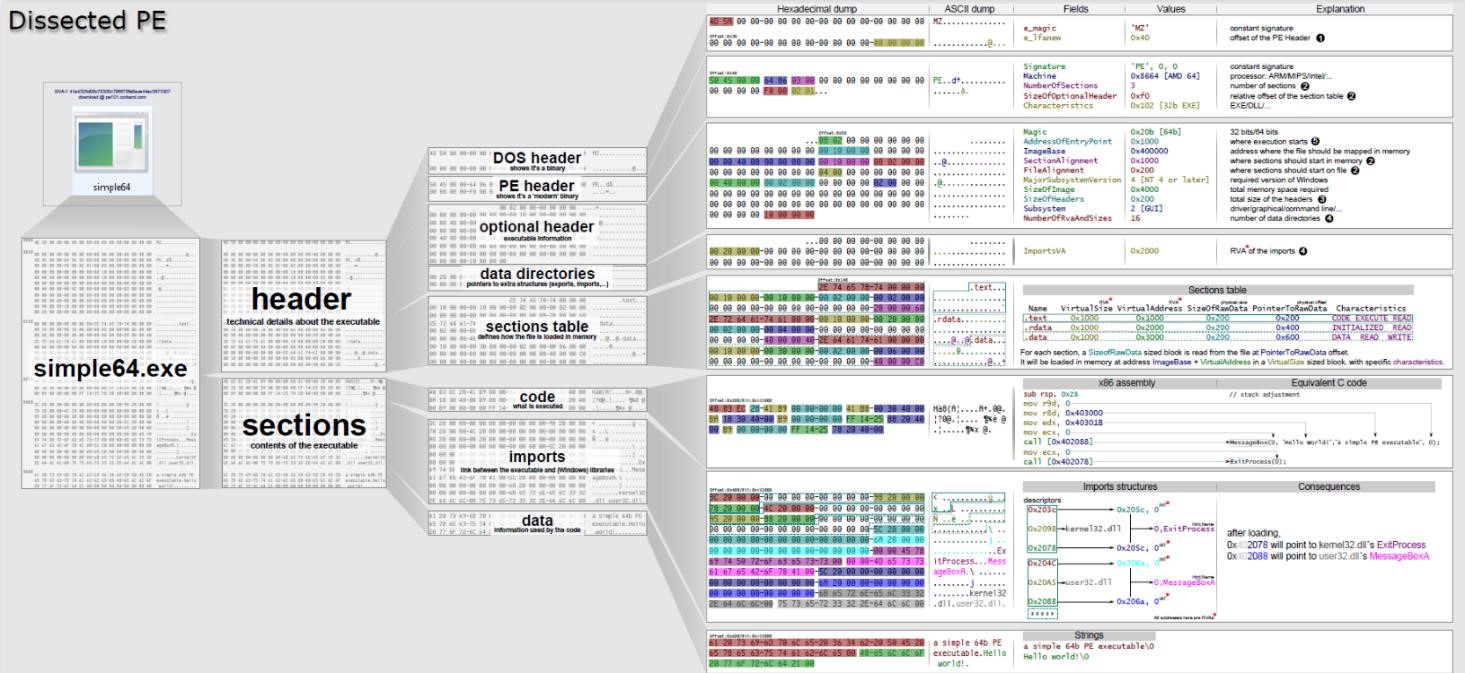
Using the same Amazon Sagemaker instance for model deployment, the project made use of AWS Sagemaker for both model construction and training. An intuitive web application is developed so that a distant user can submit their executable (.exe) file and discover the hazard. The project was written in Python, and the model generation and imp

lementation were done using ML libraries such as sklearn, pefile, nltk, etc.

# Introduction

## PE Files

Windows operating systems use PE files as a file type for storing executable code and related information. These files include all of the essential information required to run the programme, including resources, machine instructions, imported libraries, and metadata. PE files are mostly used for drivers, programmes, and dynamic link libraries (DLLs). They follow a standard format with headers that provide information about the file's architecture, entry point, and section organisation. Understanding the PE file format is essential for tasks like malware detection, reverse engineering, and software analysis since it allows you to look at and change executable files.



## Random Forest Classifier

Suitable for both regression and classification applications, the Random Forest classifier is a flexible and strong machine learning technique. It is a member of the ensemble learning family, which creates predictions by combining several different individual models. Decision trees are the individual models in the Random Forests scenario. In Random Forest, the term "forest" refers to a grouping of decision trees, each of which is constructed separately and makes judgements in response to a set of input features.

The main idea behind Random Forests is to provide randomness in both the training and prediction phases, resulting in a diversified set of decision trees. A random subset of the training data and a random subset of features are used to build each tree during training for every split. This unpredictability lowers the possibility of overfitting and enhances the model's overall performance by decorrelationing the individual trees. During prediction, each tree's output is aggregated using a technique known as voting or averaging, and the final prediction is determined by taking the average (for regression tasks) or the most frequent class (for classification tasks).

Random Forests' primary goal is to provide unpredictability into both the training and prediction stages, producing a diverse collection of decision trees. Each tree is constructed during training for each split using a random subset of features and a random subset of training data. By decorrelationing the individual trees, this unpredictability reduces the likelihood of overfitting and improves the overall performance of the model. Voting or averaging is a technique used to aggregate each tree's output during prediction. The final prediction is then established by picking the most frequent class (for classification tasks) or the average (for regression tasks).

**Task Approach:**

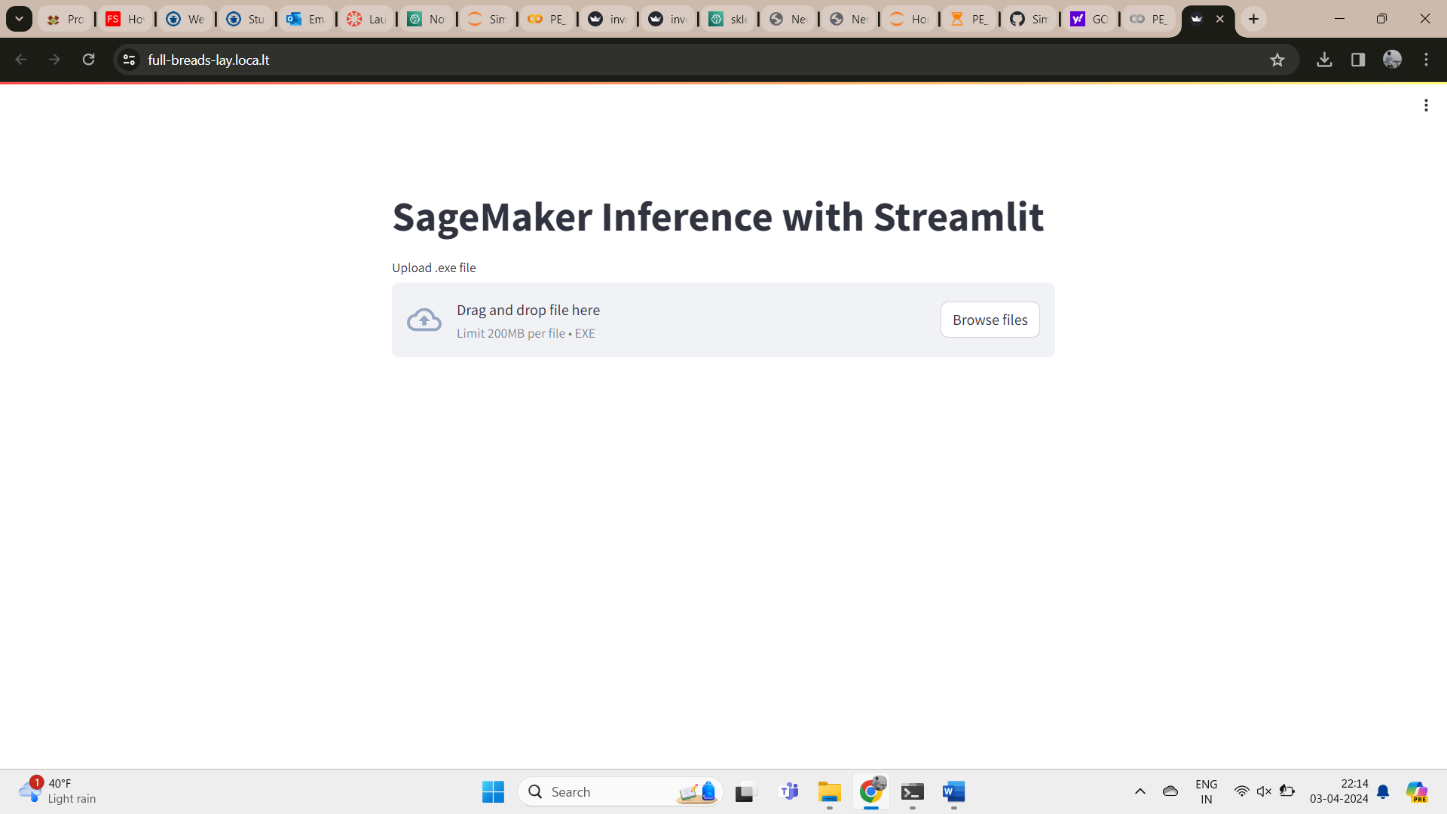
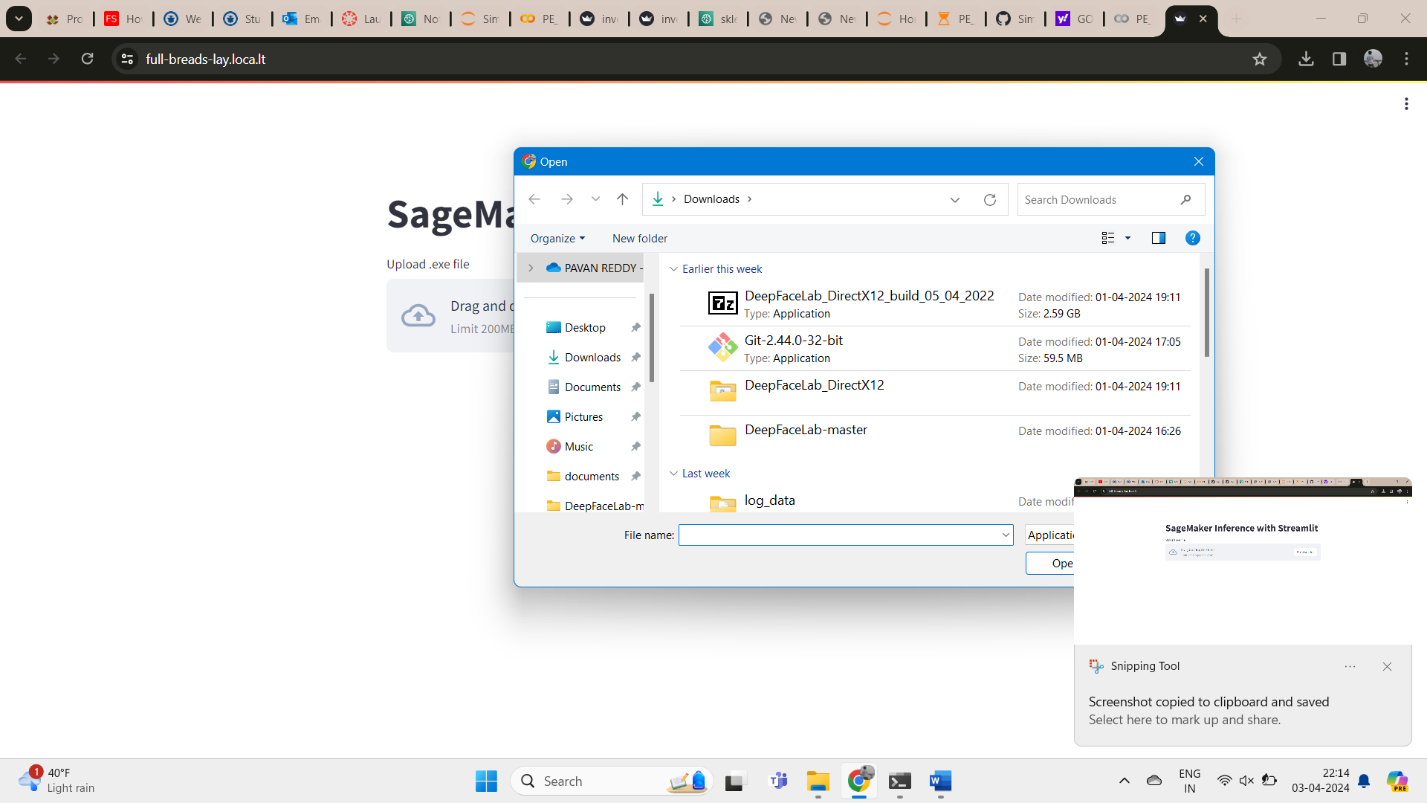
1. **Building and Training the Model:** n AWS Sagemaker, training and development use the scikit-learn 1.2.1 version. A correctly labelled dataset of binary feature vectors was used to train the Random Forest binary classifier. The trained model is subsequently dropped into a joblib file.

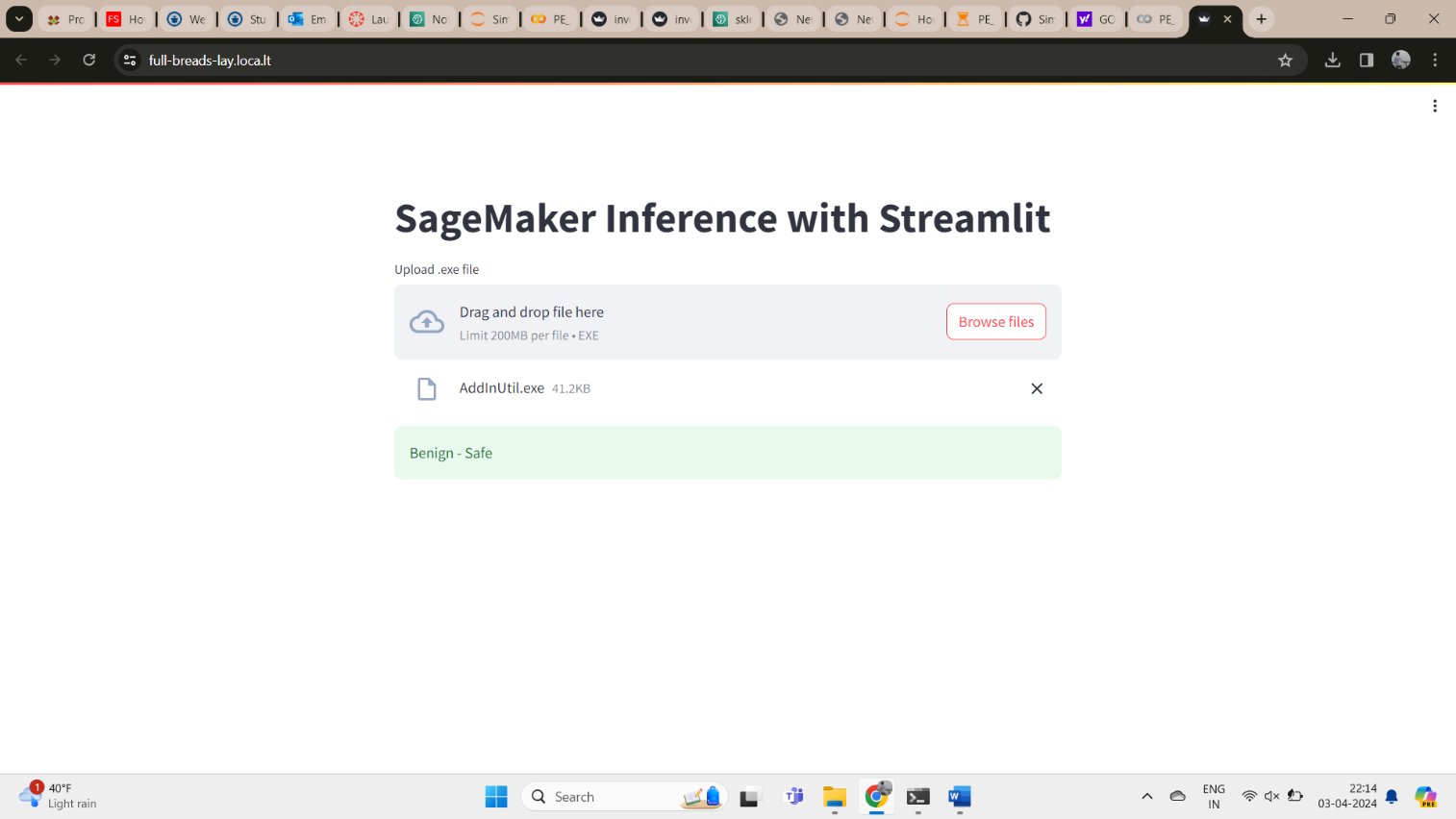
1. **Deploying the Model as a Cloud API:**

The trained model was deployed using Amazon Sagemaker, which produced an endpoint for a cloud-based API for real-time prediction. Using the stored joblib file, the model is loaded, and Sagemaker is used to put it up for deployment.Module SKLearn. The model is deployed into an Endpoint that has been setup and established.

1. **Creating a Client Application:**

A user-friendly interface was created for a streamlit web application. Users can upload executable (.exe) files to the webclient. The webclient uses pefile lib and other pretrained data to extract the necessary features from the.exe file. The features are then transformed to a JSON format and sent to the deployed API. The classification results (Malware - Danger or Benign - Safe) are subsequently shown by the programme. I launched the streamlit application in Google Colab and utilised it for the client deployment.





1. **Ember 2018 dataset performance check:** At least 200 malware and benign samples are obtained from the Ember 2018 dataset. The relevant features are then extracted from them, given to the API, and the replies are logged to verify the model's performance with the Malconv model detection.

# Project Results

The project successfully achieved its intended outcomes:

1. **Trained Malware detection model:** It was possible to create a malware detection model that was trained to distinguish between dangerous and benign PE files.
2. **Deployed Cloud API:** The trained model is put to use on Amazon Sagemaker, where it serves as an online real-time prediction API.
3. **Web Client:** a web-based interface designed to allow users to upload and verify whether their files are dangerous.

**Accuracy: 0.9915254237288136**

# Result and Conclusion:

The data above demonstrate how accurate my models are. With an accuracy of 0.9915254237288136, the model is roughly 99.15% accurate when it comes to predicting the kind of exe file most of the time.

The project's goal of creating and implementing a cloud-based PE static malware detection API was accomplished. The research shows how machine learning can effectively classify malware and how cloud platforms like Google Colab and Amazon Sagemaker may be used to create scalable and user-friendly applications.

**Resources:**

* <https://github.com/endgameinc/ember>
* <https://github.com/endgameinc/ember/tree/master/malconv>
* <https://github.com/UNHSAILLab/S24-AISec/tree/main/Midterm%20Tutorial>
* <https://sagemaker-examples.readthedocs.io/en/latest/intro.html>
* https://sagemaker[examples.readthedocs.io/en/latest/frameworks/pytorch/get\_started\_mnist\_train\_outputs.html](https://sagemaker-examples.readthedocs.io/en/latest/frameworks/pytorch/get_started_mnist_train_outputs.html)
* <https://docs.aws.amazon.com/sagemaker/latest/dg/deploy-model.html>
* <https://github.com/RamVegiraju/Pre-Trained-Sklearn-SageMaker>
* <https://youtu.be/ueI9Vn747x4?si=KSFTvR9hBnU0u0DO>
* <https://youtu.be/oOqqwYI60FI?si=3WKd-iDz93mm1Vbe>