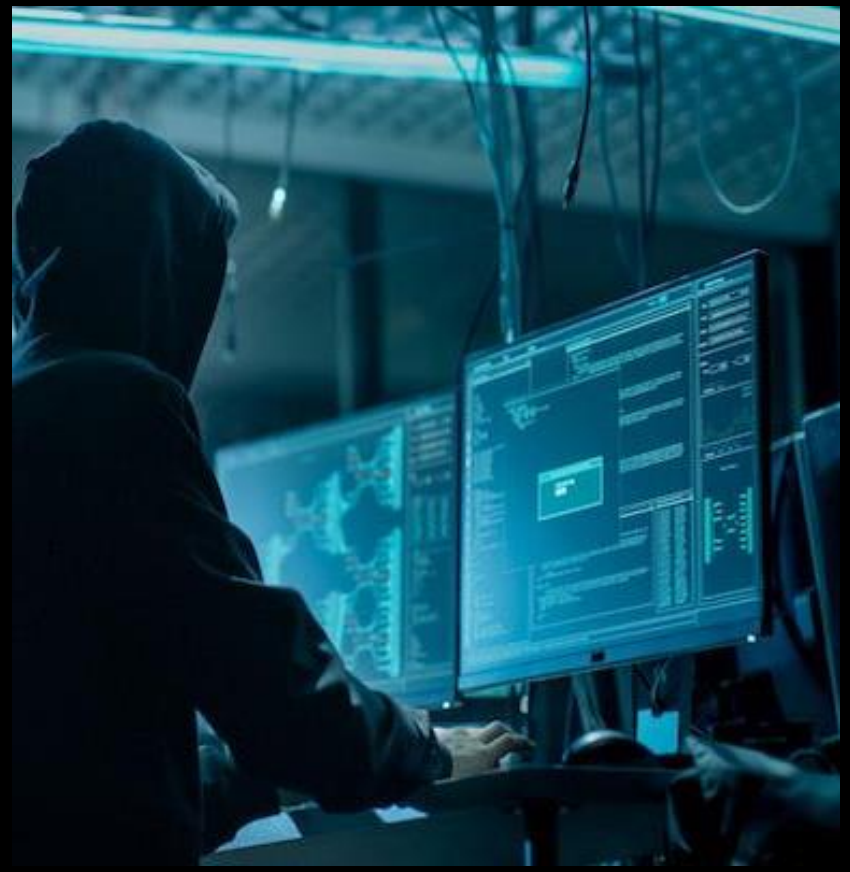
M3-BA

A MULTI-LABEL MULTI-OUTPUT MALWARE BEHAVIOR ANALYZER

- RICCARDO DEIDDA 70/90/00639
- ALESSIO MURGIONI 70/90/00641

INTRODUCTION

Our project consists of a **multi-label, multi-output** malware behavior classifier that can determine whether a single sample exhibits characteristics of ransomware, password stealers, viruses, and more; **without** manual static or dynamic analysis, which is both time-consuming and risky.



The project code is available on GitHub at:

RDbtx/Mlproject

THE DATASET

The project uses the EMBER2024 dataset, also available on GitHub, as both dependency and training/testing set.

Because the full dataset is nearly **48 GB** before vectorization, we chose to use only the **Win64** subset, which contains about **1 million samples.** The data is also split into training and test sets by collection year, with the **training set one year older** than the test set.

Feature extraction, labeling, and vectorization were all handled by **thrember**, the internal EMBER2024 library.

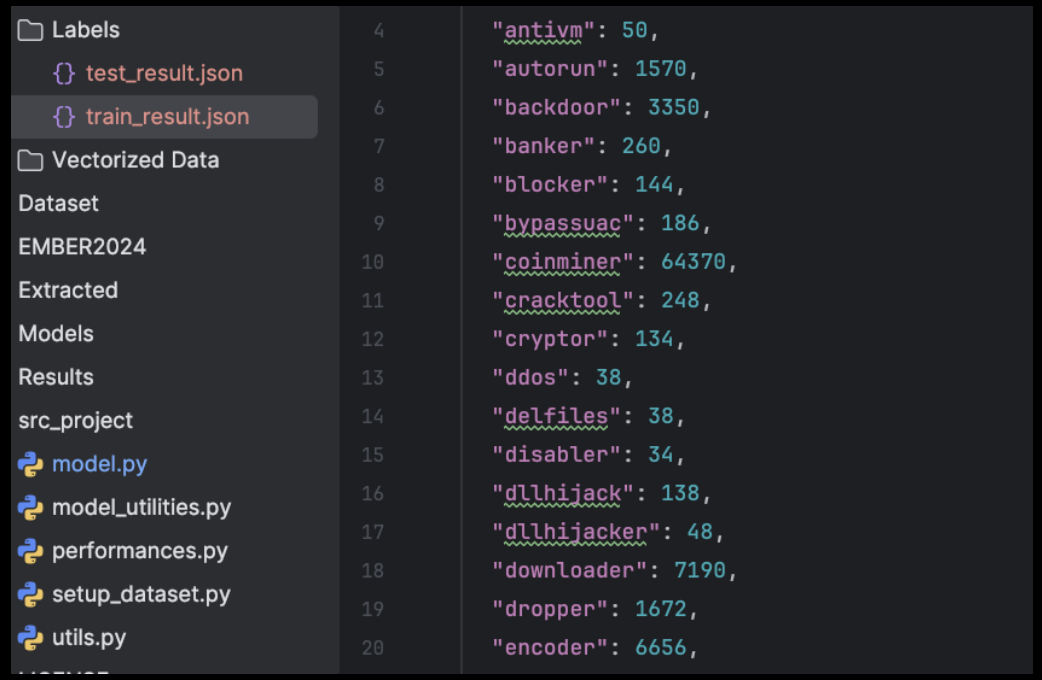
Subset	Total Size
Win32 train	23.7 GB
Win32 test	4.9 GB
Win64 train	12.9 GB
Win64 test	2.5 GB
.NET train	1.8 GB
.NET test	425 MB
APK train	1.0 GB
APK test	234 MB
PDF train	197 MB
PDF test	46 MB
ELF train	100 MB
ELF test	24 MB
challenge	126 MB

THE MAJOR PROBLEMS

EMBER was originally designed for malware recognition as a **binary malicious/benign** dataset. The **behavior** section we rely on is still experimental, which led to several issues:

- **Severe class imbalance** across behavior labels.
- **Numerous samples without behavior labels**, reducing effective training data.
- **Label mismatch** between training and test set.
- **Missing post-vectorization label vectors.**

Despite these hurdles, we implemented some labeling workarounds that allowed us to proceed and mitigate the missing label vector issue.



IMPLEMENTATION

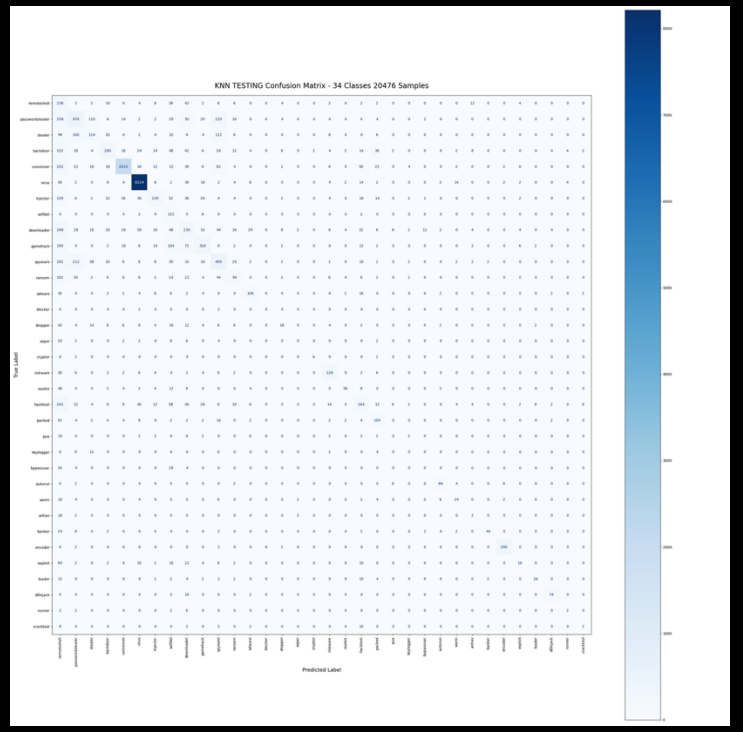
Our project is implemented with the **scikit-learn** library, which let us evaluate multiple **multi-label, multi-output**methods. The pipeline is organized as follows:

- begin with dataset analysis and labeling correction
- Shape and consistency fixing of both training and test set
- Model training and performances computation
- Testing phase and report of the final results

```
if __name__ == "__main__":
    DATASET_DIR = "../Dataset"
    EXTRACTED_DATA_DIR = "../Extracted"
    RESULTS_DIR = "../Results"

    x_train, y_train, x_test, y_test, x_challenge, y_challenge = feature_loading(EXTRACTED_DATA_DIR,
                                                                                 desired_datasets: ["x_train.npy", "y_train.npy",
                                                                                 "x_test.npy", "y_test.npy"])

    print("\n----DATASET ANALYSIS----")
    print("x_train shape: ", x_train.shape)
    print("y_train shape: ", y_train.shape)
    print("x_test shape: ", x_test.shape)
    print("y_test shape: ", y_test.shape)

    # modify the third variable of the set_generation function if you want to use a smaller dataset
    train_labels, x_train, y_train = set_generation(x_train, y_train, len(x_train), scenario: "TRAINING")
    test_labels, x_test, y_test = set_generation(x_test, y_test, len(x_test), scenario: "TESTING")

    # uncomment this section if you want your training and test set to be a split of the original training set
    # x_train, x_test, y_train, y_test, test_labels = training_set_split( x_train, y_train,train_labels, test_size=0.33)

    print("\n----DATASET ANALYSIS----")
    print("x_train shape: ", x_train.shape)
    print("y_train shape: ", y_train.shape)
    print("x_test shape: ", x_test.shape)
    print("y_test shape: ", y_test.shape)

    # this section is used to reshape the test set since seems like that the test set has additional labels
    # that are not present in the training set.
    train_labels, test_labels, x_train, y_train, x_test, y_test = shape_fixer( min_samples_per_train_label: 100, train_labels, test_labels, x_train,
                                                                               y_train, x_test, y_test)

    subset_analysis(x_train, y_train, scenario: "TRAINING", train_labels)
    subset_analysis(x_test, y_test, scenario: "TESTING", test_labels)

    # model training
    trained_model, train_predictions = model_train(lghtbm_multi, train_labels, x_train, y_train)
    # save_model(trained_model, "GRAD BOOST")

    test_predictions = model_test(trained_model, test_labels, x_test, y_test)
```

FIRST ATTEMPT - KNN

Given our hardware constraints, we began with a **K-Nearest Neighbors (KNN)** baseline, which is natively multi-output in scikit-learn. However, the **severe label imbalance** degraded test performance, as visible in the images.



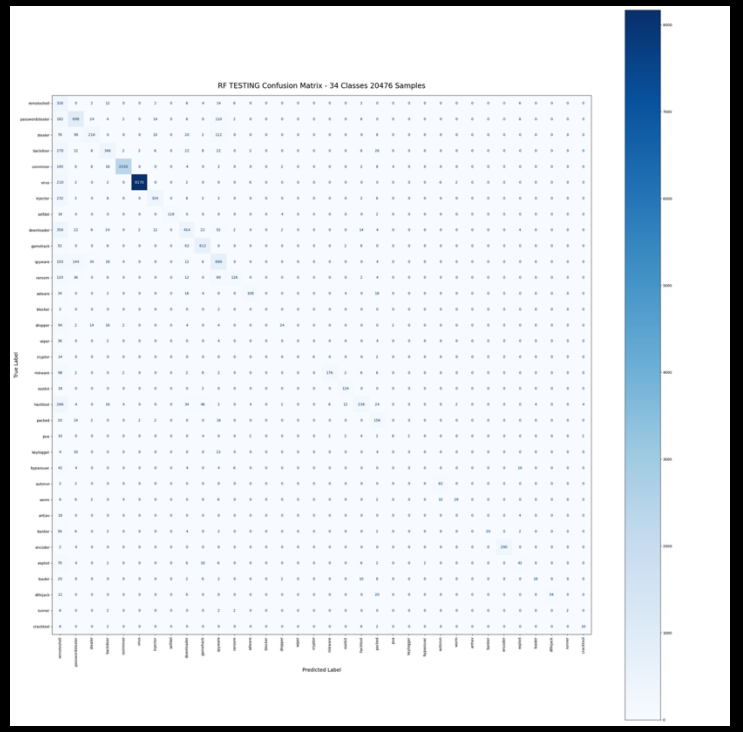

```
----KNN TESTING PERFORMANCES----
Accuracy: 0.6828
Precision: 0.4581
Recall: 0.3803
F1 micro Score: 0.6828
F1 macro Score: 0.3887
Hamming Loss: 0.3172
```

These results indicate KNN ends to favor majority behaviors and struggles to recover minority labels. Consequently, we moved to classifiers that are **more robust to imbalance**

SECOND ATTEMPT – RF

As a second option, we moved to a **Random Forest** classifier, which prior work, such as Felan Carlo C. Garcia et al. has shown to perform well in malware classification. Since Random Forest isn't inherently multi-output, we **wrapped it** with a **MultiOutputClassfier model**, effectively training **one independent model per behavior label**.

This approach delivered **better performance than KNN**, confirming Random Forest's greater robustness to class imbalance. However, the gains were still short of our target.

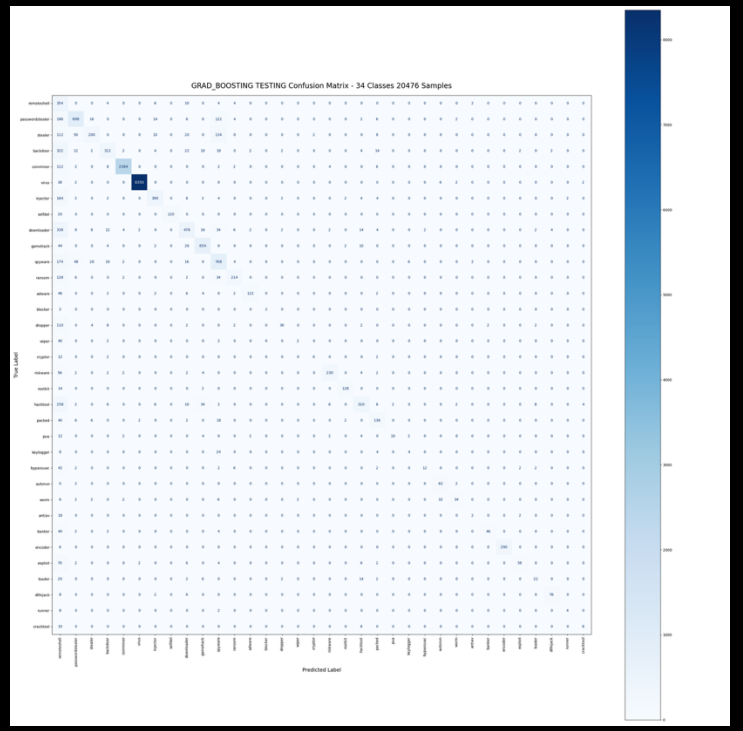



```
----RF TESTING PERFORMANCES----
Accuracy: 0.7791
Precision: 0.6745
Recall: 0.4697
F1 micro Score: 0.7791
F1 macro Score: 0.5105
Hamming Loss: 0.2209
```

FINAL ATTEMPT – GB

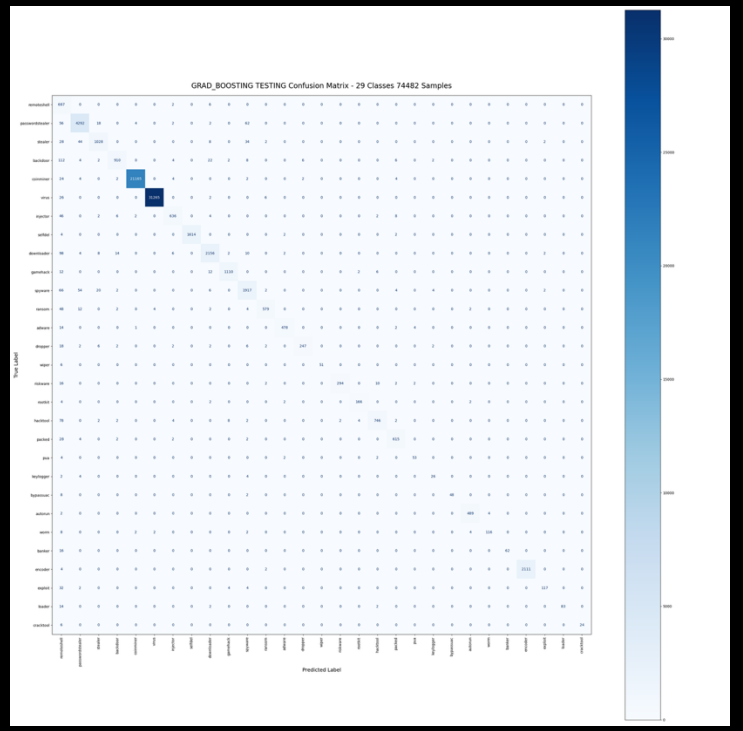
Our final step to improve the baseline was a **Gradient Boosting** approach using **LightGBM**. As with Random Forest, we wrapped the base estimator in scikit-learn's **OneVsRestClassifier** to obtain the required **multilabel** behavior.

This configuration delivered the **best performance** among all our models, confirming gradient boost's resilience to imbalance. Given the **time gap** between training and test sets and the dataset's **heavy imbalance**, we consider these results satisfactory.




```
----GRAD BOOSTING TESTING PERFORMANCES----
Accuracy: 0.8173
Precision: 0.7836
Recall: 0.5498
F1 micro Score: 0.8173
F1 macro Score: 0.6059
Hamming Loss: 0.1827
```

SPLIT TRAINING ATTEMPT - GB



To verify that the model itself was sound, we also evaluated it on **same-year** data by splitting the original test set into **2/3 for training** and **1/3 for testing**.

As we can see, the resulting **confusion matrix shows an almost perfectly diagonal pattern** and the metrics were very high, indicating strong behavior discrimination when train and test share the same temporal distribution. This confirms the model's capability and underscores the impact of **time differences in malware classification.**

```
----GRAD BOOSTING TESTING PERFORMANCES----
Accuracy: 0.9812
Precision: 0.9507
Recall: 0.9001
F1 micro Score: 0.9812
F1 macro Score: 0.9195
Hamming Loss: 0.0188
```

RESULTS

- **Gradient Boosting** achieved the best overall scores, though there's still room for improvement.
- **KNN** heavily underperforms on unbalanced datasets and under **temporal drift** between training and testing years.
- **Random Forest** trailed Gradient Boosting, but remains viable: results are likely to improve with **hyperparameter tuning** and **better classification strategy**, such as **per-label thresholding** to boost underrepresented labels.

FINAL REMARKS

Since the project was mainly hindered by the dataset structure, originally not designed for our current implementation, the **best improvements** will likely come from **enhancing the dataset itself** by:

- Fixing **labeling inconsistencies**
- Reducing **sample imbalance**
- Providing a reliable **label vector**

Given that **we were conservative with processing power**, further advancements on our side could also come from:

- Per-label **threshold calibration**
- Better **hyperparameter tuning**

THANK YOU FOR THE ATTENTION !

- RICCARDO DEIDDA 70/90/00639
- ALESSIO MURGIONI 70/90/00641