**Trường Đại học Công nghệ Thông tin – ĐHQG Tp. HCM**
**Khoa Mạng máy tính và truyền thông**

# BÁO CÁO TỔNG KẾT ĐỒ ÁN MÔN HỌC

**Môn học:** **Cơ chế hoạt động của mã độc**

**Tên chủ đề:** **Applying NLP techniques to malware detection in a practical environment**

*Mã nhóm: G17* *Mã đề tài: S21*

**Lớp**: **NT230.O21.ANTT**

## 1. THÔNG TIN THÀNH VIÊN NHÓM:

| STT | Họ và tên | MSSV | Email |
|---|---|---|---|
| 1 | Nguyễn Phương Trinh | 21521581 | 21521581@gm.uit.edu.vn |
| 2 | Nguyễn Thị Minh Châu | 21520645 | 21520645@gm.uit.edu.vn |
| 3 | Trần Minh Duy | 21522010 | 21522010@gm.uit.edu.vn |

## 2. TÓM TẮT NỘI DUNG THỰC HIỆN:

A. Chủ đề nghiên cứu trong lĩnh vực Mã độc:

☑ Phát hiện mã độc

☐ Đột biến mã độc

☐ Khác: ……

B. Liên kết lưu trữ mã nguồn của nhóm:

Mã nguồn của đề tài đồ án được lưu tại: [NT230.N21.ANTT]-Project_G17_S21

C. Tên bài báo tham khảo chính:

M. Mimura and R. Ito, "Applying NLP techniques to malware detection in a practical environment," International Journal of Information Security, vol. 21, no. 2, pp. 279-291, Apr. 2022, doi: 10.1007/s10207-021-00553-8.

D. Dịch tên Tiếng Việt cho bài báo:

Áp dụng kỹ thuật xử lý ngôn ngữ tự nhiên để phát hiện phần mềm độc hại trong môi trường thực tế

E. Tóm tắt nội dung chính:

Bài báo này trình bày về việc áp dụng các kỹ thuật xử lý ngôn ngữ tự nhiên (NLP) vào việc phát hiện malware trong môi trường thực tế. Các điểm chính như sau:

1. Các tệp thực thi vẫn là phương tiện phổ biến được dùng trong tấn công điểm cuối. Những tệp thực thi này thường bị mã hóa để tránh các chương trình diệt virus. Phân tích động của tất cả các tệp đáng ngờ từ internet rất tốn thời gian, vì vậy cần có một phương pháp lọc nhanh.

2. Với sự phát triển gần đây của các kỹ thuật NLP, các chuỗi ký tự in được trở nên hiệu quả hơn trong việc phát hiện malware. Sự kết hợp của các chuỗi ký tự in và các kỹ thuật NLP có thể được sử dụng làm phương pháp lọc nhanh.

3. Bài báo này áp dụng các kỹ thuật NLP vào việc phát hiện malware. Kết quả nghiên cứu cho thấy rằng các chuỗi ký tự in cùng với các kỹ thuật NLP là hiệu quả trong việc phát hiện malware trong môi trường thực tế.

4. Tập dữ liệu của nghiên cứu bao gồm hơn 500.000 mẫu thu thập từ nhiều nguồn. Kết quả thực nghiệm cho thấy phương pháp của họ hiệu quả không chỉ với các phân loại mới của malware hiện có, mà còn với malware mới.

5. Phương pháp này cũng hiệu quả đối với malware đã được đóng gói và các kỹ thuật chống gỡ lỗi.

F. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong bài báo:

Trong bài báo này, tác giả sử dụng các kỹ thuật chính sau đây để phát hiện phần mềm độc hại:

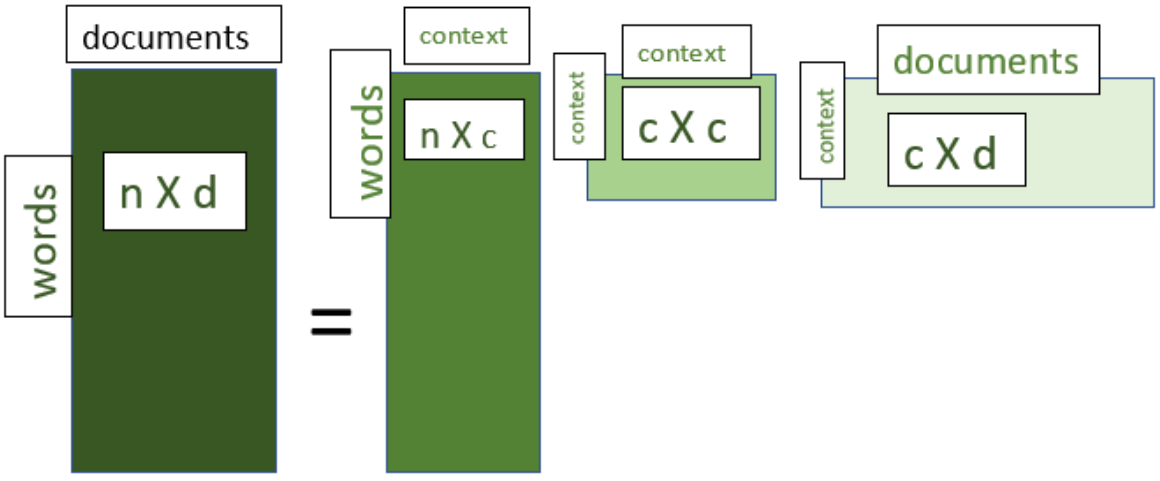
1. Bag-of-Words (BoW)

- Kỹ thuật BoW được sử dụng để chuyển đổi các chuỗi ký tự có thể in được từ các mẫu phần mềm độc hại thành các vector đặc trưng. Mô hình này tạo ra một biểu diễn vector không có thứ tự từ tần suất của các từ xuất hiện trong tệp thực thi, giúp trích xuất các đặc trưng cơ bản cho quá trình phân loại. Nhờ vậy mỗi tệp thực thi có thể được chuyển đổi thành một vector để thực hiện tính toán.


Hình 1. Minh họa về Bag-of-Words (Bow)

2. Latent Semantic Indexing (LSI)

- Ma trận tạo ra bởi BoW được nhân với trọng số TF-IDF, sau đó áp dụng LSI để giảm số chiều của các vector đặc trưng bằng kĩ thuật SVD (Singular Value Decomposition). Việc nhân trọng số và SVD giúp nhấn mạnh những từ quan trọng và bỏ đi những phần không cần thiết, từ đó có được cấu trúc ngữ nghĩa ẩn trong dữ liệu.



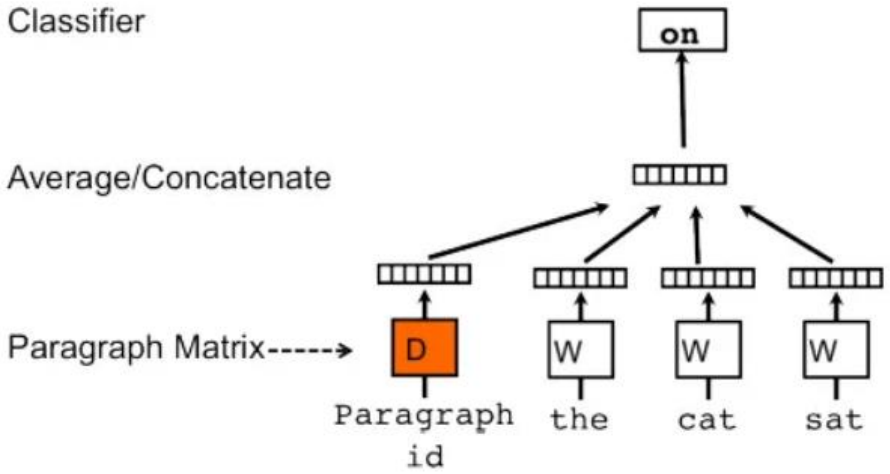

Hình 2. Minh họa về kiến trúc LSI

3. Paragraph Vector (Doc2Vec)

- Doc2Vec là mở rộng của mô hình Word2Vec, một mạng neural nông. Word2Vec chuyển đổi mỗi từ thành một vectơ trong không gian ngữ nghĩa. Các vectơ từ được định

vị trong không gian vectơ sao cho các từ có chung ngữ cảnh trong kho ngữ liệu sẽ nằm gần nhau trong không gian. *queen = king – man + woman* là một ví dụ về thao tác sử dụng mỗi vectơ từ được tạo bởi Word2vec. Doc2Vec là Word2Vec áp dụng thêm Paragraph Vector để biểu diễn tài liệu.

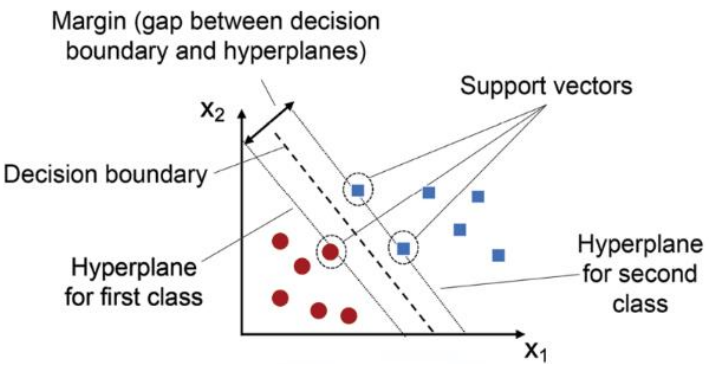
- Không như BoW và LSI chỉ tập trung vào tần suất từ mà không quan tâm tới vị trí, mối quan hệ giữa các từ, Doc2Vec tạo ra vector biểu diễn ngữ cảnh toàn diện cho cả tệp thực thi. Phương pháp này giúp mô hình hiểu được ngữ cảnh tổng thể của tệp thực thi, cung cấp thông tin ngữ nghĩa phong phú hơn cho quá trình phân loại.



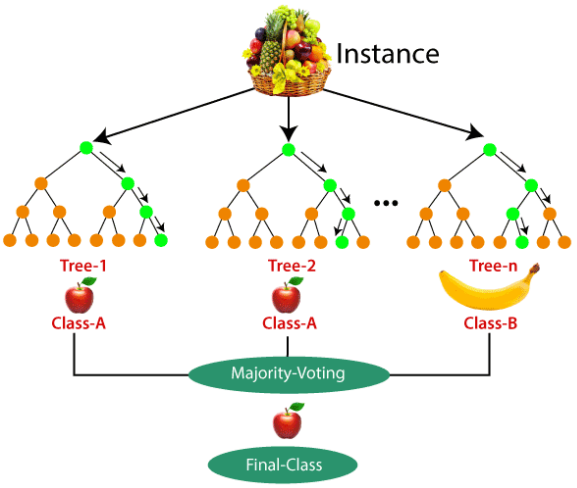

*Hình 3. Minh họa về kiến trúc Doc2Vec*

4. Các mô hình phân loại (SVM, RF, XGB, MLP, CNN)

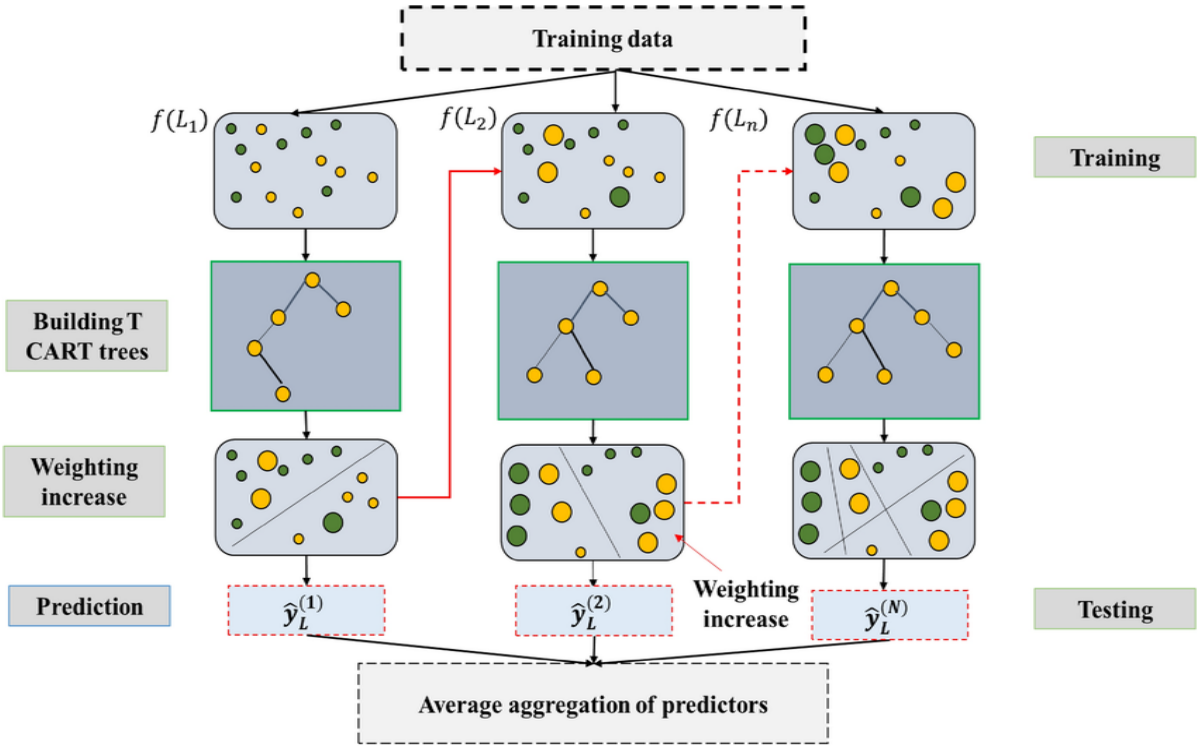
- Sau khi chuyển đổi các chuỗi ký tự thành vector đặc trưng bằng các kỹ thuật trên, các mô hình phân loại khác nhau như Support Vector Machine (SVM) (Hình 4), Random Forest (RF) (Hình 5), XGBoost (XGB) (Hình 6), Multi-Layer Perceptron (MLP) (Hình 7), và Convolutional Neural Network (CNN) (Hình 8) được sử dụng để dự đoán nhãn của các mẫu phần mềm. Mỗi mô hình có cách tiếp cận riêng để xử lý và phân loại dữ liệu, đảm bảo độ chính xác cao trong việc phát hiện phần mềm độc hại.



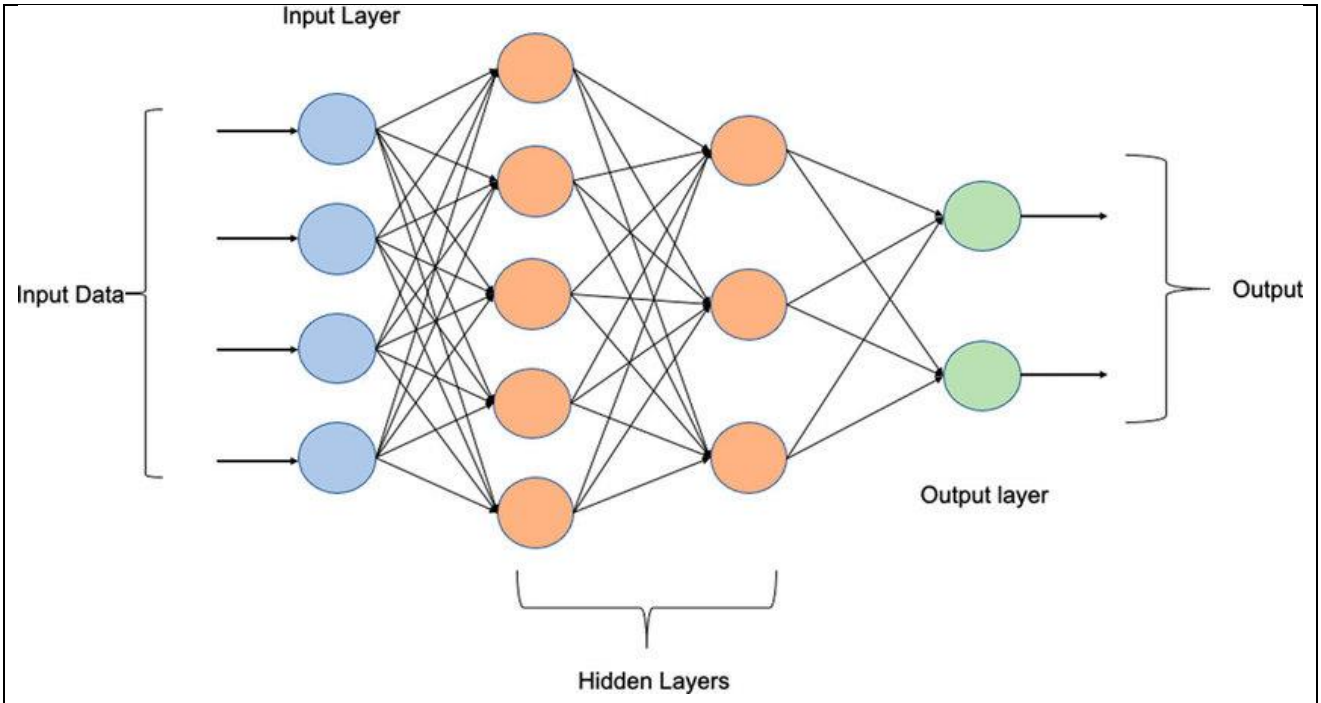

Hình 4. Minh họa về mô hình phân loại SVM



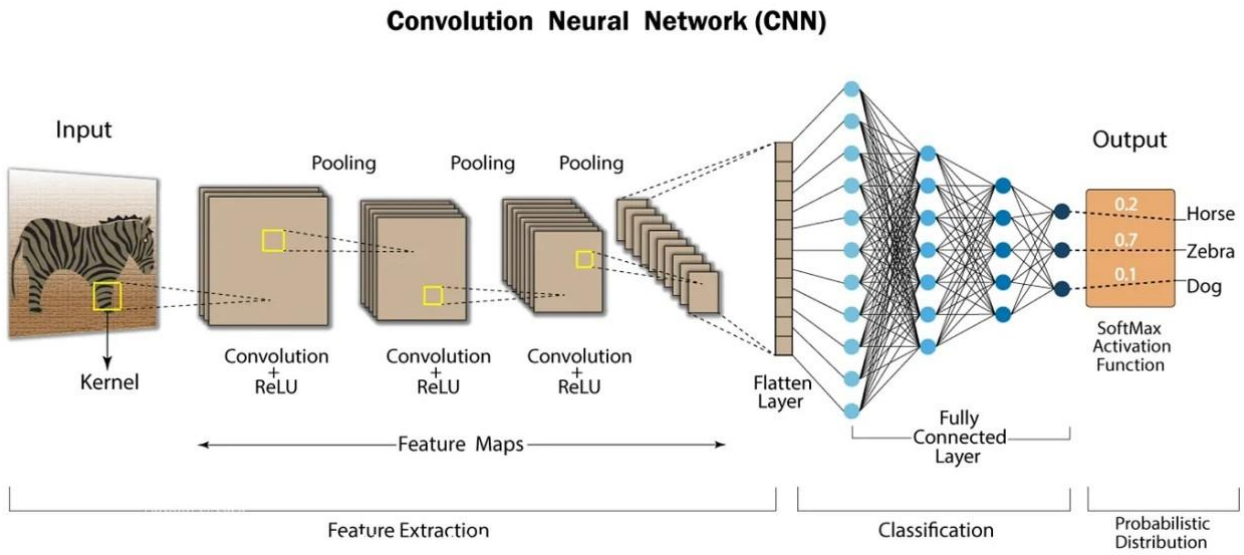

Hình 5. Minh họa về mô hình phân loại Random Forest (RF)




Hình 6. Minh họa về mô hình phân loại XGBoost (XGB)

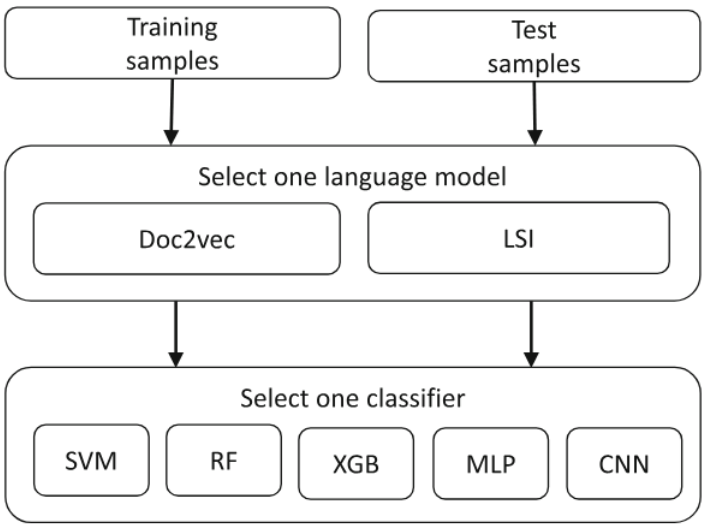



*Hình 7. Minh họa về mô hình phân loại Multi Layer Perceptron (MLP)*




Hình 8. Minh họa về mô hình phân loại Convolution Neural Network (CNN)

Các kỹ thuật này phối hợp với nhau để tạo thành một phương pháp phát hiện phần mềm độc hại hiệu quả, từ việc trích xuất đặc trưng cho đến quá trình phân loại và dự đoán.




Hình 9. Sơ đồ kiến trúc và các thành phần chính của hệ thống trong bài báo

G. Môi trường thực nghiệm của bài báo:

- **Cấu hình máy tính**: Hệ điều hành Windows 10, CPU Intel Core i7-5820K 3.3GHz, RAM 32GB DDR4 và ổ cứng Serial ATA 3 HDD

- **Các công cụ hỗ trợ sẵn có** trong giai đoạn hiện thực phương pháp:

+ Gensim: cung cấp các mô hình LSI và Doc2vec

+ Scikit-learn: cung cấp các bộ phân loại SVM và RF

+ XGBoost: cung cấp bộ phân loại XGB

+ Chainer: dùng để hiện thực MLP và CNN

+ PEiD: công cụ phát hiện hầu hết các kỹ thuật đóng gói và chống gỡ lỗi phổ biến trong tệp PE.

- **Ngôn ngữ lập trình** để hiện thực phương pháp: Python 2.7

- **Đối tượng nghiên cứu** (chương trình phần mềm dùng để kiểm tra tính khả thi của phương pháp/tập dữ liệu):

+ Tập dữ liệu: bao gồm hơn 500,000 mẫu từ nhiều nguồn khác nhau

+ FFRI dataset: một phần của tập dữ liệu MWS, chứa các log được thu thập từ hệ thống phân tích động malware Cuckoo sandbox và bộ phân tích tĩnh. Dữ liệu này được viết dưới dạng JSON và được phân loại từ năm 2013 đến 2019 dựa trên năm thu thập.

+ Hybrid Analysis (HA): một trang web phổ biến phân phối malware, cung cấp hàng chục ngàn mẫu.

- **Tiêu chí đánh giá** tính hiệu quả của phương pháp: Accuracy, Precision, Recall, F1 score, Receiver Operating Characteristics (ROC) curve, Area under the ROC Curve (AUC), thời gian training and testing.

H. Kết quả thực nghiệm của bài báo:

Kết quả thực nghiệm cho thấy phương pháp kết hợp chuỗi ký tự in được và kỹ thuật xử lý ngôn ngữ tự nhiên (NLP) đã chứng minh hiệu quả trong việc phát hiện phần mềm độc hại (malware) trong môi trường thực tế. Bộ dữ liệu gồm hơn 500,000 mẫu từ nhiều nguồn khác nhau đã được sử dụng để kiểm tra mô hình. Kết quả cho thấy phương pháp này có khả năng phát hiện không chỉ các biến thể của phần mềm độc hại hiện có mà còn phát hiện phần mềm độc hại mới. Phương pháp này cũng hiệu quả đối với các malware đã được đóng gói và các kỹ thuật chống gỡ lỗi.

**Ưu điểm**:

+ Phương pháp hiệu quả trong việc phát hiện cả malware mới và các biến thể của malware hiện có.

+ Hiệu quả trong việc đối phó với các kỹ thuật đóng gói và chống gỡ lỗi của malware.

**Nhược điểm**:

+ Tác giả tự nhận xét bộ dữ liệu có thể không đại diện chính xác cho toàn bộ số malware phổ biến do có thể còn thiếu một số mẫu thực tế, mặc dù vậy bộ dữ liệu đã bao gồm hơn 500,000 mẫu được thu thập từ nhiều nguồn.

+ Thiếu phân tích chi tiết và so sánh với các phương pháp khác.

+ Sử dụng packer detector PEid có tỷ lệ âm tính giả cao (khoảng 30%).

## I. Công việc/tính năng/kỹ thuật mà nhóm thực hiện lập trình và triển khai cho demo:

Kế hoạch của nhóm, các công việc mà nhóm dự định thực hiện:

1. Tìm hiểu, đọc bài báo và tài liệu liên quan để hiểu rõ nội dung, mục đích của bài báo; xác định cách triển khai mà bài báo đã làm, tài nguyên, môi trường thực hiện.

2. Tìm kiếm và thu thập các dataset mà tác giả đã sử dụng.

3. Tìm hiểu và triển khai các thuật toán của tác giả, trong đó có mô hình LSI, Doc2Vec và framework chainer.

4. Thực hiện tối ưu hóa mô hình.

5. Phân tích đánh giá mở rộng, vẽ biểu đồ, đồng thời so sánh kết quả với tác giả.

6. Làm slide và viết báo cáo cuối kỳ.

7. Viết chương trình web app demo phát hiện mã độc từ dữ liệu thu được và model đã training.

Các công việc đã thực hiện và kết quả:

1. Tìm hiểu, đọc bài báo và tài liệu liên quan để hiểu rõ nội dung, mục đích của bài báo; xác định cách triển khai mà bài báo đã làm, tài nguyên, môi trường thực hiện.
   ➔ Kết quả: Tất cả các thành viên đã đọc và nắm rõ nội dung, mục đích, cách thực hiện, kết quả mà bài báo trình bày.
2. Tìm kiếm và thu thập các dataset mà tác giả đã sử dụng.
   ➔ Kết quả: Không thu được dataset từ nguồn tác giả cung cấp, phải tự thu thập, đã xoay sở xử lý được định dạng của dataset giống với dataset FFRI mà bài báo đã sử dụng.
3. Tìm hiểu và triển khai các thuật toán của tác giả, trong đó có mô hình LSI, Doc2Vec và framework chainer.
   ➔ Kết quả: Hoàn thành tốt. Chi tiết được nhóm trình bày ở phần dưới
4. Thực hiện tối ưu hóa mô hình
   ➔ Kết quả: Đã hoàn thành.
5. Phân tích đánh giá mở rộng, vẽ biểu đồ, đồng thời so sánh kết quả với tác giả
   ➔ Kết quả: Hoàn thành một phần. Chi tiết kết quả được trình bày ở phần dưới.
6. Làm slide và viết báo cáo cuối kỳ
   ➔ Kết quả: Hoàn thành.
7. Viết chương trình web app demo phát hiện mã độc từ dữ liệu thu được và model đã training.
   ➔ Kết quả: Đã có thể xây dựng ứng dụng cơ bản sử dụng mô hình đã được huấn luyện để dự đoán mã độc.

### J. Các khó khăn, thách thức hiện tại khi thực hiện:

1. Tác giả không cung cấp dataset và code, nhóm phải tự thu thập dataset và code lại theo thuật toán của tác giả.

2. Nhóm không lấy được dataset từ các nguồn mà tác giả đã thu thập: đã liên hệ nhưng tại vì dataset FFRI cần phải được nhóm tác giả chấp nhận và cấp phép. Nhóm đã liên hệ với nhóm tác giả nhưng không nhận được phản hồi nên không thể truy cập được dataset FFRI.

3. Framework chainer cần hiểu sâu về cấu trúc của các thuật toán deep learning để sử dụng, cần biết cách sử dụng nhiều component để tạo và train một model: *dataset*, *iterator*, *model*, *optimizer*, *updater*, *trainer*, và nhiều *extension* khác.

## 3. TỰ ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH SO VỚI KẾ HOẠCH THỰC HIỆN:

Mức độ hoàn thành: 95%

## 4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

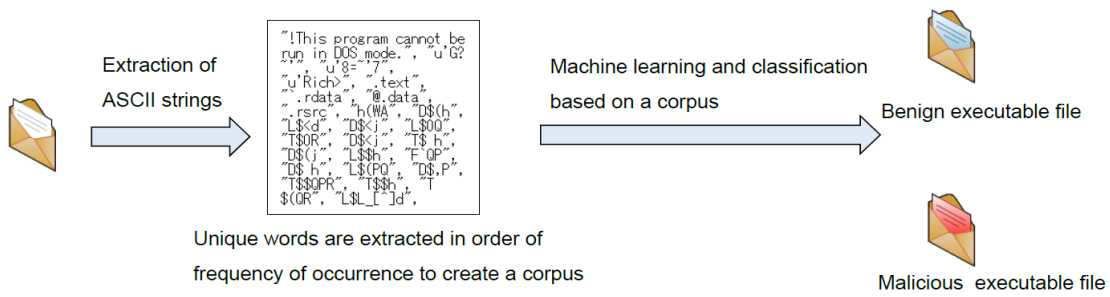
| STT | Công việc | Phân công nhiệm vụ |
|---|---|---|
| 1 | Phân công công việc | Trần Minh Duy |
| 2 | Viết báo cáo giữa kỳ | Nguyễn Phương Trinh |
| 3 | Tìm kiếm và thu thập dataset | Nguyễn Thị Minh Châu |
| 4 | Tìm hiểu mô hình và triển khai | Trần Minh Duy |
| 5 | Tối ưu hóa mô hình | Nguyễn Phương Trinh |
| 6 | Phân tích, so sánh, đánh giá kết quả, vẽ đồ thị biểu diễn | Cả nhóm |
| 7 | Tổng hợp bài báo cáo, thiết kế slide và thuyết trình | Nguyễn Phương Trinh |
| 8 | Viết web app demo phát hiện mã độc | Trần Minh Duy |
| 9 | Viết báo cáo cuối kỳ | Cả nhóm |

# BÁO CÁO TỔNG KẾT CHI TIẾT

**Phần bên dưới của báo cáo này là tài liệu báo cáo tổng kết - chi tiết của nhóm thực hiện cho đề tài này.**

## A. PHƯƠNG PHÁP THỰC HIỆN

- Kiến trúc chính của bài báo:
  + Mô hình phát hiện trong bài báo sử dụng các kỹ thuật xử lý ngôn ngữ tự nhiên (NLP) và bao gồm hai thành phần chính: các mô hình ngôn ngữ và các bộ phân loại. Cấu trúc tổng thể của hệ thống được mô tả trong Hình 9 và *Hình 10*.
  + Trong đó, các mô hình ngôn ngữ được sử dụng để trích xuất các đặc trưng từ các mẫu phần mềm độc hại và mẫu lành tính. Các bộ phân loại được huấn luyện với các đặc trưng trích xuất bởi các mô hình ngôn ngữ. Nhiều bộ phân loại khác nhau được xem xét, mỗi loại có đặc điểm riêng.*Hình 9*




Hình 10. Sơ đồ tóm tắt luồng hoạt động chính của bài báo

- Thành phần chính của hệ thống trong bài báo:
  + Mô hình ngôn ngữ: LSI, Doc2Vec
  + Mô hình phân loại: SVM, RF, XGB, MLP, CNN (có các thông số như *Bảng 1. Thông số các mô hình phân loại trong hệ thống*)

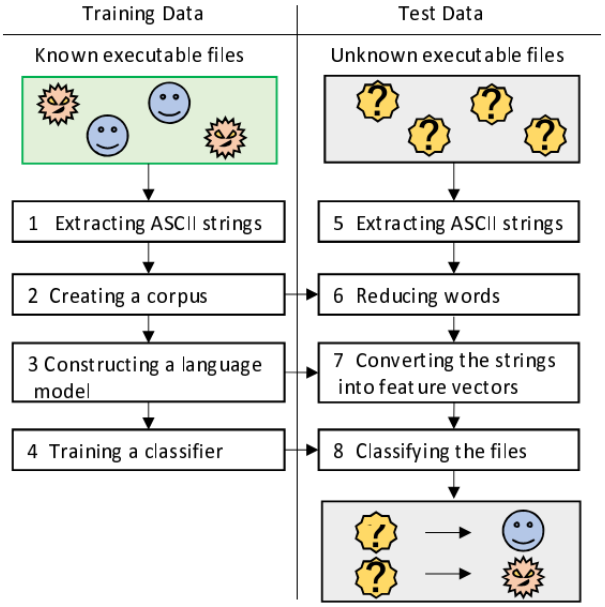
**Bảng 1. Thông số các mô hình phân loại trong hệ thống**

| Classifier | Parameter | Optimum value | |
|---|---|---|---|
| | | Doc2Vec | LSI |
| | kernel | rbf | rbf |
| **SVM** | C | 100 | 100 |
| | gamma | 0.01 | 0.1 |
| **RF** | n_estimators | 392 | 442 |

| | | | |
|---|---|---|---|
| | n_jobs | 14 | 32 |
| | max_depth | 11 | 3 |
| **XGB** | min_child_weight | 11 | 5 |
| | subsample | 0.5 | 0.8 |
| | colsample_tree | 0.7 | 0.6 |
| **MLP** | optimizer | Adam | Adam |
| | epoch | 40 | 40 |
| **CNN** | optimizer | Adam | Adam |
| | epoch | 40 | 40 |

- Hai giai đoạn chính của hệ thống trong bài báo (tổng quan *ở Hình 11*):
  + Giai đoạn huấn luyện:
    - Trích xuất chuỗi: Trích xuất tất cả các chuỗi in được (ASCII) từ cả mẫu phần mềm độc hại và mẫu lành tính.
    - Chọn từ: Tách các chuỗi trích xuất thành các từ và chọn những từ xuất hiện thường xuyên.
    - Xây dựng mô hình:
      - Nếu sử dụng Doc2vec, xây dựng mô hình Doc2vec từ các từ đã chọn.
      - Nếu sử dụng LSI, xây dựng mô hình TF-IDF từ các từ đã chọn, sau đó xây dựng mô hình LSI.
    - Chuyển đổi đặc trưng: Chuyển đổi các chuỗi in được thành các vector đặc trưng sử dụng mô hình ngôn ngữ đã xây dựng.
    - Huấn luyện bộ phân loại: Huấn luyện bộ phân loại đã chọn sử dụng các vector đặc trưng có nhãn từ các bước trước (*Hình 12*)

  + Giai đoạn kiểm tra:
    - Trích xuất chuỗi: Trích xuất các chuỗi in được từ các mẫu không xác định và tách chúng thành các từ.
    - Chuyển đổi đặc trưng: Chuyển đổi các từ đã trích xuất thành các đặc trưng sử dụng mô hình ngôn ngữ đã xây dựng trong giai đoạn huấn luyện.
    - Phân loại: Bộ phân loại đã được huấn luyện sẽ kiểm tra các vector đặc trưng và dự đoán liệu các mẫu có phải là phần mềm độc hại hay không (*Hình 13*)




Hình 11. Minh họa 2 giai đoạn chính của hệ thống trong bài báo

**Algorithm 1** training

```
/* Extract printable strings */
for all malware samples m do
    mw ← extract printable strings from m
end for
for all benign samples b do
    bw ← extract printable strings from b
end for
imw ← select frequent words from m
ibw ← select frequent words from b
if Doc2vec then
    /* Construct a Doc2vec model */
    construct a Doc2vec model from imw, ibw
else
    /* Construct a LSI model */
    construct a TF-IDF model from imw, ibw
    construct a LSI model from the TF-IDF
end if
/* Convert printable strings into vectors */
for all malware samples mw do
    if Doc2vec then
        mv ← Doc2vec(mw)
    else
        mv ← LSI(mw)
    end if
end for
for all benign samples bw do
    if Doc2vec then
        bv ← Doc2vec(bw)
    else
        bv ← LSI(bw)
    end if
end for
/* Train classifiers with the labeled vectors */
if SVM then
    Train SVM(mv, bv)
else if RF then
    Train RF(mv, bv)
else if XGB then
    Train XGB(mv, bv)
else if MLP then
    Train MLP(mv, bv)
else
    Train CNN(mv, bv)
end if
return
```

Hình 12. Thuật toán giai đoạn Training các mô hình phân loại

**Algorithm 2** test

```
/* Extract printable strings */
for all unknown samples u do
    uw ← extract printable strings from u
end for
/* Convert printable strings into vectors */
for all unknown samples uw do
    if Doc2vec then
        uv ← Doc2vec(uw)
    else
        uv ← LSI(uw)
    end if
end for
/* Predict labels with the trained classifiers */
for all unknown vectors uv1 do
    if SVM then
        label ← SVM(uv)
    else if RF then
        label ← RF(uv)
    else if XGB then
        label ← XGB(uv)
    else if MLP then
        label ← MLP(uv)
    else
        label ← CNN(uv)
    end if
end for
return
```

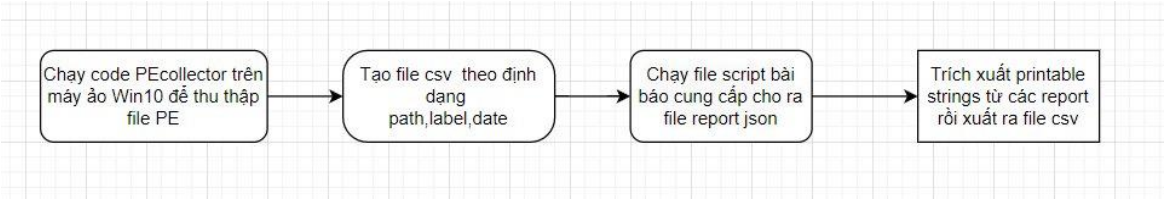
Hình 13. Thuật toán giai đoạn Testing các mô hình phân loại

- Nhóm đã triển khai toàn bộ các kiến trúc trên cũng như các thành phần, giai đoạn của hệ thống trong bài báo. Ngoài ra, nhóm còn tự thu thập data và xử lý để cho data có format giống với dataset FFRI mà nhóm tác giả bài báo đã sử dụng. Chi tiết thực hiện ở phần B dưới đây.

## B. CHI TIẾT CÀI ĐẶT, HIỆN THỰC

- Môi trường thực nghiệm: Google Colab
- Ngôn ngữ sử dụng: Python 3.10
- Các thư viện sử dụng:
  + numpy, pandas cho xử lý dữ liệu.
  + gensim cho mô hình LSI.
  + scikit-learn cho các bộ phân loại SVM, RF, XGB.
  + chainer cho MLP và CNN
  + matplotlib để vẽ biểu đồ.

- Quá trình chuẩn bị dữ liệu:
  + Đầu tiên ta truy cập vào link github mà bài báo dùng để tạo lại dataset có format giống với format của dataset FFRI. Ta cần tạo một file .csv gồm 3 cột là path, label, date tương ứng với path của file PE sample, label nó là 0 (bengin) hay 1 (malware) và date là ngày modified của file sample.
  + Đối với sample benign thì nhóm em chạy code PEcollector.jar trên máy ảo Win10 để tiến hành thu thập các file sample benign. Về date thì em lấy luôn ngày modified trên máy lúc thu thập được.




Hình 14. Workflow tạo dataset benign

+ Còn với các file sample malware thì nhóm sẽ sắp xếp theo thời gian để phục vụ cho việc phân tích time series sau này. Theo trình tự thì trước tiên nhóm sẽ tiến hành crawl mã hash của file malware từ VirusShare. Ở đây nhóm sẽ lấy theo họ là Ransomware và WannaCry. Sử dụng Selenium để có thể cào một cách tự động chứ không cần cào thủ công.

```
[4]
search_box = driver.find_element("name", 'search')
search_box.send_keys('wanna extension:exe after:2023-01-02T03:04:05Z')
submit_button = driver.find_element(By.XPATH, "//input[@type='Submit']")
submit_button.click()
```

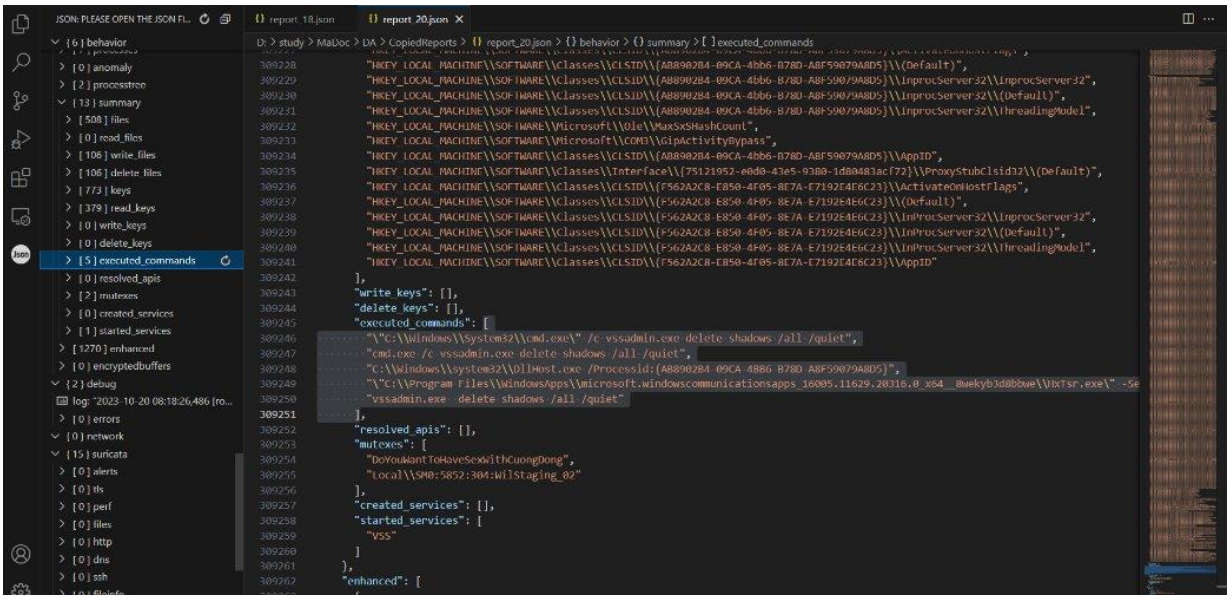
Hình 15. Code search sample theo từ khoá là wanna với extension là file .exe và thời gian modified là sau ngày 02/01/2023

+ Tiếp theo ta sẽ download sample theo list file gồm các mã hash đã lấy được ở trên. Ta cũng sẽ áp dụng Selenium để có thể download tự động. Ở đây ta sẽ download bằng API VirusShare cung cấp sẵn. Vì VirusShare có rate limit nên ta cũng sẽ đặt rate limit là 4 request trong 75 giây để không bị timeout trong lúc download.

```python
@sleep_and_retry
@limits(calls=CALLS, period=RATE_LIMIT)
def download_hash(hash, path):
    result = v.download(hash, path)
    return 0

parent_folder = "malsam"

def download_file(url, folder):
    filename = url.split("/")[-1]
    filepath = os.path.join(folder, filename)
    response = requests.get(url)
    with open(filepath, "wb") as file:
        file.write(response.content)
    print(f"Downloaded {filename} to {folder}")

def process_subfolders(folder):
    count = 1
    for root, subfolders, files in os.walk(folder):
        for subfolder in subfolders:
            subfolder_path = os.path.join(root, subfolder)
            file_name = subfolder + ".txt"
            text_file_path = os.path.join(subfolder_path, file_name)
            if os.path.isfile(text_file_path):
                with open(text_file_path, "r") as file:
                    for line in file:
                        print(count)
                        count += 1
                        print(f'{root}/{subfolder}/{file_name}: {line.rstrip()}')
                        result_path = f'{root}/{subfolder}'
                        if os.path.exists(f'{result_path}/VirusShare_{line.rstrip()}.zip'):
                            print("Downloaded")
                            continue
                        else:
                            print("Now downloading")
                            download_hash(line.rstrip(), result_path)
```

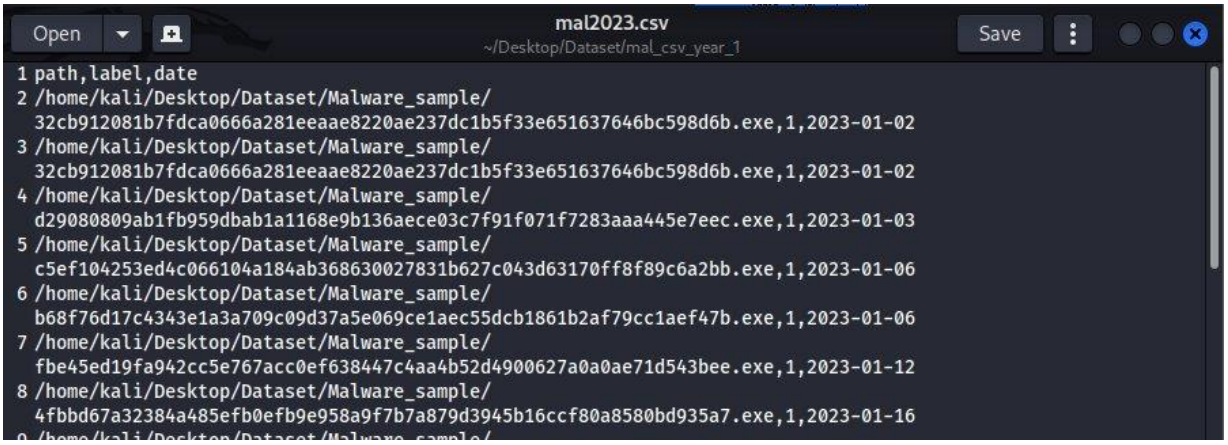
Hình 16. Code download sample xuống theo list hash có sẵn

+ Sau khi có file sample thì ta sẽ đưa vào môi trường Cuckoo Sandbox để nó tiến hành phân tích động file malware. Hoàn thành việc phân tích thì ta được các file report dưới dạng json bao gồm các thông tin hành vi chi tiết của file malware. Ta thấy rằng trong file này các strings nằm rải rác khắp nơi và nó không đúng định dạng của dataset FFRI nên ta chỉ lấy phần timestamp lọc ra để phục vụ cho việc tạo file .csv



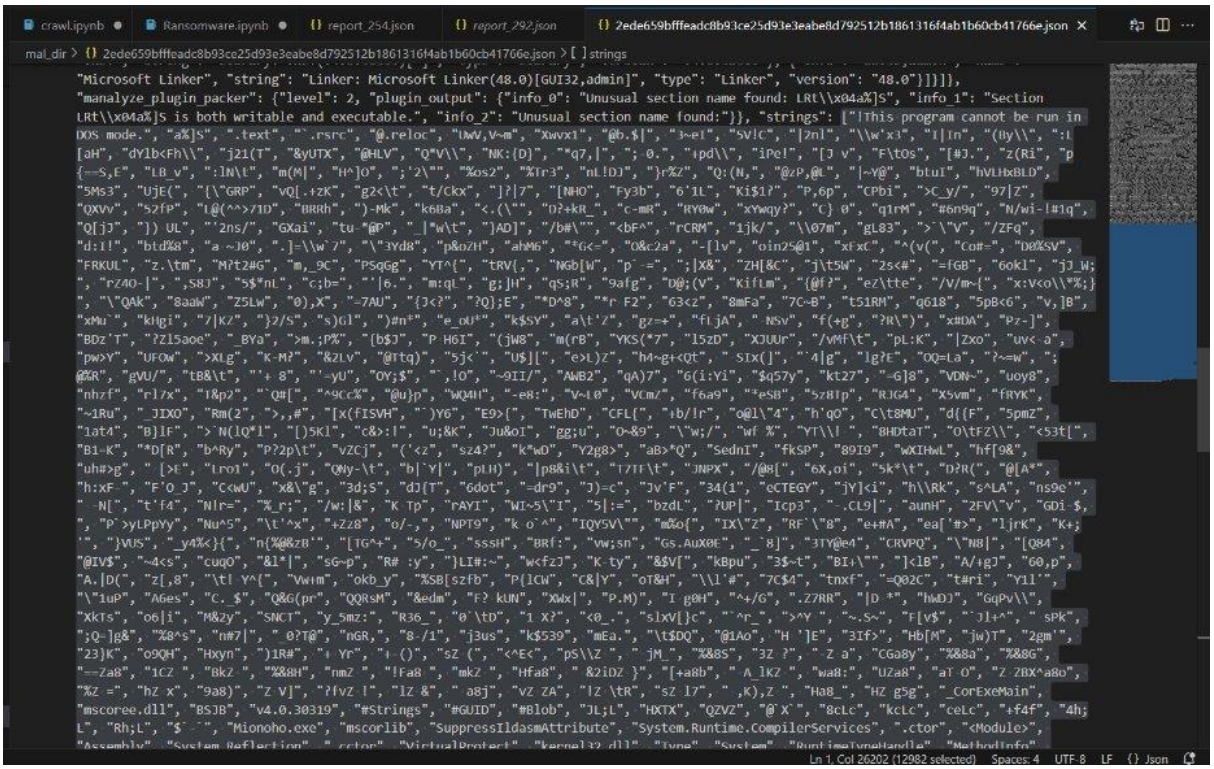
Hình 17. File json report từ Cuckoo Sandbox

+ Trích xuất được timestamp xong ta sẽ cho nó vào cột date trong file .csv và ta đã có một file .csv hoàn chỉnh của cả dataset benign và malware




Hình 18. File csv theo đúng định dạng đầu vào yêu cầu của code tạo dataset

+ Chạy các file docker có trong link github để có thể tạo được file report theo đúng định dạng của dataset FFRI



Hình 19. File report sau khi chạy code mà bài báo đưa ra

+ Từ các file report đã có ta sẽ trích xuất thêm một lần nữa để tổng hợp lại tất cả các printable strings có trong các file report vào một file csv duy nhất

Hình 20. File csv sau khi đã tổng hợp lại từ các file json

+ Kết quả sau khi tạo dataset thì ta đã có các tập dataset với số lượng như sau:

- Malware train + test = 501 sample
- Benign train: 255 sample
- Malware unknown1 + unknown2: 639 sample
- Benign unknown1 + unknown2: 170 sample
- Trong đó, tệp malware unknown1 có timestamp trước tệp unknown2 (sử dụng cả 2 tệp để kiểm tra khả năng phát hiện malware mới cho các mô hình)

- Hiện thực hệ thống:
+ Với dataset là các file .csv chứa các printable strings đã được trích xuất từ các sample PE, thu được từ quá trình chuẩn bị dữ liệu trên. Thực hiện sử dụng chúng để làm đầu vào cho quá trình huấn luyện các mô hình máy học phân loại malware.
+ Như đã trình bày trên phần kiến trúc và thành phần của hệ thống bài báo, nhóm chia phần hiện thực hệ thống thành 5 phần: 1. Trích xuất chuỗi in được; 2. Chọn các từ có tần suất xuất hiện cao và tạo corpus; 3. Khởi tạo language model; 4. Gán nhãn và huấn luyện các mô hình máy học phân loại mã độc (Hình 12) và 5. Kiểm tra mức độ nhận biết các loại mã độc mới (chưa được dùng để huấn luyện các mô hình phân loại) (Hình 13). Cụ thể hiện thực từng phần như sau:

- Phần 1: Trích xuất chuỗi in được: Tất cả các chuỗi ASCII từ các mẫu phần mềm độc hại và lành tính được trích xuất. Chuỗi này sau đó được chuyển thành danh sách từ malware_words và benign_words. Ngoài ra các xử lý được cụ thể trong đoạn code sau đây:

```
import csv
def Strings(file_path):
    strings = []
    with open(file_path, newline='') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            for cell in row:
                strings.append(cell[:MAX_STRINGLEN])

    # Now limit the amount of strings
    strings = strings[:MAX_STRINGCNT]

    return strings

def String2word(string_list):
    # Join the list of strings into a single string
    full_string = ' '.join(string_list)
    # Split the full string into words
```

```
    return full_string.lower().split()

MAX_FILESIZE = 16*1024*1024
MAX_STRINGCNT = 2048
MAX_STRINGLEN = 1024

malware_files = ['/content/drive/MyDrive/ProjectMal/train (1).csv',
'/content/drive/MyDrive/ProjectMal/test (1).csv']
benign_files =
['/content/drive/MyDrive/ProjectMal/train_ben_data.csv']
malware_words = []
benign_words = []

import sys
csv.field_size_limit(sys.maxsize)

for file_path in malware_files:
    strings = Strings(file_path)
    words = String2word(strings)
    malware_words.extend(words)

for file_path in benign_files:
    strings = Strings(file_path)
    words = String2word(strings)
    benign_words.extend(words)

del strings, words, file_path
```

- Phần 2: Chọn các từ có tần suất xuất hiện cao và tạo corpus: Chọn các từ phổ biến nhất từ các danh sách từ malware_words và benign_words; và đối với mỗi mô hình ngôn ngữ sẽ có chỉ định số lượng từ phổ biến nhất (với LSI là 9000 và Doc2Vec là 500) được lưu vào 'benign_frequent_words' và 'malware_frequent_words'. Sau đó, giảm danh sách các từ phổ biến từ các mẫu dữ liệu văn bản mã độc và không mã độc, sau đó kết hợp chúng lại để tạo thành một corpus

```
from collections import Counter

def Select_frequent_words(samples, num_unique_words):
    word_counts = Counter()
    for words in samples:
        word_counts.update(words)

    # Select top num_unique_words most common words
```

```
        frequent_words = [word for word, _ in
word_counts.most_common(num_unique_words)]
    return frequent_words

if language_model == "Doc2Vec":
        benign_frequent_words = Select_frequent_words(benign_words,
num_unique_words=500)
      malware_frequent_words = Select_frequent_words(malware_words,
num_unique_words=500)
else:
        benign_frequent_words = Select_frequent_words(benign_words,
num_unique_words=9000)
      malware_frequent_words = Select_frequent_words(malware_words,
num_unique_words=9000)

def Reduce_words(words, frequent_words):
    reduced_words = []
    for file_words in words:
        reduced_file_words = [word for word in file_words if word in
frequent_words]
        reduced_words.append(reduced_file_words)
    return reduced_words

reduced_benign_words = Reduce_words(benign_words,
benign_frequent_words)
reduced_malware_words = Reduce_words(malware_words,
malware_frequent_words)
corpus = reduced_benign_words + reduced_malware_words
```

- Phần 3: Khởi tạo language model (LSI và Doc2Vec): khởi tạo 2 mô hình ngôn ngữ này từ corpus output bước 2. Mô hình đang dùng (LSI/Doc2Vec) sẽ phụ thuộc vào giá trị của model_type. Doc2Vec tạo ra các vector đại diện cho các document, trong khi LSI sử dụng các latent topics để mô hình hóa mối quan hệ giữa các từ trong các document

```
def tagged_document(list_of_list_of_words):
    for i, list_of_words in enumerate(list_of_list_of_words):
        yield TaggedDocument(list_of_words, [i])

def Construct_language_model(corpus, model_type):
    if model_type == "Doc2Vec":
        data_for_training = list(tagged_document(corpus))
        model = Doc2Vec(vector_size=400, alpha=0.075, min_count=2,
window=1, epochs=20)
```

```
        model.build_vocab(data_for_training)
        model.train(data_for_training,
total_examples=model.corpus_count, epochs=model.epochs)
    else:
        # Construct a dictionary from corpus
        dictionary = Dictionary(corpus)
        # Convert the corpus into a Bag-of-words corpus
        corpus_bow = [dictionary.doc2bow(words) for words in
corpus]
        # Construct the TF-IDF model
        tfidf = TfidfModel(corpus_bow)
        # Transform the whole corpus
        corpus_tfidf = tfidf[corpus_bow]
        # Construct the LSI model
        model = LsiModel(corpus_tfidf, id2word=dictionary,
num_topics=800)
    return model

# Construct language model
language_model = Construct_language_model(corpus, language_model)
```

- Phần 4: Gán nhãn và huấn luyện các mô hình máy học phân loại mã độc: gồm các nội dung: vector hóa (nếu language_model là Doc2Vec, nó sử dụng infer_vector để lấy vector đại diện cho từng từ (word_list). Nếu không (LSI), nó sử dụng doc2bow để chuyển đổi word_list thành một Bag-of-Words (bow), sau đó sử dụng mô hình để chuyển đổi thành vector. Kết quả là danh sách các vector tương ứng với các từ trong words); gán nhãn cho dữ liệu '1' cho malware và '0' cho benign, loại bỏ các dòng chứa NaN, chia dữ liệu train test (tỉ lệ 8:2); sau đó xây dựng các mô hình máy học với các thông số như *Bảng 1.*

```
def Convert_to_vectors(words, language_model):
    vectors = []
    for word_list in words:
        if isinstance(language_model, Doc2Vec):
            vector = language_model.infer_vector(word_list)
        else:
            bow = language_model.id2word.doc2bow(word_list)
            vector = [value for _, value in language_model[bow]]
        vectors.append(vector)
    return vectors

# Convert malware strings to vectors
malware_vectors = Convert_to_vectors(reduced_malware_words,
language_model)
```

```
# Convert benign strings to vectors
benign_vectors = Convert_to_vectors(reduced_benign_words,
language_model)

# Labels for malware and benign samples
malware_labels = [1] * len(malware_vectors)  # 1 for malware
benign_labels = [0] * len(benign_vectors)  # 0 for benign

df = pd.DataFrame(malware_vectors + benign_vectors)

df['label'] = malware_labels + benign_labels
print("Shape before dropna:", df.shape)

df.dropna(inplace = True)
print("Shape after dropna:", df.shape)

X = df.drop('label', axis = 1)
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

X_train = np.array(X_train, dtype=np.float32)
y_train = np.array(y_train, dtype=np.int32)
X_test = np.array(X_test, dtype=np.float32)
y_test = np.array(y_test, dtype=np.int32)

…
```

- Phần 5: Kiểm tra mức độ nhận biết các loại mã độc mới: Với 2 tệp unknown_1.csv và unknown_2.csv. Thực hiện sử dụng lại model đã được huấn luyện từ 4 bước trên để kiểm tra khả năng nhận biết, phân loại với các malware mới, chưa có trong data đã dùng để train mô hình phân loại

- Sau khi đã hoàn thành việc huấn luyện mô hình học máy thì nhóm tiến hành lưu lại mô hình rồi cho vào ứng dụng để nó thực hiện việc đánh giá và phát hiện mã độc. Ở đây nhóm chúng em sử dụng nền tảng Streamlit để xây dựng một ứng dụng cơ bản với chức năng dự đoán mã độc thông qua các mô hình có sẵn:

  + Tạo một số title để trên UI:

```
st.title("Malware Detection")
st.subheader("Detect Malware in PE Files")
```

+ Ở đây ta có 2 model language là LSI và Doc2Vec nên ta tạo một ô option để chọn một trong hai model

```
model_type = ["LSI", "Doc2Vec"]
LANGUAGE_MODEL = st.sidebar.selectbox("Choose Option", model_type)
```

+ Các hàm được sử dụng để phân tích file PE được tải lên:

```
def Strings(file_path):
    data = open(file_path, "rb").read(MAX_FILESIZE)
    strings = []
    for s in re.findall(b"[\x1f-\x7e]{6,}", data):
        strings.append(s.decode("utf-8"))
    for s in re.findall(b"(?:[\x1f-\x7e][\x00]){6,}", data):
        strings.append(s.decode("utf-16le"))

    strings = strings[:MAX_STRINGCNT]
    for idx, s in enumerate(strings):
        strings[idx] = s[:MAX_STRINGLEN]

    return strings

def String2word(string_list):
    return ' '.join(string_list).lower().split()

def Reduce_words(words, frequent_words):
    return [[word for word in file_words if word in frequent_words] for
file_words in words]

def Convert_to_vectors(words, language_model):
    vectors = []
    for word_list in words:
        if isinstance(language_model, Doc2Vec):
            vector = language_model.infer_vector(word_list)
        else:
            bow = language_model.id2word.doc2bow(word_list)
            vector = [value for _, value in language_model[bow]]
        vectors.append(vector)
    return vectors
```

+ Rồi tiếp đến là load model tuỳ vào lựa chọn của người dùng sẽ là “Doc2Vec” hoặc là “LSI”

```
def load_language_model(filename):
```

```
    return Doc2Vec.load(filename) if filename.endswith("Doc2Vec") else
LsiModel.load(filename + '.lsi')
language_model = load_language_model(save_path+LANGUAGE_MODEL)
```

+ Bây giờ đến phần xử lí mô hình. Trong 5 mô hình mà nhóm đã huấn luyện thì nhóm sẽ chia thành 2 loại đó là Random Forest, SVM và XGBoost với lại MLP và CNN dựa vào đặc điểm khác nhau của các loại mô hình.

+ Ta sẽ xử lí phần load model MLP và CNN. Vì đây là các mô hình học sâu (deep learning), bao gồm nhiều lớp nơ-ron (neural layers) nên ta cần định nghĩa lại các lớp khi load model có sẵn. Hai hàm MLP() và CNN() chúng em dùng để xử lí phần model này như sau:

```
def MLP(n_units, n_out):
    layer = ch.Sequential(L.Linear(n_units), F.relu)
    model = layer.repeat(2)
    model.append(L.Linear(n_out))
    return model

class CNN(ch.Chain):
    def __init__(self, n_out):
        super(CNN, self).__init__()
        with self.init_scope():
             self.conv1 = L.Convolution1D(None, 32, ksize=3, stride=1,
pad=1)
              self.conv2 = L.Convolution1D(32, 64, ksize=3, stride=1,
pad=1)
            self.fc1 = L.Linear(None, 512)
            self.fc2 = L.Linear(512, n_out)

    def forward(self, x):
        h = F.relu(self.conv1(x))
        h = F.max_pooling_1d(h, 2, 2)
        h = F.relu(self.conv2(h))
        h = F.max_pooling_1d(h, 2, 2)
        h = F.relu(self.fc1(h))
        h = self.fc2(h)
        return h
```

+ Khi load model CNN và MLP lên thì ta sử dụng hai hàm kể trên để có thể định nghĩa lại các lớp của nó

```
mlp_model = L.Classifier(MLP(40, 1), lossfun=F.sigmoid_cross_entropy,
accfun=F.binary_accuracy)
```

```
ch.serializers.load_npz(save_path+f"_{LANGUAGE_MODEL}_mlp_model.npz",
mlp_model)
cnn_model = L.Classifier(CNN(1), lossfun=F.sigmoid_cross_entropy,
accfun=F.binary_accuracy)
ch.serializers.load_npz(save_path+f"_{LANGUAGE_MODEL}_cnn_model.npz",
cnn_model)
```

+ Còn loại mô hình còn lại bao gồm Random Forest, SVM và XGBoost thì không cần phải xử lí lại mô hình mà ta chỉ cần load từ file .pkl lên là được

```
svm_model = joblib.load(save_path+f"_{LANGUAGE_MODEL}-SVM.pkl")
rf_model = joblib.load(save_path+f"_{LANGUAGE_MODEL}-RF.pkl")
xgb_model = joblib.load(save_path+f"_{LANGUAGE_MODEL}-XGB.pkl")
```

+ Sau khi đã load được model thì ta chỉ cần thêm các phần phụ trong ứng dụng như chỗ để upload file. Sau khi upload được file lên thì ta tiến hành chạy các hàm xử lí strings rồi đưa nó vào các model đã được load lên để phân tích.

```
uploaded_file = st.file_uploader("Choose a PE file", type=["exe",
"dll"])

if uploaded_file is not None:
    with open("uploaded_file.exe", "wb") as f:
        f.write(uploaded_file.getbuffer())

    unknown_words = [String2word(Strings('uploaded_file.exe'))]
            reduced_unknown_words = Reduce_words(unknown_words,
benign_frequent_words + malware_frequent_words)
        unknown_vectors = Convert_to_vectors(reduced_unknown_words,
language_model)
    unknown_vectors = np.array(unknown_vectors, dtype=np.float32)

    svm_predictions = svm_model.predict(unknown_vectors)
    rf_predictions = rf_model.predict(unknown_vectors)
    xgb_predictions = xgb_model.predict(unknown_vectors)
                                    mlp_predictions =
np.where(mlp_model.predictor(unknown_vectors).array < 0, 0, 1)
                                    cnn_predictions =
np.where(cnn_model.predictor(unknown_vectors.reshape(-1, 1,
unknown_vectors.shape[1])).array < 0, 0, 1)

    all_predictions = {
        'SVM': svm_predictions.flatten(),
        'Random Forest': rf_predictions.flatten(),
        'XGBoost': xgb_predictions.flatten(),
```

```
        'MLP': mlp_predictions.flatten(),
        'CNN': cnn_predictions.flatten()
    }
```

+ Kết quả dự đoán file PE có chứa mã độc hay không của các mô hình phân loại được hiển thị (hình ảnh GUI được trình bày ở mục C):

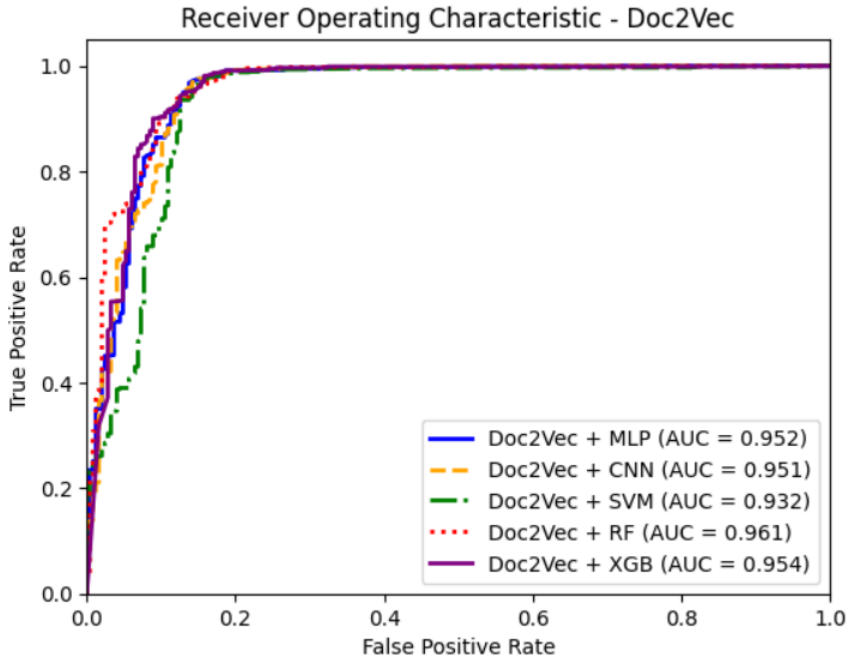
```
    st.info("Malware Detection Results:")

    # Create a layout with columns for better visualization
    cols = st.columns(len(all_predictions))

    # Iterate through predictions and display them in separate columns
    for idx, (model, prediction) in enumerate(all_predictions.items()):
        with cols[idx]:
            st.subheader(model)
            if prediction[0] == 1:
                st.error("Malware Detected")
            else:
                st.success("No Malware Detected")
```

## C. KẾT QUẢ THỰC NGHIỆM

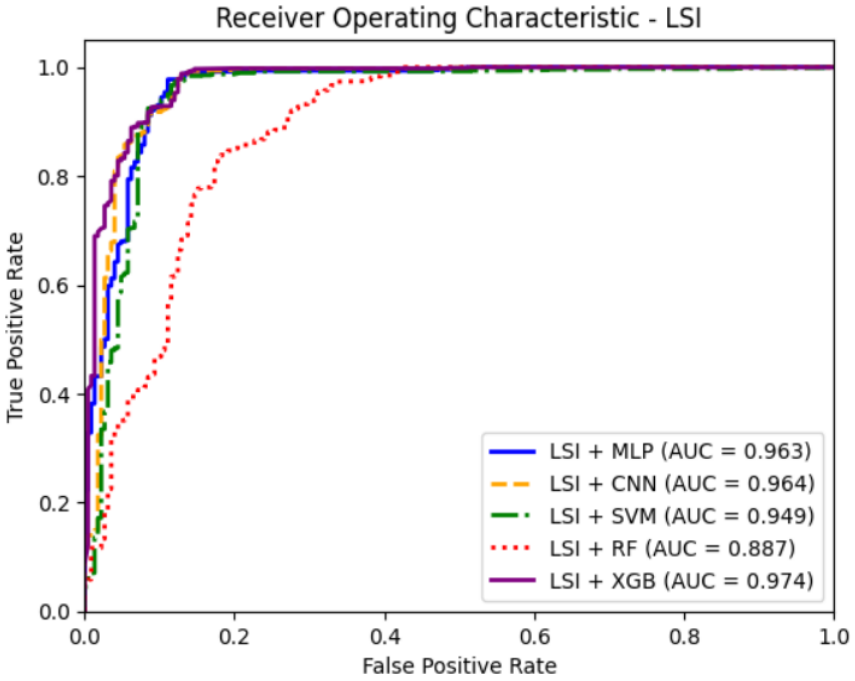
- Thực nghiệm toàn bộ phần B. Hiện thực hệ thống ở trên, thu được các kết quả sau đây:

**Bảng 2. Kết quả huấn luyện của các mô hình phân loại**

| | SVM | | RF | | XGB | | MLP | | CNN | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Doc2Vec** | **LSI** | **Doc2Vec** | **LSI** | **Doc2Vec** | **LSI** | **Doc2Vec** | **LSI** | **Doc2Vec** | **LSI** |
| **Accuracy** | 0.942 | 0.956 | 0.947 | 0.907 | 0.951 | 0.966 | 0.948 | 0.960 | 0.949 | 0.964 |
| **Precision** | 0.942 | 0.955 | 0.949 | 0.917 | 0.950 | 0.967 | 0.948 | 0.960 | 0.949 | 0.963 |
| **Recall** | 0.942 | 0.956 | 0.947 | 0.907 | 0.951 | 0.966 | 0.948 | 0.960 | 0.949 | 0.964 |
| **F1-score** | 0.940 | 0.956 | 0.944 | 0.895 | 0.950 | 0.966 | 0.948 | 0.960 | 0.948 | 0.963 |



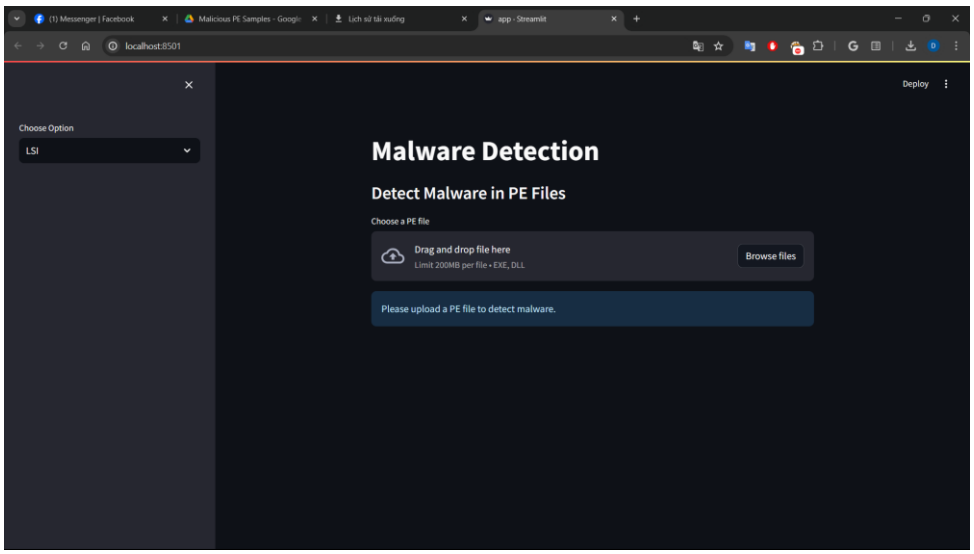

Hình 20. Đường cong ROC khi dùng model language – Doc2Vec



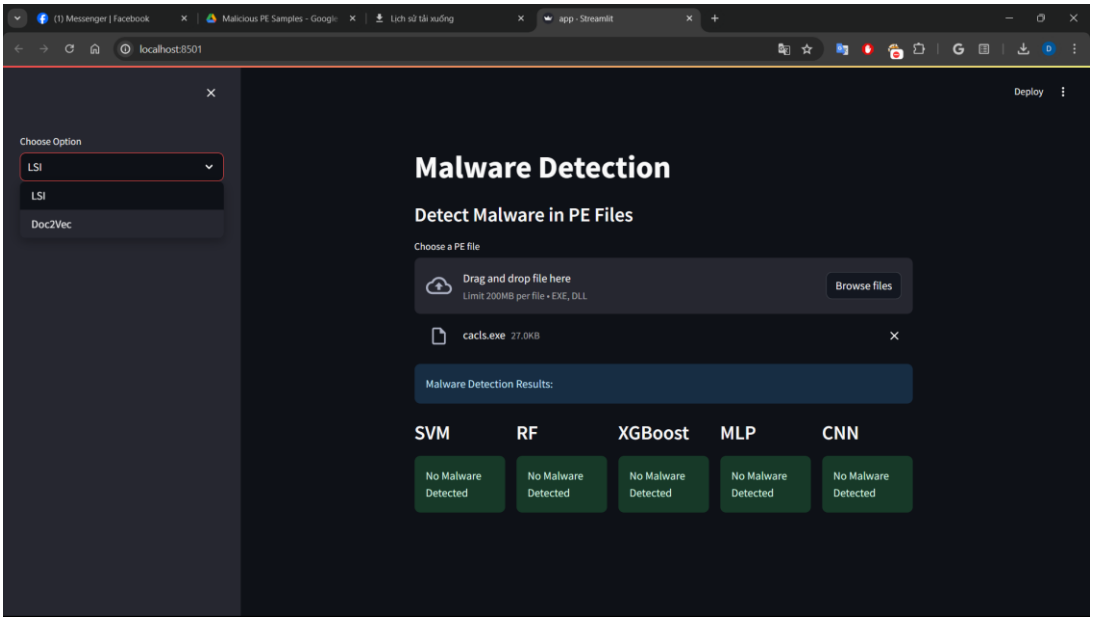

Hình 21. Đường cong ROC khi dùng model language – LSI

**Bảng 3. Kết quả phân loại mã độc mới của các mô hình đã được huấn luyện**

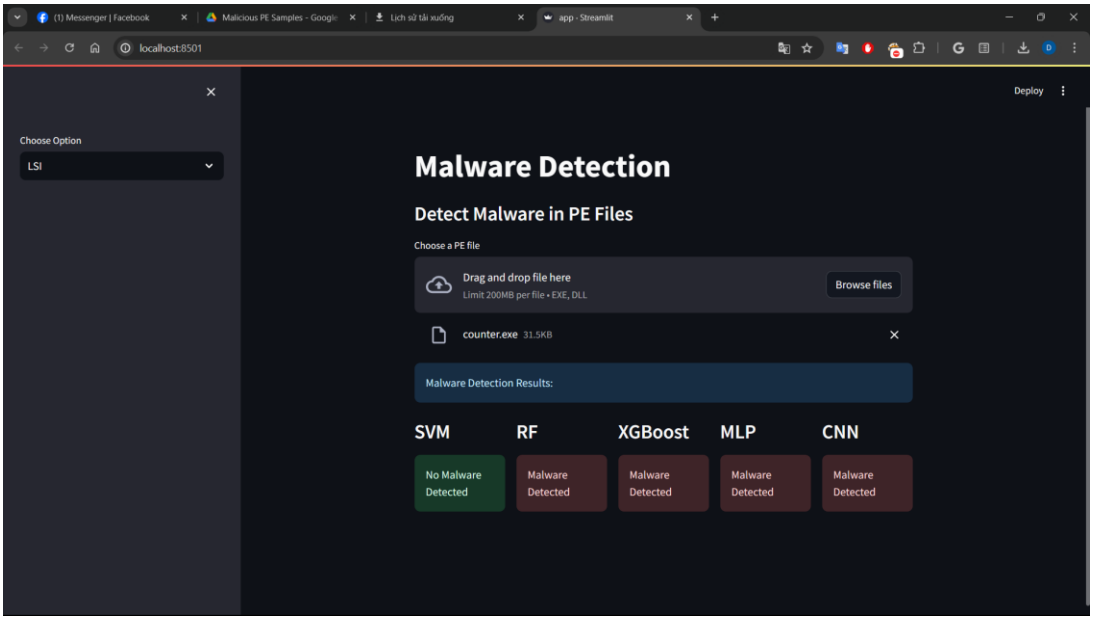
| | SVM | | RF | | XGB | | MLP | | CNN | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Doc2Vec** | **LSI** | **Doc2Vec** | **LSI** | **Doc2Vec** | **LSI** | **Doc2Vec** | **LSI** | **Doc2Vec** | **LSI** |
| **unknown1** | 0.620 | 0.754 | 0.688 | 0.732 | 0.678 | 0.778 | 0.584 | 0.756 | 0.686 | 0.785 |
| **unknown2** | 0.628 | 0.606 | 0.679 | 0.672 | 0.650 | 0.681 | 0.610 | 0.710 | 0.652 | 0.685 |



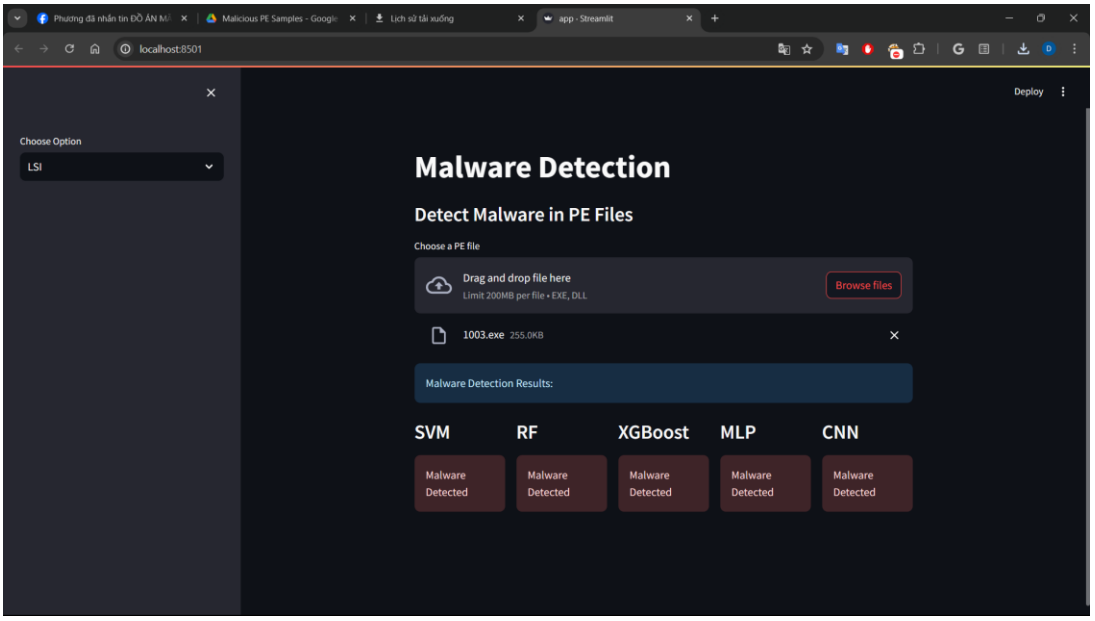

Hình 22. Giao diện web dùng để kiểm tra file PE có chứa mã độc hay không (sử dụng các mô hình đã huấn luyện)




Hình 23. Kết quả khi upload 1 file benign calcs.exe lên web




Hình 24. Kết quả khi upload file malware counter.exe lên web




Hình 25. Kết quả khi upload file malware 1003.exe lên web

- Nhận xét:
- + Có thể thấy rằng với language model LSI đưa ra được các kết quả cao hơn là Doc2Vec. Sự kết hợp tốt nhất là LSI + XGB (trong paper là LSI + SVM) (**Bảng 2**)
  + Độ chính xác và độ nhạy cao, có thể phát hiện được nhiều malware đúng mà không bị báo động giả quá nhiều (*Hình 20* và *Hình 21*)
  + Các kết quả trong quá trình học khá ấn tượng dù với dataset tự thu thập và tạo theo script của tác giả
  + Về phần kiểm tra khả năng phát hiện trên tệp malware mới, chưa từng được training thì kết quả thu được không được cao như paper (trong paper đề cập đến detection rate new malware của LSI + SVM trung bình là 0.973) (***Bảng 3***).
  + Tệp unknown2 có timestamp mới hơn so với unknown1 cũng tương ứng với kết quả phát hiện thấp hơn hoàn toàn phù hợp với việc malware thay đổi và tiến hóa theo thời gian cũng như khả năng thích ứng của mô hình đã học còn hạn chế; cần được mở rộng thêm dataset huấn luyện theo thời gian để đưa ra những kết quả phát hiện tốt nhất
- Thực nghiệm phần xây dựng ứng dụng ta có được một ứng dụng chạy localhost có chức năng phân tích một file pe sample được upload lên xem nó có phải là mã độc hay không dựa trên các mô hình học máy đã được huấn luyện từ bài báo. Kết quả phân loại benign/malware của 5 mô hình được học có tính chính xác cao.

## D. HƯỚNG PHÁT TRIỂN

- Hướng phát triển:
  + Hiện tại, dataset được sử dụng có thể chưa đủ đa dạng và chưa thể hiện đúng môi trường thực tế. Việc mở rộng và cập nhật dataset liên tục sẽ giúp cải thiện độ chính xác của các mô hình phân loại. Điều này bao gồm thu thập thêm dữ liệu từ các nguồn khác nhau và bao gồm các mẫu mã độc mới nhất.
  + Tìm hiểu và áp dụng các mô hình học sâu hiện đại hơn cho xử lý ngôn ngữ tự nhiên như Transformer-based models (BERT, GPT) có thể mang lại hiệu quả cao hơn trong việc phát hiện mã độc.
  + Tối ưu hóa các mô hình để giảm thời gian huấn luyện và dự đoán, tiết kiệm tài nguyên tính toán.
  + Đánh giá và cải thiện tính bền vững của mô hình đối với các thay đổi và tiến hóa của mã độc theo thời gian.

- Tính ứng dụng của đề tài:
  + Tầm quan trọng trong bảo mật thông tin: Với sự gia tăng không ngừng của các cuộc tấn công mạng, việc phát hiện và ngăn chặn mã độc là một phần quan trọng trong chiến lược bảo mật thông tin của bất kỳ tổ chức nào.
  + Nâng cao hiệu quả phát hiện mã độc: Việc sử dụng các mô hình học máy tiên tiến có thể cải thiện độ chính xác và hiệu quả trong việc phát hiện mã độc so với các phương pháp truyền thống. Các mô hình này có thể được triển khai trong các hệ thống bảo mật hiện tại, như tường lửa, hệ thống phát hiện xâm nhập (IDS), và các giải pháp antivirus để cung cấp một lớp bảo vệ toàn diện hơn.

**HẾT**