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: PE Windows Malware Detection

*Mã nhóm: **G04** Mã đề tài: **S17***

Lớp: NT230.Q11.ANTN

1. THÔNG TIN THÀNH VIÊN NHÓM:

(Sinh viên liệt kê tất cả các thành viên trong nhóm)

STT	Họ và tên	MSSV	Email
1	Nguyễn Ngọc Diệu Duyên	23520401	23520401@gm.uit.edu.vn
2	Đoàn Việt Khải	23520673	23520673@gm.uit.edu.vn
3	Nguyễn Hoàng Bảo Minh	23520938	23520938@gm.uit.edu.vn

2. TÓM TẮT NỘI DUNG THỰC HIỆN:[1]

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

☐ Phát hiện lỗ hổng bảo mật phần mềm

☐ Khai thác lỗ hổng bảo mật phần mềm

☐ Lập trình an toàn

☑ Khác: Phát hiện phần mềm độc hại PE Windows

B. Tên bài báo tham khảo chính:

[1] P. Feng *et al.*, "DawnGNN: Documentation augmented windows malware detection using graph neural network," *Computers & Security*, vol. 140, p. 103788, May 2024, doi: https://doi.org/10.1016/j.cose.2024.103788.

[1] Ghi nội dung tương ứng theo mô tả

C. Dịch tên Tiếng Việt cho bài báo:

DawnGNN: Phát hiện mã độc Windows tăng cường bằng Tài liệu API sử dụng Mạng nơ-ron Đồ thị

D. Tóm tắt nội dung chính:

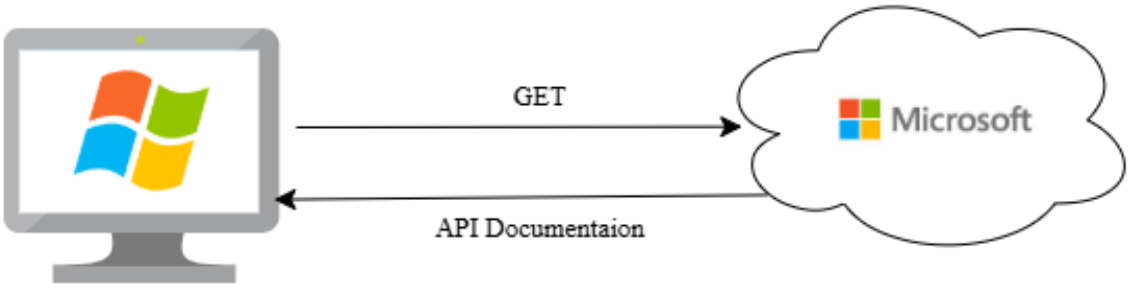
Trong suốt những thập kỉ qua, số lượng phần mềm độc hại đã tăng một cách nhanh chóng, đi kèm với sự phức tạp và tinh vi. Theo báo cáo của Sonic Wall năm 2023, 172146 mẫu chưa từng thấy đã được phát hiện trong một năm, hơn 956 biến thể mới trong một ngày. Các phương pháp phân tích tĩnh mã độc dễ dàng bị vượt qua bởi các phương pháp Làm rối mã (Code Obfuscation), Đóng gói (Packer), Biến đổi đa hình (Metamorphism), … Các phương pháp phân tích động trong những nghiên cứu gần đây chủ yếu tập trung đến tên API và tần suất được sử dụng. Nghiên cứu này đề xuất mô hình DawnGNN nhằm khai thác sâu các đặc trưng ngữ nghĩa của API như Phân loại (Category), Hành động (Action), Đối tượng thao tác (Operation Object), … Mục tiêu là trích xuất tài liệu để xây dựng Semantic Chain (Chuỗi ngữ nghĩa), sau đó nhúng các đặc trưng này vào đồ thị luồng gọi chương trình (API Call Graph) để thực hiện phân loại mã độc thông qua mạng nơ – ron đồ thị. Điểm đột phá của nghiên cứu nằm ở việc khai thác tài liệu chính thức về Windows API, một nguồn tài liệu chưa từng được sử dụng trong các nghiên cứu trước đây. Bằng cách sử dụng mô hình BERT đã được tiền huấn luyện, hệ thống tính toán và đưa ra các đặc trưng ngữ nghĩa tương ứng với từng Windows API. Chuỗi API trích xuất từ chương trình được chuyển thành đồ thị có hướng, thể hiện sự tương tác qua lại giữa các API. Các đặc trưng BERT trích xuất ở trên được nhúng vào đồ thị API, đưa vào mạng nơ-ron đồ thị Graph Attention Network – GAT để nắm bắt các thông tin then chốt, tập trung vào các hành vi cốt lõi của malware, và bỏ qua các API rác. Mô hình DawnGNN được thực nghiêm trên 3 bộ tập dữ liệu công khai gồm: MalBehavD-V1, PE_APICALLS, APIMDS đã chứng mình được hiệu suất của cơ chế

tăng cường ngữ nghĩa dựa trên BERT và mô hình đề suất trong phát hiện phần mềm độc hại trên Windows (Windows Malware).

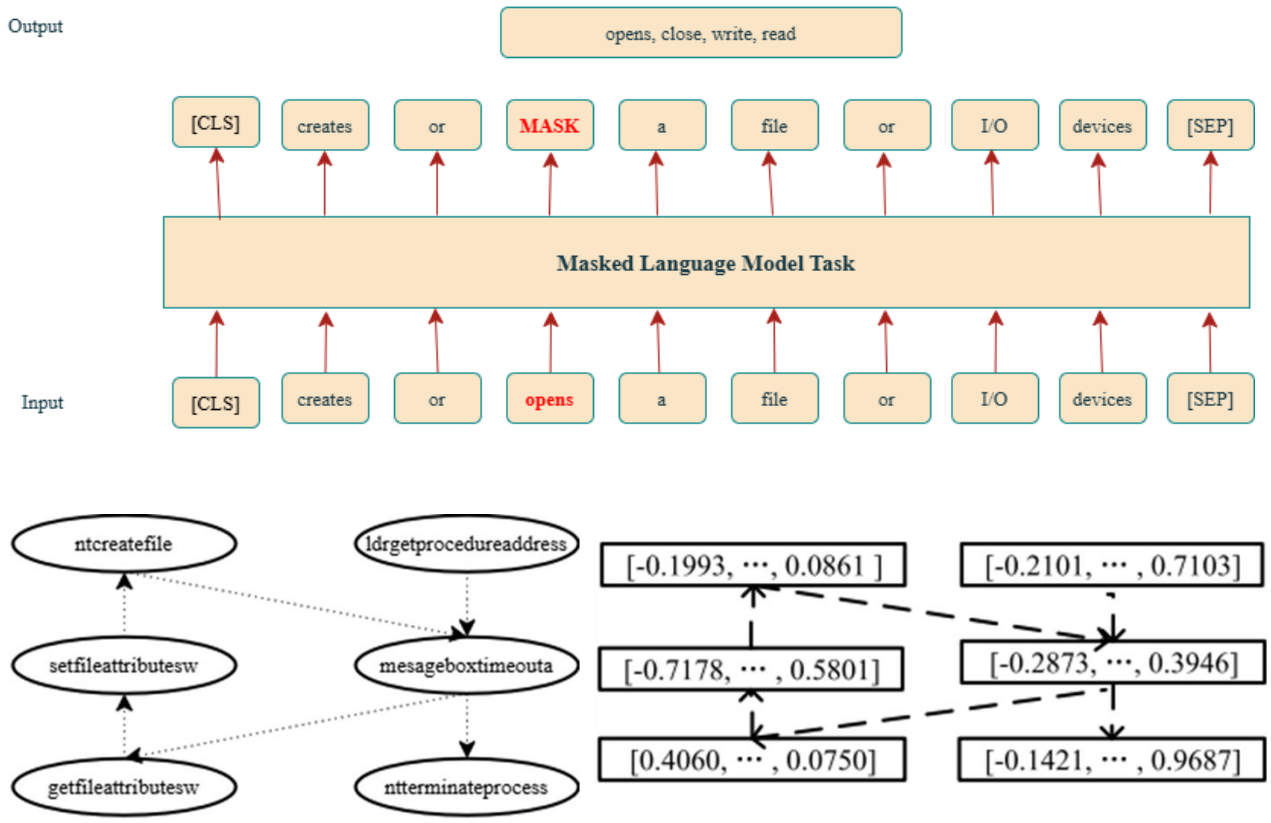
E. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong bài báo:

Bài báo sử dụng 3 kỹ thuật cho 3 giai đoạn chính trong phương pháp đề xuất, bao gồm:

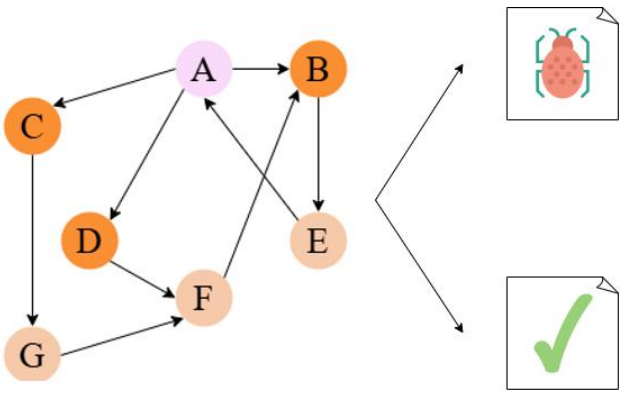
- **Kỹ thuật 01: API Sequence to Graph**
  - **Vai trò:** Biểu diễn chuỗi API thành đồ thị tương tác qua lại, thể hiện phụ thuộc cấu trúc (Stuctural Dependences) giữa các API.



  - **Nhiệm vụ:** Từ chuỗi lời gọi API của chương trình đã thu thập từ trước, xem thứ tự của chuỗi API như một biểu diễn quan hệ giữa các lời gọi. Nút của đồ thị tương ứng với các API không trùng lặp trong chuỗi, và cạnh tương ứng với mối quan hệ giữa hai API liền kề, có chiều từ API được gọi trước đến API được gọi sau.
- **Kỹ thuật 02: BERT**
  - **Vai trò:** Tạo ra API embeddings thông qua quá trình học ngữ nghĩa từ các mô tả của tài liệu API chính thức.
  - **Nhiệm vụ:** Tiền huấn luyện BERT bằng Mask-Language-Modeling (MLM Task) giúp BERT làm quen với mối quan hệ ngữ cảnh giữa các từ khác nhau trong tài liệu API. Huấn luyện chính thức trên dữ liệu về API đã crawl và xem trạng thái ẩn (Hidden State) của lớp cuối cùng (The last layer) là Embedding ngữ nghĩa của API tương ứng.



- **Kỹ thuật 03: GAT (Một loại mô hình đồ thị GNN)**
  - **Vai trò:** Trích xuất các thông tin cấu trúc từ đồ thị và thực hiện phân loại phần mềm độc hại Windows API.
  - **Nhiệm vụ:** Nhận vào đồ thị API đã được nhúng API Embedding tương ứng với từng nút, thông qua cơ chế truyền thông điệp (Message Pasing) và tập trung vào các nút quan trọng (Multi – head Attention). Từ dữ liệu nhận được của các nút láng giềng, cập nhật biểu diễn của nút đó và phân loại chương trình tương ứng thành phần mềm độc hại hay phần mềm vô hại.



F. Môi trường thực nghiệm của bài báo:

- Cấu hình máy tính: Intel(R) Core (TM) i7-12700 CPU @ 2.10 GHz, 16.0 GB RAM, NVIDIA RTX 3060 và 512 GB ổ cứng.
- Hệ điều hành: Ubuntu 20.04 (64-bit)
- Các công cụ hỗ trợ sẵn có:
  - Hugging Face Transformers: Tải và tinh chỉnh (Fine-tuning) các tham số của mô hình BERT – base – uncased và BERT – small.
  - Pytorch: Xây dựng vecto đặc trưng ứng với mỗi API và huấn luyện mạng nơron đồ thị (GNN/GAT).
  - PyTorch Geometric (PyG): Xây dựng và huấn luyện mạng nơ-ron đồ thị (GNN/ GAT).
  - Pandas & Numpy: Tính toán và đọc vào dữ liệu.
  - Scikit – learrn: Tính toán các chỉ số đánh giá (Metrics) và chạy mô hình Baseline.
- Ngôn ngữ lập trình để hiện thực phương pháp: Python
- Đối tượng nghiên cứu:
  - Mô hình Baseline: One-hot Encoding, Random Forest, Long Short-term Memory, Graph Convolutional Network, Graph Isomophism Network.
  - Tập dữ liệu được sử dụng:

Tập dữ liệu	Mẫu độc hại (Malicious)	Mẫu lành tính (Benign)	Ghi chú
MalBehavD-V1	1.285	1.285	Cân bằng
PE_APICALLS	452	101	Mất cân bằng nhẹ
APIMDS	23.080	300	Mất cân bằng nghiêm trọng

Tiêu chí đánh giá tính hiệu quả của phương pháp: Sử dụng 5 metrics

- Precision (Độ chuẩn xác): Tỉ lệ mẫu thực sự là mã độc trên Tổng số mẫu được mô hình dự đoán là mã độc.

- Recall (Độ phủ): Tỉ lệ mẫu được phát hiện thành công trên Tổng số mẫu là mã độc thực tế có trong tập dữ liệu.
- True negative rate (Tỷ lệ âm tính thật): Tỉ lệ mẫu bình thường được dự đoán đúng trên Tổng số mẫu bình thường thực tế.
- Accuracy (Độ chính xác): Tỉ lệ số mẫu được dự đoán đúng (Gồm cả Mẫu độc hại và Mẫu bình thường) trên Tổng số mẫu thực nghiệm.
- F1-score (Trung bình giữa Precision và Recall): Giá trị trung bình điều hòa giữa Precision và Recall, dùng để đánh giá năng lực tổng thể của mô hình.

G. Kết quả thực nghiệm của bài báo:

❖ **Tính toán độ hiệu quả của hai thành phần chính (BERT + GAT):**

- Tăng cường ngữ nghĩa bằng BERT cải thiện hiệu quả phát hiện phần mềm độc hại, với API Embeddings càng lớn, càng chứa được nhiều thông tin ngữ nghĩa chính xác hơn.

Table III
Detection performance comparison with different detection methods in dataset MalBehavD-V1.

Detection method	Precision	Recall	TNR	Acc	F1
one-hot + RF	0.9009	0.9494	0.8908	0.9203	0.9231
one-hot + LSTM	0.9150	0.9469	0.9106	0.9283	0.9295
one-hot + GAT	0.9195	0.9586	0.9074	0.9339	0.9368
Word2Vec + LSTM	0.9323	0.9661	0.9277	0.9441	0.9475
Word2Vec + GAT	0.9559	0.9532	0.9436	0.9432	0.9543
$BERT_{small}$ + LSTM	0.9609	0.9584	0.9512	0.9508	0.9595
$BERT_{small}$ + GAT	0.9667	0.9756	0.9527	0.9607	0.9683
$BERT_{base}$ + LSTM	0.9632	0.9615	0.9553	0.9535	0.9618
$BERT_{base}$ + GAT	0.9697	0.9788	0.9556	0.9638	0.9711

The $BERT_{small}$ and $BERT_{base}$ represent *different versions* `small` and `base` of the pre-trained BERT encoding mechanism.

❖ **So sánh các thuật toán GNN (GIN, GCN):**

- GAT với cơ chế chú ý đa đầu (Multi – head Attention) cho phép mô hình tổng hợp thông tin từ các nút hàng xóm và tập trung vào tương tác của các API nhạy cảm, từ đó biểu hiện vượt trội hơn GCN và GIN.

Table VI
The detection performance comparison with three GNN models.

Algorithms	Precision	Recall	TNR	Acc	F1
GCN	0.9401	0.9666	0.9388	0.9406	0.9524
GIN	0.9506	0.9701	0.9469	0.9509	0.9603
GAT	**0.9697**	**0.9788**	**0.9556**	**0.9638**	**0.9711**

❖ **So sánh với các hướng tiếp cận khác:**

➢ So sánh DawnGNN với các phương pháp phát hiện mã độc Windows để xác thực sự hiểu quả của giải pháp đề xuất.

Table VII
Detection performance comparison with existing approaches.

Dataset	Approaches	Behavior Feature	Feature Vectorization Method	ML/DL Algorithm	Detection Accuracy
MalBehavD-V1	MalDy [36]	Behavior reports from analysis sandbox	N-grams + Feature Hashing + TF-IDF	Ensemble learning	0.9559
	MalDetConv [26]	API call sequences	Word2vec	Hybrid model CNN-BiGRU	0.9610
	DawnGNN	API call sequences + API documentation	BERT-based semantic enhancement	GAT	**0.9638**
PE_APICALLS	MalDy [36]	Behavior reports from analysis sandbox	N-grams + Feature Hashing + TF-IDF	Ensemble learning	0.9518
	MalDetConv [26]	API call sequences	Word2vec	Hybrid model CNN-BiGRU	0.9573
	DawnGNN	API call sequences + API documentation	BERT-based semantic enhancement	GAT	**0.9762**
APIMDS	Amer and Zelinka [7]	API call sequences	Word2Vec + clustering similarity	Markov chain model	0.9990
	Ki et al. [32]	API call sequences	DNA sequence alignment	Similarity matching	0.9980
	Tran and Sato [37]	API call sequences	TF-IDF	SVM	0.9619
	MalDetConv [26]	API call sequences	Word2vec	Hybrid model CNN-BiGRU	0.9993
	DawnGNN	API call sequences + API documentation	BERT-based semantic enhancement	GAT	0.9975

❖ **Khả năng của giải pháp:** Mô hình có khả năng cập nhật thông tin định kỳ thông qua Crawl tài liệu của Microsoft, tiền xử lý và huấn luyện lại BERT. Từ đó, giải quyết bài toán thiếu hụt ngữ nghĩa mà các phương pháp cũ đang gặp phải, đồng thời có khả năng phân biệt tốt hơn các mã độc mới (Zero – day).

❖ **Ưu điểm:**

✓ Định nghĩa các API, phiên bản, tình trạng sử dụng từ Microsoft liên tục được cập nhật.

✓ Không gánh nặng tính toán ở giai đoạn dự đoán.

✓ Trích xuất ra được mối quan hệ và sự tác động qua lại giữa các lời gọi API.

✓ Độ chính xác cao nhờ kết hợp trích xuất ngữ nghĩa các API (BERT) và mạng nơ – ron đồ thị (GAT).

❖ **Nhược điểm:**

- ✓ Không hiệu quả với các mã độc sử dụng kỹ thuật trốn tránh (Evasion techniques) như Anti-Sandbox, Anti-VM.
- ✓ Attacker đổi tên hoặc sử dụng API không được định nghĩa trong tài liệu chính thống.
- ✓ Cần bypass được cơ chế chống lấy cắp dữ liệu (Crawl) của trang web Microsoft từ đó cập nhật các API Embedding.
- ✓ Chương trình gọi các API rác với một tần suất cao có thể làm ảnh hưởng đến khả năng phân loại của GAT.

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

❖ Các công việc nhóm đã thực hiện cho đề tài:

- ➢ Xây dựng Dictionary cho API: Crawl dữ liệu Microsoft chính thức và làm sạch thành công 37100 API (Tên và Định nghĩa tương ứng).
- ➢ Tiền huấn luyện BERT: Sử dụng Masked Language Model Task.
- ➢ Huấn luyện BERT chính thức: Trích xuất lớp ẩn cuối cùng (Hidden Layer) trở thành Vector đặc trưng tương ứng với API.
- ➢ Tiền xử lý dữ liệu: Phân tích 3 bộ dataset tiến hành thực (Phân bổ các lớp) và Làm sạch (Xóa trùng, Dòng/ Cột trống).
- ➢ Tiến hành thực nghiệm 01: Tái hiện luồng chạy chính của tác giả với 3 bộ dữ liệu (MalBehavD-V1, PE_APICALLS, APIMDS) trên 2 mô hình BERT-based-uncased và BERT-small.
- ➢ Tiến hành thực nghiệm 02: Chứng minh sự ưu việt của DawnGNN so với các mô hình đơn/ kết hợp GCN, GIN, WORD2VEC + GAT, BERTbase + LSTM.
- ➢ Tiến hành thực nghiệm 03: Kiểm tra hiệu quả của mô hình trên Dataset mới Malware Analysis Datasets: API Call Sequences.

❖ Kết quả của công việc:

- Về ngữ nghĩa của API: Hoàn thành bộ nhúng ngữ nghĩa (Semantic Embedding) cho 37.100 API, giúp mô hình hiểu bản chất hành vi thay vì chỉ tên hay tần suất.
- Về khả năng phân loại và hiệu suất: DawnGNN đạt kết quả cao nhất về Recall và F1-score, chứng minh khả năng nhận diện chương trình độc hại.
- Về tính tổng quát và ổn đinh: Khả năng phân loại chương trình độc hại hiệu quả trên tập dữ liệu mới.

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

Xây dựng toàn bộ mô hình gồm Mã nguồn và Quy trình từ đầu.

Việc xử lý và làm sạch 37100 API từ tại liệu chính thức được xử lý bằng tay 80%.

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

Nhóm đã hoàn thành 16/16 nhiệm vụ

- Hiểu sâu toàn bộ phương pháp và kiến trúc kết hợp nhiều mô hình
- Thực nghiệm được flow làm việc chính theo paper
- Thực nghiệm bổ sung để so sánh và chứng minh tính vượt trội của BERT+GAT so với các baseline khác
- Thực nghiệm mở rộng: làm việc trên dataset mới hoàn toàn để tính minh khả năng tổng quát hóa

Mức độ tự đánh giá: 100%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Đọc hiểu paper	Cả ba thành viên
2	Tìm hiểu ngữ cảnh, hạn chế của các phương pháp kiểm thử đương thời	Nguyễn Hoàng Bảo Minh

3	Nghiên cứu sâu cơ chế BERT	Nguyễn Ngọc Diệu Duyên
4	Nghiên cứu sâu cơ chế GAT+GNN	Đoàn Việt Khải
5	Đặt vấn đề về những điều chưa chặt chẽ ở quy trình BERT – GAT - GNN	Đoàn Việt Khải
6	Tìm hiểu, phân tích và đưa ra toàn bộ quy trình BERT – GAT - GNN một cách toàn vẹn nhất (scope, mục tiêu & cơ chế từng bước, các vấn đề xung đột,…)	Nguyễn Ngọc Diệu Duyên
7	Phân tích những PaintPoint (Packer, Obfuscation) để làm bật lên giá trị của phương pháp DawnGNN.	Nguyễn Ngọc Diệu Duyên Nguyễn Hoàng Bảo Minh
8	Hiện thực hóa API Graph Constructor: Trích xuất mối quan hệ từ chuỗi API có sẵn trong Dataset để xây dựng Ma trận kề (Adjacency Matrix).	Đoàn Việt Khải
9	Chuẩn bị tài nguyên + set up thực nghiệm	Nguyễn Hoàng Bảo Minh
10	Phân tích bốn bộ dataset (cấu trúc, đặc điểm, cách xử lí dữ liệu, phân phối dữ liệu)	Đoàn Việt Khải
11	Làm sạch dữ liệu văn bản (Text Cleaning) cho mô tả API trước khi đưa vào mô hình BERT	Nguyễn Ngọc Diệu Duyên Nguyễn Hoàng Bảo Minh
12	Thực nghiệm 1: Tái hiện flow của Paper và kiểm chứng sự chênh lệch hiệu suất giữa BERTbase (768d) và BERTsmall (512d).	Nguyễn Ngọc Diệu Duyên
13	Thực nghiệm 2: - Vấn đề 1: So sánh sự ưu việt của GAT so với các kiến trúc đồ thị khác như GCN, GIN. - Vấn đề 2: Triển khai các thực nghiệm so sánh với: GCN, GIN, WORD2VEC + GAT, BERTbase + LSTM	- Vấn đề 1: Nguyễn Ngọc Diệu Duyên - Vấn đề 2: Đoàn Việt Khải Nguyễn Hoàng Bảo Minh
14	Thực nghiệm 3: Thực hiện thực nghiệm với Dataset mới từ GitHub để kiểm chứng tính tổng quát của mô hình trên các dữ liệu lạ.	Cả ba thành viên

15	Phân tích và so sánh các kết quả. Lí giải các chỉ số Precision (Độ chuẩn xác), Recall (Độ phủ), True negative rate (Tỷ lệ âm tính thật), Accuracy (Độ chính xác), F1-score (Trung bình giữa Precision và Recall)	Cả ba thành viên
16	Làm slide, thuyết trình, viết báo cáo chi tiết, làm poster	Cả ba thành viên

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 DawnGNN được tác giả đưa ra gồm 4 thành phần chính:
  - Bộ phận xây dựng đồ thị API (API Graph Constructor): Từ chuỗi lời gọi API trong các Dataset có sẵn, chuyển đổi thành đồ thị có hướng g = {V, D}. Trong đó, nút (V) là các API duy nhất và cạnh (D) thể hiện trình tự gọi nhau.
  - Lớp vector nhúng API (API2Vec embedding layer):
    - ✓ Thu thập tài liệu về 32763 Windows API từ Microsoft.
    - ✓ Tiền huấn luyện mô hình BERT với nhiệm vụ Masked Language Model để hiểu sâu ngữ cảnh của dữ liệu API Call.
    - ✓ Huấn luyện BERT chính thức và lấy ra trạng thái ẩn của lớp cuối cùng làm vector nhúng ngữ nghĩa (Semantic Embedding).
  - Bộ phận phân loại mạng nơ – ron đồ thị (GNN classifier):
    - ✓ Sử dụng mô hình GAT (Cơ chế Multi – head Attention) tổng hợp thông tin đồ thị và ngữ nghĩa BERT thành một vector đại diện. Sau đó, sử dụng Multi – layer Perception (MLP) phân loại nhãn: Độc hại hoặc Bình thường.
- Nhóm đã hiện thực hóa toàn bộ quy trình được trình bày trong bài báo:
  - Bước 01: Thu thập tài liệu mô tả API
    - ✓ Sử dụng thư viện BeautifulSoup để crawl đường dẫn tới các thư viện API.
    - ✓ Truy cập sâu vào từng địa chỉ, sử dụng thư viện Selenium WebDriver để crawl dữ liệu API và Xpath để bóc tách chính xác tên và mô tả API.
  - Bước 02: Huấn luyện BERT với MLM Task và Huấn luyện chính thức:
    - ✓ Tiền xử lý dữ liệu: Loại bỏ đi các từ thừa thải trong mô tả (Tên API, Viết tắt, Chú thích).

CopyFileW

The CopyFileW (Unicode) function (winbase.h) copies an existing file to a new file.

~~The CopyFileW~~ ~~(Unicode)~~ ~~function~~ ~~(winbase.h)~~ **copies an existing file to a new file**

- ✓ Token hóa dữ liệu với AutoTokenizer, chia khối dữ liệu (Block size: 128).
- ✓ Thực hiện tiền huấn luyện với Nhiệm vụ MLM, xác xuất dữ liệu API được dùng là 15%, trong đó 80% từ được che bởi tag [MASK], 10% thay bằng giá trị ngẫu nhiên khác và 10% giữ không đổi. Thông qua quá trình dự đoán các giá trị bị che giấu, tinh chỉnh các tham số bên trong, làm việc hiệu quả hơn với ngữ cảnh liên quan đến API.
- ✓ Huấn luyện chính thức trên dữ liệu API và trích xuất lớp ẩn cuối cùng (Last Hidden Layer) đại diện cho ngữ nghĩa của từng API ở dạng vector 768 chiều (Mô hình BERT-base-uncased) và 512 chiều (Mô hình BERT -small).

➢ Bước 03: Xây dựng đồ thị API

- ✓ Duyệt qua từng chuỗi API trong các Dataset để xác định các cặp API được gọi kế tiếp nhau.
- ✓ Tính toán trọng số cạnh: Sử dụng Counter để đếm tần suất xuất hiện của mỗi cặp – sự chuyển đổi API.
- ✓ Mỗi API duy nhất là một nút (Node) và gán ID, đưa vào Tensor.
- ✓ Ánh xạ vector nhúng (API Embedding) được chuẩn bị từ trước vào từng nút tương ứng trên đồ thị.
- ✓ Đóng gói tất cả thông số về cấu trúc liên kết, trọng số, thuộc tính nút vào Data của thư viện PyTorch Geometric.

➢ Bước 04: Huấn luyện với mô hình phân loại DawnGNN

- ✓ Sử dụng GATConv với hàm kích hoạt ELU, mô hình học mối quan hệ giữa các API lân cận theo cơ chế Multi – head Attention và tổng hợp thông tin.
- ✓ Sử dụng Global Add Pooling để cộng tổng các vector từ tất cả các nút thành vector duy nhất đại điện cho toàn bộ chương trình.

- ✓ Sử dụng Perceptron đa lớp (Multi – layer Perceptron) để phân loại Chương trình độc hại (Xác suất > 0.5) hoặc Chương trình bình thường.

B. Chi tiết cài đặt, hiện thực

- ❖ Cấu hình phần cứng, môi trường và công cụ phần mềm:

	Phần cứng	**Phần mềm**
Môi trường 01: Crawl API	Hệ điều hành: Ubuntu 20.04 LTS. GPU: NVIDIA GeForce RTX 3060 Ti. VRAM: 8GB CPU: 12 Cores	Selenium: 4.38.0 Requests: 2.32.5 BeautifulSoup: 4.13.5 LXML version: 6.0. 2.0 Pandas version: 2.3.2
Môi trường 02: Huấn luyện BERT + DawnGNN	Hệ điều hành: Window 11 RAM: 16GB Ổ cứng: 1TB SSD VGA: Nvidia Geforce RTX 4050 6GB GDDR6 + MUX switch	Torch: 2.6.0 Torchvision: 0.21.0 Torch_geometric: 2.7.0 Transformers: 4.57.1 Gensim: 4.4.0 Numpy: 2.3.3 Pandas: 2.3.3 Scikit-learn: 1.7.2 Tqdm: 4.67.1 Matplotlib: 3.10.7 Seaborn: 0.13.2 Datasets: 4.4.1

- ❖ Ngôn ngữ lập trình: Python 3.12.6
- ❖ Chuẩn bị dữ liệu: Dữ liệu cho quá trình phân loại gồm 4 bộ dataset
  - ➢ MalBehavD-V1:
    - ✓ Cấu trúc: dataset gồm 177 cột
      - Cột đầu tiên: chứa mã hash của ứng dụng
      - Cột labels: chứa nhãn phân loại của ứng dụng, nếu ứng dụng "clean" thì gán nhãn "0", nếu ứng dụng là malware thì gán nhãn "1"
      - Cột 0 đến Unnamed 174: chứa các APIs mà ứng dụng sử dụng, mỗi cột chứa một API

sha256	labels	0	1	2	....	Unnamed 173	Unnamed 174
5c18291c481a192ed5...	0	LdrUnloadDll	RegCloseKey	NtOpenSection	.....	NaN	NaN
4683faf3da550ffb594...	0	NtOpenMutant	NtOpenSection	CoUninitialize	.....	NaN	NaN
9a0aea1c7290031d7c...	0	GetForegroundWindow	LoadStringW	GetFileType	.....	NaN	NaN
.....	.....	.....	.....	.....	.....	.....	.....
e0f3e4d5f50afd9c31e...	1	CreateToolhelp32Snapshot	NtOpenSection	CreateThread	.....	NaN	NaN
ec2b6d29992f13e740...	1	CreateToolhelp32Snapshot	NtOpenSection	CreateThread	.....	NaN	NaN

    - ✓ Kích thước Dataset: Dataset chứa 2570 dòng và 177 cột (tiến hành xem bằng lệnh shape trong thư viện pandas của python)

```
print("Original Dataset size:", df.shape)
✓ 0.0s
Original Dataset size: (2570, 177)
```

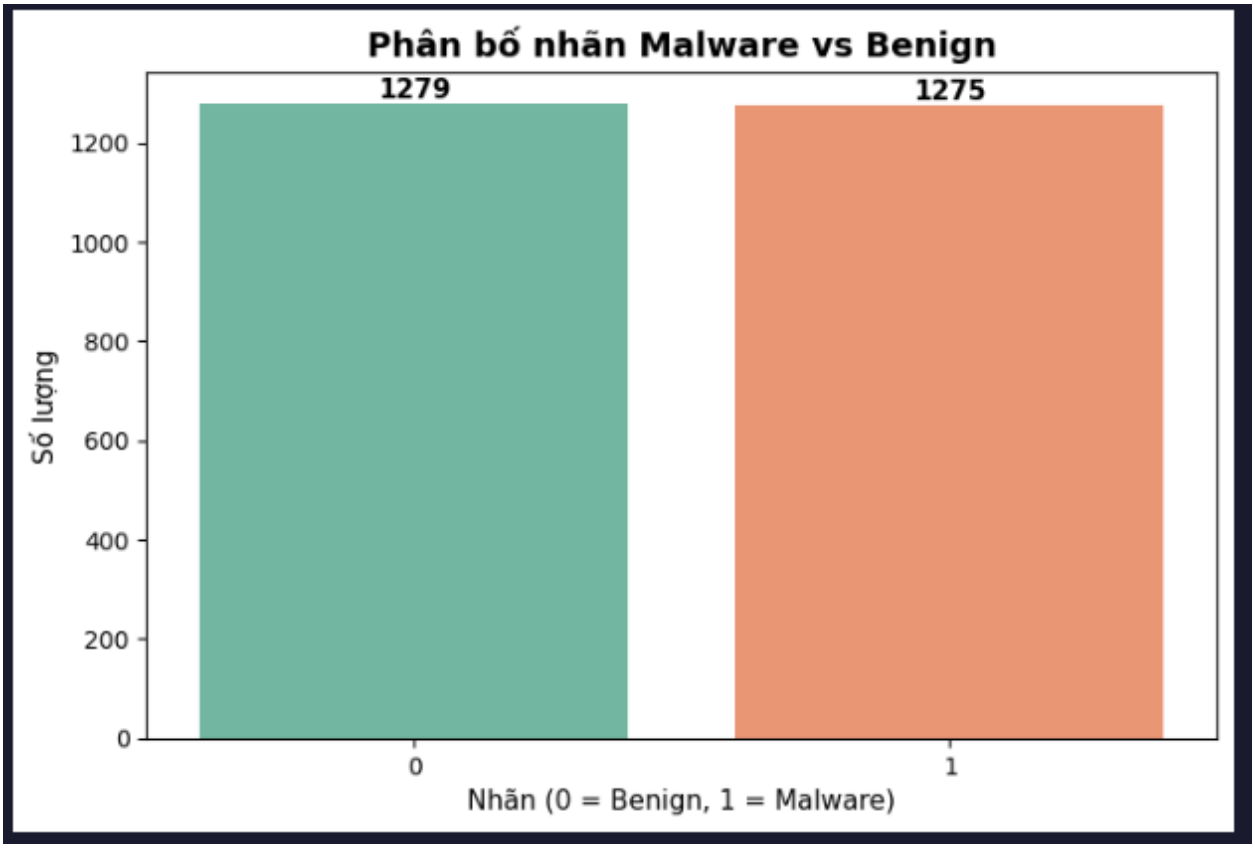
    - ✓ Đặc điểm:
      - Dataset có những cột unnamed, đó là vì có nhiều ứng dụng sử dụng nhiều API trong khi một số khác lại sử dụng ít API hơn, vì mỗi API nằm một cột nên dư ra nhiều cột unnamed như vậy (ví dụ ứng dụng 1 sử dụng 100 API, nhưng ứng dụng 2 sử dụng 120 API thì sẽ có 20 cột unnamed). Vì đặc tính của python, những giá trị bị thiếu trong cột unnamed sẽ có dạng là NaN, khi processing cần xóa đi
      - Dataset cũng có một số dòng bị trùng lặp như các dòng tô đỏ, ta cũng cần xóa trước khi đưa vào huấn luyện

sha256	labels	0	1	2	....	Unnamed 173	Unnamed 174
5c18291c481a192ed5...	0	LdrUnloadDll	RegCloseKey	NtOpenSection	.....	NaN	NaN
4683faf3da550ffb594...	0	NtOpenMutant	NtOpenSection	CoUninitialize	.....	NaN	NaN
9a0aea1c7290031d7c...	0	GetForegroundWindow	LoadStringW	GetFileType	.....	NaN	NaN
.....	.....	.....	.....	.....	.....	.....	.....
e0f3e4d5f50afd9c31e...	1	CreateToolhelp32Snapshot	NtOpenSection	CreateThread	.....	NaN	NaN
ec2b6d29992f13e740...	1	CreateToolhelp32Snapshot	NtOpenSection	CreateThread	.....	NaN	NaN

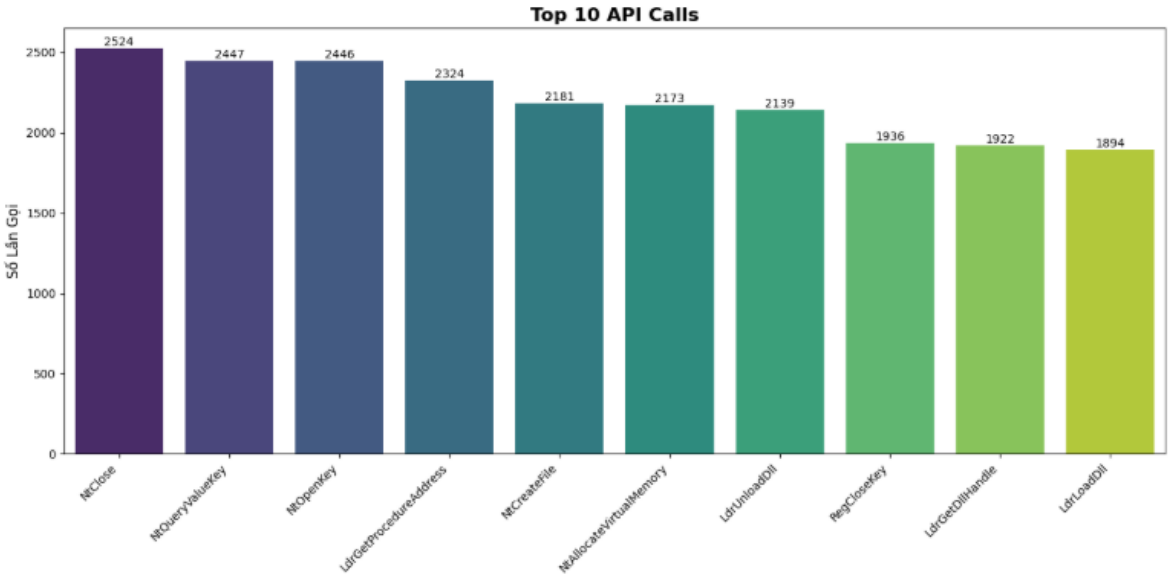
      - Mặc dù các dòng này đến từng hai ứng dụng khác nhau (mã hash khác nhau), nhưng dữ liệu từ cột hash không có lợi cho quá trình huấn luyện, vì vậy ta cũng xóa cột hash đi.
      - Sau khi xử lí, kích thước thật của dataset

```
df.drop_duplicates(inplace=True)
print("Dataset size after removing duplicates:", df.shape)
✓ 0.0s
Dataset size after removing duplicates: (2554, 177)
```

✓ Phân phối dữ liệu:



- Nhìn chung đây là dataset cân bằng, dữ liệu từ hai nhãn tương đương nhau, dẫn đến model học tốt, không bị thiên vị class nào.
- Top 10 API phổ biến (được các ứng dụng benign lẫn malware dùng nhiều nhất)



- Các ứng dụng dù benign hay malware cũng đều sử dụng các API này, dẫn đến một điều, các ứng dụng dù là malware cũng sử dụng những API phổ biến mà ứng dụng benign gọi tới, để tăng khả năng "bình thường hóa" trước các công cụ anti-virus. Đây là một ưu điểm của dataset này, đánh giá rất cao hiệu quả của model khi xem thử "có thể phân biệt giữa một benign thuần và một malware đang cố gắng giả thành benign không"

➢ PE_APICALLS:

✓ Cấu trúc: dataset gồm hai cột là

- Malware: chứa phân loại của một ứng dụng, nếu ứng dụng đó "clean" thì nằm trong nhóm "benign"; nếu ứng dụng đó là malware thì chỉ ra phân loại malware đó (ví dụ backdoor, virus, worm,…)
- APIcalls: chứa những chuỗi API mà ứng dụng sử dụng

Malware	APIcalls
Backdoor	SetUnhandledExceptionFilter, GetSystemTimeAsFileTime, NtDelayExecution, GetSystemTimeAsFileTime, NtDelayExecution, GetSystemTimeAsFileTime,...
benign	LdrGetDllHandle, NtTerminateProcess, NtClose, LdrGetDllHandle, NtClose, NtClose, NtClose, NtClose, NtClose,...
Trojan	NtOpenFile, NtCreateSection, NtClose, LdrLoadDll, LdrGetProcedureAddress, LdrLoadDll, LdrGetProcedureAddress, LdrGetDllHandle,...
Virus	LdrLoadDll, LdrGetProcedureAddress, LdrGetProcedureAddress, LdrGetProcedureAddress, LdrGetProcedureAddress, LdrGetProcedureAddress,...
Worm	NtProtectVirtualMemory, SHGetFolderPathW, NtQueryAttributesFile, GetSystemTimeAsFileTime, NtDelayExecution, MoveFileWithProgressW,...
Backdoor	GetSystemTimeAsFileTime, NtDelayExecution, GetSystemTimeAsFileTime, NtDelayExecution, GetSystemTimeAsFileTime, NtDelayExecution,...
Worm	NtProtectVirtualMemory, SHGetFolderPathW, NtQueryAttributesFile, GetSystemTimeAsFileTime, NtDelayExecution, MoveFileWithProgressW,...

✓ Kích thước: Dataset có 552 dòng và 2 cột (xem bằng lệnh shape trong thư viện pandas của python)

```
print("Original dataset size:", df.shape)
]  ✓ 0.1s
Original dataset size: (552, 2)
```

✓ Đặc điểm:

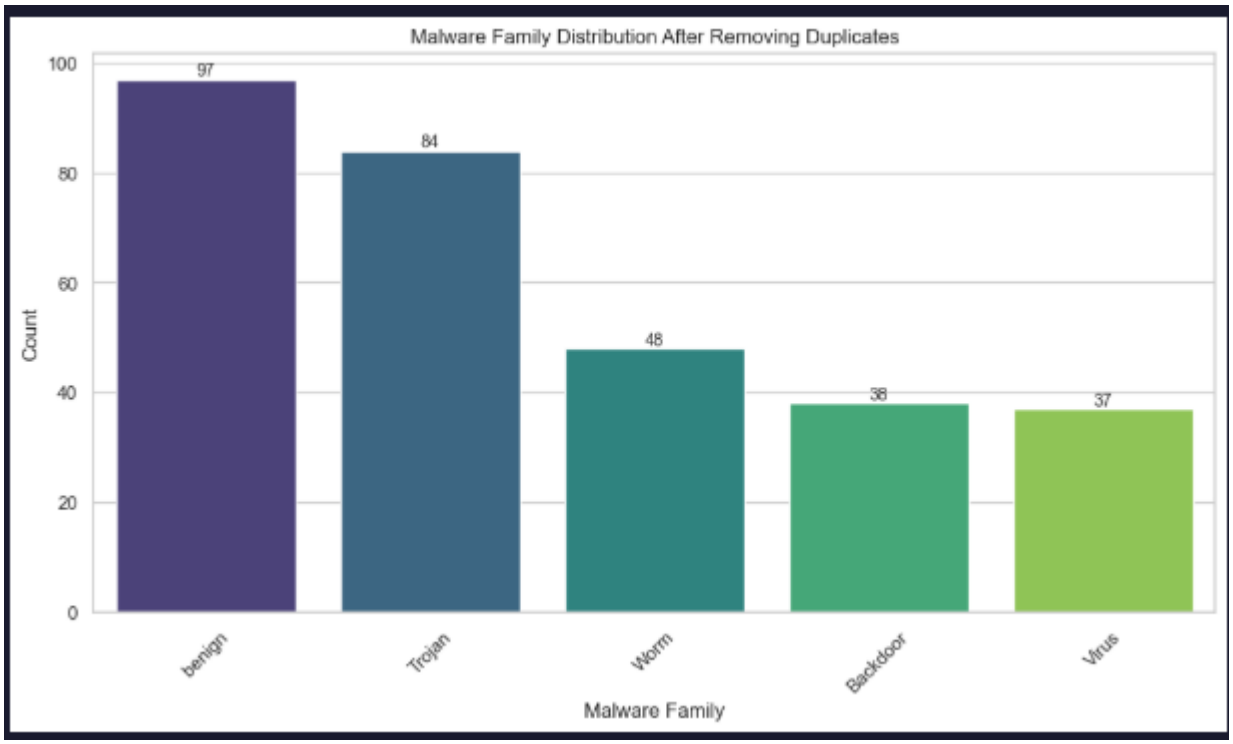
- Dataset có rất nhiều dòng bị trùng lặp (duplicate): điển hình như hai dòng tô đỏ

Malware	APIcalls
Backdoor	SetUnhandledExceptionFilter, GetSystemTimeAsFileTime, NtDelayExecution, GetSystemTimeAsFileTime, NtDelayExecution, GetSystemTimeAsFileTime,...
benign	LdrGetDllHandle, NtTerminateProcess, NtClose, LdrGetDllHandle, NtClose, NtClose, NtClose, NtClose, NtClose,...
Trojan	NtOpenFile, NtCreateSection, NtClose, LdrLoadDll, LdrGetProcedureAddress, LdrLoadDll, LdrGetProcedureAddress, LdrGetDllHandle,...
Virus	LdrLoadDll, LdrGetProcedureAddress, LdrGetProcedureAddress, LdrGetProcedureAddress, LdrGetProcedureAddress, LdrGetProcedureAddress,...
Worm	NtProtectVirtualMemory, SHGetFolderPathW, NtQueryAttributesFile, GetSystemTimeAsFileTime, NtDelayExecution, MoveFileWithProgressW,...
Backdoor	GetSystemTimeAsFileTime, NtDelayExecution, GetSystemTimeAsFileTime, NtDelayExecution, GetSystemTimeAsFileTime, NtDelayExecution,...
Worm	NtProtectVirtualMemory, SHGetFolderPathW, NtQueryAttributesFile, GetSystemTimeAsFileTime, NtDelayExecution, MoveFileWithProgressW,...

- Đòi hỏi trong quá trình xử lí cần loại bỏ những duplicate này.
- Kích thước thật sau khi loại bỏ duplicate: gồm 304 dòng và 2 cột

```
df.drop_duplicates(inplace=True)
df.shape
```
✓ 0.0s
```
(304, 2)
```

✓ Phân phối dữ liệu:



- Nhìn chung phân phối các class khá đều nhau, không có class nào chiếm số lượng quá nhiều. Vì vậy dataset này rất phù hợp để huấn luyện và phân loại malware.

➢ APIMDS:

✓ **Cấu trúc:** dataset gồm 597 cột, tương tự bộ MalBehavD-V1

- Cột labels: chứa nhãn phân loại của ứng dụng, nếu ứng dụng "clean" thì gán nhãn "0", nếu ứng dụng là malware thì gán nhãn "1"
- Cột hash: chứa mã hash của ứng dụng
- Cột 0 đến Unnamed 594: chứa các APIs mà ứng dụng sử dụng, mỗi cột chứa một API

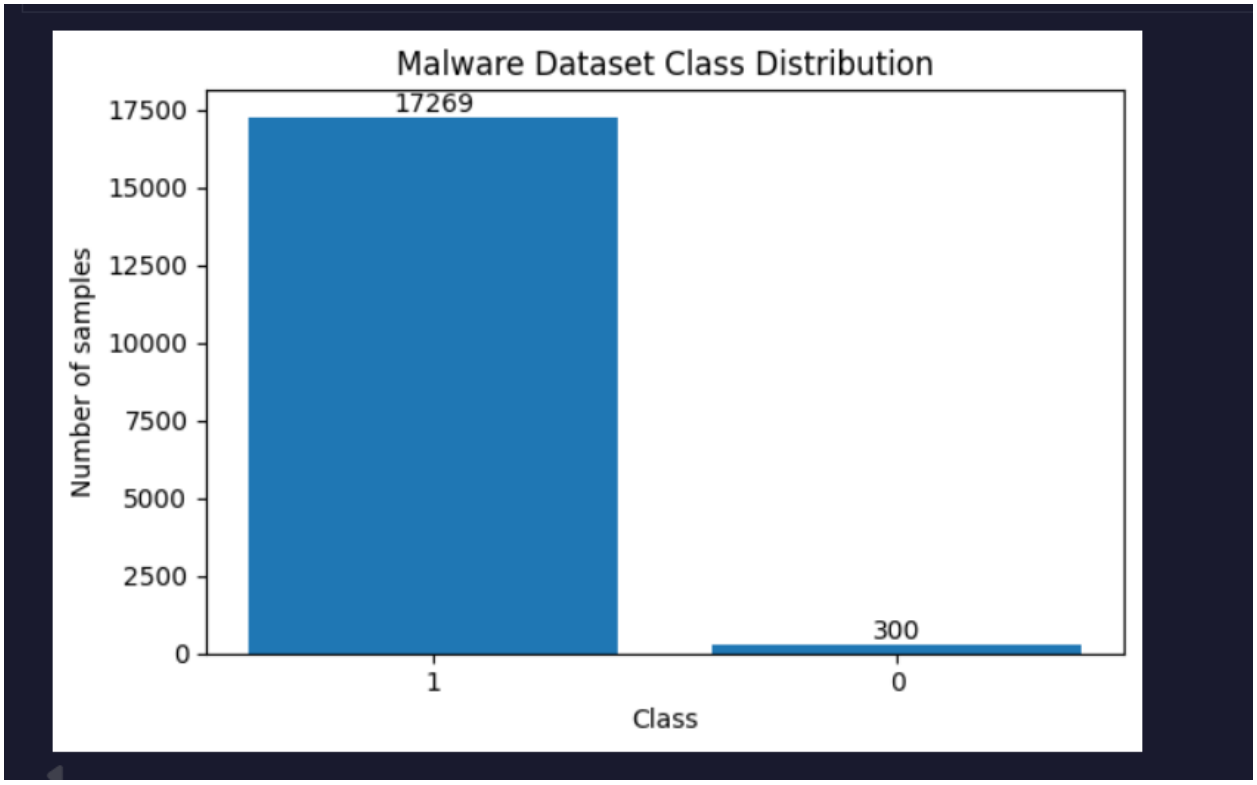
label	hash	0	1	...	594
1	034691b9a0558842....	CreateEventW	GetCommandLineA	...	...
1	0f74250f63874840...	CreateEventW	DisableThreadLibraryCalls	...	...
...	...	...	...	...	...
0	1772d592e6115cc84f....	RegCloseKey	RegQueryValueExW	...	...

✓ **Kích thước:** Dataset chưa 17569 dòng và 597 cột (tiến hành xem bằng lệnh shape trong thư viện pandas của python)

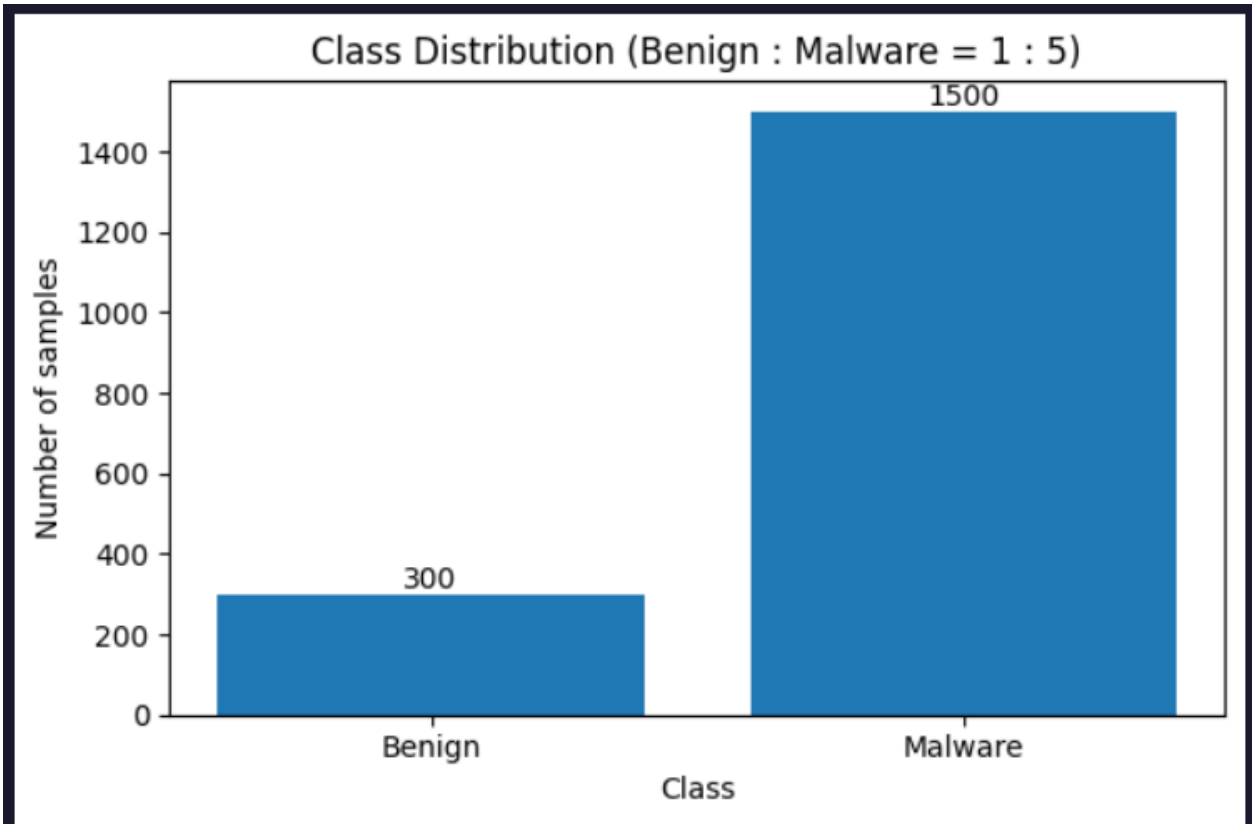
```
print("Original Dataset size:", df.shape)
✓ 1.2s
Original Dataset size: (17569, 597)
```

✓ **Đặc điểm:**
- Nhìn chung thì dataset không có dòng bị trùng
- Cần xóa cột hash vì nó là đặc điểm không có lợi trong quá trình huấn luyện
- Cái khó khăn trong thu thập dataset đó là yêu cầu chạy các ứng dụng để trích xuất API, nhưng nhóm đã tìm được một repository chứa các API đã trích xuất này, nên tận dụng luôn
- Repository: Link

✓ **Phân phối dữ liệu:**



- Vấn đề lớn nhất của bộ dataset này là dữ liệu quá mất cân bằng (malware gấp ~55 lần so với benign), nếu không xử lí mà dùng luôn để huấn luyện thì dẫn đến model học quá nhiều kiến thức từ malware, dẫn đến xu hướng thiên vị:
- ***"Khi đưa vào một app cần dự đoán thì dễ gán nhãn là malware, vì tỉ lệ malware chiếm đa số, nên model lựa chọn cách học an toàn".***
- Dẫn đến sai bản chất của quá trình huấn luyện và gán nhãn.
- Vì vậy, nhóm sẽ chia thành các subset, mỗi subset theo tỉ lệ 1: 5 như sau:
  - Benign chiếm một phần (khoảng 300 samples)
  - Malware chiếm năm phần (khoảng 1500 samples)
  - Đặt seed_random = 42 để mỗi lần random đều cho về cùng một kết quả
- Phân phối lại dữ liệu khi đưa vào huấn luyện:



- Lúc này, dataset nhìn có vẻ cân bằng hơn, đã có thể dùng huấn luyện.

➢ Malware Analysis Datasets: API Call Sequences

✓ **Cấu trúc:** Dataset gồm 102 cột như sau

- Cột hash: chứa mã hash của ứng dụng
- Cột malware: gán nhãn ứng dụng là "0" nếu nó là benign, gán nhãn là "1" nếu nó làm malware
- Cột từ t0 đến t99: chứa các API mà ứng dụng gọi tới (ở dạng số nguyên), mỗi API là một cột

hash	t0	t1	...	t99	Malware
071e8c3f8922e18...	112	274	...	35	1
33f8e6d08a6aae9...	82	208	...	112	1
....	...	....	...	...	...
654139d715abcf7...	82	280	...	141	1
078c9d4e7be4819...	112	274	...	71	1

✓ **Kích thước:** Dataset chưa 43876 dòng và 102 cột (tiến hành xem bằng lệnh shape trong thư viện pandas của python)

```
data = pd.read_csv("dynamic_api_call_sequence_per_malware_100_0_306.csv")
print("Original data size:", data.shape)

✓ 0.3s
Original data size: (43876, 102)
```

✓ **Đặc điểm:**

- Dataset này đang bị "số nguyên hóa" API, dẫn đến từ số nguyên cần phải trích xuất được tên của API đó
- Tiến hành tìm hiểu dataset trên IEEE thì phát hiện mô tả API như sau:

INSTRUCTIONS:

* FEATURES * Column name: hash Description: MD5 hash of the example Type: 32 bytes string Column name: t_0 ... t_99 Description: API call Type: Integer (0-306) Column name: malware Description: Class Type: Integer: 0 (Goodware) or 1 (Malware) API Calls: ['NtOpenThread', 'ExitWindowsEx', 'FindResourceW', 'CryptExportKey', 'CreateRemoteThreadEx', 'MessageBoxTimeoutW', 'InternetCrackUrlW', 'StartServiceW', 'GetFileSize', 'GetVolumeNameForVolumeMountPointW', 'GetFileInformationByHandle', 'CryptAcquireContextW', 'RtlDecompressBuffer', 'SetWindowsHookExA', 'RegSetValueExW',

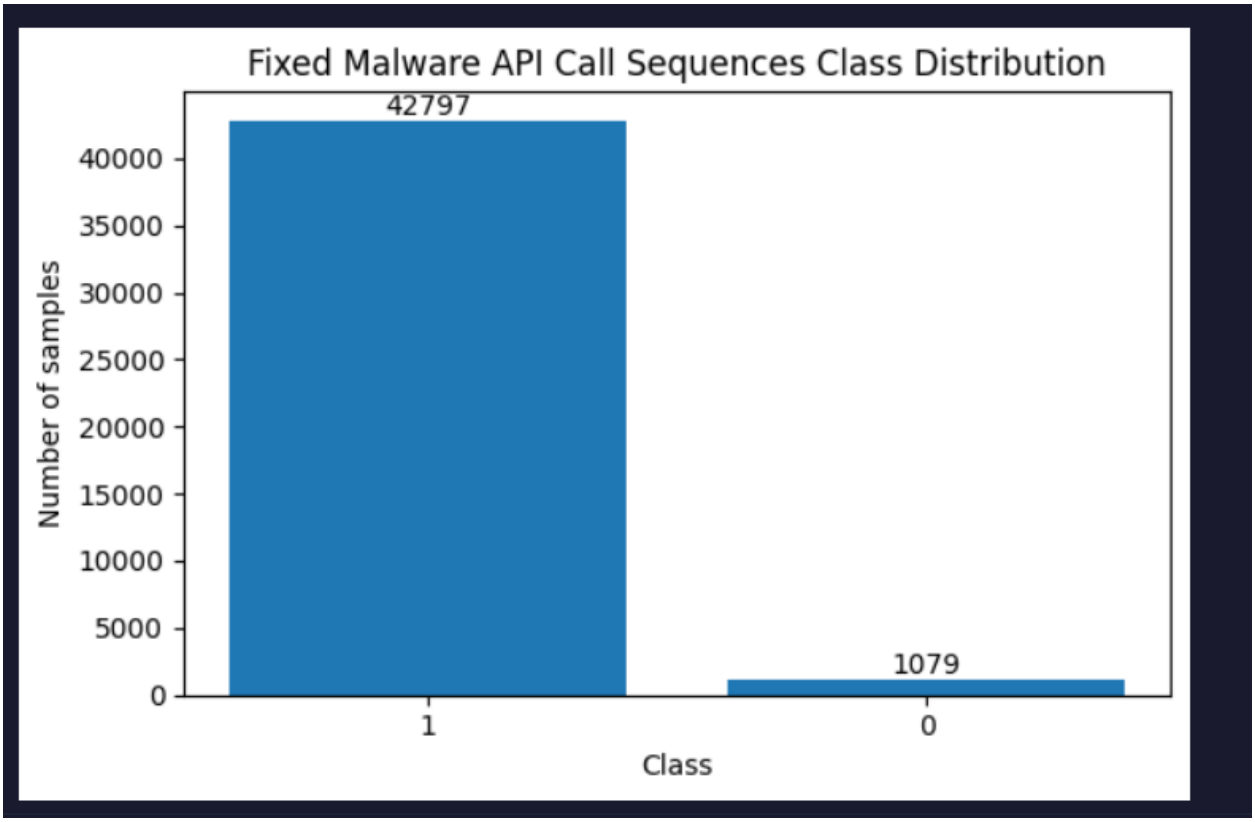
- Phần mô tả cung cấp danh sách API bị "số hóa" và danh sách tên API tương ứng, từ hai danh sách này, nhóm sẽ xử lí và mapping lại cho đúng tên API.
- Dạng gốc:

hash	t0	t1	...	t99	Malware
071e8c3f8922e18...	112	274	...	35	1
33f8e6d08a6aae9...	82	208	...	112	1
....	...	....	...	...	...

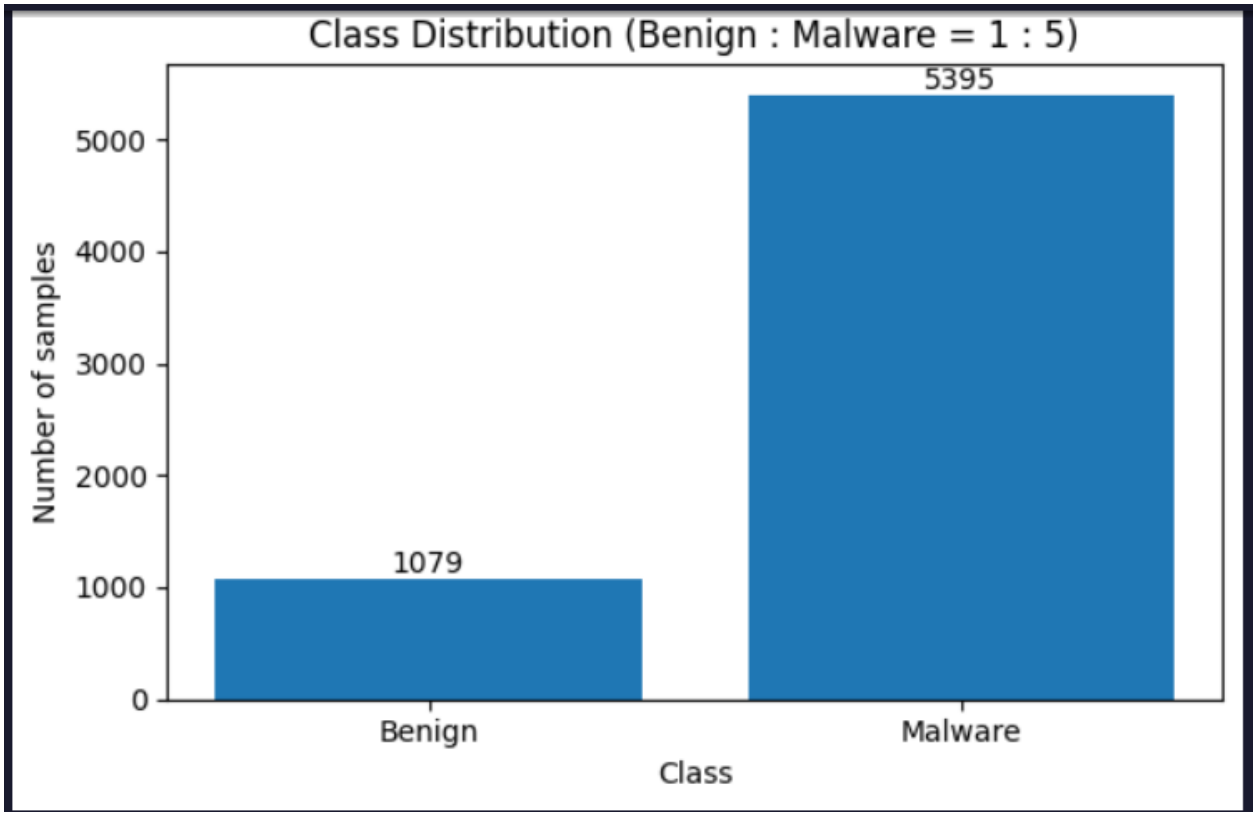
- Sau khi xử lí:

hash	t0	t1	...	t99	Malware
071e8c3f8922e18...	RegOpenKeyExA	NtOpenKey		GetSystemMetrics	1
33f8e6d08a6aae9...	GetSystemTimeAsFileTime	NtAllocateVirtualMemory		RegOpenKeyExA	1
....	...	....	...	...	...

✓ **Phân phối dữ liệu:**



- Vấn đề lớn nhất của bộ dataset này là dữ liệu quá mất cân bằng (malware gấp ~40 lần so với benign), nếu không xử lí mà dùng luôn để huấn luyện thì dẫn đến model học quá nhiều kiến thức từ malware, dẫn đến xu hướng thiên vị:
- ***"Khi đưa vào một app cần dự đoán thì đề gán nhãn là malware, vì tỉ lệ malware chiếm đa số, nên model lựa chọn cách học an toàn".***
- Dẫn đến sai bản chất của quá trình huấn luyện và gán nhãn.
- Vì vậy, nhóm sẽ chia thành các subset, mỗi subset theo tỉ lệ 1: 5 như sau:
  - Benign chiếm một phần (khoảng 1079 samples)
  - Malware chiếm năm phần (khoảng 5395 samples)
  - Đặt seed_random = 42 để mỗi lần random đều cho về cùng một kết quả
- Phân phối lại dữ liệu khi đưa vào huấn luyện:



- Lúc này, dataset nhìn có vẻ cân bằng hơn, đã có thể dùng huấn luyện.

C. Kết quả thực nghiệm

	Model	Dataset	Metrics	Goals
Experimental Setup 01	BERT based uncased, BERT small	MalBehavD-V1, PE_APICALLS, APIMDS	Accuracy, Precision, F1 Score, Recall	Hiện thực Flow chính của tác giả.
Experimental Setup 02	GCN, GIN, WORD2VEC + GAT, BERTbase + LSTM	MalBehavD-V1	Accuracy, Precision, F1 Score, Recall	Chứng minh sự ưu việt của DawnGNN.
Experimental Setup 03	BERT based uncased	Malware Analysis Datasets: API Call Sequences	Accuracy, Precision, F1 Score, Recall, TNR	Kiểm tra hiệu quả của mô hình trên Dataset mới.

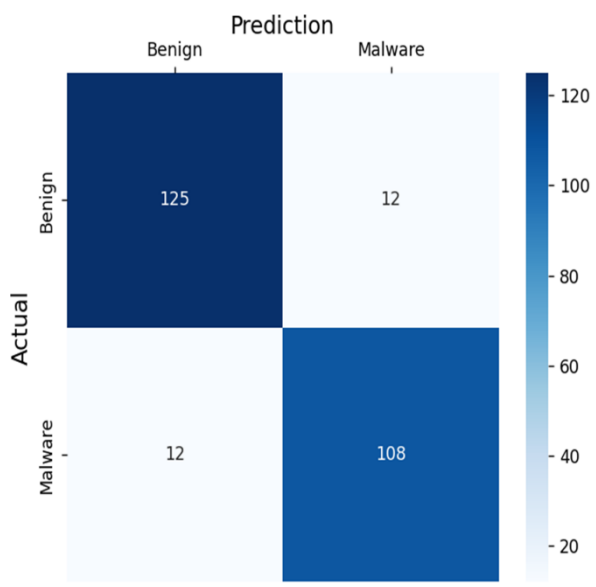
❖ Thực nghiệm 01:

➢ Đối với tập dữ liệu MalBehavD-V1: Sử dụng $BERT_{base}$ thay cho $BERT_{small}$ đã cải thiện độ chính xác tổng thể thêm 2.5% - 4.7%. Điều này chứng minh rằng, các

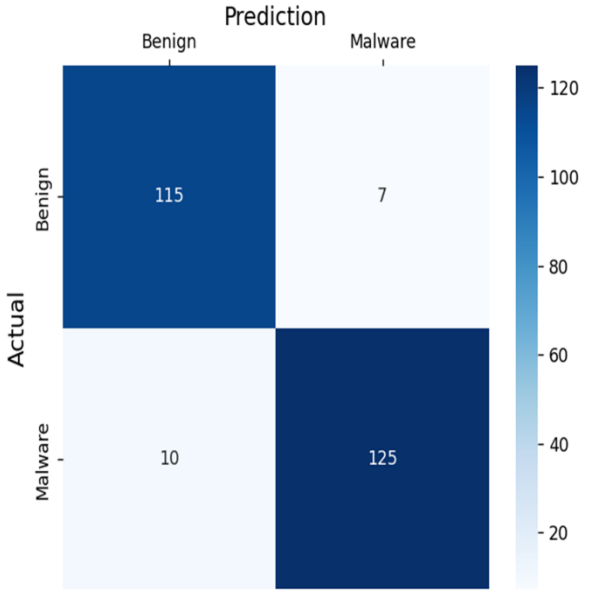
không gian vecor cao chiều của $BERT_{base}$ lưu trữ được nhiều đặc trưng phức tập hơn thì mô hình có khả năng phân loại tốt hơn.

- Đối với tập dữ liệu APIMDS: Thực nghiệm với một phần dữ liệu theo tỉ lệ 1:5, mô hình đạt 100% ở tất cả chỉ số đánh giá.
- Đối với tập dữ liệu PE_APICALLS: Mô hình đạt chỉ số Precision tỉ lệ 100%. Điều này thể hiện rằng, mọi mẫu mà mô hình dự đoán là mã độc đều chính xác là mã độc, không xảy ra trường hợp báo động giả (False Positive).
- Chú thích: Của nhóm tác giả | **Của nhóm thực hiện**

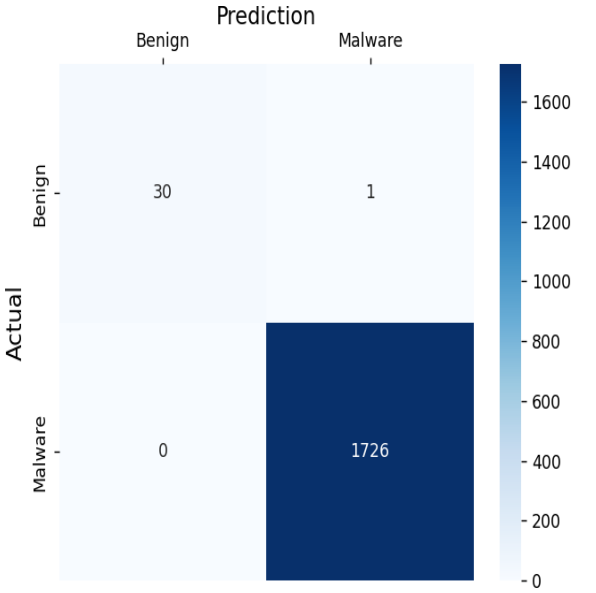
Model	Precision	Recall	F1 - Score	Accuracy	Dataset
BERTsmall + GAT	0.9667	0.9756	0.9683	0.9607	MalBehavD-V1
BERTsmall + GAT	**0.9066**	**0.9000**	**0.9000**	**0.9000**	**MalBehavD-V1**
BERTbase + GAT	0.9697	0.9788	0.9711	0.9638	MalBehavD-V1
BERTbase + GAT	**0.9470**	**0.9259**	**0.9363**	**0.9339**	**MalBehavD-V1**
BERTbase + GAT	0.9722	1.0000	0.9855	0.9762	PE_APICALLS
BERTbase + GAT	**0.9545**	**1.0000**	**0.9767**	**0.9677**	**PE_APICALLS**
BERTbase + GAT	0.9969	1.0000	0.9984	0.9975	APIMDS
BERTbase + GAT (Full Dataset)	**0.9994**	**1.0000**	**0.9997**	**0.9994**	**APIMDS**
BERTbase + GAT	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**APIMDS**



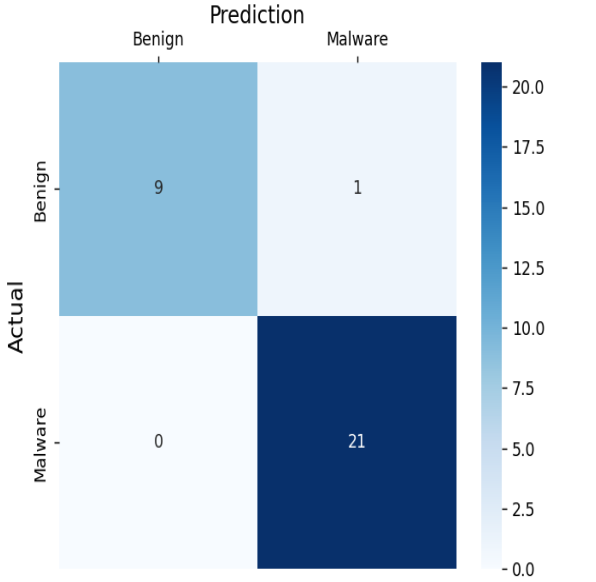
Ảnh: Confusion Matrix cho $BERT_{base}$ + GAT trên MalBehavD-V1



Ảnh: Confusion Matrix cho $BERT_{small}$ + GAT trên MalBehavD-V1



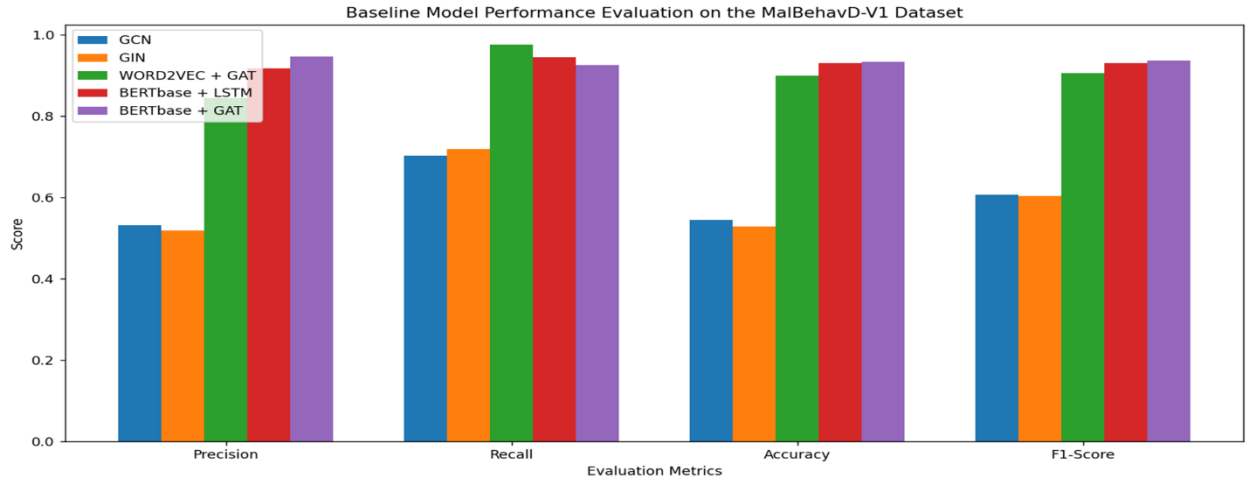
Ảnh: Confusion Matrix cho $BERT_{base}$ + GAT trên APIMDSẢnh: Confusion Matrix cho $BERT_{base}$ + GAT trên APIMDS



Ảnh: Confusion Matrix cho $BERT_{base}$ + GAT trên PE_APICALLSẢnh: Confusion Matrix cho $BERT_{base}$ + GAT trên PE_APICALLS

- Kết quả thực nghiệm trên tập dữ liệu MalBehavD-V1 cho thấy sự chênh lệch lớn trong khả năng phân loại giữa các kiến trúc mô hình.
- Các dòng thuật toán GNN thuần túy như GCN và GIN nhận diện với độ chính xác chỉ đạt xấp xỉ 51% - 53%.
- Khi kết hợp BERTbase với GAT, mô hình đạt độ chính xác 93.39% và chỉ số F1-Score 93.63%. Kết quả này minh chứng rằng việc nhúng đặc trưng từ $BERT_{base}$ cung cấp thông tin ngữ nghĩa phong phú, cho phép GAT tập trung vào các nút

quan trọng trong đồ thị gọi API, và có khả năng phân loại vượt trội so với các phương pháp truyền thống như Word2Vec hay LSTM trong đa số các chỉ số.



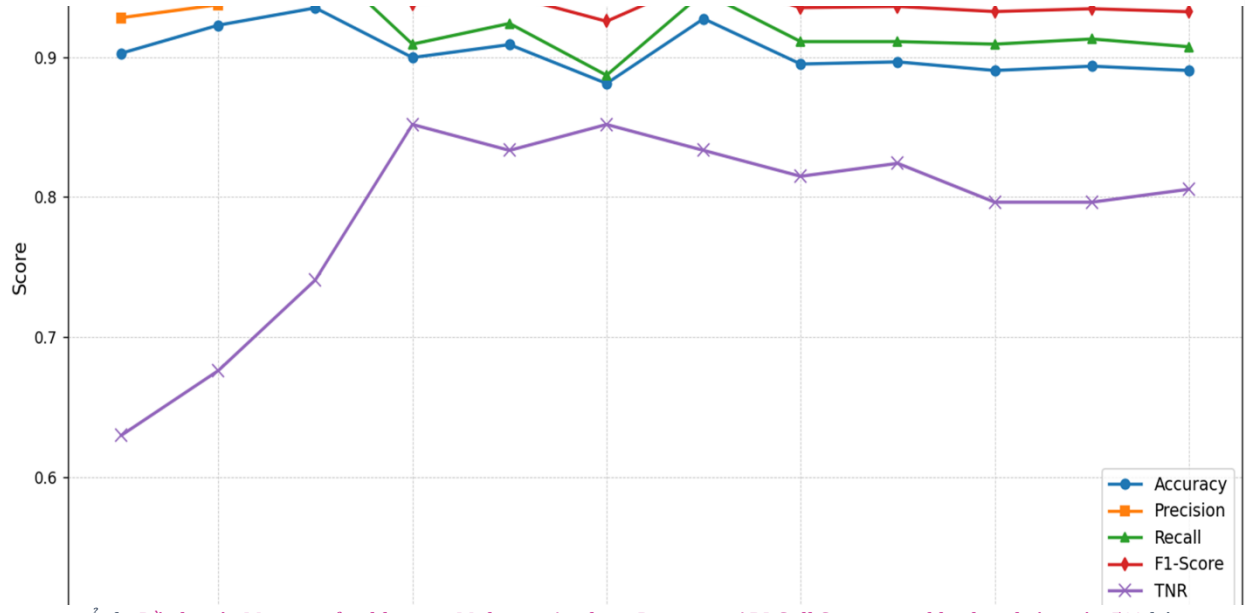
Ảnh: So sánh hiệu quả của BERTbase + GAT so với các môn hình đơn lẻ và mô hình kết hợp

Model	Precision	Recall	F1 - Score	Accuracy	Dataset
GCN	0.5325	0.7031	0.6061	0.5447	MalBehavD-V1
GIN	0.5198	0.7188	0.6032	0.5292	MalBehavD-V1
WORD2VEC + GAT	0.8446	0.9765	0.9058	0.8988	MalBehavD-V1
BERTbase + LSTM	0.9167	0.9453	0.9307	0.9300	MalBehavD-V1
BERTbase + GAT	**0.9470**	**0.9259**	**0.9363**	**0.9339**	**MalBehavD-V1**

- Thực nghiệm 03:
  - Khi huấn luyện trên nguyên bộ, các chỉ số duy trì ở mức từ 95% đến 98% và trạng thái định trong các Epoch từ 50 đến 300. Tuy nhiên, True Negative Rate dao động mạnh từ 56% đến 78%, phản ảnh rằng mô hình có xu hướng dự đoán chương trình bình thường thành chương trình độc hại.

- Khi huấn luyện trên khoảng 5% dữ liệu, các chỉ số từ khoảng 89% đến 96% với biên độ dao động lớn tầm 4%. Song, tỉ lệ True Negative Rate ổn định hơn, từ 63% đến 85%.

Ảnh: Đồ thi các Metrics của dataset Malware Analysis Datasets: API Call Sequences khi thực hiên trên nguyên bộ



Ảnh: Đồ thị các Metrics của ddataset Malware Analysis Datasets: API Call Sequences khi thực hiện trên 5% bộ

D. Hướng phát triển

- Hướng phát triển tiềm năng của đề tài:
  - Một thành phần (Module) trong cơ chế phòng thủ nhiều lớp của Mạng. Khi Host – Based Intrusion Detection System (HIDS) phát hiện chương trình khả không khớp với bất kì Signature đang có, đưa mẫu vào Sandbox. Thực thi chương trình, trích xuất API và đưa vào mô hình DawnGNN phân tích luồng gọi API và đưa ra xử lý tự động (Cho phép/ Theo dõi/ Chặn hoặc Xóa).
  - Áp dụng kỹ thuật Chắc lọc kiến thức (Knowledge Distillation) để nén mô hình BERT và GAT nhỏ gọn hơn, cho phép triển khai mô hình ngay tại các thiết bị đầu cuối (Edge devices) hoặc các máy chủ có cấu hình hạn chế.
  - Xây dựng cơ chế Phân tích lai (Hybrid Analysis): Khắc phục điểm yếu của phân tích động đối với mã độc sử dụng kỹ thuật Anti-Sandbox. Bằng cách kết hợp thêm các đặc trưng Phân tích tĩnh (Opcode, Strings, Import Table), hệ thống vẫn có thể phát hiện nguy cơ ngay cả khi mã độc "ngủ đông" hoặc không kích hoạt hành vi trong môi trường giám sát.
  - Tăng cường khả năng chống tấn công đối kháng (Adversarial Robustness): Nâng cao tính bền vững của mô hình GNN trước các kỹ thuật chèn nhiễu (Noise Injection). Sử dụng phương pháp Huấn luyện đối kháng (Adversarial Training) để giúp mô hình không bị đánh lừa bởi việc kẻ tấn công chèn các API vô hại vào luồng thực thi nhằm thay đổi cấu trúc đồ thị.
  - Mở rộng không gian ngữ nghĩa API (Semantic Expansion): Tối ưu hóa module API2Vec bằng cách khai thác thêm thông tin về Tham số (Parameters) và Giá trị trả về (Return Values) thay vì chỉ dựa vào mô tả chức năng. Điều này giúp phân biệt chính xác các hành vi tinh vi (ví dụ: phân biệt việc mở file hệ thống để ghi đè so với mở file log thông thường).
- Tính ứng dụng của đề tài:
  - Hỗ trợ ra quyết định tự động theo thời gian thực: Đóng vai trò là thành phần cốt lõi trong các hệ thống SOAR (Security Orchestration, Automation and

Response), giúp rút ngắn thời gian từ lúc phát hiện đến lúc ngăn chặn mối đe dọa (Mean Time to Respond - MTTR).

- Tối ưu hóa vận hành cho Trung tâm giám sát an ninh (SOC):
  - ✓ Giảm thiểu tỷ lệ cảnh báo sai (False Positives) nhờ khả năng hiểu sâu ngữ cảnh hành vi của mã độc thay vì chỉ dựa trên dấu hiệu nhận biết đơn giản.
  - ✓ Giảm tải khối lượng công việc thủ công cho các chuyên gia phân tích mã độc (Malware Analysts), cho phép họ tập trung vào các mối đe dọa phức tạp hơn.

HẾT