

# SeGDroid :

Phương pháp phát hiện mã độc dựa trên đồ thị gọi hàm đặc biệt.

## Tóm tắt :

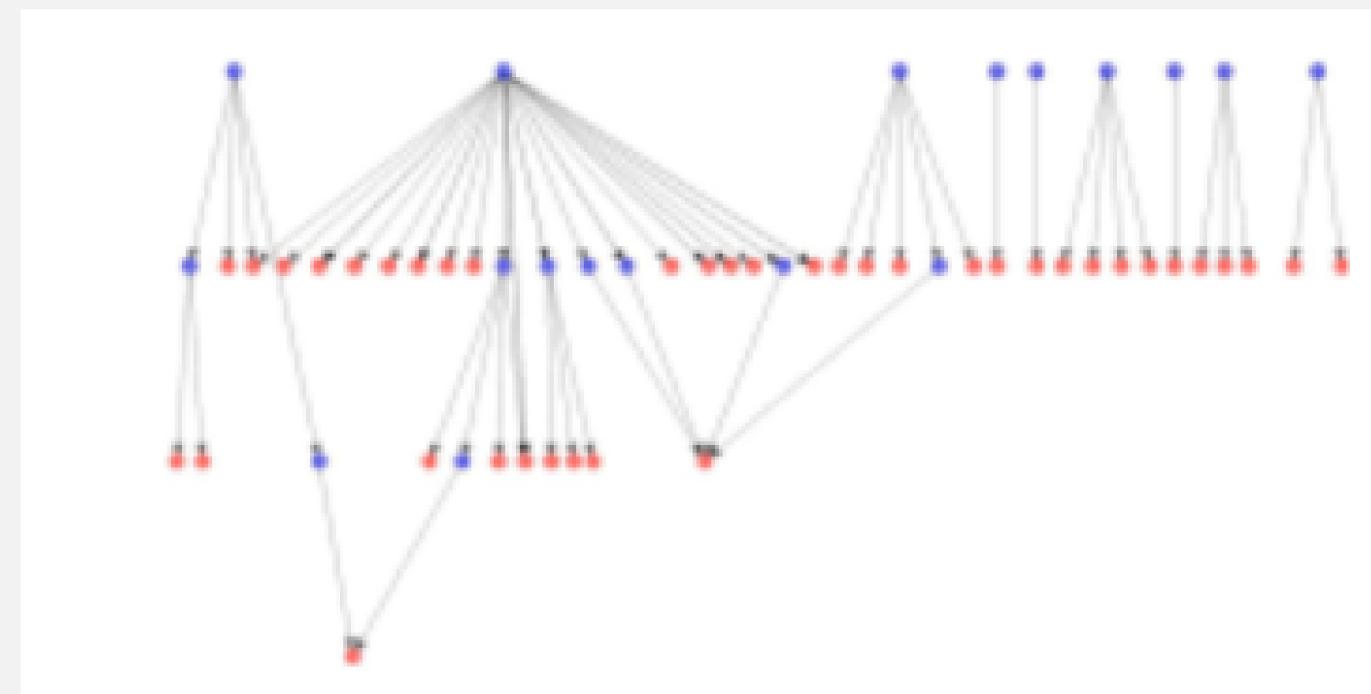
Báo cáo đưa ra một mô hình phát hiện các mã độc Android dựa trên mô hình học kiếń thức trích xuất từ các đồ thị gọi hàm đặc biệt.

Các đồ thị gọi hàm đặc biệt (FCGs) được xây dựng thông qua phương pháp cắt tỉa đồ thị, các nốt của đồ thị sẽ được biểu diễn qua các phương pháp API2vec, Opcode2vec và độ lường trọng số.

Thuật toán GNN GraphSAGE được áp dụng để huấn luyện có kết quả F-score 98% trong việc phát hiện mã độc và F-Score 96% trong việc phân loại mã độc

# I. Xây dựng đồ thị gọi hàm.

Sau khi giải nén và giải mã một tập tin APK, file bytecode dạng DEX sẽ được chuyển thành dạng đọc được là Smali Code.



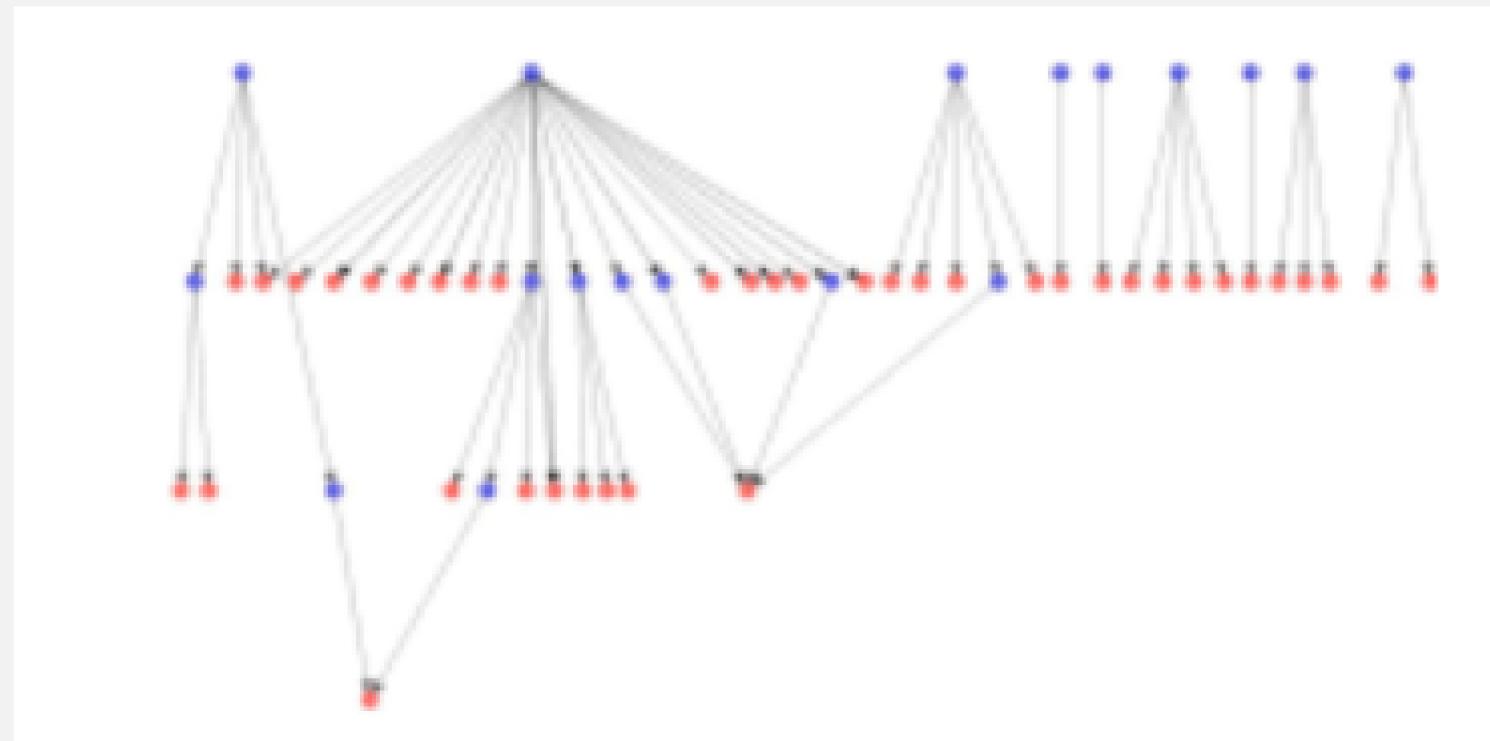
Các đồ thị gọi hàm được xây dựng từ Smali Code sẽ có dạng  $FCG = \langle V, E \rangle$ .

- Trong đó  $V$  là tập các hàm.
- $E$  là tập các cạnh có hướng biểu diễn hàm gọi và hàm được gọi.

# I. Xây dựng đồ thị gọi hàm.

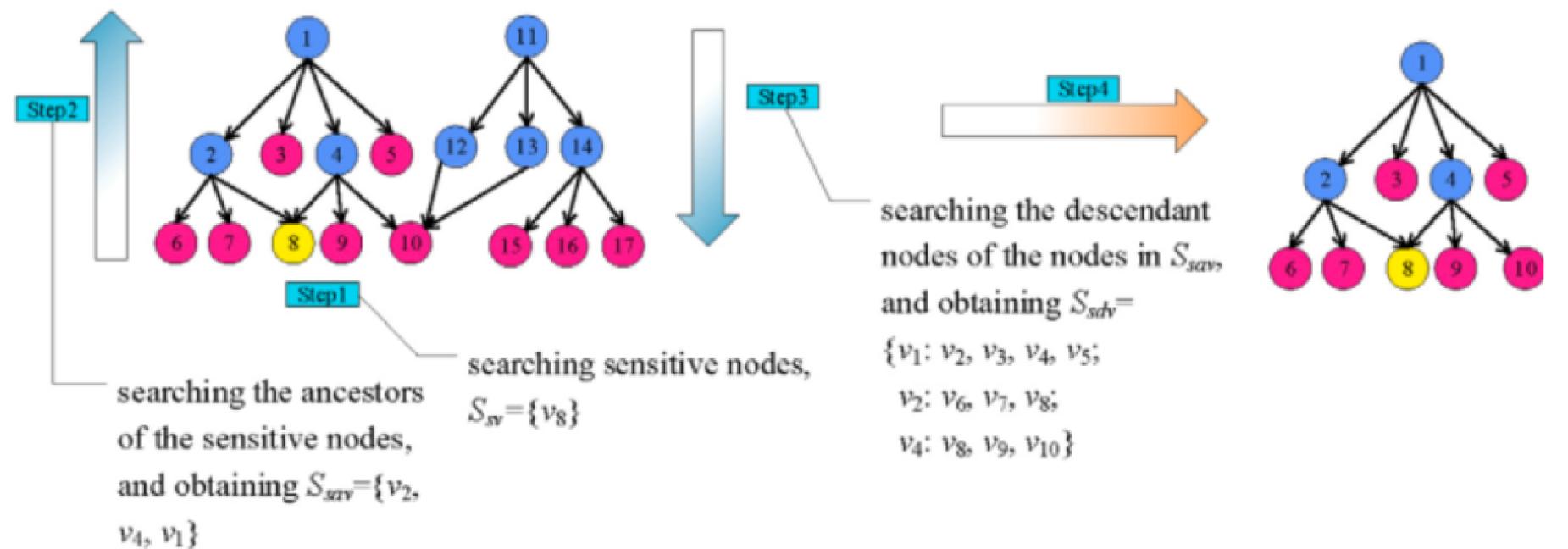
Màu xanh : Nốt trong, đại diện cho các hàm gọi đến các hàm khác, có thể là hàm tự định nghĩa.

Màu đỏ : Nốt ngoài, các hàm không gọi đến các hàm khác.



- **Đi theo hướng phân tích tĩnh**
- **Đối với các nốt trong sẽ được chuyển thành Opcode để hạn chế hành động làm nhiễu.**
- **Các nốt ngoài thường là các nốt thực hiện các nhiệm vụ nên có thể dựa vào API mà chúng tương tác để phát hiện hành vi mã độc.**
- **Cần cắt tỉa để giảm kích thước của đồ thị và xử lý mất cân bằng.**

# II. Cắt tỉa đồ thị gọi hàm.



1. Tìm kiếm các nốt có tương tác đến các API quan trọng trong tập Sensitive API
2. Tìm kiếm các nốt cha của nốt có tương tác đến Sensitive API đưa vào tập A.
3. Tìm kiếm tất cả nốt con của tất cả các nốt trong tập A.
4. Loại bỏ các nốt còn lại, ta được đồ thị gọi hàm đặc biệt (FCSs)

Tập các Sensitive API :

Tập dữ liệu các API đặc biệt thực hiện các hành động nhạy cảm, liên quan đến các quyền riêng tư, bảo mật, việc tương tác đến các API này đặt ra nghi vấn là hành vi của mã độc.

# III. Biểu diễn các nốt của FCSs.

Sau khi tạo ra đồ thị gọi hàm đặc biệt, cần biểu diễn các nốt dưới dạng dữ liệu mà máy tính có thể học được kiến thức từ chúng.

## API2vec

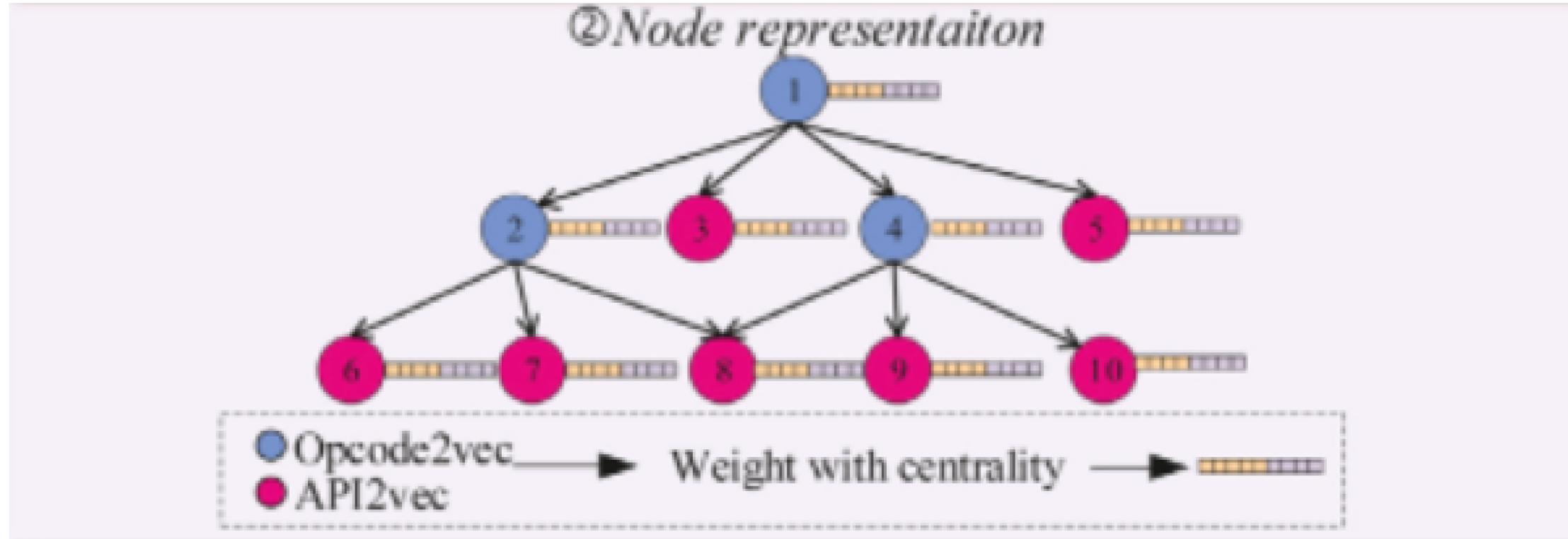
- Dùng cho các nốt ngoài.
- Từ điển (corpus) được xây dựng dựa trên tên package API để đảm bảo ổn định thay vì tên phương thức API
- Các API trong cùng 1 gói thường có cùng mục đích.
- Word2vec áp dụng để tạo feature vector cho mỗi từ trong tên của package
- Trung bình cộng các Vector biểu diễn cho mỗi từ trong tên Package tạo thành feature vector của nốt ngoài đó.

## Opcde2vec

- Dùng cho các nốt trong.
- Từ điển (corpus) được xây dựng từ opcode được chuyển đổi từ tập huấn luyện
- Mỗi Opcode trong chuỗi Opcode được chuyển từ hàm được biểu diễn bởi nốt trong sẽ được xem như 1 từ và áp dụng Word2vec để tạo feature vector.
- Trung bình cộng các Vector biểu diễn cho mỗi Opcode trong 1 chuỗi opcode là feature vector cho nốt trong đó.

Cả 2 phương pháp đều dựa trên Word2Vec vậy nên các vector gần nhau sẽ gần nhau về mặt ý nghĩa kiến thức chúng mang lại.

### III. Biểu diễn các nốt của FCSs.



Sau khi các nốt được biểu diễn dưới dạng các vector, các trọng số sẽ được tính toán dựa trên phương pháp trung tâm để đánh giá tầm quan trọng của các nốt.

$$d_i = \frac{\deg(i)}{n - 1}$$

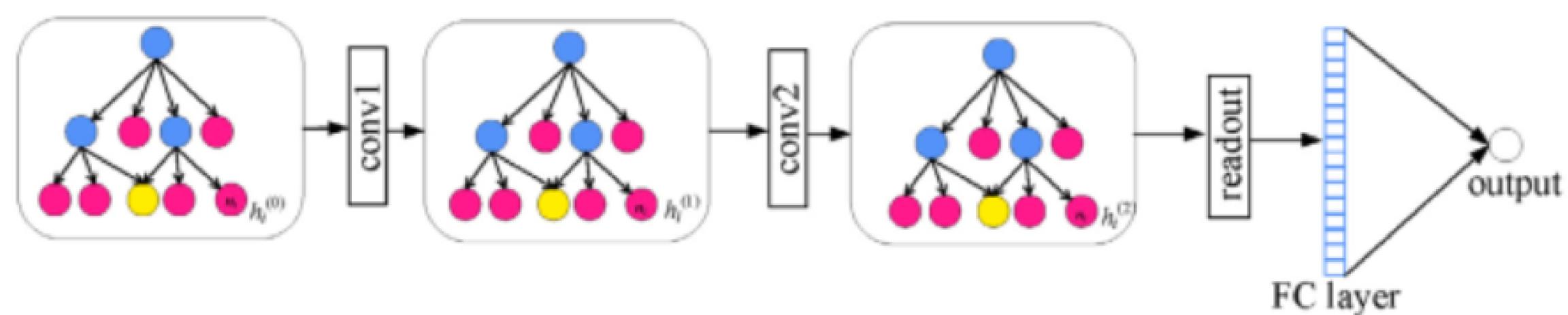
- $\deg(i)$  là bậc của nốt i
- n là tổng số nốt của đồ thị gọi hàm

$$h_i = d_i * V^i$$

- $V$  là vector được biểu diễn qua API2vec và opcode2vec của nốt i
- $h_i$  là vector cuối cùng của nốt i sau quá trình biểu diễn.

# IV. Chọn phương pháp học.

- GNN được lựa chọn vì trong quá trình chuyển đồ thị thành vector nhúng, thuật toán có dựa vào cấu trúc topo của đồ thị.
- Trong nhiều dạng của thuật toán GNN được áp dụng, nhiều thử nghiệm chỉ ra GraphSAGE hiệu quả nhất trong tác vụ xác định mã độc.



- 2 Lớp convolution được dùng để học kiến thức từ các FCGs.
- Lớp Dense cuối cùng sẽ nhận kết quả từ readout và đưa ra dự đoán.

# IV. Thủ nghiệm.

## 1. Tập dữ liệu

2 Tập dữ liệu được dùng là

- CICMal2020 dataset : với 5 loại APK gồm Benign, Adware, Banking malware, SMS malware và Riskware số lượng lần lượt là 4043, 1511, 2282, 4821 và 3938.
- MalRadar dataset : với 15 loại APK chứa mã độc và số lượng theo bảng dưới đây, phần benign còn thiếu sẽ được lấy từ AndroZoo.

The number of samples in each class of MalRadar dataset.

Families	#apps	Families	#apps	Families	#apps
KBuster	54	FAKEBANK	80	GhostClicker	181
ZNIU	59	Lucy	80	HiddenAd	287
SpyNote	63	GhostCtrl	109	LIBSKIN	240
Joker	72	EventBot	124	Xavier	589
FakeSpy	74	MilkyDoor	208	RuMMS	795

# IV. Thủ nghiệm.

## 2.Cấu hình huấn luyện

- Tỉ lệ Train / Test : 80% / 20%, trong đấy 80% tập training set được dùng để huấn luyện mô hình, 20% dùng để kiểm thử.
- Thêm hàm check ngưỡng nếu số nốt của đồ thị dưới 8000 sẽ không tiến hành cắt tỉa.

## 3.Cấu hình mô hình học máy dùng GraphSAGE.

- Số lớp Convolution bằng 2.
- Số nốt học tập mỗi lớp [64,32].
- Learning rate : 0.001.
- Trình tối ưu hóa : ADAM
- Số Epchos : 100.

# IV. Thủ nghiệm.

## 4. Đánh giá hiệu quả các phần.

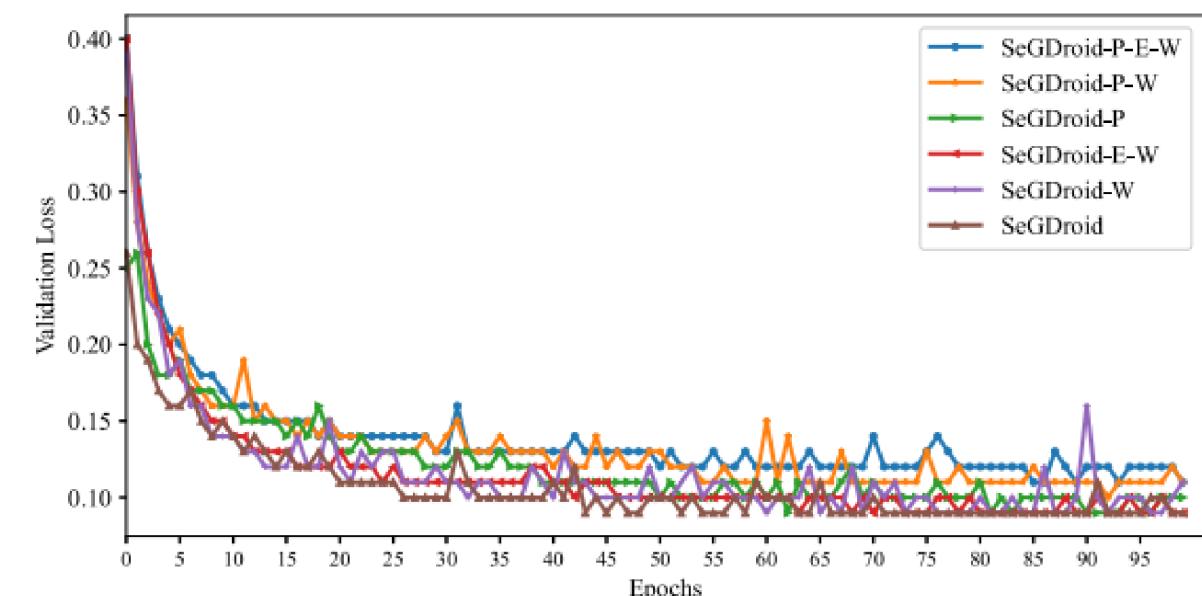
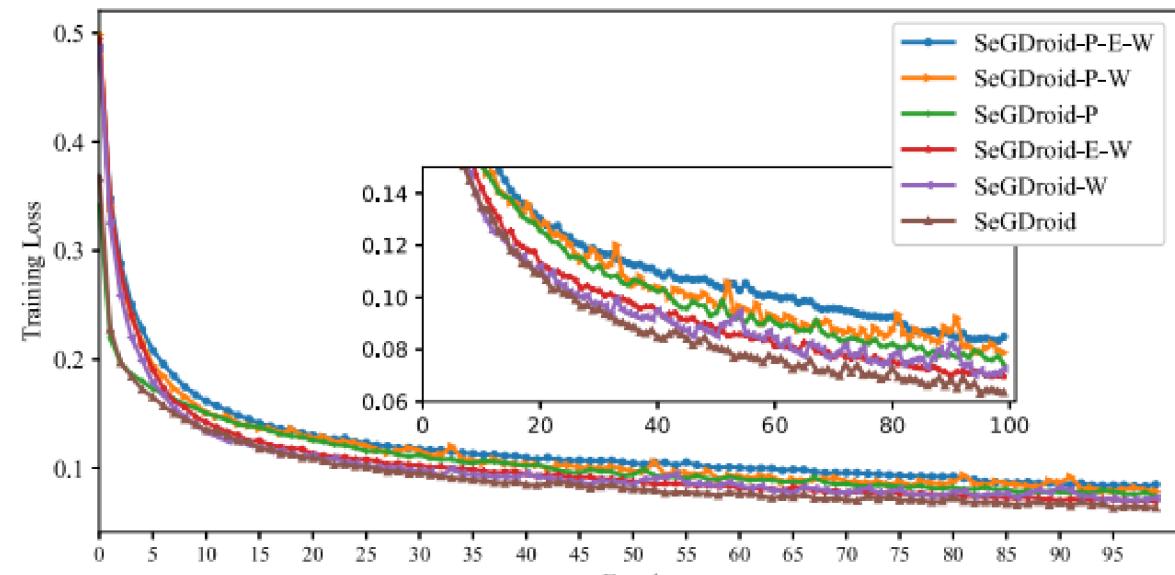
Áp dụng cả cắt tỉa, biểu diễn vector và đặt trọng số luôn cho kết quả training loss và validation loss thấp nhất và ổn định qua các epochs.

Các giá trị Accuracy, Prec, Recall và F-score phần lớn đạt giá trị cao nhất khi áp dụng cả 3.

**Table 2**  
The malware detection results of variant models.

Methods	Acc.	Prec.(m)	Rec.(m)	F-score(m)	Prec.(b)	Rec.(b)	F-score(b)
SeGDroid-P-E-W	0.9715	0.9808	0.9816	0.9812	0.9423	0.9399	0.9411
SeGDroid-P-W	0.9732	0.9832	0.9815	0.9824	0.9432	0.9482	0.9457
SeGDroid-P	0.9757	0.9874	0.9804	0.9839	0.9408	0.9612	0.9509
SeGDroid-E-W	0.9715	0.9732	0.9896	0.9813	0.9657	0.9149	0.9396
SeGDroid-W	0.9768	0.9841	0.9852	0.9846	0.9540	0.9054	0.9522
SeGDroid	<b>0.9807</b>	<b>0.9931</b>	<b>0.9812</b>	<b>0.9871</b>	0.9439	<b>0.9790</b>	<b>0.9611</b>

SeGDroid P-E-W : là loại bỏ  
• P ( Cắt tỉa )  
• E (Biểu diễn vector)  
• W ( Đặt trọng số )  
SeGDroid : Áp dụng cả 3.



# IV. Thủ nghiệm.

## 5. Đánh giá hiệu quả cắt tỉa.

Benign-A : Số nốt trong đồ thị gọi hàm từ 1 APK benign sau khi cắt tỉa.

Benign-B : Số nốt trong đồ thị gọi hàm từ 1 APK benign trước khi cắt tỉa.

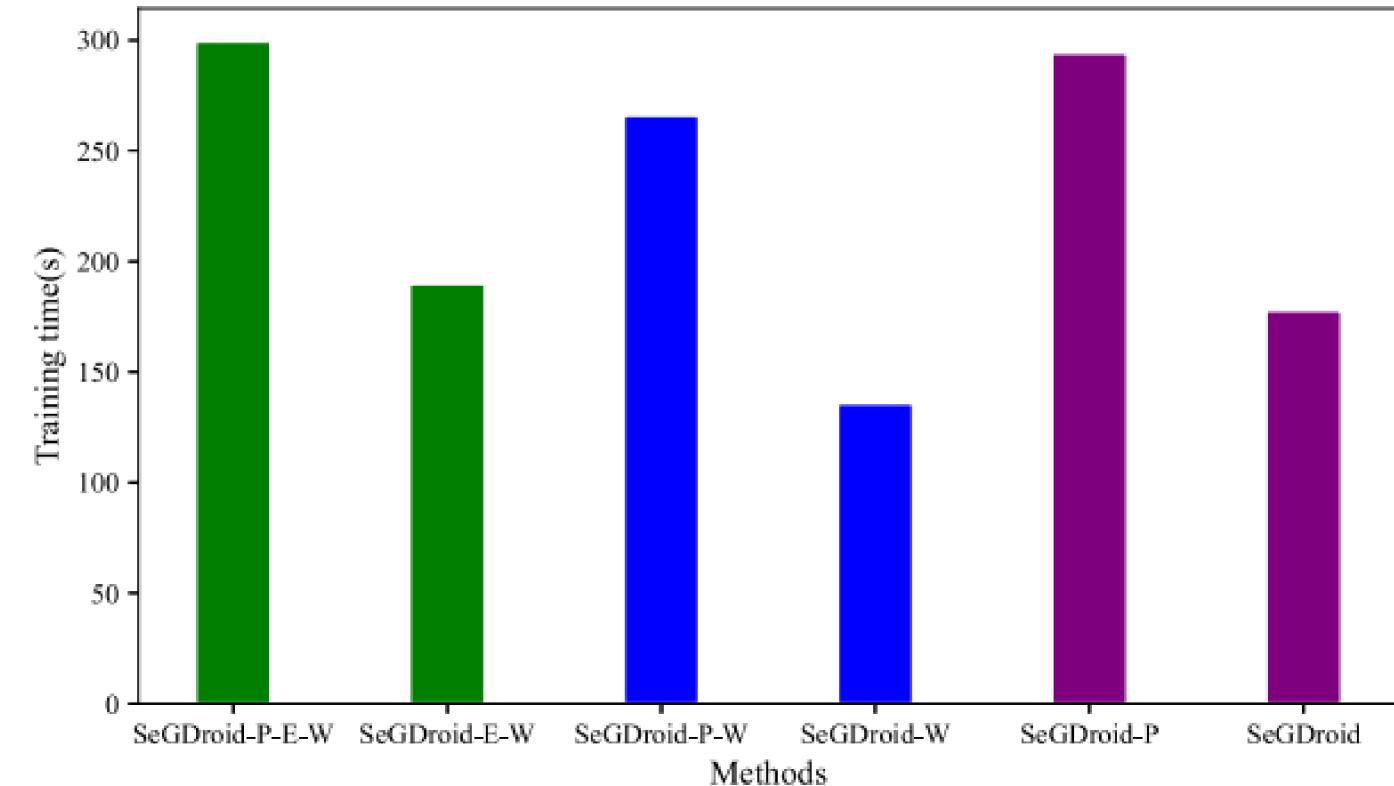
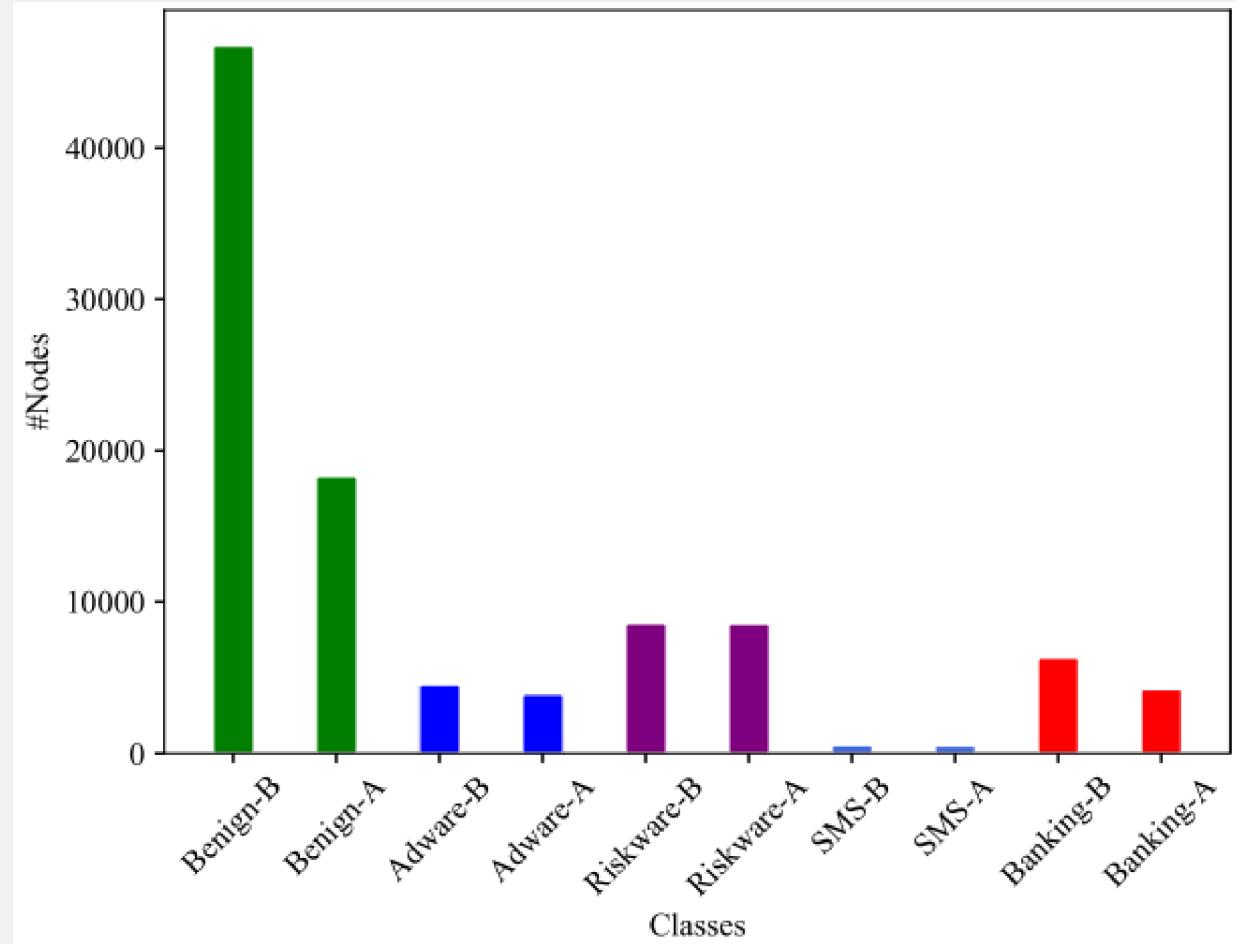


Fig. 10. The training time of different models.

=> Cắt tỉa hiệu quả trong việc giảm bớt số note không quan trọng và giảm thời gian Training.

# IV. Thủ nghiệm.

## 6. So sánh với các mô hình khác.

Đối với tập dữ liệu CIC:

Kết quả phân loại nhị phân khi so sánh với các phương pháp khác

Binary classification results obtained by different methods.

Methods	Acc.	Prec.(m)	Rec.(m)	F-score(m)	Prec.(b)	Rec.(b)	F-score(b)
Permission	0.9470	0.8380	0.9690	0.8987	0.9895	0.9400	0.9641
MaMaDroid	0.9645	0.9717	0.9817	0.9767	0.9410	0.9107	0.9256
MalScan	0.9789	0.9857	0.9865	0.9861	0.9577	0.9553	0.9565
GraphSAGE-Occ	0.9049	0.9357	0.8696	0.9014	0.8782	0.9402	0.9081
SeGDroid	0.9837	0.9889	0.9896	0.9892	0.9677	0.9653	0.9665

# IV. Thủ nghiệm.

## 6. So sánh với các mô hình khác.

Đối với tập dữ liệu MalRadar :

Kết quả phân loại nhị phân khi so sánh với các phương pháp khác

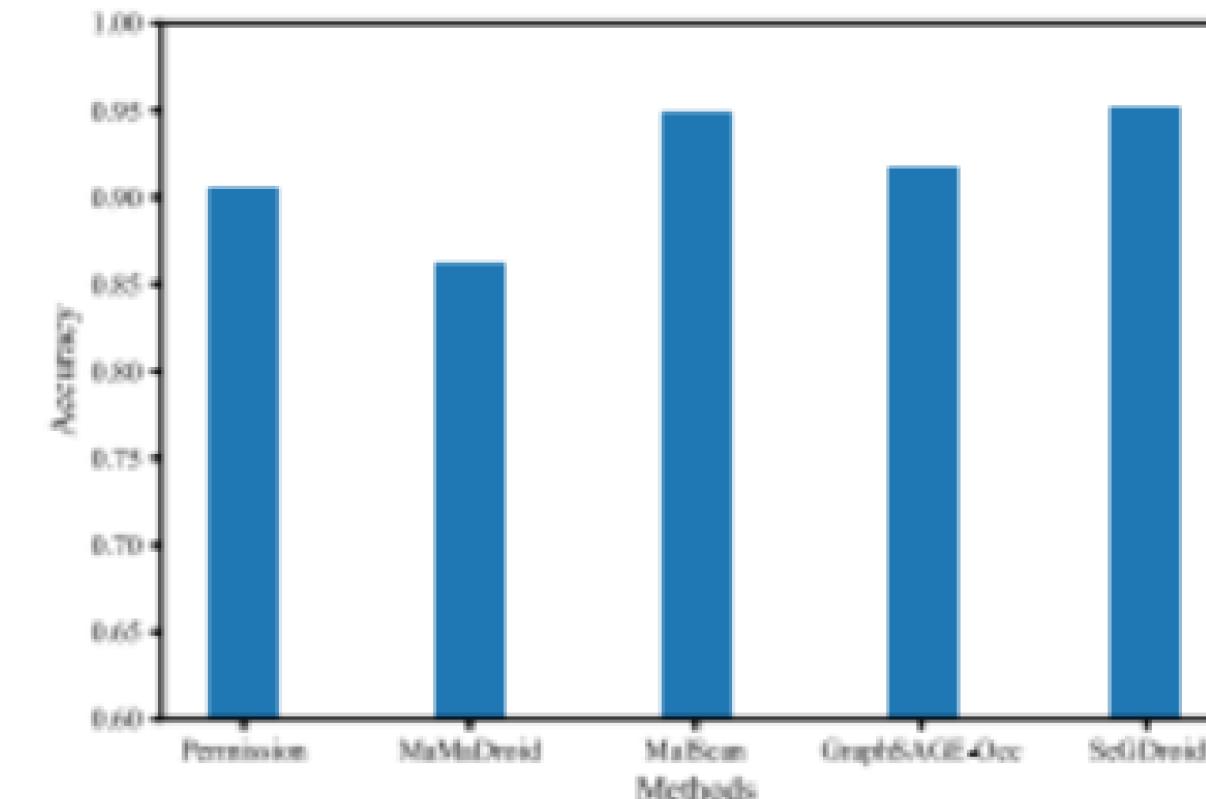
The binary classification results on MalRadar dataset.

Methods	Acc.	Prec.(m)	Rec.(m)	F-score(m)	Prec.(b)	Rec.(b)	F-score(b)
Permission	0.9739	0.9723	<b>0.9933</b>	0.9827	<b>0.9791</b>	0.9167	0.9468
MaMaDroid	0.9727	0.9866	0.9867	0.9816	0.9333	0.9608	0.9469
MalScan	<b>0.9813</b>	0.9867	0.9883	<b>0.9875</b>	0.9653	0.9606	0.9630
GraphSAGE-Occ	0.9763	0.9882	0.9800	0.9841	0.9423	0.9655	0.9538
SeGDroid	<b>0.9813</b>	<b>0.9899</b>	0.9850	<b>0.9875</b>	0.9563	<b>0.9704</b>	<b>0.9633</b>

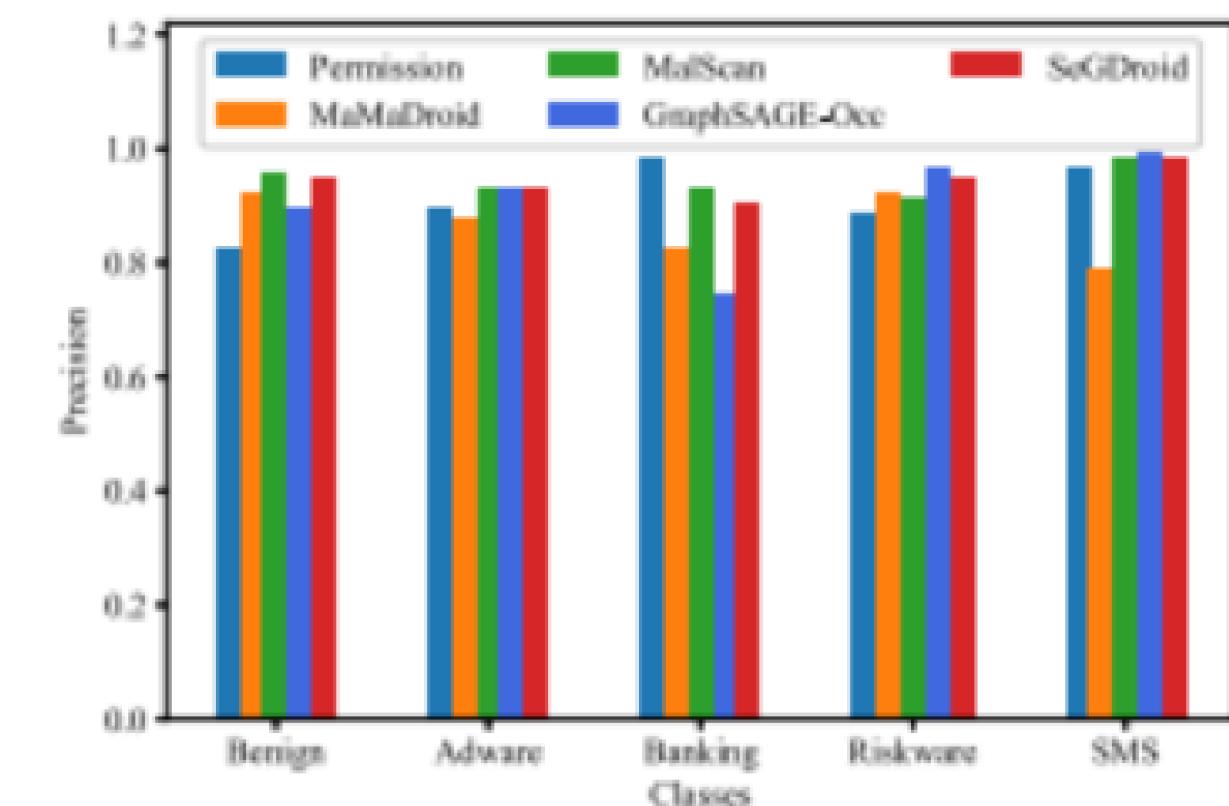
# IV. Thủ nghiệm.

## 6. So sánh với các mô hình khác.

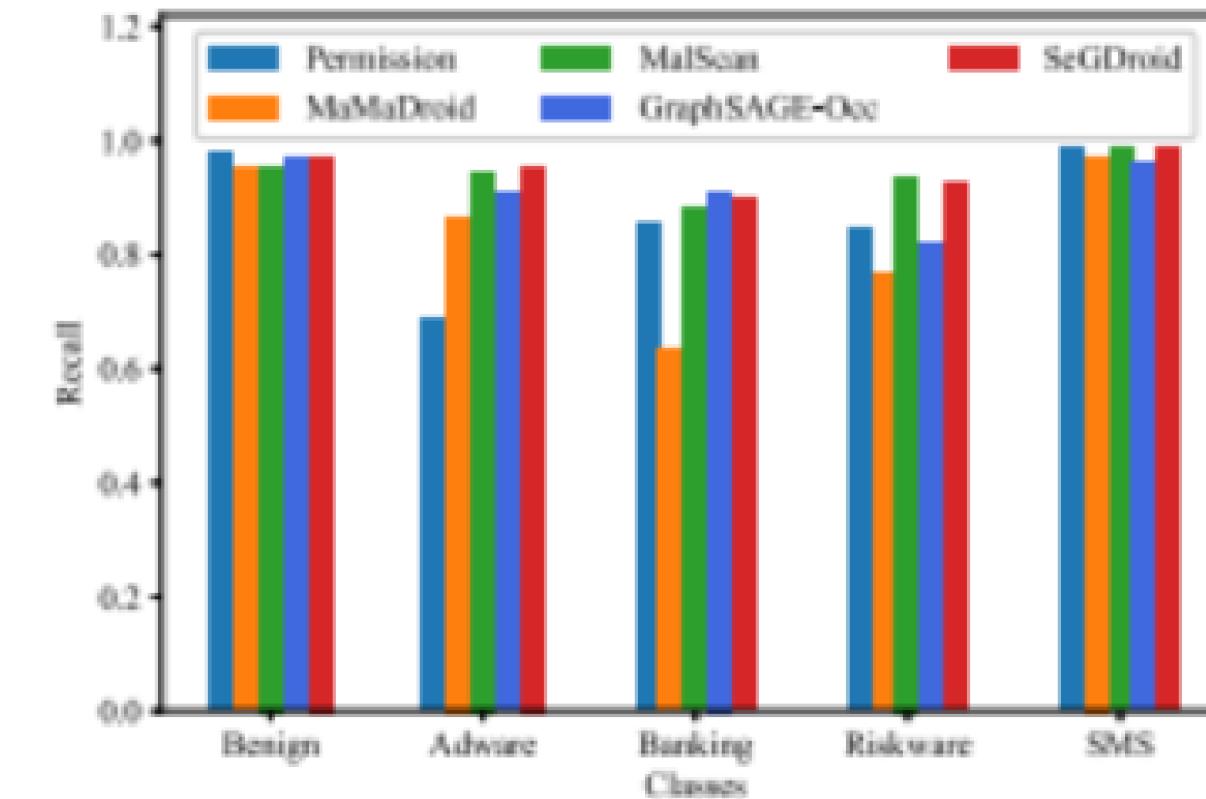
Đối với tập dữ liệu CIC :  
Kết quả phân loại Benign và các loại APK độc.



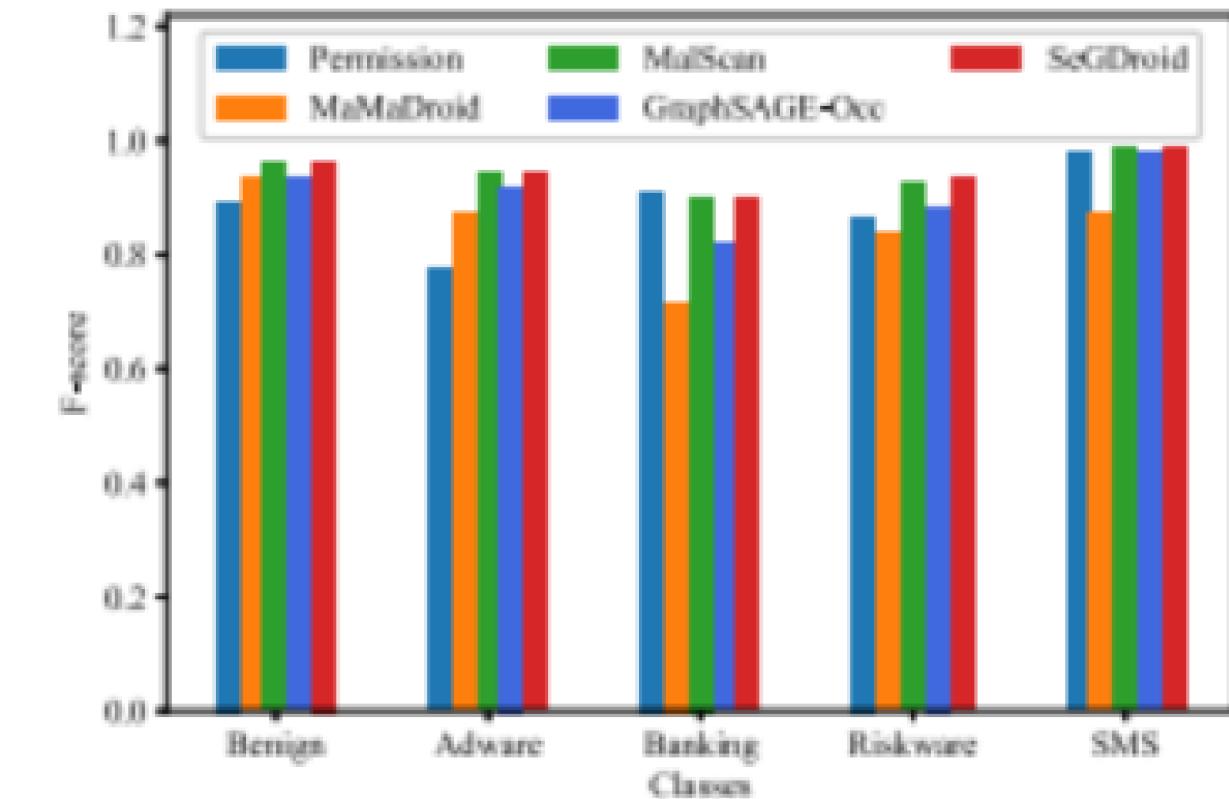
(a) Accuracy



(b) Precision



(c) Recall



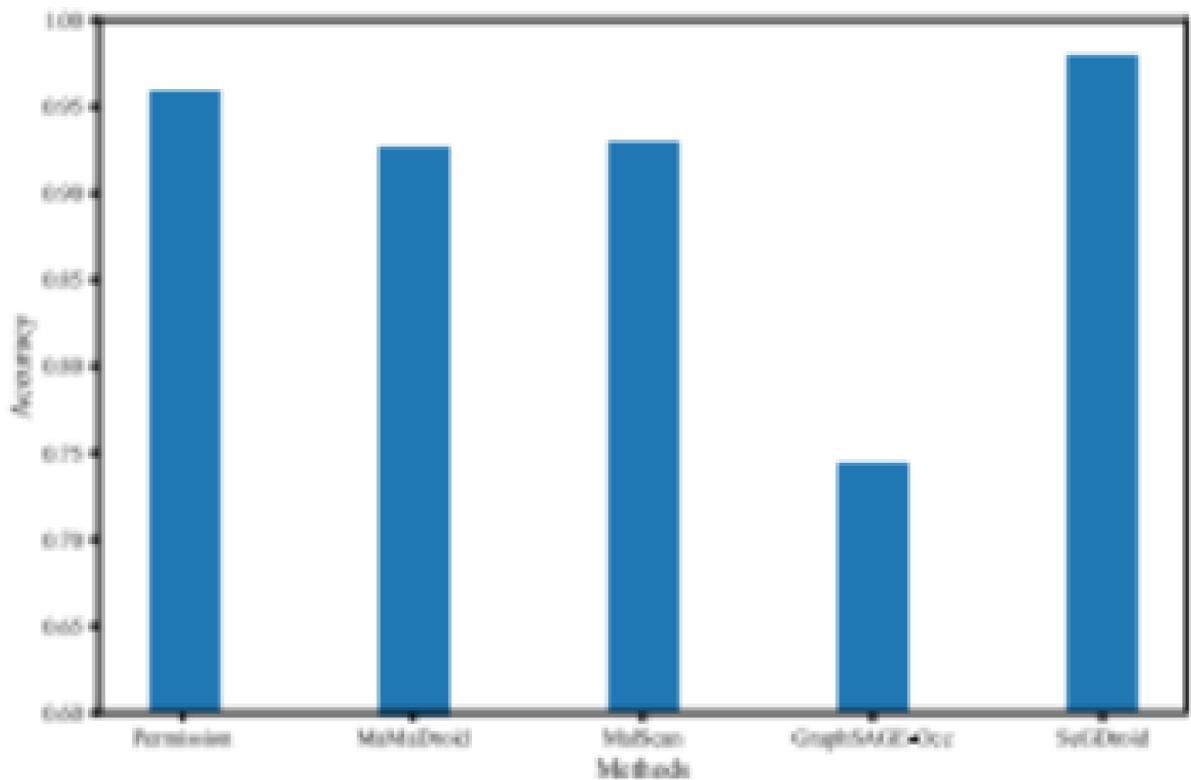
(d) F-score

Fig. 11. The results of category classification on CIC.

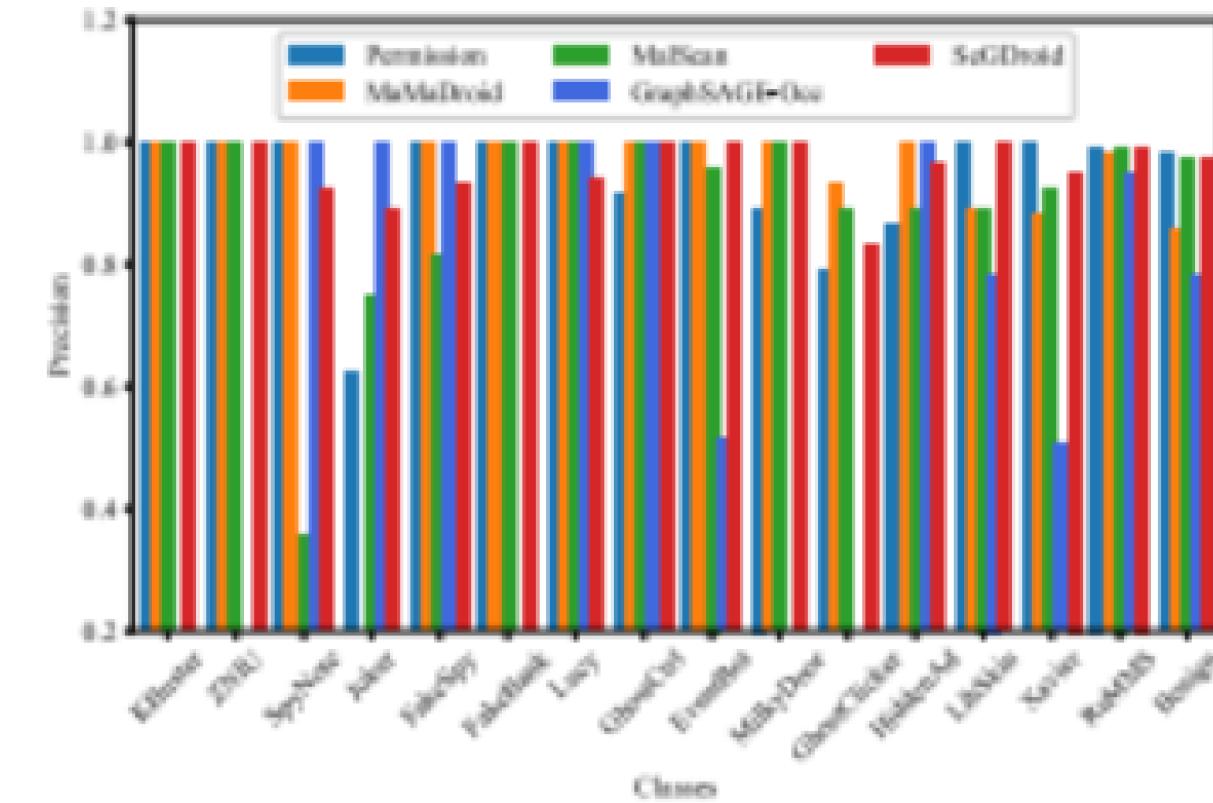
# IV. Thủ nghiệm.

## **6. So sánh với các mô hình khác.**

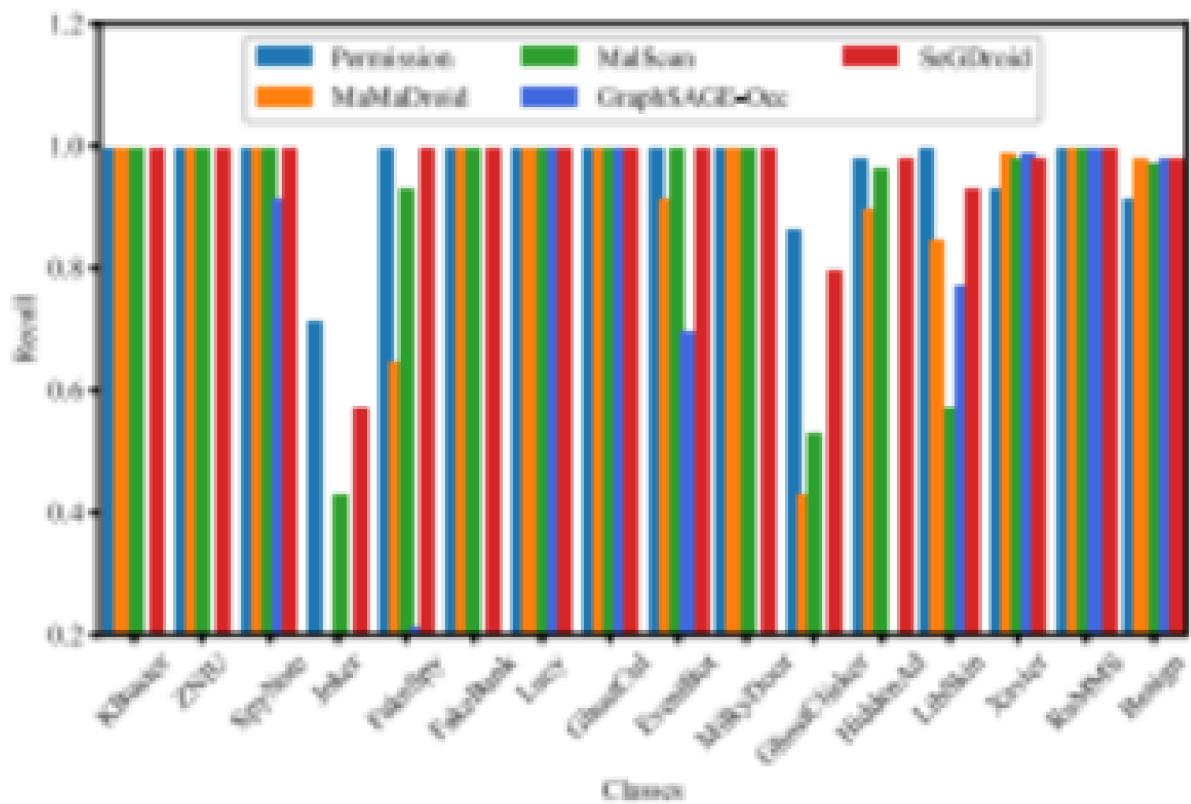
# Đối với tập dữ liệu MalRadar: Kết quả phân loại Benign và các loại APK độc.



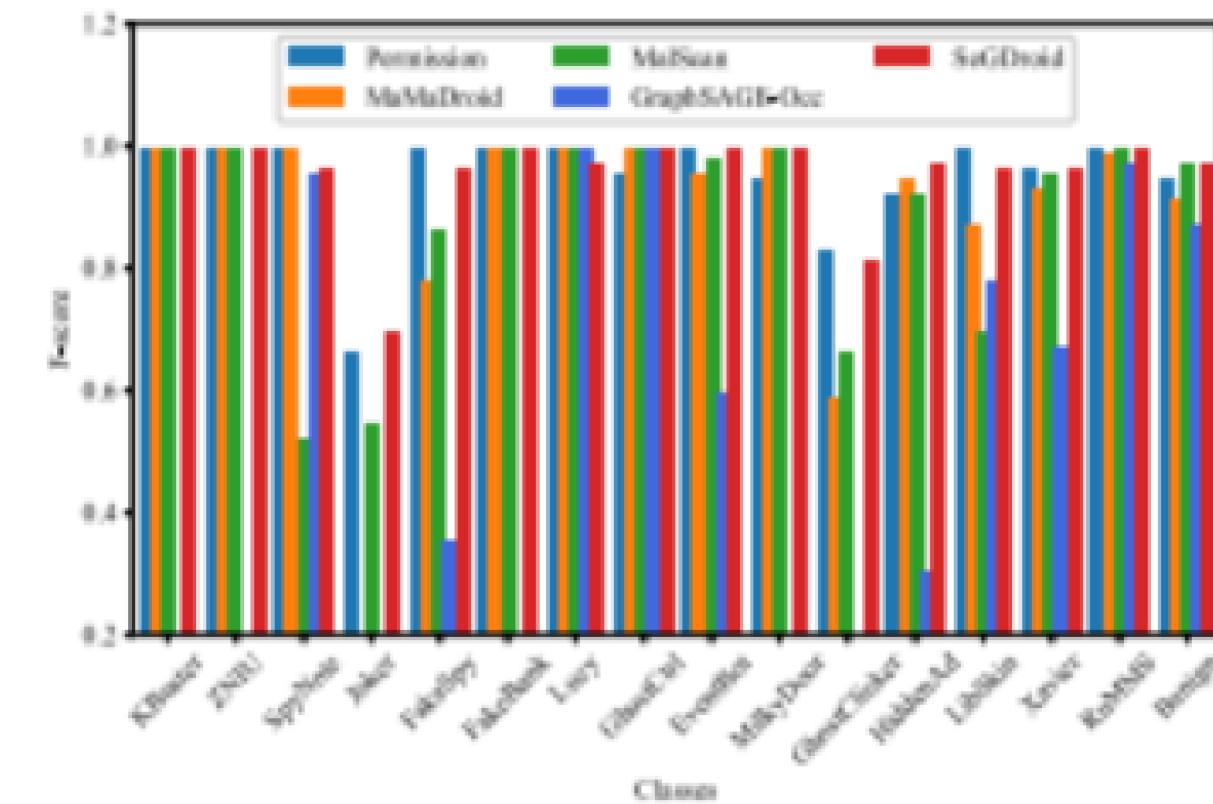
(a) Accuracy



### (b) Precision



(c) Recall



(d) F-score

**Fig. 12.** The family classification results on the MalRadar dataset.

# IV. Thủ nghiệm.

## 7. So sánh với các dạng khác của thuật toán GNN.

Kết quả phân loại nhị phân của GraphSAGE khi so sánh với các phương pháp khác

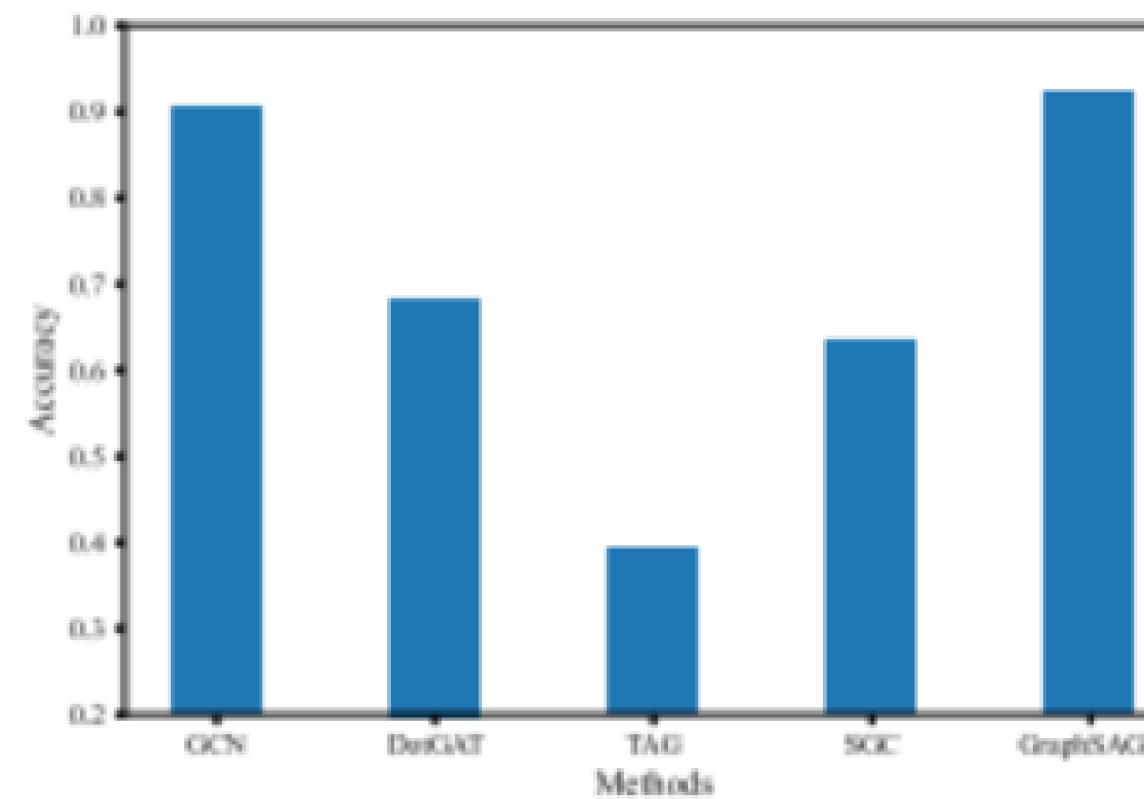
The binary classification results obtained by different graph learning algorithms.

Methods	Acc.	Prec.(m)	Rec.(m)	F-score(m)	Prec.(b)	Rec.(b)	F-score(b)
GCN	0.9710	0.9882	0.9732	0.9807	0.9208	0.9641	0.9420
DotGAT	0.9257	0.9800	0.9205	0.9493	0.7927	0.9418	0.8609
TAG	0.9586	0.9736	0.9716	0.9726	0.9127	0.9183	0.9155
SGC	0.9372	0.9682	0.9481	0.9580	0.8488	0.9035	0.8753
GraphSAGE	0.9807	0.9931	0.9812	0.9871	0.9439	0.9790	0.9611

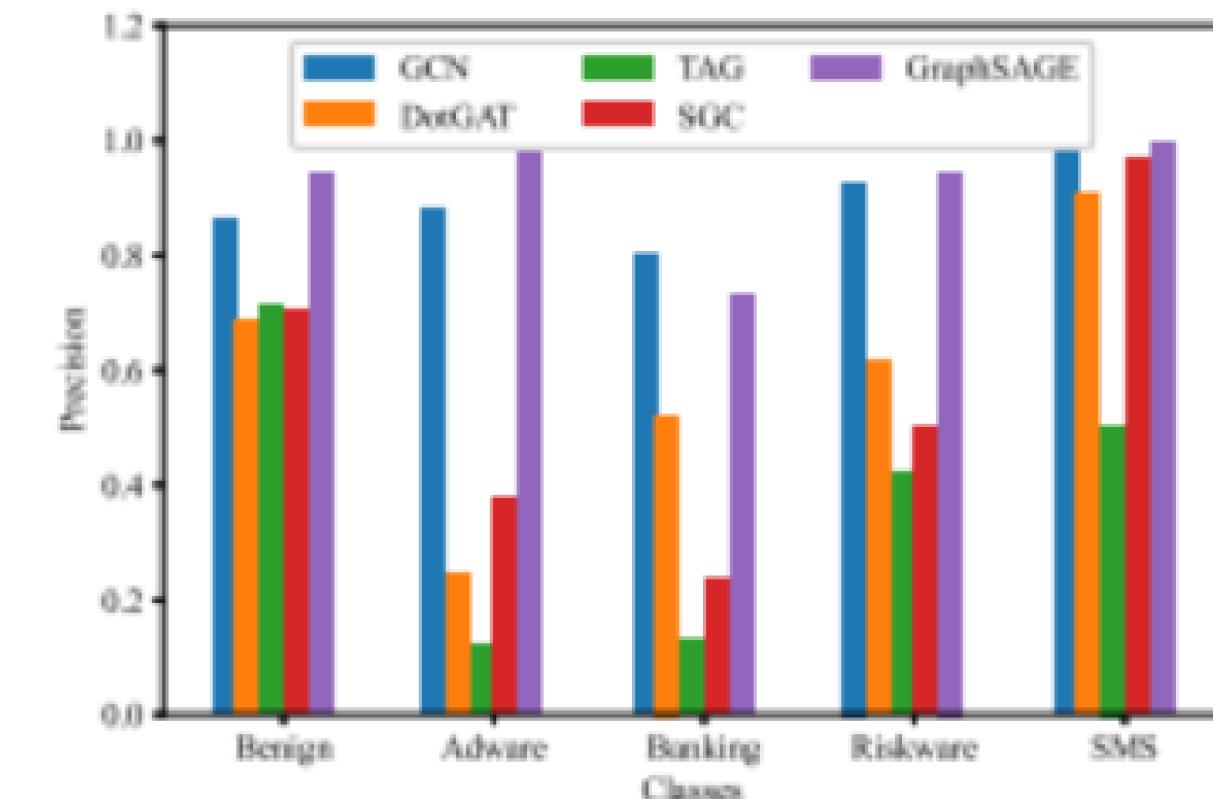
# IV. Thủ nghiệm.

## 7. So sánh với các dạng khác của thuật toán GNN.

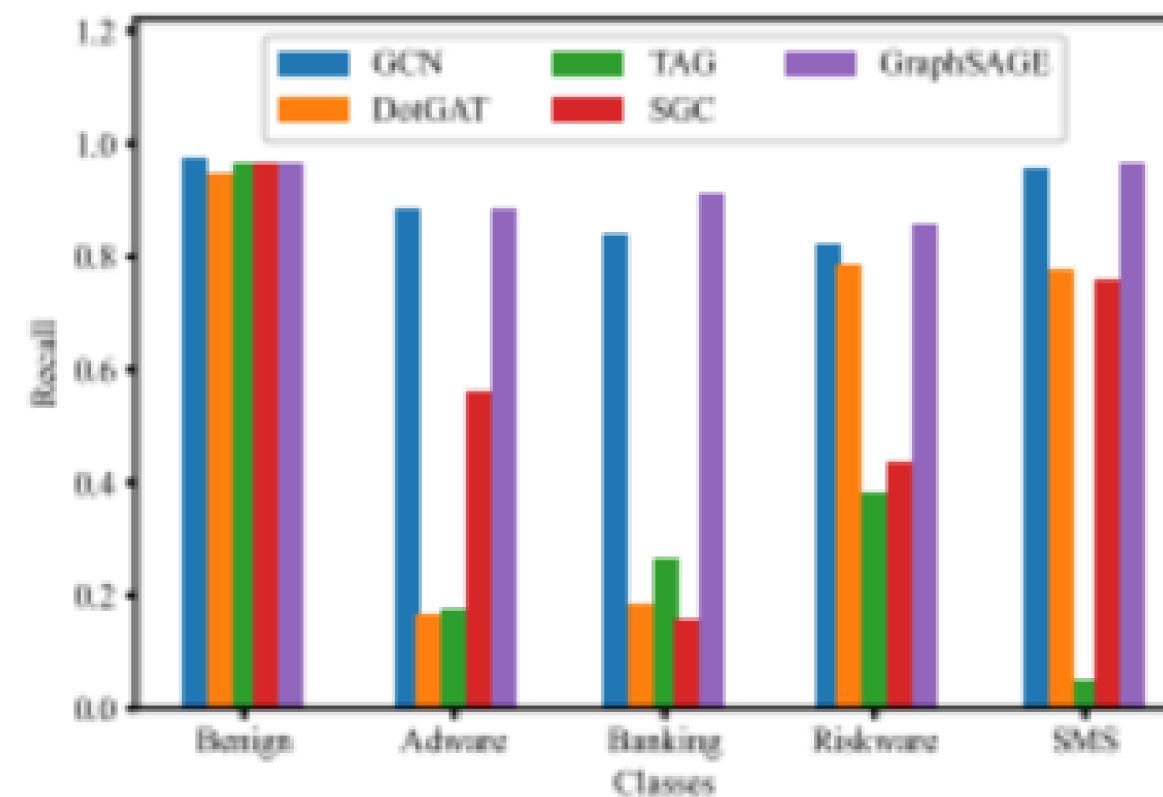
Kết quả phân loại Benign và các loại APK độc.



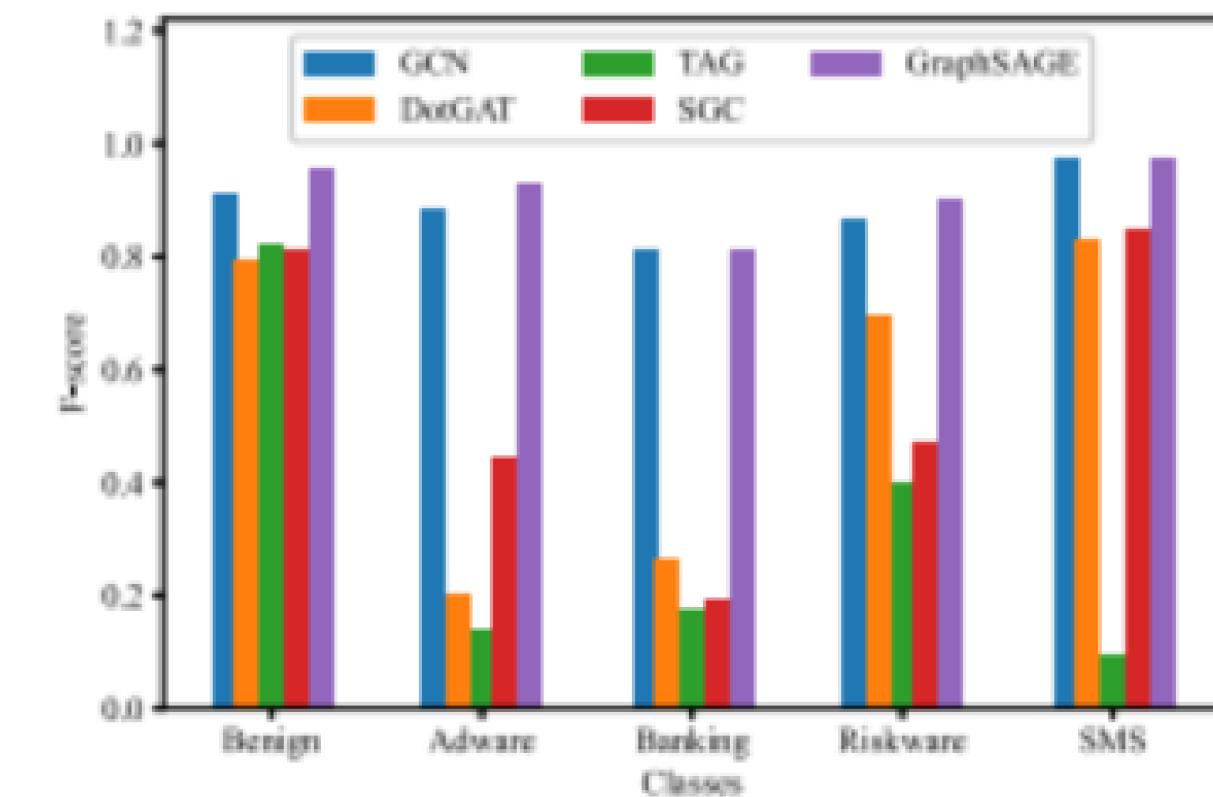
(a) Accuracy



(b) Precision



(c) Recall



(d) F-score

Fig. 13. The category classification results obtained by different graph learning algorithms.