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: Advanced Persistent Threat Attack Detection

Mã nhóm: G03 - Mã đề tài: S20

Lớp: NT230.P21.ANTT

1. THÔNG TIN THÀNH VIÊN NHÓM:

STT	Họ và tên	MSSV	Email
1	Lại Quan Thiên	22521385	22521385@gm.uit.edu.vn
2	Mai Nguyễn Nam Phương	22521164	22521164@gm.uit.edu.vn
3	Trần Thế Hữu Phúc	22521143	22521143@gm.uit.edu.vn
4	Hồ Diệp Huy	22520541	22520541@gm.uit.edu.vn

2. TÓM TẮT NỘI DUNG THỰC HIỆN:[1]

A. Chủ đề nghiên cứu trong lĩnh vực Học Máy:

☐ Dev Track

☑ Research Track

B. Tên đề tài

Magic: Detecting Advanced Persistent Threats Via Masked Graph Representation Learning

C. Liên kết lưu trữ mã nguồn của nhóm:

Mã nguồn của đề tài đồ án được lưu tại:

- Mã nguồn được fork từ tác giả: https://github.com/WanThinnn/MAGIC.git
- Mã nguồn bổ sung thêm giao diện GUI, gọi API từ mã nguồn gốc: https://github.com/WanThinnn/WAGIC.git

D. Tên tài liệu tham khảo chính:

[1] Z. Jia, Y. Xiong, Y. Nan, Y. Zhang, J. Zhao, and M. Wen, "MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning," *in Proc. 33rd USENIX Security Symposium (USENIX Security)*, Philadelphia, PA, USA, Aug. 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2310.09831

E. Tóm tắt nội dung chính:

Đề tài trình bày **MAGIC**, một mô hình học máy tự giám sát (self-supervised) để phát hiện Advanced Persistent Threats (APT) mà không cần dữ liệu tấn công đã gán nhãn. Đầu tiên, hệ thống chuyển các log thành một đồ thị nguồn gốc (**provenance graph)**—

[1] Ghi nội dung tương ứng theo mô tả

mỗi thực thể (process, file, socket…) là một nút, mỗi tương tác (read, write, exec…) là một cạnh, rồi loại bỏ nhiễu bằng cách gộp cạnh trùng. Tiếp theo, MAGIC áp dụng **masked graph learning**: ngẫu nhiên che một phần đặc trưng nút, dùng **Graph Attention Network (GAT)** kết hợp **autoencoder** để học embedding của hành vi bình thường. Embedding này được lưu trong **K-D Tree** làm tham chiếu. Khi có log mới, MAGIC tạo embedding và đo khoảng cách đến thư viện nhúng, những điểm quá xa ngưỡng bị đánh dấu dị thường - tương ứng với hoạt động APT. Cuối cùng, cơ chế thích ứng (**feedback adaptation**) cho phép cập nhật mô hình dựa trên phản hồi của chuyên gia để giảm false-positive và bắt kịp những thay đổi mới của hệ thống.

F. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong đề tài:

1. Provenance Graph Construction: Chuyển log hệ thống thành đồ thị có hướng; gộp cạnh trùng để giảm nhiễu.

2. Masked Graph Representation Learning

- Masking: Ẩn ngẫu nhiên thuộc tính nút để ép mô hình học ngữ cảnh.
- Graph Attention Network (GAT): Học từ hàng xóm với trọng số attention.
- Autoencoder: Tái cấu trúc đồ thị để tối ưu embedding.

3. Anomaly Detection via K-D Tree: Lưu embedding "bình thường" vào K-D Tree; so sánh khoảng cách embedding mới → cảnh báo bất thường.

4. Feedback Adaptation: Cập nhật embedding lành tính theo phản hồi của analyst, giảm false-positive và thích ứng với thay đổi hệ thống.

G. Môi trường thực nghiệm của đề tài:

- **Cấu hình máy tính:**
  + Mô hình được huấn luyện bằng GPU NVIDIA Tesla T4 12GB trên Google Colab;
  + Sau khi huấn luyện, mô hình được triển khai và thử nghiệm tại máy cá nhân với GPU Nvidia RTX 3050 6GB vRAM.
- **Các công cụ hỗ trợ sẵn có:**
  + Google Colab dùng cho giai đoạn huấn luyện mô hình;
  + Local Environment dùng cho giai đoạn triển khai thử nghiệm mô hình đã huấn luyện;
- **Ngôn ngữ lập trình sử dụng:** Python 3.8.10.
- **Thư viện, framework sử dụng:**
  + PyTorch (>=1.12.1);
  + DGL (>=1.0.0);
  + Scikit-learn (>=1.2.2);

+ Các thư viện Python bổ trợ khác.

- Đối tượng nghiên cứu:

+ Các bộ dữ liệu:

- StreamSpot (600 log batches, tấn công Drive-by-download);
- Unicorn Wget (chuỗi tấn công supply-chain ngụy trang);
- DARPA E3 (TRACE, THEIA, CADETS): gần 70 triệu interaction từ mạng doanh nghiệp bị APT tấn công;
- FiveDirections (tập dữ liệu thực tế nhưng phân phối khác biệt).

+ Tất cả được sử dụng để kiểm tra tính khả thi và hiệu quả của MAGIC trong nhiều kịch bản.

- Tiêu chí đánh giá:

+ Độ chính xác phân loại (Accuracy);

+ AUC, F1-score, Precision, Recall;

+ Tỷ lệ phát hiện mẫu độc hại (TPR) và tỷ lệ báo động giả (FPR);

+ Hiệu suất mô hình trên tập lớn và đa dạng (scalability, robustness).

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

- Công việc thực hiện:

+ Cài đặt lại MAGIC từ mã nguồn gốc;

+ Huấn luyện và kiểm thử mô hình trên các tập dữ liệu công khai;

+ Tích hợp giao diện GUI từ WAGIC để hỗ trợ demo trực quan;

+ Tạo script xử lý log, parser, tiền xử lý dữ liệu;

+ Thử nghiệm thêm trên tập dữ liệu mới (FiveDirections).

- Kết quả so sánh:

+ MAGIC đạt kết quả xuất sắc trên các tập dữ liệu gốc với AUC > 0.97, F1 ~ 0.99, Precision > 0.91, Recall > 0.97;

+ Với tập dữ liệu FiveDirections, mô hình gặp khó khăn do sự khác biệt lớn trong phân phối đặc trưng:

- AUC ~ 0.75;
- Recall = 0 → Không phát hiện được mẫu độc hại nào;
- Precision = 1.0 → Không có dự đoán sai là độc hại.

+ Kết quả này khẳng định mô hình hoạt động mạnh trong môi trường phù hợp nhưng cần điều chỉnh khi áp dụng dữ liệu thực tế "thô".

- Ưu điểm:

+ Khả năng phát hiện APT hiệu quả ở cả mức hệ thống và thực thể;

+ Hiệu suất cao, ít bỏ sót và báo động giả thấp trên dữ liệu đã chuẩn hoá.

- **Nhược điểm:**

+ Mô hình phụ thuộc nhiều vào chất lượng dữ liệu đầu vào;

+ Chi phí tính toán cao ở giai đoạn phát hiện nếu không tối ưu (KNN chiếm ~99% thời gian inference);

+ Cần tinh chỉnh khi triển khai với tập dữ liệu mới hoặc chưa được xử lý kỹ.

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

- Quá trình thu thập và xử lý dữ liệu gặp nhiều khó khăn:

+ Một số dữ liệu ở định dạng nhị phân/phức tạp (DARPA, FiveDirections);

+ Thiếu nhãn rõ ràng trong một số tập (cần tạo parser tùy biến).

- Chi phí huấn luyện mô hình cao nếu triển khai local:

+ MAGIC đòi hỏi GPU để xử lý tập dữ liệu lớn;

+ Tốn tài nguyên khi dùng KNN để phát hiện outlier.

- Hiệu năng trên dữ liệu thực tế (FiveDirections) chưa tốt:

+ Mô hình chưa tổng quát hóa đủ tốt khi áp dụng lên log thô thực tế;

+ Cần cải tiến masking strategy và phương pháp huấn luyện để nâng cao khả năng thích ứng.

- Tích hợp giao diện GUI đòi hỏi xử lý tương thích với backend gốc (MAGIC), phát sinh lỗi khi gọi API ban đầu, cần chỉnh sửa cấu trúc source code.

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

95%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

Xem chi tiết tại: Đánh Giá - Google Drive

STT	Công việc	Phân công nhiệm vụ
1	Nhóm trưởng; lên kế hoạch, phân công, điều hành nhóm thực hiện đồ án môn học.	Lại Quan Thiên
2	Đề xuất/thực hiện những ý tưởng mang tính chiến lược cho nhóm, Training Model, Chuẩn hoá Code	Mai Nguyễn Nam Phương
3	Thực hiện đầy đủ đồ án, thiết kế Poster, Training Model, Chuẩn hoá Code	Trần Thế Hữu Phúc
4	Thực hiện đầy đủ đồ án, thiết kế Slide, Training Model, Chuẩn hoá Code	Hồ Diệp Huy

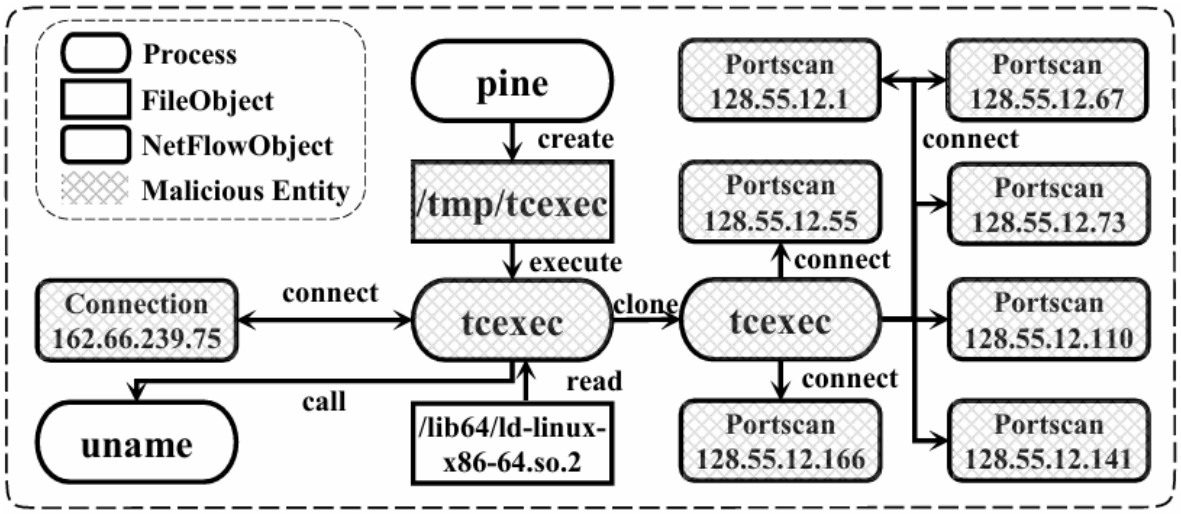
BÁO CÁO TỔNG KẾT CHI TIẾT

A. PHƯƠNG PHÁP THỰC HIỆN

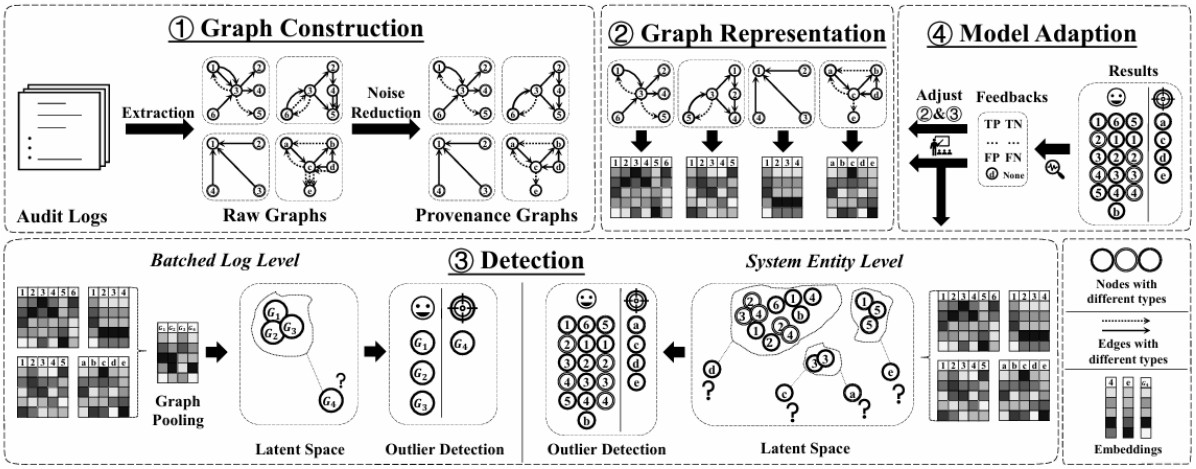
MAGIC là một phương pháp phát hiện APT tự giám sát mới lạ, tận dụng học biểu diễn đồ thị bị che mặt và các phương pháp phát hiện ngoại lệ, đồng thời có khả năng thực hiện phát hiện đa mức độ chi tiết hiệu quả trên khối lượng lớn nhật ký kiểm toán. Quy trình của **MAGIC** bao gồm ba thành phần chính: (1) xây dựng đồ thị nguồn gốc, (2) một mô-đun biểu diễn đồ thị và (3) một mô-đun phát hiện. Nó cũng cung cấp một tùy chọn (4) cơ chế thích ứng mô hình. Trong quá trình huấn luyện, **MAGIC** chuyển đổi dữ liệu huấn luyện bằng (1), học nhúng đồ thị bằng (2) và ghi nhớ các hành vi lành tính trong (3). Trong quá trình suy luận, **MAGIC** chuyển đổi dữ liệu mục tiêu bằng (1), thu được nhúng đồ thị bằng (2) đã được huấn luyện và phát hiện các ngoại lệ thông qua (3). Hình 2 đưa ra tổng quan về kiến trúc của **MAGIC**.

Các luồng nhật ký kiểm toán do phần mềm kiểm toán hệ thống thu thập thường được lưu trữ theo đợt. Trong quá trình xây dựng đồ thị nguồn gốc (1), **MAGIC** chuyển đổi các nhật ký này thành các đồ thị nguồn gốc tĩnh. Các thực thể hệ thống và tương tác giữa chúng được trích xuất và chuyển đổi thành các nút và cạnh tương ứng. Một số kỹ thuật giảm độ phức tạp được sử dụng để loại bỏ thông tin dư thừa.

Các đồ thị nguồn gốc đã được xây dựng sau đó được đưa qua mô-đun biểu diễn đồ thị (2) để thu được các nhúng đầu ra (tức là các biểu diễn vectơ toàn diện của các đối tượng). Được xây dựng dựa trên các bộ mã hóa tự động đồ thị bị che mặt và tích hợp tái cấu trúc cấu trúc dựa trên mẫu, mô-đun biểu diễn đồ thị nhúng, lan truyền và tổng hợp các thuộc tính của nút và cạnh vào các nhúng đầu ra, chứa cả nhúng nút và nhúng trạng thái hệ thống.



Hình 1. Minh hoạ một đồ thị nguồn gốc của một cuộc tấn công APT thực tế, khai thác lỗ hổng Pine Backdoor. Tất cả các thực thể và tương tác không liên quan đến tấn công đã được loại bỏ khỏi đồ thị nguồn gốc.



Hình 2. Tổng quan về quy trình phát hiện của MAGIC.

Mô-đun biểu diễn đồ thị chỉ được huấn luyện trên nhật ký kiểm toán lành tính để mô hình hóa các hành vi hệ thống lành tính. Khi thực hiện phát hiện APT trên nhật ký kiểm toán có khả năng chứa tấn công, **MAGIC** sử dụng các phương pháp phát hiện ngoại lệ dựa trên các nhúng đầu ra để phát hiện các ngoại lệ trong hành vi hệ thống (3). Tùy thuộc vào mức độ chi tiết của tác vụ, các nhúng khác nhau được sử dụng để hoàn thành việc phát hiện APT. Đối với các tác vụ ở cấp độ nhật ký theo đợt, các nhúng trạng thái hệ thống, phản ánh các hành vi chung của toàn bộ hệ thống, là mục tiêu phát hiện. Một ngoại lệ trong các nhúng như vậy có nghĩa là trạng thái hệ thống tương ứng của nó chưa từng thấy và có khả năng độc hại, điều này tiết lộ một tín hiệu APT trong lô đó. Đối với các tác vụ ở cấp độ thực thể hệ thống, mục tiêu phát hiện là các nhúng nút đó, đại diện cho hành vi của các thực thể hệ thống. Các ngoại lệ trong nhúng nút cho thấy các thực thể hệ thống đáng ngờ và phát hiện các mối đe dọa APT ở mức độ chi tiết cao hơn.

Trong các thiết lập phát hiện thực tế, **MAGIC** có hai ứng dụng được thiết kế trước. Đối với mỗi lô nhật ký được thu thập bởi phần mềm kiểm toán hệ thống, người dùng có thể trực tiếp sử dụng tính năng phát hiện ở cấp độ thực thể của **MAGIC** để xác định chính xác các thực thể độc hại trong lô, hoặc thực hiện phát hiện hai giai đoạn, như đã nêu trong Phần 2.3. Trong trường hợp này, **MAGIC** đầu tiên quét một lô và xem liệu có tín hiệu độc hại tồn tại trong lô hay không (phát hiện cấp độ nhật ký theo lô). Nếu nó cảnh báo dương tính, **MAGIC** sau đó thực hiện phát hiện ở cấp độ thực thể để xác định các thực thể hệ thống độc hại ở mức độ chi tiết cao hơn. Phát hiện ở cấp độ nhật ký theo lô ít đòi hỏi tính toán hơn đáng kể so với phát hiện ở cấp độ thực thể. Do đó, một quy trình hai giai đoạn như vậy có thể giúp người dùng của **MAGIC** tiết kiệm tài nguyên tính toán và tránh các báo động giả mà không ảnh hưởng đến độ chi tiết phát hiện của **MAGIC**. Tuy nhiên, nếu người dùng ưu tiên phát hiện chi tiết trên tất cả các thực thể hệ thống, thì quy trình trước đó vẫn là một tùy chọn khả dụng.

Để đối phó với sự trôi dạt khái niệm và các cuộc tấn công chưa từng thấy, một cơ chế thích ứng mô hình tùy chọn được sử dụng để cung cấp các kênh phản hồi cho người dùng (4).

Kết quả phát hiện được các nhà phân tích bảo mật kiểm tra và xác nhận được đưa trở lại **MAGIC**, giúp nó thích ứng với những thay đổi trong hành vi hệ thống lành tính theo cách bán giám sát. Trong những điều kiện như vậy, **MAGIC** đạt được kết quả phát hiện thậm chí còn hứa hẹn hơn, điều này được thảo luận trong Phần **C. KẾT QUẢ THỰC NGHIỆM**. Hơn nữa, **MAGIC** có thể dễ dàng được áp dụng cho việc phát hiện APT trực tuyến trong thế giới thực nhờ khả năng tự thích ứng với sự trôi dạt khái niệm và chi phí tính toán tối thiểu.

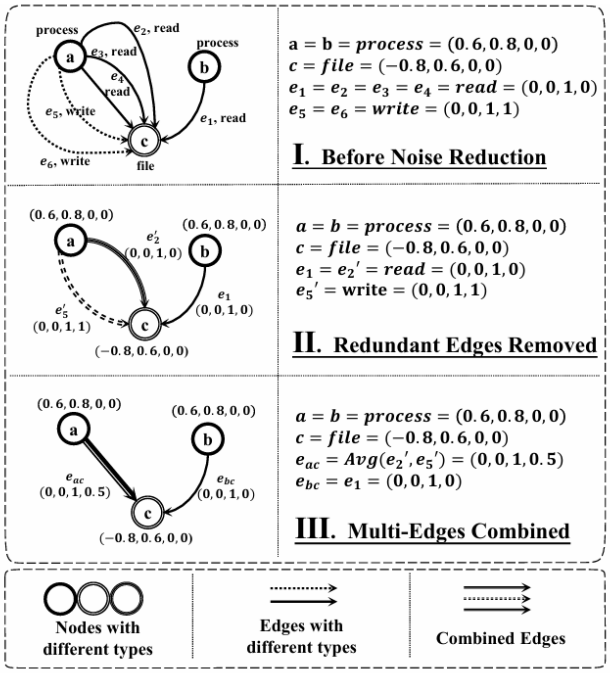
A.1. Xây dựng đồ thị nguồn gốc

MAGIC đầu tiên xây dựng một đồ thị dòng dõi từ nhật ký kiểm toán thô trước khi thực hiện biểu diễn đồ thị và phát hiện APT. MAGIC tuân theo ba bước để xây dựng một đồ thị dòng dõi nhất quán và được tối ưu hóa, sẵn sàng cho việc biểu diễn đồ thị. Phân tích cú pháp nhật ký. Bước đầu tiên là chỉ cần phân tích cú pháp từng mục nhật ký, trích xuất các thực thể hệ thống và các tương tác hệ thống giữa chúng. Sau đó, một đồ thị dòng dõi nguyên mẫu có thể được xây dựng với các thực thể hệ thống là các nút và các tương tác là các cạnh. Bây giờ ta trích xuất thông tin phân loại liên quan đến các nút và cạnh. Đối với định dạng nhật ký đơn giản cung cấp nhãn thực thể và tương tác, MAGIC sử dụng trực tiếp các nhãn này.

Đối với một số định dạng cung cấp các thuộc tính phức tạp của các thực thể và tương tác đó, MAGIC áp dụng băm đa nhãn (ví dụ: xxhash) để chuyển đổi các thuộc tính thành nhãn. Ở giai đoạn này, đồ thị dòng dõi là một đa đồ thị có hướng. Nhóm tác giả đã thiết kế một ví dụ để minh họa cách họ xử lý đồ thị dòng dõi thô sau khi phân tích cú pháp nhật ký trong Hình 3. Nhúng ban đầu. Ở giai đoạn này, họ chuyển đổi nhãn nút và cạnh thành vectơ đặc trưng có kích thước cố định (tức là nhúng ban đầu) có chiều d, trong đó d là chiều ẩn của mô-đun biểu diễn đồ thị. Họ áp dụng nhúng tra cứu, thiết lập mối quan hệ một-một giữa nhãn nút/cạnh với các vectơ đặc trưng d chiều. Như được minh họa trong Hình 3 (I và II), các quy trình a và b có cùng nhãn, do đó chúng được ánh xạ tới cùng một vectơ đặc trưng, trong khi a và c được nhúng vào các vectơ đặc trưng khác nhau vì chúng có nhãn khác nhau. Nhóm tác giả lưu ý rằng số lượng nhãn nút/cạnh duy nhất có thể có được xác định bởi nguồn dữ liệu (tức là định dạng nhật ký kiểm toán).

Do đó, nhúng tra cứu hoạt động trong một thiết lập quy nạp và không cần học các nhúng cho các nhãn chưa từng thấy. Đó là giảm nhiễu, với đầu vào dự kiến của mô-đun biểu diễn đồ thị của MAGIC là các đồ thị đơn. Do đó, cần kết hợp nhiều cạnh giữa các cặp nút. Nếu nhiều cạnh có cùng nhãn (cũng chia sẻ cùng một nhúng ban đầu) tồn tại giữa một cặp nút, loại bỏ các cạnh dư thừa để chỉ một trong số chúng còn lại. Sau đó, kết hợp các cạnh còn lại thành một cạnh cuối cùng. Lưu ý rằng giữa một cặp nút, các cạnh có nhiều nhãn khác nhau có thể vẫn còn. Sau khi kết hợp, nhúng ban đầu của cạnh duy nhất kết quả được thu được bằng cách lấy trung bình các nhúng ban đầu của các cạnh còn lại. Để minh họa cho thấy cách giảm nhiễu của việc kết hợp đa cạnh và cách nó ảnh hưởng đến các nhúng ban đầu của cạnh trong Hình 3 (II và III). Đầu tiên, ba tương tác đọc và hai tương tác ghi giữa a và c được hợp

nhất thành một cho mỗi nhãn. Sau đó, kết hợp chúng lại với nhau, tạo thành một cạnh eac với nhúng ban đầu bằng trung bình nhúng ban đầu của các cạnh còn lại (e′2 và e′5).

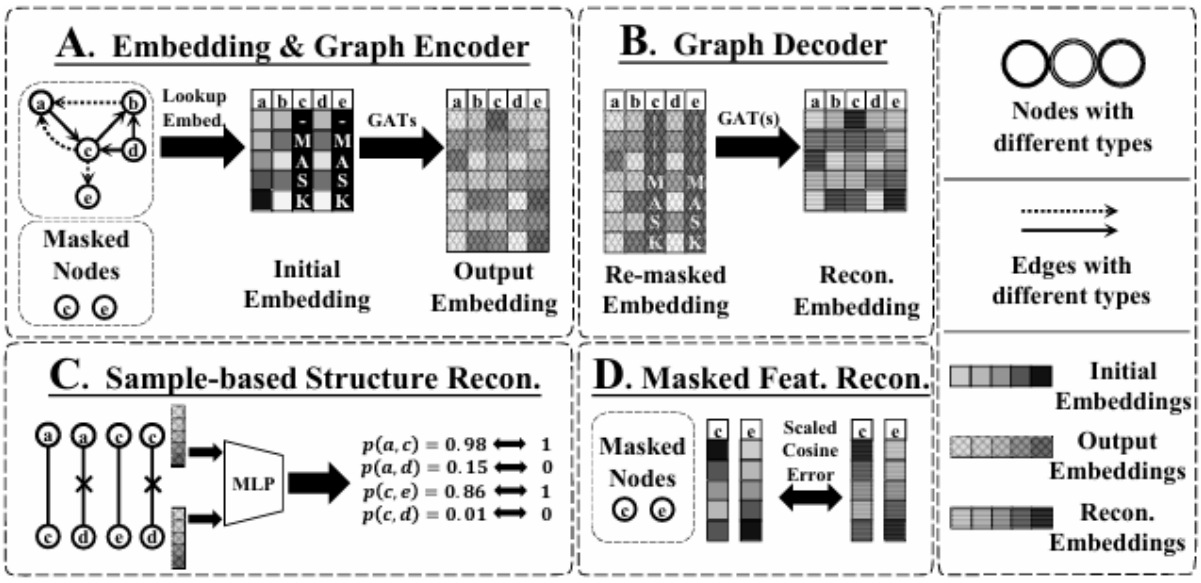


Hình 3. Ví dụ về các bước xây dựng đồ thị của MAGIC.

Sau khi thực hiện ba bước trên, **MAGIC** đã hoàn thành việc xây dựng một đồ thị dòng dõi nhất quán và bảo toàn thông tin, sẵn sàng cho các tác vụ tiếp theo. Trong quá trình xây dựng đồ thị dòng dõi, ít thông tin bị mất vì **MAGIC** chỉ làm tổn hại đến ngữ nghĩa ban đầu bằng cách khái quát hóa các mô tả chi tiết về các thực thể và tương tác hệ thống thành các nhãn. Tuy nhiên, trung bình 79,60% tổng số cạnh đã được giảm trên bộ dữ liệu DARPA E3 Trace, giúp tiết kiệm thời gian huấn luyện và mức tiêu thụ bộ nhớ của **MAGIC**.

A.2. Mô-đun biểu diễn đồ thị

MAGIC sử dụng một mô-đun biểu diễn đồ thị để thu được các nhúng chất lượng cao từ các đồ thị dòng dõi đã được đặc trưng hóa. Như được minh họa trong Hình 4, mô-đun biểu diễn đồ thị bao gồm ba giai đoạn: một quy trình che mặt để ẩn một phần các đặc trưng nút (tức là các nhúng ban đầu) cho mục đích tái cấu trúc (Phần 4.2.1), một bộ mã hóa đồ thị tạo ra các nhúng đầu ra của nút và trạng thái hệ thống bằng cách lan truyền và tổng hợp các đặc trưng đồ thị (Phần 4.2.2), một bộ giải mã đồ thị cung cấp các tín hiệu giám sát cho việc huấn luyện mô-đun biểu diễn đồ thị thông qua tái cấu trúc đặc trưng bị che mặt và tái cấu trúc cấu trúc dựa trên mẫu (Phần 4.2.3). Bộ mã hóa và giải mã tạo thành một bộ mã hóa tự động đồ thị bị che mặt, vượt trội trong việc tạo ra các nhúng nhanh và tiết kiệm tài nguyên.



Hình 4. Mô-đun biểu diễn đồ thị của MAGIC

A.2.1. Che đặc trưng

Trước khi huấn luyện mô-đun biểu diễn đồ thị, thuật toán thực hiện che mặt trên các nút, để bộ mã hóa tự động đồ thị bị che mặt có thể được huấn luyện dựa trên việc tái cấu trúc các nút này. Các nút bị che mặt được chọn ngẫu nhiên, bao phủ một tỷ lệ nhất định của tất cả các nút. Các nhúng ban đầu của các nút bị che mặt như vậy được thay thế bằng một mã thông báo mặt nạ đặc biệt xmask để che đi mọi thông tin ban đầu về các nút này. Tuy nhiên, các cạnh không bị che mặt vì các cạnh này cung cấp thông tin quý giá về mối quan hệ giữa các thực thể hệ thống. Tóm lại, với các nhúng ban đầu của nút xn, thuật toán sẽ che mặt các nút như sau:

$$emb_n = \begin{matrix} x_n, & n \notin \tilde{N} \\ x_{mask}, & n \in \tilde{N} \end{matrix}$$

trong đó $\tilde{N}$ là các nút bị che mặt được chọn ngẫu nhiên, emb_n là nhúng của nút n sẵn sàng cho việc huấn luyện mô-đun biểu diễn đồ thị. Quá trình che mặt này chỉ xảy ra trong quá trình huấn luyện. Trong quá trình phát hiện, thuật toán sẽ không che mặt bất kỳ nút nào.

A.2.2. Bộ mã hóa đồ thị

Các nhúng ban đầu thu được từ các bước xây dựng đồ thị chỉ xem xét các đặc trưng thô. Tuy nhiên, các đặc trưng thô không đủ để mô hình hóa các hành vi chi tiết của các thực thể hệ thống. Thông tin ngữ cảnh của một thực thể, chẳng hạn như vùng lân cận của nó, mối quan hệ nhiều bước và các mẫu tương tác của nó với các thực thể hệ thống khác đóng vai trò quan trọng để thu được các nhúng thực thể chất lượng cao. Ở đây, Nhóm tác giả sử dụng và mở rộng các bộ mã hóa tự động đồ thị bị che mặt để tạo ra các nhúng đầu ra theo cách tự giám

sát. Bộ mã hóa tự động đồ thị bị che mặt bao gồm một bộ mã hóa và một bộ giải mã. Bộ mã hóa tạo ra các nhúng đầu ra bằng cách lan truyền và tổng hợp các đặc trưng đồ thị, và bộ giải mã tái cấu trúc các đặc trưng đồ thị để cung cấp các tín hiệu giám sát cho việc huấn luyện. Kiến trúc bộ mã hóa-giải mã như vậy duy trì thông tin ngữ cảnh và ngữ nghĩa trong các nhúng được tạo ra, trong khi chi phí tính toán của nó giảm đáng kể thông qua học bị che mặt.

Bộ mã hóa của mô-đun biểu diễn đồ thị của MAGIC chứa nhiều lớp mạng lưới chú ý đồ thị (GAT) xếp chồng lên nhau. Chức năng của một lớp GAT là tạo ra các nhúng nút đầu ra theo cả các đặc trưng (nhúng ban đầu) của chính nút đó và các nút lân cận của nó. Khác với các GNN thông thường, GAT giới thiệu một cơ chế chú ý để đo lường tầm quan trọng của các nút lân cận đó.

Để giải thích chi tiết, một lớp GAT nhận các nhúng nút được tạo ra bởi các lớp trước đó làm đầu vào và lan truyền các nhúng từ các nút nguồn đến các nút đích thành các thông điệp dọc theo các tương tác. Thông điệp chứa thông tin về nút nguồn và tương tác giữa nguồn và đích:

$$MSG(src, dst) = W_{msg}^{T}(h_{src}||emb_{e}).$$

Và cơ chế chú ý được sử dụng để tính toán các hệ số chú ý giữa nguồn thông điệp và đích của nó:

$$\alpha(src, dst) = LeakyReLU(W_{as}^{T} h_{src} + W_{am} MSG(src, dst)),$$
$$a(src, dst) = Softmax(\alpha(src, dst)).$$

Sau đó, đối với nút đích, GAT tổng hợp các thông điệp từ các cạnh đến để cập nhật nhúng nút của nó bằng cách tính tổng có trọng số của tất cả các thông điệp đến. Trọng số chính là các hệ số chú ý:

$$AGG(h_{dst}, h_{\mathcal{N}}) = W_{self} h_{dst} + \sum_{i \in \mathcal{N}} a(i, dst) MSG(i, dst),$$
$$h_{n}^{l} = AGG^{l}(h_{n}^{l-1}, h_{\mathcal{N}_n}^{l-1}).$$

trong đó $h^l{}_n$ là nhúng ẩn của nút n ở lớp thứ l của GAT, $h^{l-1}{}_n$ là của lớp $l-1$ và N_n là vùng lân cận một bước của n. Đầu vào của lớp GAT đầu tiên là các nhúng nút ban đầu. embe là nhúng cạnh ban đầu và không đổi trong suốt mô-đun biểu diễn đồ thị. Was, Wam, Wsel f, Wmsg là các tham số có thể huấn luyện. Nhúng nút được cập nhật tạo thành một sự trừu tượng hóa chung về hành vi tương tác một bước của nút.

Nhiều lớp GAT như vậy được xếp chồng lên nhau để thu được nhúng nút cuối cùng *hn*, được nối với nhúng nút gốc và đầu ra của tất cả các lớp GAT:

$$h_{n} = emb_{n}||h_{n}^{1}||\cdots||h_{n}^{l}.$$

·||· biểu thị phép nối. Càng nhiều lớp GAT được xếp chồng lên nhau, phạm vi lân cận càng rộng và mẫu tương tác nhiều bước của một nút càng được biểu diễn xa hơn trong nhúng của

nó. Do đó, bộ mã hóa đồ thị kết hợp hiệu quả các đặc trưng ban đầu của nút và các hành vi tương tác nhiều bước để trừu tượng hóa các hành vi của thực thể hệ thống thành các nhúng nút. Bộ mã hóa đồ thị cũng áp dụng phép gộp trung bình cho tất cả các nhúng nút để tạo ra một nhúng toàn diện của chính đồ thị, tóm tắt toàn bộ trạng thái của hệ thống:

$$h_G = \frac{1}{|N|} \sum_{n_i \in N} h_{n_i}.$$

Các nhúng nút và các nhúng trạng thái hệ thống được tạo ra bởi bộ mã hóa đồ thị được coi là đầu ra của mô-đun biểu diễn đồ thị, được sử dụng trong các tác vụ tiếp theo trong các kịch bản khác nhau.

A.2.3. Bộ giải mã đồ thị

Bộ mã hóa đồ thị không cung cấp các tín hiệu giám sát hỗ trợ quá trình huấn luyện mô hình. Trong các bộ mã hóa tự động đồ thị điển hình, một bộ giải mã đồ thị được sử dụng để giải mã các nhúng nút và giám sát quá trình huấn luyện mô hình thông qua tái cấu trúc đặc trưng và tái cấu trúc cấu trúc. Tuy nhiên, các bộ mã hóa tự động đồ thị bị che mặt loại bỏ tái cấu trúc cấu trúc để giảm chi phí tính toán. Bộ giải mã đồ thị là sự kết hợp của cả hai, tích hợp tái cấu trúc đặc trưng bị che mặt và tái cấu trúc cấu trúc dựa trên mẫu để xây dựng một hàm mục tiêu tối ưu hóa mô-đun biểu diễn đồ thị.

Cho các nhúng nút *hn* thu được từ bộ mã hóa đồ thị, bộ giải mã đầu tiên che mặt lại các nút đã được che mặt và chuyển đổi chúng thành đầu vào của quá trình tái cấu trúc đặc trưng bị che mặt:

$$h_n^* = \begin{cases} W^* h_n, & n \notin \widetilde{N} \\ W^* v_{remask}, & n \in \widetilde{N} \end{cases},$$

Sau đó, bộ giải mã sử dụng một lớp GAT tương tự như đã mô tả ở trên để tái cấu trúc các nhúng ban đầu của các nút bị che mặt, cho phép tính toán một tổn thất tái cấu trúc đặc trưng:

$$x_n^* = AGG^*(h_n^*, h_{\mathcal{N}_n}^*),$$
$$L_{fr} = \frac{1}{|\widetilde{N}|} \sum_{n_i \in \widetilde{N}} (1 - \frac{x_{n_i}^T x_{n_i}^*}{||x_{n_i}|| \cdot ||x_{n_i}^*||})^\gamma.$$

trong đó L_{fr} là tổn thất tái cấu trúc đặc trưng bị che mặt thu được bằng cách tính toán tổn thất cosin được chia tỷ lệ giữa các nhúng ban đầu và được tái cấu trúc của các nút bị che mặt. Tổn thất này thay đổi đáng kể giữa các mẫu dễ và khó, giúp tăng tốc quá trình học một cách hiệu quả. Mức độ chia tỷ lệ như vậy được kiểm soát bởi một siêu tham số γ.

Trong khi đó, tái cấu trúc cấu trúc dựa trên mẫu nhằm mục đích tái cấu trúc cấu trúc đồ thị (tức là dự đoán các cạnh giữa các nút). Thay vì tái cấu trúc toàn bộ ma trận kề, có độ phức tạp O(N2), tái cấu trúc cấu trúc dựa trên mẫu áp dụng lấy mẫu đối lập trên các cặp nút và dự đoán xác suất cạnh giữa các cặp đó. Chỉ các nút không bị che mặt tham gia vào tái cấu trúc cấu trúc. Các mẫu dương được xây dựng với tất cả các cạnh hiện có giữa các nút không

bị che mặt, và các mẫu âm được lấy mẫu trong số các cặp nút không có cạnh hiện có giữa chúng.

Một MLP hai lớp đơn giản được sử dụng để tái cấu trúc các cạnh giữa các mẫu cặp nút, tạo ra một xác suất cho mỗi mẫu. Tổn thất tái cấu trúc có dạng một tổn thất entropy chéo nhị phân đơn giản trên các mẫu đó:

$$prob(n,n') = \sigma(MLP(h_n||h_{n'})),$$
$$L_{sr} = -\frac{1}{|\hat{N}|}\sum_{n\in\hat{N}}(log(1-prob(n,n^-)) + log(prob(n,n^+))).$$

trong đó $(n,n-)$ và $(n,n+)$ lần lượt là các mẫu âm và dương, và $\hat{N} = N - \tilde{N}$ là tập hợp các nút không bị che mặt. Tái cấu trúc cấu trúc dựa trên mẫu chỉ cung cấp sự giám sát cho các nhúng đầu ra. Thay vì sử dụng tích vô hướng, MAGIC sử dụng một MLP để tính toán xác suất cạnh vì các thực thể tương tác không nhất thiết phải tương tự nhau về hành vi. Ngoài ra, Nhóm tác giả không buộc mô hình phải học cách dự đoán xác suất cạnh. Chức năng của việc tái cấu trúc cấu trúc như vậy là tối đa hóa thông tin hành vi chứa trong các nhúng nút đã được trừu tượng hóa, để một MLP đơn giản là đủ để kết hợp và diễn giải thông tin đó thành xác suất cạnh.

Hàm mục tiêu cuối cùng $L = L_{fr} + L_{sr}$ kết hợp L_{fr} và L_{sr} và cung cấp các tín hiệu giám sát cho mô-đun biểu diễn đồ thị, cho phép nó học các tham số theo cách tự giám sát.

A.3. Mô-đun phát hiện

Dựa trên các nhúng đầu ra được tạo ra bởi mô-đun biểu diễn đồ thị, MAGIC sử dụng các phương pháp phát hiện ngoại lệ để thực hiện phát hiện APT theo cách không giám sát. Như đã giải thích chi tiết trong các phần trước, các nhúng này tóm tắt các hành vi hệ thống ở các mức độ chi tiết khác nhau. Mục tiêu của mô hình phát hiện là xác định các thực thể hoặc trạng thái hệ thống độc hại chỉ với kiến thức tiên nghiệm về các hành vi hệ thống lành tính. Các nhúng được tạo ra thông qua học biểu diễn đồ thị có xu hướng tạo thành các cụm nếu các thực thể tương ứng của chúng có các hành vi tương tác tương tự trong đồ thị [19, 25–27, 32]. Do đó, các ngoại lệ trong các nhúng trạng thái hệ thống cho thấy các hành vi hệ thống bất thường và đáng ngờ. Dựa trên hiểu biết sâu sắc này, tác giả phát triển một phương pháp phát hiện ngoại lệ đặc biệt để thực hiện phát hiện APT.

Trong quá trình huấn luyện, các nhúng đầu ra lành tính đầu tiên được trừu tượng hóa từ các đồ thị dòng dõi huấn luyện. Những gì mô-đun phát hiện làm ở giai đoạn này chỉ đơn giản là ghi nhớ các nhúng đó và sắp xếp chúng trong một Cây K-D. Sau khi huấn luyện, mô-đun phát hiện tiết lộ các ngoại lệ trong ba bước: tìm kiếm k-láng giềng gần nhất, tính toán độ tương tự và lọc. Với một nhúng mục tiêu, mô-đun phát hiện đầu tiên thu được k-láng giềng gần nhất của nó thông qua tìm kiếm Cây K-D. Quá trình tìm kiếm như vậy chỉ mất thời gian log(N), trong đó N là tổng số nhúng huấn luyện đã ghi nhớ. Sau đó, một tiêu chí tương tự được áp dụng để đánh giá độ gần gũi của nhúng mục tiêu với các láng giềng của nó và tính

toán một điểm bất thường. Nếu điểm bất thường của nó cao hơn một siêu tham số θ, nhúng mục tiêu được coi là một ngoại lệ và thực thể hệ thống hoặc trạng thái hệ thống tương ứng của nó là độc hại. Một quy trình làm việc ví dụ của mô-đun phát hiện được chính thức hóa như sau, sử dụng khoảng cách Euclid làm tiêu chí tương tự:

$$\mathcal{N}_x = KNN(x)$$
$$dist_x = \frac{1}{|\mathcal{N}_x|} \sum_{x_i \in \mathcal{N}_x} ||x - x_i||$$
$$score_x = \frac{dist_x}{\overline{dist}}$$
$$result_x = \begin{cases} 1, & score_x \geq \theta \\ 0, & score_x < \theta \end{cases}$$

trong đó dist là khoảng cách trung bình giữa các nhúng huấn luyện và k-láng giềng gần nhất của chúng. Khi thực hiện phát hiện cấp độ nhật ký theo lô, mô-đun phát hiện ghi nhớ các nhúng trạng thái hệ thống lành tính phản ánh trạng thái hệ thống và phát hiện xem nhúng trạng thái hệ thống của một đồ thị dòng dõi mới đến có phải là một ngoại lệ hay không. Khi thực hiện phát hiện cấp độ thực thể hệ thống, mô-đun phát hiện thay vào đó ghi nhớ các nhúng nút lành tính chỉ ra hành vi của thực thể hệ thống và khi có một đồ thị dòng dõi mới đến, nó phát hiện các ngoại lệ trong các nhúng của tất cả các thực thể hệ thống.

A.4. Cơ chế thích ứng mô hình

Để một bộ phát hiện APT hoạt động hiệu quả trong các kịch bản phát hiện thực tế, cần phải xem xét đến sự trôi dạt khái niệm. Khi đối mặt với các hành vi hệ thống lành tính nhưng chưa từng thấy trước đây, **MAGIC** tạo ra kết quả phát hiện dương tính giả, điều này có thể gây hiểu lầm cho các ứng dụng tiếp theo (ví dụ: điều tra tấn công và khôi phục câu chuyện). Các công trình gần đây giải quyết vấn đề này bằng cách quên đi dữ liệu lỗi thời hoặc điều chỉnh mô hình của chúng cho phù hợp với những thay đổi hệ thống lành tính thông qua cơ chế thích ứng mô hình. **MAGIC** cũng tích hợp cơ chế thích ứng mô hình để chống lại sự trôi dạt khái niệm và học hỏi từ các dương tính giả được xác định bởi các nhà phân tích bảo mật. Hơi khác so với các công trình khác chỉ sử dụng dương tính giả để huấn luyện lại mô hình, **MAGIC** có thể được huấn luyện lại với tất cả các phản hồi. Như đã thảo luận trong các phần trước, mô-đun biểu diễn đồ thị trong **MAGIC** mã hóa các thực thể hệ thống thành các nhúng theo cách tự giám sát, mà không cần biết nhãn của nó. Bất kỳ dữ liệu chưa từng thấy nào, bao gồm cả các âm tính thật, đều là dữ liệu huấn luyện có giá trị cho mô-đun biểu diễn đồ thị để nâng cao khả năng biểu diễn của nó trên các hành vi hệ thống chưa từng thấy.

Mô-đun phát hiện chỉ có thể được huấn luyện lại với các phản hồi lành tính để theo kịp những thay đổi trong hành vi hệ thống. Và khi nó ghi nhớ ngày càng nhiều phản hồi lành tính, hiệu quả phát hiện của nó sẽ giảm xuống. Để giải quyết vấn đề này, Nhóm tác giả cũng triển khai một cơ chế giảm giá trên mô-đun phát hiện. Khi khối lượng các nhúng đã ghi nhớ vượt

quá một lượng nhất định, các nhúng sớm nhất sẽ đơn giản bị loại bỏ khi các nhúng mới đến được học. MAGIC cung cấp cơ chế thích ứng mô hình như một giải pháp tùy chọn cho sự trôi dạt khái niệm và các hành vi hệ thống chưa từng thấy. Nên điều chỉnh **MAGIC** cho phù hợp với những thay đổi của hệ thống bằng cách cung cấp các mẫu dương tính giả đã được xác nhận cho cơ chế thích ứng mô hình của **MAGIC**.

B. CHI TIẾT CÀI ĐẶT, HIỆN THỰC

B.1. Cấu hình máy tính và môi trường

- **Hệ điều hành**: Ubuntu
- **Phần cứng**:

+ Mô hình được huấn luyện bằng cách sử dụng **một GPU NVIDIA Tesla T4 của Google Colab 12GB vRAM**.

+ Ngoài ra, sau khi huấn luyện xong, việc sử dụng mô hình sẽ được thực hiện tại Local, với 1 GPU **Nvidia RTX 3050 6GB vRAM**

- **Phiên bản Python**: Python 3.8.10
- **Thư viện**: Việc triển khai sử dụng thư viện **PyTorch (>=1.12.1), DGL (>=1.0.0), Scikit-learn (>=1.2.2)** và các thư viện bổ trợ khác của Python, xem Requirements

B.2. Chuẩn bị dữ liệu

B.2.1. Bộ dữ liệu

Nhóm đánh giá hiệu quả của **MAGIC** trên ba bộ dữ liệu công khai: bộ dữ liệu StreamSpot, bộ dữ liệu Unicorn Wget và bộ dữ liệu DARPA Engagement 3. Các bộ dữ liệu này khác nhau về dung lượng, nguồn gốc và độ chi tiết. Bằng cách kiểm tra hiệu suất của **MAGIC** trên các bộ dữ liệu này, ta có thể so sánh **MAGIC** với càng nhiều phương pháp phát hiện APT hiện đại càng tốt và khám phá tính phổ quát và khả năng ứng dụng của **MAGIC**. Mô tả chi tiết về ba bộ dữ liệu như sau.

Dataset	Scenario	Malicious	#Log pieces	Avg. #Entity	Avg. #Interaction	Size(GB)
StreamSpot	CNN		100	8,989	294,903	0.9
	Download		100	8,830	310,814	1.0
	Gmail		100	6,826	37,382	0.1
	VGame		100	8,636	112,958	0.4
	YouTube		100	8,292	113,229	0.3
	Attack	✓	100	8,890	28,423	0.1
Unicorn Wget	Benign		125	265,424	975,226	64.0
	Attack	✓	25	257,156	949,887	12.6

Bảng 1. Bộ dữ liệu cho phát hiện cấp độ nhật ký theo lô.

Dataset	Scenario	Malicious	#Node	#Edge	Size (GB)
DARPA E3 Trace	Benign		3,220,594	4,080,457	15.40
	Extension Backdoor	✓	732		
	Pine Backdoor	✓	67,345		
	Phishing Executable	✓	5		
DARPA E3 THEIA	Benign		1,598,647	2,874,821	17.91
	Attack	✓	25,319		
DARPA E3 CADETS	Benign		1,614,189	3,303,264	18.38
	Attack	✓	12,846		

Bảng 2. Bộ dữ liệu cho phát hiện cấp độ thực thể hệ thống.

B.2.1.1. Bộ dữ liệu StreamSpot

Bộ dữ liệu StreamSpot (Xem Bảng 1) là một bộ dữ liệu mô phỏng được thu thập và công khai bởi StreamSpot bằng hệ thống kiểm toán SystemTap. Bộ dữ liệu StreamSpot chứa 600 lô nhật ký kiểm toán giám sát các lệnh gọi hệ thống trong 6 kịch bản duy nhất. Năm trong số các kịch bản đó mô phỏng hành vi người dùng lành tính, trong khi kịch bản tấn công mô phỏng một cuộc tấn công drive-by-download. Bộ dữ liệu được coi là một bộ dữ liệu tương đối nhỏ và vì không có nhãn của các mục nhật ký và các thực thể hệ thống nào được cung cấp, nhóm sẽ thực hiện phát hiện ở cấp độ nhật ký theo lô trên bộ dữ liệu StreamSpot tương tự như các công trình trước đây.

B.2.1.2. Bộ dữ liệu Unicorn Wget

Bộ dữ liệu Unicorn Wget (Xem Bảng 1) chứa các cuộc tấn công mô phỏng được thiết kế bởi Unicorn. Cụ thể, nó chứa 150 lô nhật ký được thu thập bằng Camflow, trong đó 125 lô lành tính và 25 lô chứa các cuộc tấn công chuỗi cung ứng. Các cuộc tấn công này, được phân loại là các cuộc tấn công lén lút, được thiết kế công phu để có hành vi tương tự như quy trình làm việc lành tính của hệ thống và được cho là khó xác định. Bộ dữ liệu này được coi là khó nhất trong số các bộ dữ liệu thử nghiệm của dự án do dung lượng lớn, định dạng nhật ký phức tạp và tính chất lén lút của các cuộc tấn công này. Tương tự như các phương pháp hiện đại, nhóm thực hiện phát hiện ở cấp độ nhật ký theo lô trên bộ dữ liệu này.

B.2.1.3. Bộ dữ liệu DARPA E3

Bộ dữ liệu DARPA Engagement 3 (Xem Bảng 2), là một phần của chương trình DARPA Transparent Computing, được thu thập từ một mạng doanh nghiệp trong một cuộc đối đầu. Các cuộc tấn công APT khai thác các lỗ hổng khác nhau được thực hiện bởi đội đỏ để đánh cắp thông tin nhạy cảm. Các đội xanh cố gắng xác định các cuộc tấn công này bằng cách kiểm tra các máy chủ mạng và thực hiện phân tích nhân quả trên chúng. Các bộ dữ liệu con là Trace, THEIA và CADETS. Ba bộ dữ liệu con này bao gồm tổng cộng 51,69GB bản ghi kiểm toán, chứa tới 6.539.677 thực thể hệ thống và 68.127.444 tương tác. Do đó, Nhóm đánh giá khả năng phát hiện ở cấp độ thực thể hệ thống của **MAGIC** và giải quyết vấn đề chi phí trên các bộ dữ liệu này.

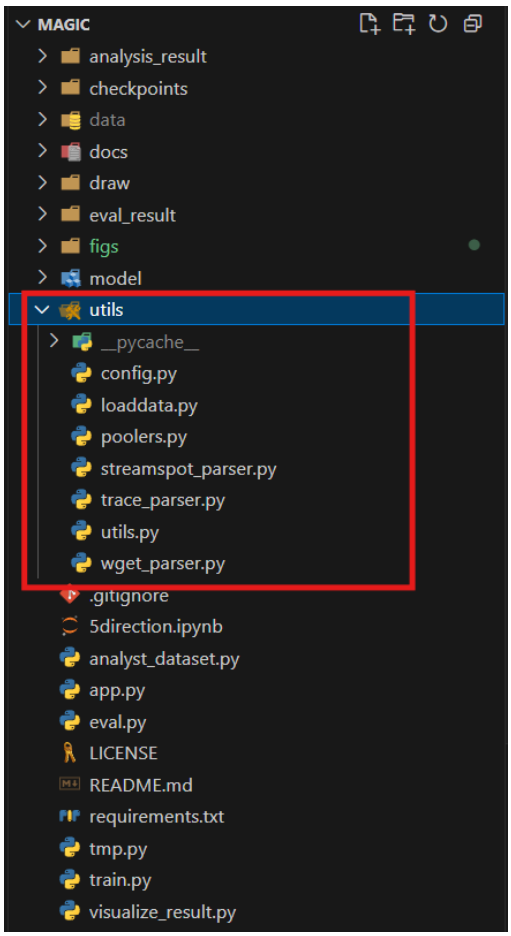
B.2.1.4. Bộ dữ liệu FiveDirections

Bộ dữ liệu DARPA E3 FiveDirections là một phần của Engagement #3 trong chương trình DARPA Transparent Computing, do nhóm FiveDirections—một trong các thực hiện TA1—thu thập và công bố để hỗ trợ nghiên cứu phát hiện APT và phân tích pháp y, mặc dù mang tính nguyên mẫu và có thể còn sai sót; dữ liệu chủ yếu nằm trong tệp ta1-fivedirections-e3-official (đã loại bỏ bản ghi lỗi cú pháp), thường ở dạng nhị phân nén hoặc tuần tự hóa, tương thích với nền tảng xử lý luồng như Kafka, và được tổ chức trong thư mục fivedirections

của kho lưu trữ chung cùng CADETS, ClearScope…, chứa event logs và telemetry; kịch bản tấn công mô phỏng chiến dịch APT với các giai đoạn truy cập ban đầu qua lỗ hổng hoặc kỹ thuật xã hội, di chuyển ngang trong mạng, và duy trì hiện diện lâu dài để thử nghiệm khả năng phát hiện, truy vết và phản ứng của hệ thống bảo mật.

B.2.2. Làm sạch dữ liệu

Dữ liệu trong dự án được làm sạch (data cleaning) chủ yếu thông qua các script Parser tương ứng từng bộ dữ liệu, nằm trong thư mục "utils/" (ví dụ: "streamspot_parser.py", "trace_parser.py", "wget_parser.py"…). Quy trình tổng quát:



- Đọc/log dữ liệu gốc từ thư mục "data/<dataset>" (thường ở dạng file .zip, .log, .csv...).

- Áp dụng các bước chuẩn hoá: – Bỏ bớt hoặc bỏ qua các dòng thiếu trường quan trọng (ví dụ thiếu IP, timestamp...).

- Loại toàn bộ dòng trùng lặp, hoặc sự kiện lặp (duplicate) theo điều kiện cài đặt trong Parser.

- Thống nhất định dạng thời gian, chuyển UTC sang timestamp hoặc nhất quán time-zone nếu cần.

Xác định nhãn (label) của sự kiện (malicious/benign), hoặc danh mục sự kiện nếu bộ dữ liệu có. Tạo cấu trúc "graph" hoặc "node-edge" (tùy dataset) bằng cách gom chung các sự kiện liên quan, liên kết theo session, thread, IP…

Nén hoặc đóng gói dữ liệu sau khi làm sạch dưới dạng file "graphs.pkl" (hoặc tương đương).

Ví dụ, với "streamspot_parser.py":

1. Giải nén "graphs.zip" trong "data/streamspot/".

2. Đọc file CSV/log, loại bỏ bản ghi rỗng hoặc không hợp lệ.

3. Chuẩn hoá thành "Node", "Edge" (ai nối với ai, timestamp...), rồi cuối cùng lưu vào "graphs.pkl".

```
'''
Module này được sử dụng để phân tích và chuyển đổi dữ liệu từ định dạng tsv sang
định dạng json cho các đồ thị trong dự án StreamSpot.
'''

import networkx as nx
from tqdm import tqdm
import json
raw_path = '../data/streamspot/'

NUM_GRAPHS = 600 # Số lượng đồ thị tối đa
node_type_dict = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'] # Danh sách các loại nút
edge_type_dict = ['i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
                  'q', 't', 'u', 'v', 'w', 'y', 'z', 'A', 'C', 'D', 'E', 'G'] #
Danh sách các loại cạnh
node_type_set = set(node_type_dict) # Tập hợp các loại nút
edge_type_set = set(edge_type_dict) # Tập hợp các loại cạnh

count_graph = 0 # Biến đếm số lượng đồ thị đã được xử lý
with open(raw_path + 'all.tsv', 'r', encoding='utf-8') as f: # Mở tệp tsv chứa dữ
liệu
    lines = f.readlines() # Đọc tất cả các dòng trong tệp
    g = nx.DiGraph() # Tạo một đồ thị có hướng
    node_map = {} # Từ điển ánh xạ các nút
    count_node = 0 # Biến đếm số lượng nút đã được xử lý
    for line in tqdm(lines): # Duyệt qua từng dòng trong tệp
        src, src_type, dst, dst_type, etype, graph_id =
line.strip('\n').split('\t') # Tách các trường trong dòng
        graph_id = int(graph_id) # Chuyển đổi id đồ thị thành số nguyên
        if src_type not in node_type_set or dst_type not in node_type_set: # Kiểm
tra loại nút
            continue # Nếu loại nút không hợp lệ, bỏ qua dòng này
        if etype not in edge_type_set: # Kiểm tra loại cạnh
            continue # Nếu loại cạnh không hợp lệ, bỏ qua dòng này
        if graph_id != count_graph: # Kiểm tra xem có phải là đồ thị mới không
            count_graph += 1 # Tăng biến đếm đồ thị
```

```
            for n in g.nodes(): # Duyệt qua tất cả các nút trong đồ thị
                g.nodes[n]['type'] = node_type_dict.index(g.nodes[n]['type']) #
Chuyển đổi loại nút thành chỉ số
            for e in g.edges(): # Duyệt qua tất cả các cạnh trong đồ thị
                g.edges[e]['type'] = edge_type_dict.index(g.edges[e]['type']) #
Chuyển đổi loại cạnh thành chỉ số
            f1 = open(raw_path + str(count_graph) + '.json', 'w', encoding='utf-8')
# Mở tệp json để lưu đồ thị
            json.dump(nx.node_link_data(g), f1) # Lưu đồ thị vào tệp json
            assert graph_id == count_graph # Kiểm tra xem id đồ thị có đúng không
            g = nx.DiGraph() # Tạo một đồ thị mới
            count_node = 0 # Đặt lại biến đếm nút
        if src not in node_map: # Kiểm tra xem nút nguồn đã được ánh xạ chưa
            node_map[src] = count_node # Nếu chưa, ánh xạ nút nguồn
            g.add_node(count_node, type=src_type) # Thêm nút nguồn vào đồ thị
            count_node += 1 # Tăng biến đếm nút
        if dst not in node_map: # Kiểm tra xem nút đích đã được ánh xạ chưa
            node_map[dst] = count_node # Nếu chưa, ánh xạ nút đích
            g.add_node(count_node, type=dst_type) # Thêm nút đích vào đồ thị
            count_node += 1 # Tăng biến đếm nút
        if not g.has_edge(node_map[src], node_map[dst]): # Kiểm tra xem cạnh đã tồn
tại chưa
            g.add_edge(node_map[src], node_map[dst], type=etype) # Nếu chưa, thêm
cạnh vào đồ thị
    count_graph += 1 # Tăng biến đếm đồ thị
    for n in g.nodes(): # Duyệt qua tất cả các nút trong đồ thị
        g.nodes[n]['type'] = node_type_dict.index(g.nodes[n]['type']) # Chuyển đổi
loại nút thành chỉ số
    for e in g.edges(): # Duyệt qua tất cả các cạnh trong đồ thị
        g.edges[e]['type'] = edge_type_dict.index(g.edges[e]['type']) # Chuyển đổi
loại cạnh thành chỉ số
    f1 = open(raw_path + str(count_graph) + '.json', 'w', encoding='utf-8') # Mở
tệp json để lưu đồ thị
    json.dump(nx.node_link_data(g), f1) # Lưu đồ thị vào tệp json
```

Tương tự, những parser khác cũng trích xuất log, bỏ trùng lặp, ghi chú link đích, xác định node (URL, host...) trước khi lưu lại. Sau quá trình này, phần lớn dữ liệu "nhiễu" sẽ bị loại bỏ, giúp tối ưu chất lượng huấn luyện và đánh giá.

B.3. Sử dụng dữ liệu

Đối với các bộ dữ liệu khác nhau, nhóm sẽ sử dụng các cách chia bộ dữ liệu khác nhau để đánh giá mô hình và chỉ sử dụng các mẫu lành tính để huấn luyện. Đối với bộ dữ liệu StreamSpot, nhóm sẽ chọn ngẫu nhiên 400 lô trong số 500 lô lành tính để huấn luyện và phần còn lại để kiểm tra, dẫn đến một bộ kiểm tra cân bằng. Đối với bộ dữ liệu Unicorn Wget, 100 lô lành tính được chọn để huấn luyện. trong khi phần còn lại dành cho việc kiểm tra. Đối với bộ dữ liệu DARPA E3, sử dụng cùng nhãn ground-truth như ThreaTrace và chia các mục nhật

ký theo thứ tự xuất hiện của chúng. 80% các mục nhật ký sớm nhất được sử dụng để huấn luyện, trong khi phần còn lại được giữ lại để kiểm tra. Trong quá trình đánh giá, hiệu suất trung bình của **MAGIC** dưới 100 hạt giống ngẫu nhiên toàn cầu được báo cáo là kết quả cuối cùng, do đó các kết quả thực nghiệm có thể chứa các phần của thực thể hệ thống/lô nhật ký.

C. KẾT QUẢ THỰC NGHIỆM

Xem Video Deomo Tại: Video Demo - Google Drive

C.1. Kịch bản 1: MAGIC trên 5 dataset gốc (streamspot, wget, trace, theia, cadets)

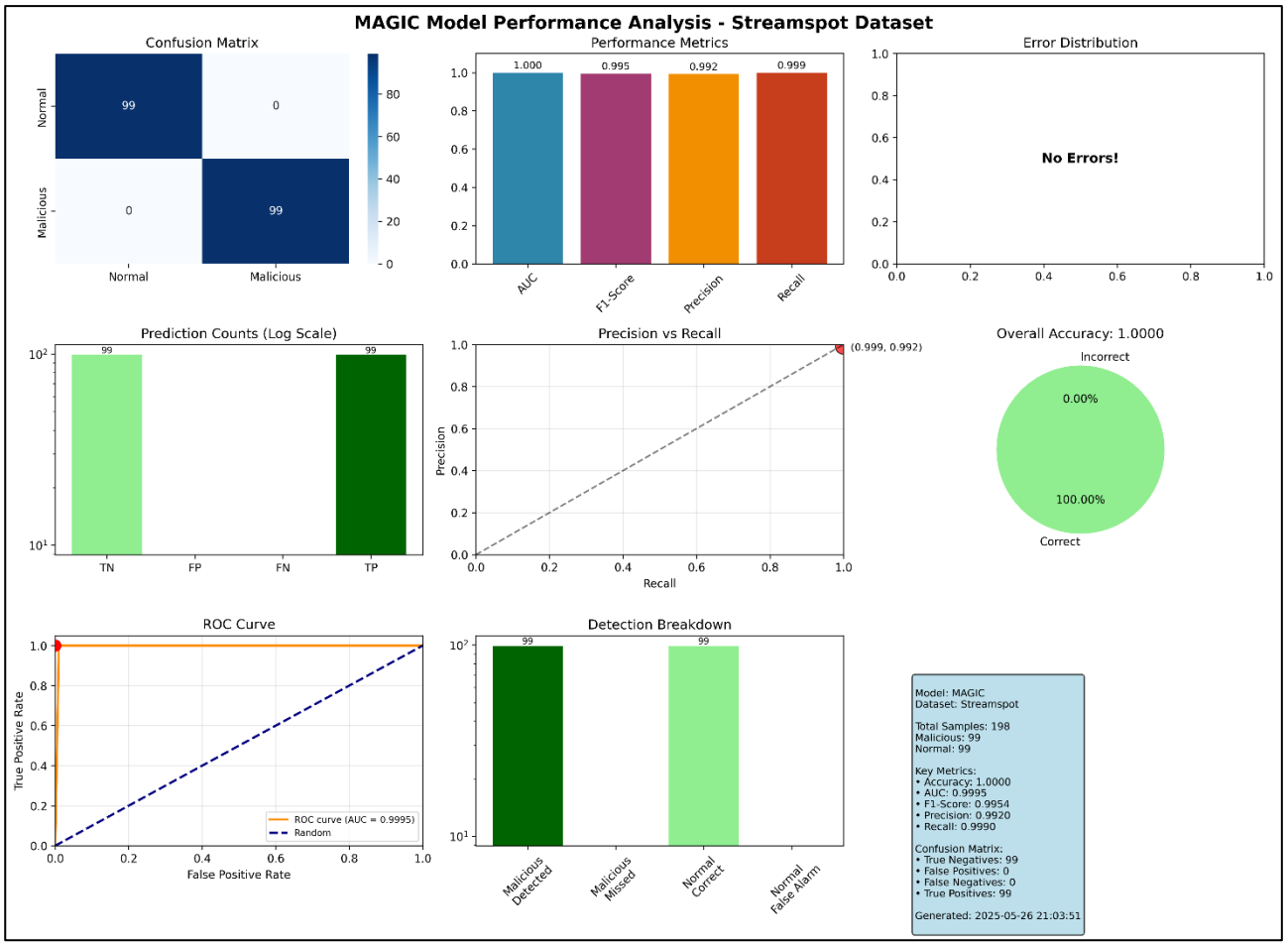
Dataset	#Test	AUC (±std)	F1 (±std)	Precision (±std)	Recall (±std)
streamspot	100	0.9995 ± 0.0007	0.9954 ± 0.0065	0.9920 ± 0.0113	0.9990 ± 0.0070
wget	25	0.9739 ± 0.0190	0.9436 ± 0.0224	0.9139 ± 0.0434	0.9776 ± 0.0317
trace	5	0.9998 ± 0.0000	0.9957	0.9917	0.9998
theia	1	0.9987 ± 0.0000	0.9911	0.9823	0.99996
cadets	1	0.9977 ± 0.0000	0.9701	0.9441	0.9977
fivedirections	7	0.7539 ± 0.0000	0.0	1.0	0.0

> **Ghi chú:** với fivedirections, F1 = 0, vì model không bắt được true positives nào.

Thực nghiệm 1: MAGIC trên 5 dataset gốc (streamspot, wget, trace, theia, cadets) đều đạt hiệu suất rất cao với AUC > 0.97, trung bình ~0.99; F1-score dao động 0.94–0.99; Precision & Recall luôn trên 0.91 và 0.97. Điều này khẳng định khả năng phân biệt bất thường rất tốt của MAGIC trong cả kịch bản batch-level và entity-level.

Thực nghiệm 2: MAGIC trên dataset tự thêm (fivedirections) cho thấy AUC ~ 0.75 nhưng Recall = 0 ⇒ mô hình chưa bắt được node độc hại nào, mặc dù Precision = 1.0. Kết quả này chỉ ra sự khác biệt lớn về phân phối đặc trưng của FiveDirections so với các dataset trong bài báo, và nhấn mạnh nhu cầu tinh chỉnh masking strategy hoặc thêm bước self-supervision phù hợp để mô hình có thể generalize tốt hơn trên data "thô" thực tế.

C.1.1. Streamspot



- Ma trận nhầm lẫn (Confusion Matrix) cho thấy khả năng phân loại của mô hình đối với hai lớp: "Normal" (Bình thường) và "Malicious" (Độc hại)

- True Negatives (TN): 99 (Dự đoán đúng "Normal" khi thực tế là "Normal")
- False Positives (FP): 0 (Dự đoán sai "Malicious" khi thực tế là "Normal")
- False Negatives (FN): 0 (Dự đoán sai "Normal" khi thực tế là "Malicious")
- True Positives (TP): 99 (Dự đoán đúng "Malicious" khi thực tế là "Malicious")

=> Mô hình đạt được phân loại hoàn hảo: không có lỗi dự đoán nào (FP = 0, FN = 0), tất cả cả 99 mẫu "Normal" và 99 mẫu "Malicious" đều được phân loại chính xác

- AUC = 1.000: Mô hình có khả năng phân biệt hoàn hảo giữa "Normal" và "Malicious"

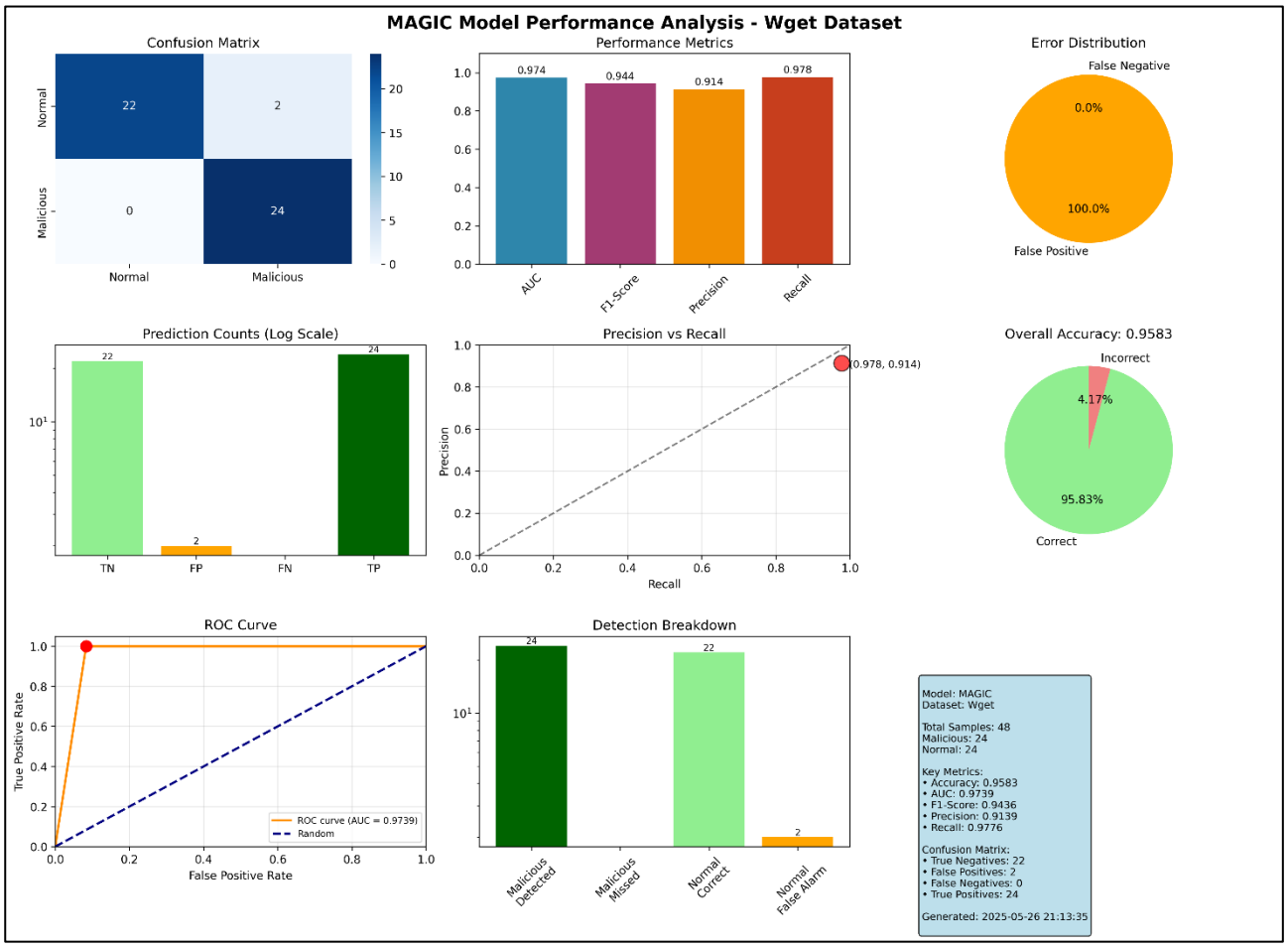
- F1-Score = 0.995: Sự cân bằng giữa Precision và Recall rất cao, cho thấy mô hình vừa chính xác vừa bao quát tốt.

- Precision = 0.992: 99.2% các dự đoán "Malicious" là đúng, chỉ có một tỷ lệ rất nhỏ sai sót (nếu có).

- Recall = 0.999: Mô hình phát hiện được 99.9% các mẫu "Malicious", gần như không bỏ sót trường hợp nào

➔ Mô hình đạt được phân loại hoàn hảo trên bộ dữ liệu Streamspot mà không có bất kỳ lỗi nào. Tất cả các mẫu Normal và Malicious đều được phân loại chính xác, thể hiện qua các chỉ số AUC, Precision, và Recall gần như hoàn hảo

C.1.2. Wget



- Ma trận nhầm lẫn (Confusion Matrix) cho thấy khả năng phân loại của mô hình đối với hai lớp: "Normal" (Bình thường) và "Malicious" (Độc hại), giảm 1 xíu so với tệp dữ liệu Streamspot

- True Negatives (TN): 22 (Dự đoán đúng "Normal" khi thực tế là "Normal")
- False Positives (FP): 2 (Dự đoán sai "Malicious" khi thực tế là "Normal")
- False Negatives (FN): 0 (Dự đoán sai "Normal" khi thực tế là "Malicious")
- True Positives (TP): 24 (Dự đoán đúng "Malicious" khi thực tế là "Malicious")

=> Mô hình đã phân loại đúng 22 mẫu "Normal" và 24 mẫu "Malicious", có 2 mẫu "Normal" bị phân loại sai thành "Malicious" (FP = 2) nhưng không có mẫu "Malicious" nào bị bỏ sót (FN = 0) là rất quan trọng

- AUC = 0.974: Mô hình có khả năng phân biệt tốt giữa "Normal" và "Malicious"

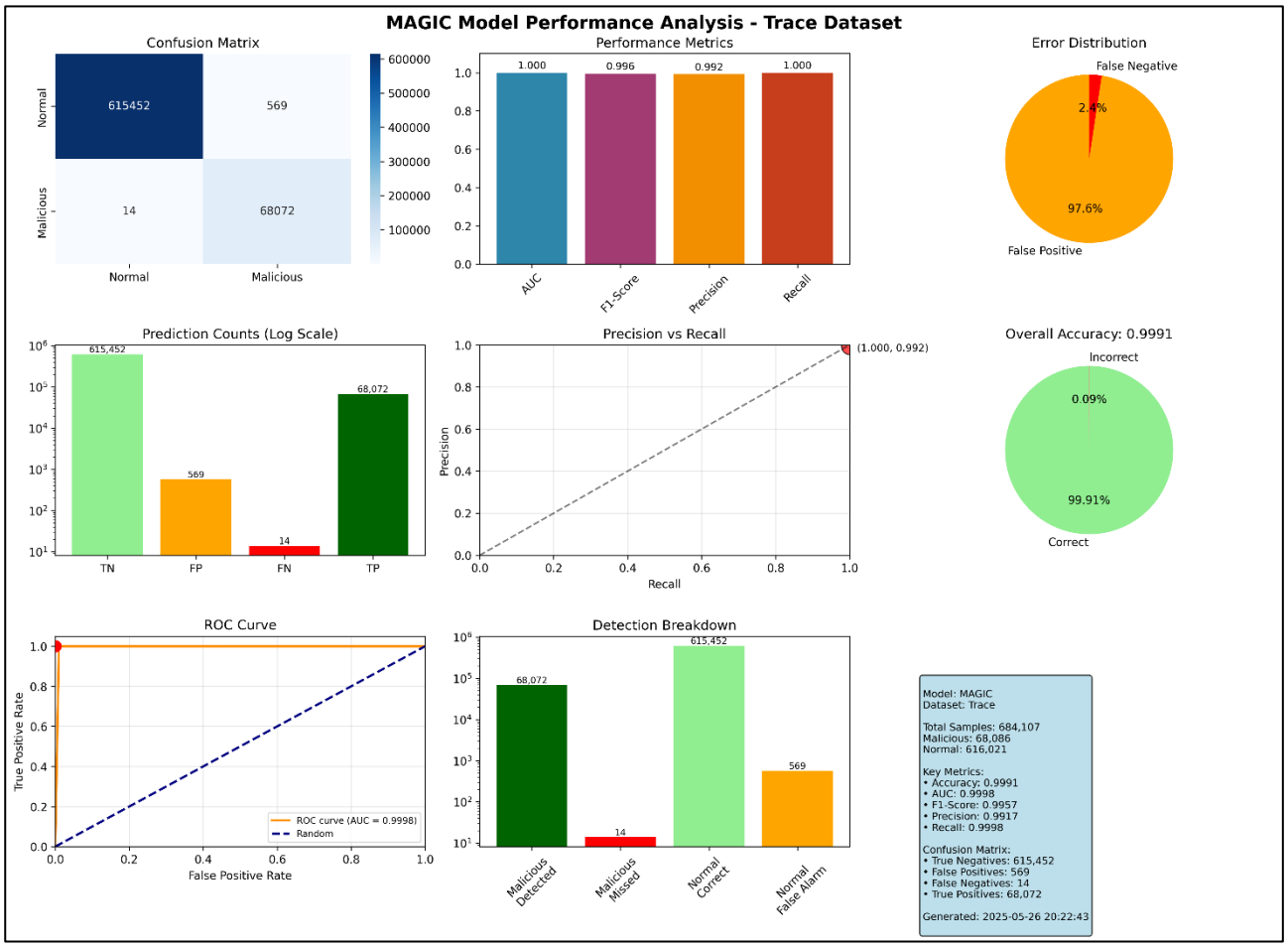
- F1-Score = 0.944: Sự cân bằng giữa Precision và Recall rất tốt, cho thấy mô hình vừa chính xác vừa bao quát

- Precision = 0.914: 91.4% các dự đoán "Malicious" là đúng, tức là có một số sai sót nhỏ (2 FP)

- Recall = 0.978: Mô hình phát hiện được 97.8% các mẫu "Malicious", gần như không bỏ sót trường hợp nào

➔ Mô hình hoạt động tốt trên bộ dữ liệu Wget với độ chính xác cao và không có False Negatives (không bỏ sót mẫu Malicious nào). Tuy nhiên, có 2 False Positives (dự đoán sai mẫu Normal thành Malicious), dẫn đến Precision thấp hơn (0.914). Điều này cho thấy mô hình có xu hướng báo động giả ở mức độ nhất định

C.1.3. Darpa TRACE



- Ma trận nhầm lẫn (Confusion Matrix) cho thấy khả năng phân loại của mô hình đối với hai lớp: "Normal" (Bình thường) và "Malicious" (Độc hại)

- True Negatives (TN): 614,552 (Dự đoán đúng "Normal" khi thực tế là "Normal")

- False Positives (FP): 569 (Dự đoán sai "Malicious" khi thực tế là "Normal")
- False Negatives (FN): 14 (Dự đoán sai "Normal" khi thực tế là "Malicious")
- True Positives (TP): 68072 (Dự đoán đúng "Malicious" khi thực tế là "Malicious")

=> Mô hình hoạt động tốt trên bộ dữ liệu Darpa Trace với độ chính xác cao, tuy vẫn xuất hiện False Negatives và False Positive. Tuy nhiên, tỉ lệ chiếm trên bộ dataset rất thấp. Điều này cho thấy mô hình vẫn hoạt động rất tốt dù vẫn còn chưa hoàn hảo

- AUC = 1: Mô hình có khả năng phân biệt hoàn hảo giữa "Normal" và "Malicious"

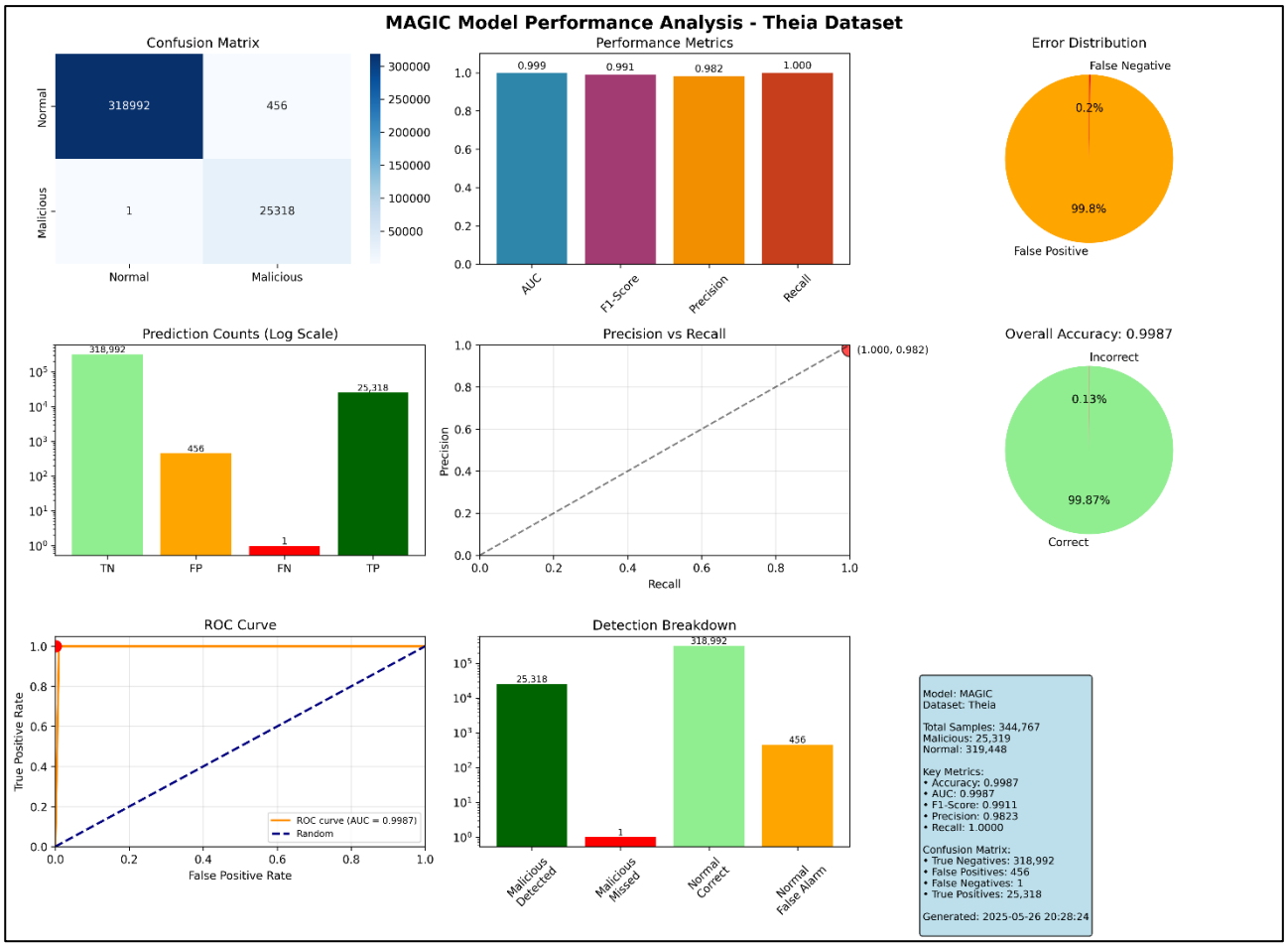
- F1-Score = 0.996: Sự cân bằng giữa Precision và Recall rất cao, cho thấy mô hình vừa chính xác vừa bao quát tốt.

- Precision = 0.992: 99.2% các dự đoán "Malicious" là đúng, chỉ có một tỷ lệ rất nhỏ sai sót (nếu có).

- Recall = 1: Mô hình phát hiện được 99.9% các mẫu "Malicious", gần như không bỏ sót trường hợp nào

➔ Trên bộ dữ liệu lớn và không cân bằng như Trace, mô hình vẫn thể hiện hiệu suất xuất sắc với độ chính xác 99.91% và Recall gần như hoàn hảo (0.9998), chỉ bỏ sót 14 mẫu Malicious. Mặc dù có 569 False Positives, tỷ lệ này rất nhỏ so với tổng số mẫu Normal (chỉ khoảng 0.09%), cho thấy mô hình quản lý tốt việc giảm thiểu báo động giả

C.1.4. Darpa THEIA



- Ma trận nhầm lẫn (Confusion Matrix) cho thấy khả năng phân loại của mô hình đối với hai lớp: "Normal" (Bình thường) và "Malicious" (Độc hại)

- True Negatives (TN): 318992 (Dự đoán đúng "Normal" khi thực tế là "Normal")
- False Positives (FP): 456 (Dự đoán sai "Malicious" khi thực tế là "Normal")
- False Negatives (FN): 1 (Dự đoán sai "Normal" khi thực tế là "Malicious")
- True Positives (TP): 25318 (Dự đoán đúng "Malicious" khi thực tế là "Malicious")

=> Mô hình phân loại chính xác 318,992 mẫu "Normal" và 25,318 mẫu "Malicious", có 456 mẫu "Normal" bị phân loại sai thành "Malicious" (FP) và chỉ 1 mẫu "Malicious" bị bỏ sót (FN), cho thấy hiệu suất rất cao với tỷ lệ lỗi rất thấp

- AUC = 0.999: Mô hình có khả năng phân biệt hoàn hảo giữa "Normal" và "Malicious"

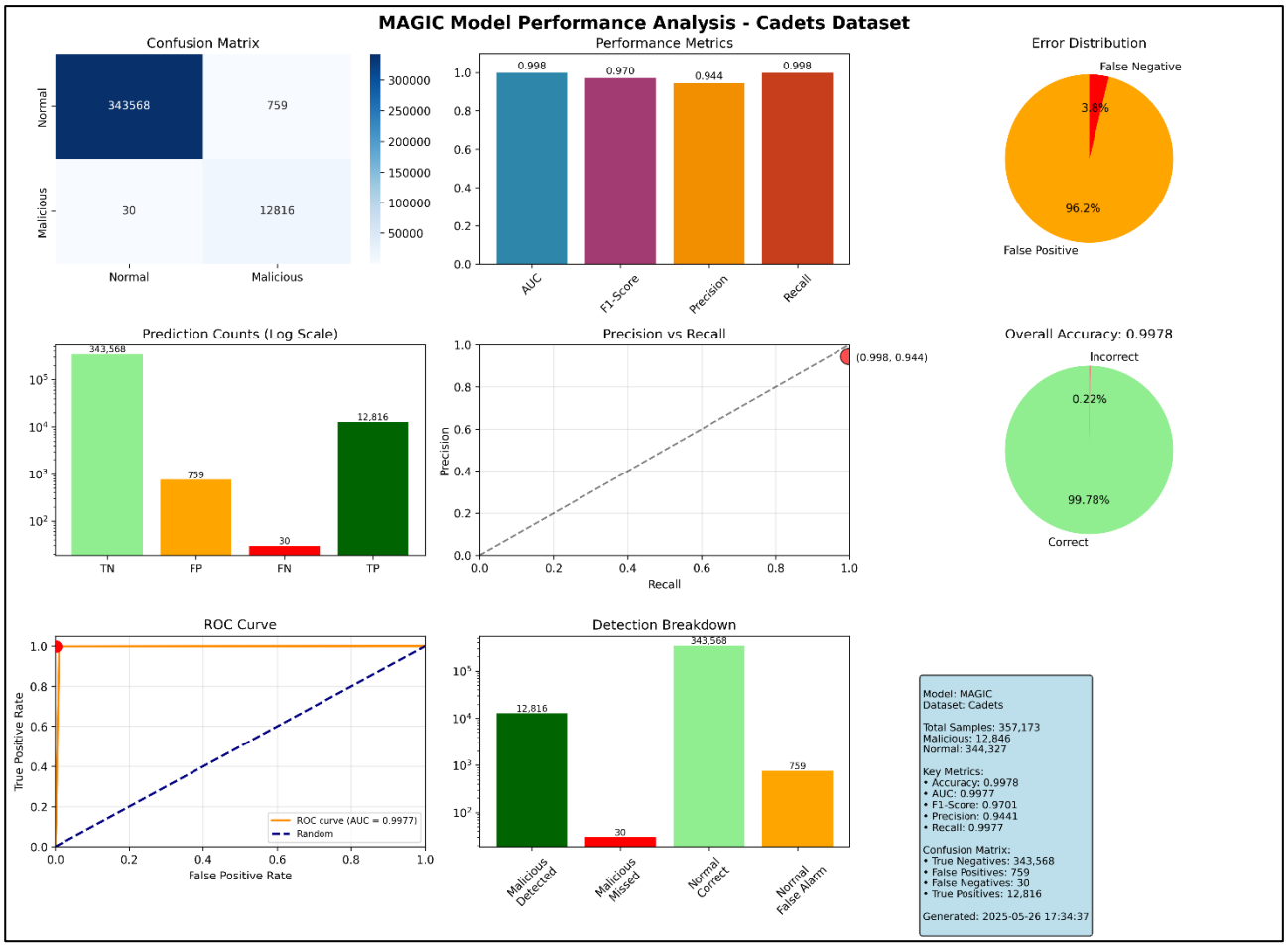
- F1-Score = 0.991: Sự cân bằng tốt giữa Precision và Recall, cho thấy mô hình vừa chính xác vừa bao quát.

- Precision = 0.982: 98.2% các dự đoán "Malicious" là đúng, với một tỷ lệ nhỏ sai sót do 456 FP

- Recall = 1: Mô hình phát hiện được 100% các mẫu "Malicious", không bỏ sót trường hợp nào

➔ Mô hình MAGIC hoạt động rất hiệu quả trên tập dữ liệu Theia, đặc biệt phù hợp cho các ứng dụng yêu cầu độ nhạy cao trong việc phát hiện mẫu Malicious mà vẫn duy trì tỷ lệ báo động giả thấp. Hiệu suất này được duy trì trên một tập dữ liệu lớn (344,767 mẫu), thể hiện khả năng mở rộng của mô hình

C.15. Darpa CADETS



- Ma trận nhầm lẫn (Confusion Matrix) cho thấy khả năng phân loại của mô hình đối với hai lớp: "Normal" (Bình thường) và "Malicious" (Độc hại)

- True Negatives (TN): 343568 (Dự đoán đúng "Normal" khi thực tế là "Normal")
- False Positives (FP): 759 (Dự đoán sai "Malicious" khi thực tế là "Normal")
- False Negatives (FN): 30 (Dự đoán sai "Normal" khi thực tế là "Malicious")
- True Positives (TP): 12816 (Dự đoán đúng "Malicious" khi thực tế là "Malicious")

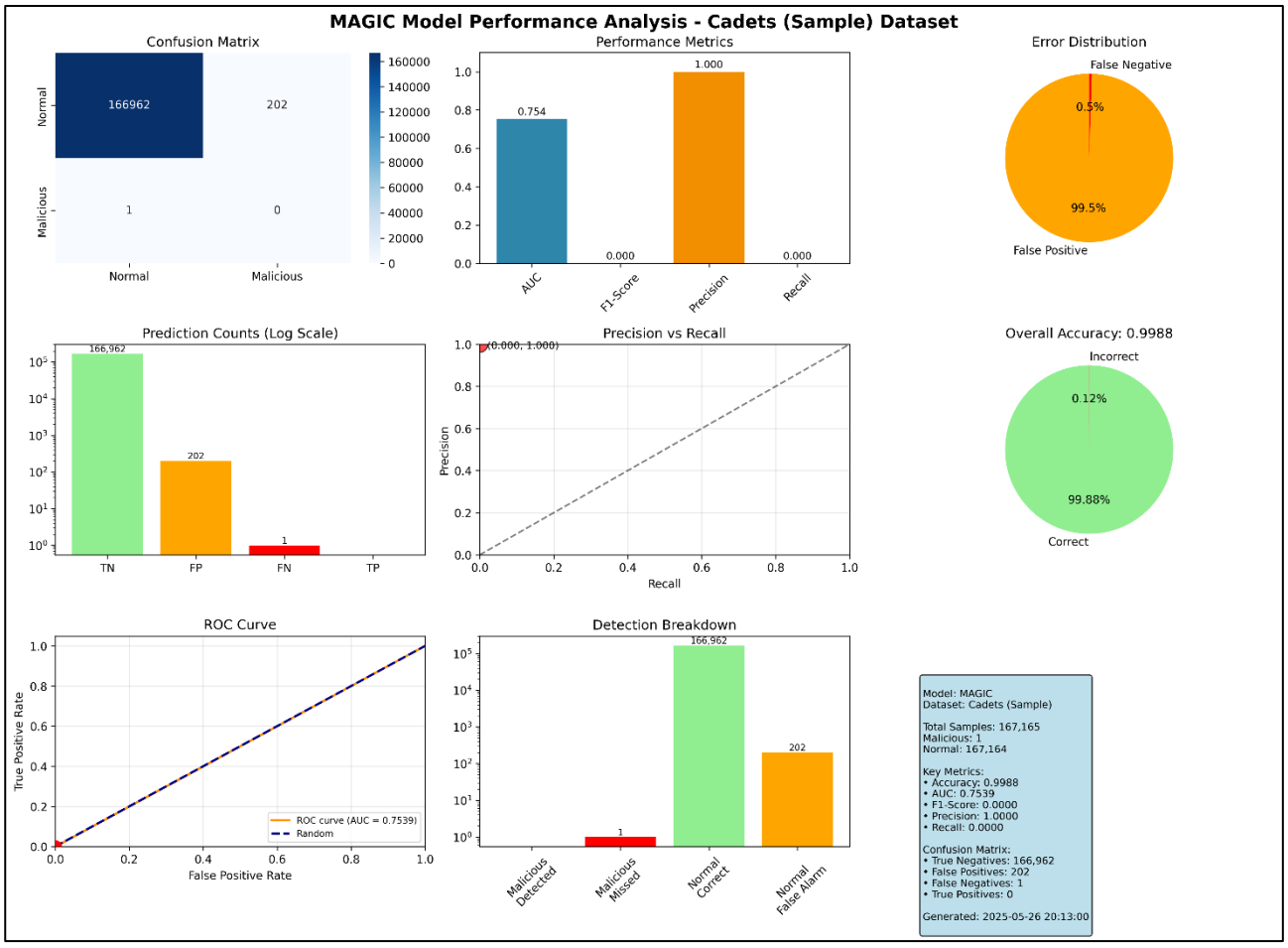
=> Mô hình phân loại chính xác 343,568 mẫu "Normal" và 12,816 mẫu "Malicious", có 759 mẫu "Normal" bị phân loại sai thành "Malicious" (FP) và 30 mẫu "Malicious" bị bỏ sót (FN), cho thấy một số lỗi nhưng tỷ lệ rất thấp so với tổng số mẫu

- AUC = 0.998: Mô hình có khả năng phân biệt hoàn hảo giữa "Normal" và "Malicious"
- F1-Score = 0.970: Sự cân bằng tốt giữa Precision và Recall, cho thấy mô hình vừa chính xác vừa bao quát.
- Precision = 0.944: 94.4% các dự đoán "Malicious" là đúng, với một tỷ lệ sai sót do 759 FP
- Recall = 0.998: Mô hình phát hiện được 99.8% các mẫu "Malicious", không bỏ sót trường hợp nào

➔ Tương tự như các tệp dataset khác của Darpa thì mô hình MAGIC hoạt động rất hiệu quả trên tập dữ liệu Cadets, đặc biệt phù hợp cho các ứng dụng yêu cầu độ nhạy cao trong việc phát hiện mẫu Malicious với tỷ lệ báo động giả thấp. Hiệu suất này được duy trì trên một tập dữ liệu lớn (357,173 mẫu), thể hiện khả năng mở rộng của mô hình

C.2. Kịch bản 2: MAGIC trên dataset tự thêm

C.2.1. Darpa FIVEDIRECTIONS



- Ma trận nhầm lẫn (Confusion Matrix) cho thấy khả năng phân loại của mô hình đối với hai lớp: "Normal" (Bình thường) và "Malicious" (Độc hại)

- True Negatives (TN): 166962 (Dự đoán đúng "Normal" khi thực tế là "Normal")
- False Positives (FP): 202 (Dự đoán sai "Malicious" khi thực tế là "Normal")
- False Negatives (FN): 1 (Dự đoán sai "Normal" khi thực tế là "Malicious")
- True Positives (TP): 0 (Dự đoán đúng "Malicious" khi thực tế là "Malicious")

=> Mô hình phân loại chính xác 166,962 mẫu "Normal" nhưng không phát hiện được mẫu "Malicious" duy nhất (TP = 0), có 202 mẫu "Normal" bị phân loại sai thành "Malicious" (FP) và 1 mẫu "Malicious" bị bỏ sót (FN), cho thấy mô hình đang gần như dự đoán tất cả các node là tấn công

- AUC = 0.754: Mô hình có khả năng phân biệt giữa "Normal" và "Malicious" ở mức trung bình, thấp hơn đáng kể so với các tập dữ liệu khác

- F1-Score = 0: Do không có True Positives, F1-Score bằng 0, phản ánh hiệu suất phân loại Malicious rất kém.

- Precision = 1: 100% các dự đoán "Malicious" là đúng, vì mô hình gần như đoán tất cả là Malicious

- Recall = 0: Mô hình không phát hiện được mẫu "Malicious" duy nhất, dẫn đến Recall bằng 0

➔ Mô hình MAGIC không hiệu quả trên tập dữ liệu Fivedirections do không thể phát hiện mẫu "Malicious" duy nhất, tập dữ liệu không phù hợp với thuật toán mô hình. Điều này cho thấy đối với các loại dữ liệu khác nhau cần được xử lí đúng cách để đạt được hiệu quả xử lí cao

D. HƯỚNG PHÁT TRIỂN

D.1. Hướng phát triển

Cải tiến mô-đun phát hiện bất thường (Outlier Detection)

- Hiện MAGIC sử dụng K-nearest neighbors (KNN) cho việc phát hiện bất thường, chiếm đến ~99% thời gian inference và có thể gặp khó khăn khi mở rộng cho tập dữ liệu rất lớn. Trong tương lai có thể:

- Áp dụng các phương pháp clustering–based hoặc tìm kiếm KNN xấp xỉ (approximate KNN) để giảm độ phức tạp tính toán.
- Chuyển một số bước sang GPU để tận dụng khả năng song song cao của phần cứng đồ họa.
- Thử nghiệm với các thuật toán khác như Isolation Forest hay One-class SVM, kèm cơ chế thích ứng concept drift hiệu quả hơn

Nâng cao khả năng chống tấn công đối kháng (Adversarial Robustness)

- Nếu kẻ tấn công biết rõ cách hoạt động bên trong của MAGIC, họ có thể thực hiện:

- MFE (Malicious Feature Evasion): Thay đổi đặc trưng đầu vào để giống benign
- MSE (Malicious Structure Evasion): Tạo thêm các cạnh kết nối với thực thể bình thường.
- BFP (Benign Feature Poisoning): Tiêm nhiễm đặc trưng “giả” vào các thực thể benign
- Hướng nghiên cứu tiếp theo bao gồm thiết kế cơ chế bảo vệ GNN và input-graph chống lại các hình thức tấn công này, hoặc phát triển chiến lược tự động phát hiện và lọc bỏ các phiên bản graph bị đầu độc

Mở rộng và tích hợp cho môi trường thực (Real-world Deployment)

- MAGIC đã chứng minh tính hiệu quả và khả năng triển khai ngay cả với CPU-only (chỉ ~2 phút để xử lý 1.37 GB log/ngày trên sub-dataset DARPA E3 Trace)

- Khả năng "tự-học" từ feedback (model adaption) giúp duy trì độ chính xác khi hệ thống cập nhật hay thay đổi hành vi bình thường

- Có thể tích hợp MAGIC vào pipeline SIEM (Security Information and Event Management) hoặc EDR (Endpoint Detection and Response) để phát hiện APT thời gian thực, đồng thời kết hợp chế độ two-stage (batch-level → entity-level) để tối ưu hiệu năng và giảm false-positive trước khi đi vào phân tích chi tiết

D.2. Tính ứng dụng

- Tính tự-lập: Không cần dữ liệu attack-labeled, chỉ dùng log benign để huấn luyện ban đầu

- Tính linh hoạt: Hỗ trợ phát hiện ở nhiều mức độ (batch-level hoặc entity-level).

- Tính hiệu quả: Chi phí tính toán thấp, khả năng mở rộng cao (logarithmic time, linear space).

- Tính thích ứng: Có cơ chế model adaption giảm false-positive khi có concept drift.

--- HẾT ---