BỘ KHOA HỌC VÀ CÔNG NGHỆ
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
CƠ SỞ THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN 2

MÔN HỌC:

KỸ THUẬT THEO DÕI, GIÁM SÁT MẠNG

ĐỀ TÀI:

Hệ thống giám sát và phân tích logfile server dựa trên Học máy

Giảng viên hướng dẫn:	**ThS. Đàm Minh Lịnh**
Lớp:	**D21CQAT01-N**
Khóa:	**2021-2026**

Sinh viên thực hiện:

Lê Khánh Bình Đức	N21DCAT013
Đinh Văn Hậu	N21DCAT017
Trần Lê Huỳnh Long	N21DCAT029

TP. Thủ Đức, tháng 05 năm 2025

Mục lục

LỜI CẢM ƠN.....
MỞ ĐẦU 1
CHƯƠNG 1 CƠ SỞ LÝ THUYẾT.....2
1.1 Tiền xử lý dữ liệu.....2
1.1.1 Dữ liệu log là gì?.....2
1.1.2 Loại bỏ dữ liệu thừa (log noise).....3
1.1.3 Tokenization, Stemming, và Stopword Removal 4
1.1.4 Chuyển log thành dạng có thể xử lý (TF-IDF, Word2Vec).....5
1.2 Phương pháp Nhúng Đặc Trưng (Feature Embedding).....6
1.2.1 Biểu diễn logfile dưới dạng chuỗi hoặc ma trận đặc trưng.....6
CHƯƠNG 2 Huấn Luyện Mô Hình 8
2.1 Các loại mô hình phổ biến 8
2.1.1 Machine Learning 8
2.1.2 Deep Learning.....11
2.2 Các bước huấn luyện mô hình 15
2.2.1 Chuẩn bị dataset 15
2.2.2 Phân tích dữ liệu.....15
2.2.3 Tiền xử lý dữ liệu.....19
2.2.4 Chuẩn bị dữ liệu trước khi huấn luyện mô hình 20
2.2.5 Huấn luyện mô hình Machine Learning 20
2.2.6 Huấn luyện mô hình Deep Learning.....30
CHƯƠNG 3 Triển Khai Hệ Thống 40
3.1 Giới thiệu 40
3.2 Mô hình hệ thống.....40
3.3 Phát hiện bất thường 43
3.3.1 Tích hợp mô hình vào hệ thống 43
3.3.2 Kết quả thực nghiệm 43
3.3.3 Hạn chế.....44
3.3.4 Đề xuất cải thiện.....44
KẾT LUẬN 45

DANH MỤC TÀI LIỆU THAM KHẢO 46

Mục lục hình ảnh

Hình 1: Giới thiệu về Random Forest 8
Hình 2: Giới thiệu về XGBoost.....9
Hình 3: Giới thiệu về SVM 10
Hình 4: Cách hoạt động của SVM.....11
Hình 5: Các lớp cơ bản của CNN.....12
Hình 6: Cách hoạt động cơ bản của CNN 13
Hình 7: Giới thiệu về RNN.....14
Hình 8: Giới thiệu về LSTM 15
Hình 9: Dataset 15
Hình 10: Số dòng và số cột.....16
Hình 11: Số lượng giá trị null trong 12 cột 16
Hình 12: Số lượng null và kiểu dữ liệu từng cột 17
Hình 13: Phân phối nhãn 18
Hình 14: Biểu đồ phân phối nhãn.....19
Hình 15: Xóa dữ liệu trùng lặp.....19
Hình 16: Ánh xạ nhãn sang số.....20
Hình 17: Chia tập train, test.....20
Hình 18: Sử dụng SMOTE 20
Hình 19: Số lượng nhãn sau khi xử lý mất cân bằng 20
Hình 20: Huấn luyện mô hình RF 21
Hình 21: Biểu đồ đặc trưng quan trọng 21
Hình 22: Cây quyết định đầu tiên trong RF 21
Hình 23: Đánh giá chỉ số của RF.....22

Hình 24: Ma trận nhầm lẫn của RF 23
Hình 25: Huấn luyện mô hình XGBoost 23
Hình 26: Đánh giá các chỉ số của XGBoost 24
Hình 27: Ma trận nhầm lẫn của XGBoost 25
Hình 28: Chuẩn hóa dữ liệu 25
Hình 29: Huấn luyện LinearSVC 26
Hình 30: Đánh giá các chỉ số của SVM 26
Hình 31: Ma trận nhầm lẫn của SVM 27
Hình 32: Chỉ số AUC (macro) của các mô hình ML 29
Hình 33: Đường cong ROC cho 3 mô hình ML 30
Hình 34: Chuẩn hóa và reshape và mã hóa one-pot 31
Hình 35: Huấn luyện mô hình CNN 32
Hình 36: Quá trình huấn luyện CNN 32
Hình 37: Báo cáo phân loại của CNN 33
Hình 38: Ma trận nhầm lẫn của CNN 33
Hình 39: Huấn luyện mô hình LSTM 34
Hình 40: Quá trình huấn luyện LSTM 35
Hình 41: Báo cáo phân loại của LSTM 35
Hình 42: Ma trận nhầm lẫn của LSTM 36
Hình 43: Điểm AUC (One to Rest) của mô hình DL 38
Hình 44: Biểu đồ ROC cho phân loại từng nhãn của LSTM 38
Hình 45: Biểu đồ ROC cho phân loại nhãn của CNN 39
Hình 46: Mô hình mạng 40
Hình 47: Các cronjob trong crontab 41
Hình 48: bash script để dump thông tin từ state_table của opnsense 41
Hình 49: bash script gửi state_table.log tới máy ubuntu chịu trách nhiệm chạy mô hình thông qua ssh có private key 41
Hình 50: Nội dung trong 1 file state_table.log 42
Hình 51: State table trên giao diện WEB GUI của OPNSense 42
Hình 52: Kết quả chạy mô hình 44

Mục lục bảng

Bảng 1: Bảng phân loại log và mục đích sử dụng 3
Bảng 2: Bảng tổng hợp chỉ số đánh giá cho mô hình ML 27
Bảng 3: Bảng so sánh về ma trận nhầm lẫn cho mô hình ML 28
Bảng 4: Bảng so sánh báo cáo phân loại cho mô hình DL 37
Bảng 5: Bảng so sánh ma trận nhầm lẫn cho mô hình DL 37
Bảng 6: Bảng thành phần trong hệ thống 40
Bảng 7: Bảng kết quả thực nghiệm 43

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin gửi lời tri ân sâu sắc đến các thầy cô Học Viện Công Nghệ Bưu Chính Viễn thông cơ sở tại TP.HCM đã tận tình dẫn dắt và truyền đạt cho chúng em rất nhiều kiến thức quý báu trong các học kỳ vừa qua.

Đặc biệt, chúng em xin gửi lời cảm ơn chân thành đến thầy Đàm Minh Lịnh. Thầy đã hướng dẫn tận tình, truyền đạt kiến thức, chỉ bảo cho em trong suốt thời gian học tập và thực hiện đề tài. Kính chúc thầy và gia đình nhiều sức khoẻ và thành công trong cuộc sống.

Và xin chân thành cảm ơn tất cả các bạn đã luôn sát cánh, giúp đỡ, động viên mình trong những thời điểm khó khăn, tiếp thêm động lực và ý chí giúp vượt qua khó khăn trong suốt quá trình học tập tại trường.

Tuy nhiên, điều kiện thời gian cũng như kinh nghiệm còn hạn chế, luận văn này không thể tránh được những thiếu sót. Chúng em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các thầy cô để chúng em có điều kiện bổ sung, nâng cao ý thức của mình, phục vụ tốt hơn công tác thực tế sau này.

TP.HCM, tháng 5 năm 2025

MỞ ĐẦU

Trong bối cảnh các hệ thống phân tán lớn ngày càng phức tạp, việc đảm bảo độ tin cậy và hiệu suất hoạt động là một thách thức không ngừng. Logfile, hay nhật ký hệ thống, đóng vai trò là xương sống thông tin, ghi lại mọi sự kiện, tương tác và bất thường xảy ra trong một hệ thống, ứng dụng, thiết bị mạng hoặc điểm cuối. Những bản ghi kỹ thuật số này được tạo ra tự động bởi máy móc, cung cấp một kho tàng dữ liệu vô giá để hiểu rõ hành vi hệ thống, xác định các vi phạm bảo mật và tối ưu hóa hiệu quả vận hành.

Log files không chỉ là công cụ giám sát mà còn là yếu tố then chốt trong việc khắc phục sự cố và theo dõi các sự kiện, sự cố bảo mật cũng như hoạt động của người dùng. Chúng cung cấp cái nhìn sâu sắc về hiệu suất và tình trạng của hệ thống, cho phép các quản trị viên xác định vấn đề, phân tích xu hướng và đảm bảo hoạt động hiệu quả của máy chủ và ứng dụng. Nếu không có khả năng phân tích log hiệu quả, việc chẩn đoán và giải quyết các vấn đề kịp thời sẽ trở nên vô cùng khó khăn, dẫn đến nguy cơ gián đoạn hoạt động, các lỗ hổng bảo mật và suy giảm hiệu suất nghiêm trọng.

Với khối lượng log ngày càng tăng theo cấp số nhân - các tệp log có thể nhanh chóng trở nên khổng lồ tùy thuộc vào lượng thông tin được ghi lại - các phương pháp phân tích thủ công hoặc các kỹ thuật học máy truyền thống đã trở nên không khả thi. Sự gia tăng về tốc độ, sự đa dạng và khối lượng dữ liệu log hiện đại đã tạo ra một thách thức lớn, vượt quá khả năng xử lý của con người và các phương pháp tính toán truyền thống. Điều này đã thúc đẩy sự phát triển nhanh chóng của các giải pháp dựa trên học sâu và học máy. Phân tích log tự động là một nhiệm vụ quan trọng trong AIOps (Artificial Intelligence for IT Operations), cung cấp những hiểu biết chính yếu giúp các kỹ sư độ tin cậy trang web (SREs) xác định và giải quyết các lỗi đang diễn ra. Học máy đang thay đổi căn bản cách chúng ta phân tích log, mở ra các khả năng mới như phát hiện bất thường để tìm các mẫu không điển hình, phân cụm tự động các vấn đề liên quan, truy vấn dữ liệu log bằng ngôn ngữ tự nhiên và cảnh báo dự đoán trước khi xảy ra lỗi. Tương lai của phân tích log đang hướng tới các công cụ dễ tiếp cận hơn, bao gồm trình tạo truy vấn trực quan thay vì ngôn ngữ truy vấn phức tạp, tạo bảng điều khiển tự động, tìm kiếm dựa trên ý định và đường dẫn khắc phục sự cố có hướng dẫn. Vấn đề cốt lõi không chỉ là phát hiện bất thường mà còn là quản lý quy mô dữ liệu khổng lồ để có thể bắt đầu phân tích có ý nghĩa. Học máy và học sâu cung cấp khả năng tự động hóa cần thiết và các kỹ thuật nhận dạng mẫu tinh vi, không thể thiếu để trích xuất thông tin chi tiết có thể hành động và giá trị từ lượng lớn dữ liệu log này, cho phép quản lý hệ thống một cách chủ động thay vì chỉ phản ứng khi sự cố xảy ra.

CHƯƠNG 1 CƠ SỞ LÝ THUYẾT

1.1 Tiền xử lý dữ liệu

Tiền xử lý dữ liệu là bước đầu tiên và quan trọng nhất trong quá trình chuẩn bị dữ liệu log để phân tích. Mục tiêu chính của giai đoạn này là giải quyết các vấn đề về chất lượng dữ liệu như giá trị bị thiếu, dữ liệu nhiễu, và chuyển đổi dữ liệu thô thành định dạng phù hợp cho các thuật toán học máy. [1] Quá trình này bao gồm các hoạt động như làm sạch dữ liệu, giảm dữ liệu, chuyển đổi dữ liệu và làm giàu dữ liệu.

1.1.1 Dữ liệu log là gì?

Dữ liệu log là các thông điệp được tạo ra bởi máy tính, mạng, tường lửa, máy chủ ứng dụng và các hệ thống công nghệ thông tin khác, ghi lại các sự kiện hoặc dấu vết kiểm toán. Chúng thường là các chuỗi văn bản không có cấu trúc, chứa thông tin quan trọng như thời gian xảy ra sự kiện, địa chỉ IP liên quan và các thông báo lỗi cụ thể.

Đặc điểm chính của dữ liệu log là chúng thường được tạo ra tự động bởi máy móc và ban đầu có thể được lưu trữ dưới dạng các tệp văn bản đơn giản. Tuy nhiên, khi quy mô sử dụng tăng lên, các tệp log có thể nhanh chóng trở nên khổng lồ, đòi hỏi phải lưu trữ trong cơ sở dữ liệu hoặc nén thành các định dạng khác để quản lý hiệu quả. [2] Log files được coi là nguồn dữ liệu chính yếu cho khả năng quan sát mạng, cung cấp cái nhìn toàn diện về hoạt động và trạng thái của hệ thống. [3]

Các loại logfile phổ biến bao gồm nhiều dạng khác nhau, mỗi loại phục vụ một mục đích chuyên biệt trong việc giám sát, khắc phục sự cố, kiểm toán và phân tích hoạt động trong các hệ thống, ứng dụng và mạng để duy trì hiệu quả hoạt động, bảo mật và tuân thủ.

Loại Log	Mô tả ngắn gọn	Mục đích chính
Log hệ thống (System logs)	Ghi lại các sự kiện và hoạt động cấp hệ thống.	Giám sát khởi động/tắt máy, lỗi phần cứng, thông báo kernel, sử dụng tài nguyên hệ thống.
Log ứng dụng (Application logs)	Ghi lại các sự kiện cụ thể của ứng dụng.	Theo dõi lỗi, cảnh báo, hành động người dùng, và các chỉ số hiệu suất ứng dụng.
Log bảo mật (Security logs)	Ghi lại các sự kiện liên quan đến bảo mật.	Phát hiện và theo dõi hoạt động độc hại, cố gắng đăng nhập, thay đổi kiểm soát truy cập, xác thực.
Log mạng (Network logs)	Ghi lại hoạt động của các thiết bị mạng.	Giám sát và kiểm soát luồng lưu lượng trong mạng.
Log kiểm toán (Audit logs)	Ghi lại các sự kiện cho mục đích kiểm toán và tuân thủ.	Đảm bảo trách nhiệm giải trình và khả năng truy vết cho các quy định.
Log cơ sở dữ liệu (Database logs)	Ghi lại các giao dịch và thay đổi trong cơ sở dữ liệu.	Theo dõi giao dịch, thay đổi dữ liệu, và các chỉ số hiệu suất.

Log sự kiện (Event logs)	Bản ghi theo thứ tự thời gian của các sự kiện đáng chú ý.	Ghi lại thông báo và hành động quản trị trong hệ thống hoặc ứng dụng.
Log truy cập (Access logs)	Ghi lại chi tiết về quyền truy cập của người dùng vào tài nguyên.	Theo dõi dấu thời gian đăng nhập/đăng xuất, tệp được truy cập, thay đổi quyền, kết nối mạng.
Log lỗi (Error logs)	Ghi lại lỗi, ngoại lệ, cảnh báo và thông tin gỡ lỗi.	Chẩn đoán và khắc phục sự cố trong phần mềm, hệ thống hoặc ứng dụng.
Log hiệu suất (Performance logs)	Giám sát các chỉ số hiệu suất hệ thống hoặc ứng dụng.	Tối ưu hóa hiệu suất và xác định các nút thắt cổ chai.
Log thay đổi (Change logs)	Ghi lại các thay đổi đối với cấu hình, cài đặt, tệp hoặc cơ sở dữ liệu.	Theo dõi sửa đổi, xác định sự khác biệt và duy trì kiểm soát phiên bản.

Bảng 1: Bảng phân loại log và mục đích sử dụng

1.1.2 Loại bỏ dữ liệu thừa (log noise)

Log noise là các dữ liệu không cần thiết, chẳng hạn như thông tin lặp lại hoặc không liên quan, có thể làm giảm đáng kể hiệu quả của quá trình phân tích. Dữ liệu nhiễu thường chứa các giá trị bị thiếu và có thể dẫn đến các mẫu và mối tương quan sai lệch trong phân tích. Điều này có thể làm sai lệch các số liệu thống kê cốt lõi như giá trị trung bình và trung vị, ảnh hưởng đến cách dữ liệu được tóm tắt và hiểu. Ngược lại, việc giảm nhiễu hiệu quả sẽ nâng cao khả năng trích xuất đặc trưng bằng cách làm rõ các cạnh và cấu trúc, từ đó đảm bảo các mô hình học máy mạnh mẽ và hoạt động tốt trong môi trường thực tế. [4]

Có nhiều kỹ thuật được áp dụng để loại bỏ log noise:

- Lọc dựa trên từ khóa hoặc phân cụm: Các phương pháp này, như sử dụng thuật toán phân cụm K-means, giúp loại bỏ nhiễu và đảm bảo dữ liệu đầu vào sạch hơn, đặc biệt quan trọng khi xử lý khối lượng log lớn từ các hệ thống phức tạp.
- Loại bỏ trùng lặp: Đây là quá trình xác định và loại bỏ các bản sao dữ liệu trùng lặp trong một tập dữ liệu hoặc hệ thống lưu trữ. Mục tiêu chính là tối ưu hóa dung lượng lưu trữ và cải thiện quản lý dữ liệu.
- Lọc thông tin nhạy cảm: Một thực hành bảo mật quan trọng là không bao giờ ghi log dữ liệu nhạy cảm như mật khẩu, khóa API/token, số thẻ tín dụng, thông tin nhận dạng cá nhân (PII), ID phiên, cookie hoặc token OAuth. Thay vào đó, chỉ nên ghi log thông tin cần thiết, ví dụ như xác nhận xác thực thành công mà không có mật khẩu, hoặc chỉ ghi log bốn chữ số cuối của thẻ tín dụng để tham chiếu mà không làm lộ dữ liệu nhạy cảm
- Xử lý giá trị thiếu: Các giá trị bị thiếu có thể làm sai lệch các xu hướng dữ liệu thực tế và ảnh hưởng đến hiệu suất của mô hình học máy.

    - Loại bỏ hàng: Xóa toàn bộ hàng chứa giá trị thiếu nếu tập dữ liệu đủ lớn, nhưng có nguy cơ mất dữ liệu quan trọng.
    - Điền giá trị: Ước tính và điền giá trị thiếu bằng cách sử dụng giá trị trung bình (mean), trung vị (median) hoặc mode của cột dữ liệu đó.
- Xử lý ngoại lệ (Handling Outliers): Ngoại lệ là các điểm dữ liệu khác biệt đáng kể so với phần còn lại của tập dữ liệu, có thể bao gồm các từ hiếm, lỗi chính tả hoặc các mục không liên quan. Các phương pháp xử lý ngoại lệ bao gồm:
    - Loại bỏ: Xóa các mục chứa ngoại lệ khỏi phân phối dữ liệu.
    - Thay thế: Xử lý ngoại lệ như các giá trị thiếu và thay thế chúng bằng các giá trị ước tính phù hợp.
    - Giới hạn (Capping): Thay thế các giá trị tối đa và tối thiểu bằng một giá trị tùy ý hoặc một giá trị từ phân phối biến.
    - Rời rạc hóa (Discretization): Chuyển đổi các biến liên tục thành các biến rời rạc bằng cách tạo ra một loạt các khoảng liên tục.
- Làm mịn dữ liệu (Data Smoothing): Nhằm mục đích lọc bỏ các yếu tố nhiễu, đóng vai trò quan trọng trong việc tinh chỉnh dữ liệu để phân tích rõ ràng hơn. Các kỹ thuật làm mịn bao gồm:
    - Trung bình động (Moving Averages): Tính toán trung bình của một tập hợp con các điểm dữ liệu trong một cửa sổ di chuyển. Phương pháp này hiệu quả cho dữ liệu chuỗi thời gian, giúp làm mịn các biến động ngắn hạn trong khi vẫn giữ các xu hướng dài hạn.
    - Làm mịn hàm mũ (Exponential Smoothing): Áp dụng trọng số giảm dần cho các điểm dữ liệu cũ hơn, nhấn mạnh các quan sát gần đây hơn. Kỹ thuật này hữu ích khi các điểm dữ liệu gần đây có liên quan hơn, ví dụ trong dự báo nhu cầu hoặc dự đoán hành vi khách hàng.
    - Làm mịn hạt nhân (Kernel Smoothing): Sử dụng trung bình có trọng số của các điểm dữ liệu lân cận để làm mịn các bất thường mà không làm mất chi tiết. Không yêu cầu kích thước cửa sổ cố định, cho phép linh hoạt hơn trong việc làm mịn dữ liệu.
    - Chuyển đổi Log (Log transformation): Giảm biến động dữ liệu bằng cách áp dụng hàm logarit, nén các giá trị cao hơn nhiều hơn các giá trị thấp hơn, hiệu quả cho dữ liệu bị lệch.

1.1.3 Tokenization, Stemming, và Stopword Removal

Để chuyển đổi dữ liệu log không có cấu trúc thành dạng có thể xử lý được bằng máy học, các kỹ thuật tiền xử lý ngôn ngữ tự nhiên (NLP) là không thể thiếu. Ba kỹ thuật cơ bản và quan trọng nhất trong số đó là tokenization, stemming và stopword removal.

- Tokenization: Là quá trình chia nhỏ văn bản log thành các đơn vị nhỏ hơn gọi là "token", ví dụ như từ hoặc cụm từ, để dễ dàng xử lý hơn. Đây là bước đầu tiên và quan trọng trong tiền xử lý dữ liệu văn bản cho các tác vụ học máy và NLP. Mục tiêu của tokenization là chuyển đổi một tài liệu văn bản không có cấu trúc thành dữ liệu số phù hợp cho các phân tích dự đoán hoặc định hướng. Nếu không có tokenization phù hợp, việc trích xuất thông tin có ý nghĩa từ dữ liệu văn bản sẽ rất

khó khăn, vì các mô hình NLP sẽ phải làm việc với toàn bộ văn bản như một chuỗi ký tự duy nhất, điều này tốn kém về mặt tính toán và khó mô hình hóa chính xác. Tokenization giúp chuẩn hóa dữ liệu đầu vào, làm cho nó dễ xử lý và phân tích hơn. Nó cũng đóng vai trò quan trọng trong việc tạo ra các word embedding (biểu diễn vector của từ), vốn là thành phần cốt lõi của nhiều mô hình NLP và các mô hình AI tạo sinh lớn như ChatGPT. [5]

- Stemming: Là kỹ thuật giảm một từ về dạng gốc của nó, ví dụ, "running" thành "run". Mặc dù từ gốc sau khi stemming có thể không phải lúc nào cũng là một từ thực hoặc đúng ngữ pháp trong tiếng Anh, nhưng nó giúp thống nhất các dạng khác nhau của cùng một từ về một dạng cơ sở chung. Sự đơn giản hóa này làm giảm độ phức tạp của dữ liệu văn bản, dẫn đến tính toán nhanh hơn và khả năng hiệu suất tốt hơn khi triển khai các thuật toán NLP, vì có ít từ duy nhất hơn để xem xét. Ví dụ, các từ "run", "runs", "running" đều có thể được stemmed về dạng gốc "run".
- Stopword Removal: Là quá trình loại bỏ các từ không mang ý nghĩa quan trọng, chẳng hạn như "the", "is", "and", để tập trung vào nội dung cốt lõi. Stop words là những từ phổ biến nhất trong một ngôn ngữ và thường không cung cấp ý nghĩa đáng kể. Việc loại bỏ chúng giúp tăng tốc độ xử lý mà không làm mất thông tin quan trọng. Bộ công cụ ngôn ngữ tự nhiên (NLTK) của Python cung cấp một danh sách các stop words được định nghĩa trước, và đôi khi cần bổ sung các stop words chuyên biệt theo lĩnh vực dựa trên bản chất của dữ liệu văn bản. [6]

1.1.4 Chuyển log thành dạng có thể xử lý (TF-IDF, Word2Vec)

Để các thuật toán học máy có thể phân tích dữ liệu log một cách hiệu quả, dữ liệu log cần được biểu diễn dưới dạng số. Hai phương pháp phổ biến để chuyển đổi văn bản log thành dạng số là TF-IDF (Term Frequency-Inverse Document Frequency) và Word2Vec.

- TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF là một thước đo thống kê được sử dụng trong xử lý ngôn ngữ tự nhiên và truy xuất thông tin để đánh giá mức độ quan trọng của một từ trong một tài liệu so với một tập hợp các tài liệu. Không giống như tần suất từ đơn giản, TF-IDF cân bằng giữa các từ phổ biến và hiếm để làm nổi bật các thuật ngữ có ý nghĩa nhất.
- TF-IDF là tích của hai thành phần:
    - **Tần suất thuật ngữ (Term Frequency - TF):** Đo lường tần suất xuất hiện của một từ trong một tài liệu log cụ thể. Có nhiều cách để định nghĩa TF, phổ biến nhất là số lần xuất hiện thô của từ trong tài liệu, hoặc tần suất tương đối (số lần xuất hiện chia cho tổng số từ trong tài liệu). TF cao cho một từ trong một log cụ thể cho thấy từ đó quan trọng đối với log đó.
    - **Tần suất tài liệu nghịch đảo (Inverse Document Frequency - IDF):** Đo lường mức độ quan trọng của một từ trong toàn bộ tập log, dựa trên mức độ phổ biến của nó. IDF giảm trọng số của các từ phổ biến xuất hiện trong nhiều tài liệu (ví dụ: "the", "is") và tăng trọng số của các từ hiếm hơn. IDF được tính bằng logarit của tổng số tài liệu trong corpus chia cho số tài liệu chứa từ đó. Nếu một từ xuất hiện trong ít tài liệu hơn, điểm IDF của nó sẽ

cao hơn, cho thấy tính độc đáo lớn hơn. Một điểm TF-IDF cao đạt được khi một từ có tần suất xuất hiện cao trong tài liệu cụ thể (TF cao) và tần suất xuất hiện thấp trong toàn bộ tập tài liệu (IDF thấp). Điều này giúp lọc bỏ các thuật ngữ phổ biến và làm nổi bật các từ khóa quan trọng, đặc trưng cho nội dung của log đó. Ví dụ, trong một hệ thống tìm kiếm, điểm TF-IDF sẽ giúp xếp hạng các tài liệu cao hơn cho một truy vấn nếu chúng chứa các từ quan trọng và hiếm. Tuy nhiên, TF-IDF có hạn chế là không tính đến ngữ nghĩa hoặc mối quan hệ giữa các từ. [7]

- Word2Vec: Word2Vec là một kỹ thuật trong xử lý ngôn ngữ tự nhiên (NLP) để thu được biểu diễn vector (embedding) của từ. Các vector này nắm bắt thông tin về ý nghĩa của từ dựa trên các từ xung quanh nó (ngữ cảnh). Trong mô hình Word2Vec, các từ xuất hiện trong ngữ cảnh tương tự được ánh xạ tới các vector gần nhau trong không gian vector đa chiều, được đo bằng độ tương đồng cosine.
- Có hai kiến trúc chính của Word2Vec:
    - CBOW (Continuous Bag of Words): Mô hình này dự đoán từ mục tiêu từ các từ ngữ cảnh xung quanh nó. Nó tổng hợp tất cả các từ ngữ cảnh và sử dụng vector kết quả để dự đoán từ mục tiêu. CBOW hiệu quả khi có tập dữ liệu nhỏ và nhanh hơn Skip-Gram.
    - Skip-Gram: Mô hình này dự đoán các từ ngữ cảnh xung quanh từ một từ mục tiêu. Tức là, nó sử dụng một từ duy nhất để dự đoán ngữ cảnh xung quanh nó. Skip-Gram hoạt động tốt với tập dữ liệu lớn và các từ hiếm, mặc dù tốn kém tính toán hơn CBOW. Các vector nhúng được tạo bằng thuật toán Word2Vec có ưu điểm so với các thuật toán trước đây như n-gram và phân tích ngữ nghĩa tiềm ẩn. Các tham số quan trọng trong huấn luyện Word2Vec bao gồm thuật toán huấn luyện (hierarchical softmax hoặc negative sampling), lấy mẫu con (sub-sampling) để loại bỏ các từ có tần suất quá cao hoặc quá thấp, số chiều của vector (thường từ 100 đến 1.000) và kích thước cửa sổ ngữ cảnh. Trong phân tích log, Word2Vec được sử dụng để biểu diễn "error" và "exception" gần nhau trong không gian vector, phản ánh mối quan hệ ngữ nghĩa của chúng. Ví dụ, LogUAD là một mô hình phát hiện bất thường sử dụng Word2Vec kết hợp với trọng số TF-IDF, cho thấy cải thiện đáng kể trong hiệu suất phát hiện bất thường log.

1.2 Phương pháp Nhúng Đặc Trưng (Feature Embedding)

Nhúng đặc trưng là một quá trình cốt lõi trong học máy, chuyển đổi dữ liệu, trong trường hợp này là log, thành một dạng biểu diễn số học có thể xử lý được, thường là dưới dạng chuỗi hoặc ma trận đặc trưng. Mục tiêu của nhúng đặc trưng là tạo ra các biểu diễn dữ liệu có chiều thấp, nhỏ gọn nhưng vẫn mã hóa được thông tin quan trọng của dữ liệu gốc, bao gồm cả chi tiết ngữ nghĩa và cấu trúc, từ đó cho phép các thuật toán học máy hoạt động hiệu quả.

1.2.1 Biểu diễn logfile dưới dạng chuỗi hoặc ma trận đặc trưng

1.2.1.1 TF-IDF hoặc Word2Vec để biểu diễn dòng log dưới dạng vector.

TF-IDF và Word2Vec là hai phương pháp chính giúp chuyển đổi các dòng log thành vector. Các vector này không chỉ đơn thuần là biểu diễn số học mà còn cố gắng giữ lại thông tin ngữ nghĩa và cấu trúc của log.

1.2.1.2 Graph Embedding để phát hiện các mối quan hệ giữa các sự kiện log.

Graph Embedding, hay nhúng đồ thị, là một phương pháp mạnh mẽ để biểu diễn các nút trong một đồ thị dưới dạng vector, mã hóa thông tin quan trọng về cấu trúc và ngữ nghĩa của đồ thị. Điều này cho phép các thuật toán học máy và mô hình hoạt động trên dữ liệu đồ thị phức tạp. Trong phân tích log, Graph Embedding đặc biệt hữu ích trong việc phát hiện các mối quan hệ phức tạp giữa các sự kiện log, ví dụ như chuỗi các sự kiện dẫn đến lỗi hệ thống. LogAI là một thư viện mã nguồn mở được thiết kế để thực hiện các tác vụ phân tích và thông minh log dựa trên AI, bao gồm cả việc sử dụng Graph Embedding.

Các lợi ích chính của việc sử dụng nhúng đồ thị cho các mạng phức tạp bao gồm:

- Tăng cường độ chính xác và hiệu quả: Nhúng đồ thị cho phép các nhà nghiên cứu và khoa học dữ liệu khám phá các mẫu ẩn trong các mạng dữ liệu lớn, giúp cải thiện đáng kể độ chính xác và hiệu quả của các thuật toán học máy.
- Ra quyết định sáng suốt hơn: Bằng cách xác định các mẫu ẩn này, các nhà nghiên cứu có thể đưa ra quyết định sáng suốt hơn và phát triển các giải pháp tốt hơn cho các vấn đề phức tạp.
- Tính toán nhanh hơn: Nhúng đồ thị chắt lọc dữ liệu có cấu trúc đồ thị phức tạp thành các số liệu đơn giản, giúp các phép toán tính toán trên chúng trở nên rất dễ dàng và nhanh chóng. Lợi ích này cho phép ngay cả các thuật toán phức tạp nhất cũng có thể được mở rộng để phù hợp với tất cả các loại tập dữ liệu. [8]

Cách các chuỗi log được chuyển đổi thành cấu trúc đồ thị để phát hiện bất thường bằng Mạng nơ-ron đồ thị (GNNs): Các chuỗi log được chuyển đổi thành cấu trúc đồ thị để phát hiện bất thường bằng cách mô hình hóa log và các phụ thuộc của chúng thành cấu trúc đồ thị. Cách tiếp cận này cho phép phát hiện và ngăn chặn các lỗi hệ thống bằng cách xem xét các phụ thuộc hiện có giữa các log, điều mà các mô hình học máy truyền thống thường bỏ qua.

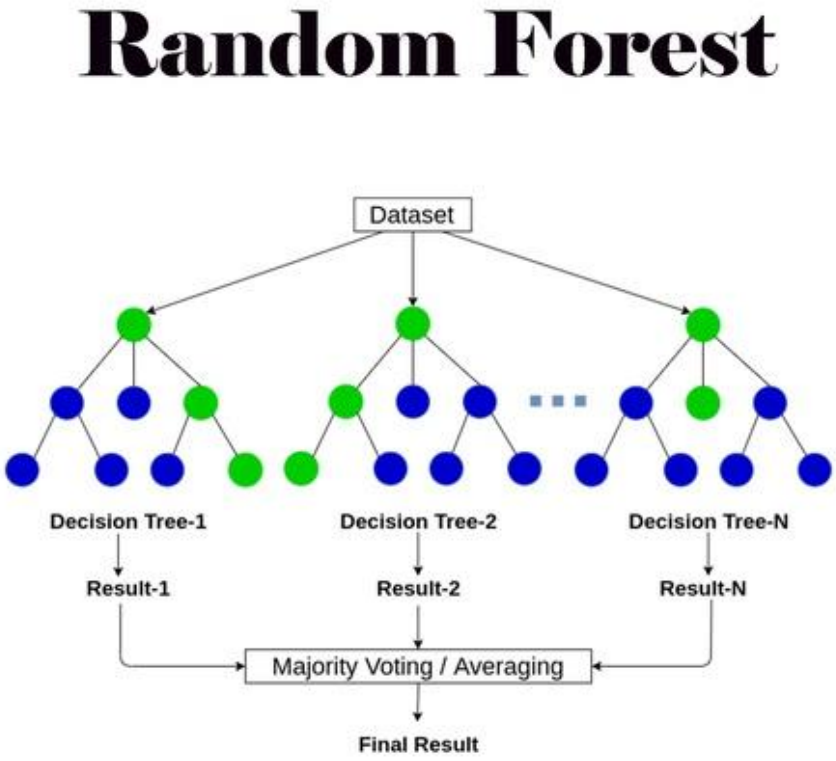
CHƯƠNG 2 HUẤN LUYỆN MÔ HÌNH

2.1 Các loại mô hình phổ biến

2.1.1 Machine Learning

2.1.1.1 Random Forest

Random Forest là một thuật toán học máy được sử dụng rộng rãi do Leo Breiman và Adele Cutler phát triển vào năm 2001, kết hợp đầu ra của nhiều cây quyết định để đạt được một kết quả duy nhất [9]. Sự phổ biến của nó bắt nguồn từ tính thân thiện với người dùng và tính linh hoạt, khiến nó phù hợp cho cả nhiệm vụ phân loại và hồi quy.



Hình 1: Giới thiệu về Random Forest

Cách hoạt động của Thuật toán Random Forest [10]

1. **Tạo nhiều cây quyết định:** Thuật toán tạo ra nhiều cây quyết định, mỗi cây sử dụng một phần ngẫu nhiên của dữ liệu.
2. **Chọn các đặc trưng ngẫu nhiên**: Khi xây dựng mỗi cây, thuật toán không xem xét tất cả các đặc trưng cùng lúc. Nó chọn ngẫu nhiên một vài đặc trưng để quyết định cách chia dữ liệu. Điều này giúp các cây khác biệt với nhau.
3. **Mỗi cây đưa ra một dự đoán**: Mỗi cây sẽ đưa ra câu trả lời hoặc dự đoán riêng của nó dựa trên phần dữ liệu mà nó đã học.

4. **Kết hợp các dự đoán**:

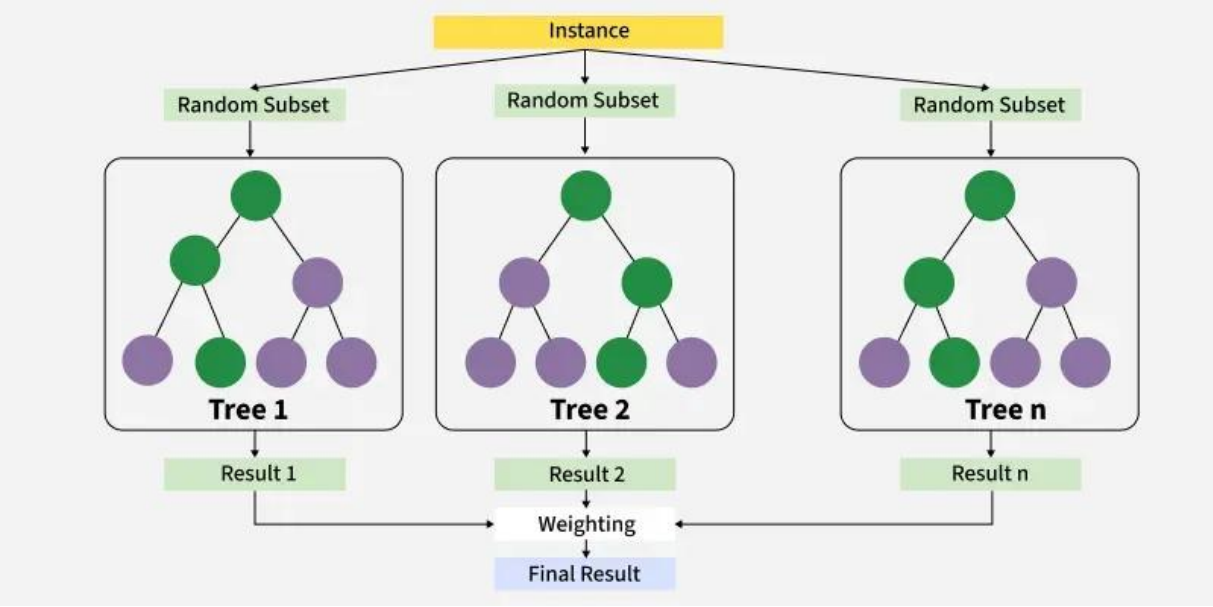
   + Đối với bài toán phân loại, thuật toán chọn nhãn mà đa số cây đồng ý

   + Đối với bài toán hồi quy, thuật toán đưa ra giá trị số là trung bình các dự đoán từ tất cả các cây.

5. **Tại sao thuật toán hoạt động hiệu quả**: Việc sử dụng dữ liệu và đặc trưng ngẫu nhiên cho mỗi cây giúp tránh overfitting (quá khớp) và làm cho dự đoán tổng thể trở nên chính xác và đáng tin cậy hơn.

2.1.1.2 XGBoost

XGBoost (eXtreme Gradient Boosting) là một thuật toán học máy theo học tập tổng hợp. Thuật toán này rất phổ biến đối với các tác vụ học có giám sát, chẳng hạn như hồi quy và phân loại. XGBoost xây dựng một mô hình dự đoán bằng cách kết hợp các dự đoán của nhiều mô hình riêng lẻ, thường là cây quyết định, theo cách lặp lại. [11]



Hình 2: Giới thiệu về XGBoost

Cách hoạt động của XGBoost [12]

XGBoost xây dựng các cây quyết định theo trình tự tuần tự, trong đó mỗi cây mới cố gắng sửa lỗi của cây trước đó. Quá trình này có thể được chia thành các bước như sau:

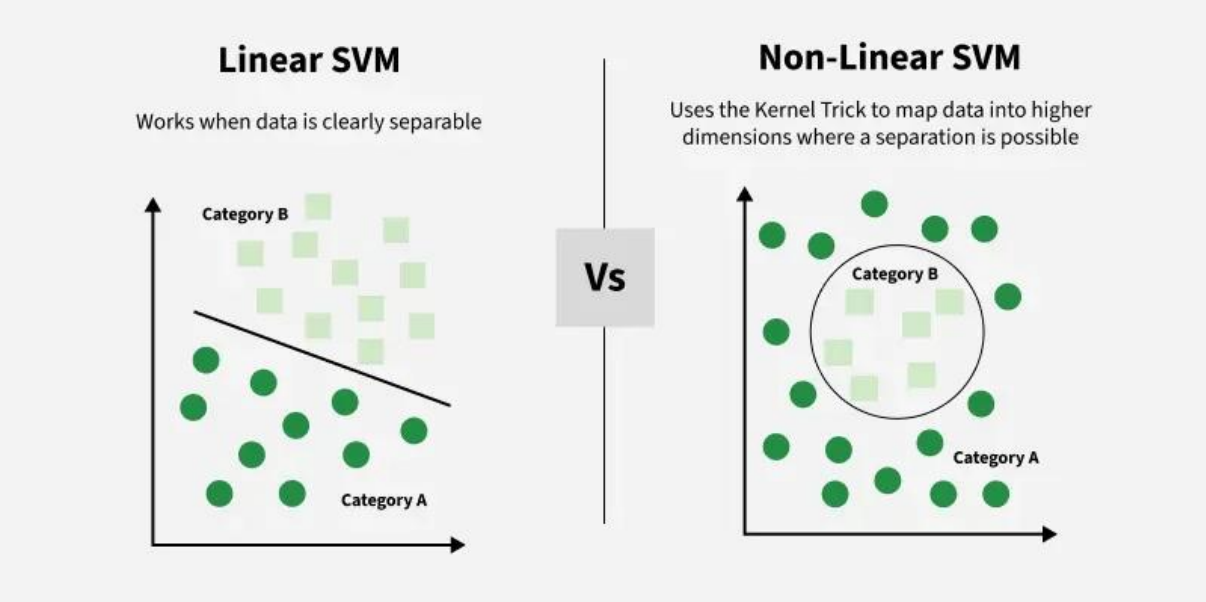
1. **Bắt đầu với mô hình cơ sở:** Cây quyết định đầu tiên được huấn luyện trên trên dữ liệu. mô hình cơ sở này đơn giản chỉ dự đoán giá trị trung bình của biến mục tiêu.
2. **Tính toán lỗi:** Sau khi huấn luyện cây đầu tiên, thuật toán tính toán sai số giữa giá trị dự đoán và giá trị thực tế.

3. **Huấn luyện cây tiếp theo:** Cây tiếp theo sẽ được huấn luyện dựa trên sai số của cây trước đó. Nghĩa là cây này học cách sửa lỗi mà cây trước mắc phải.
4. **Lặp lại quá trình:** Quá trình này được lặp đi lặp lại, với mỗi cây mới cố gắng sửa lỗi của các cây trước đó, cho đến khi đạt đến một tiêu chí dừng nhất định.
5. **Kết hợp các dự đoán:** Dự đoán cuối cùng là tổng hợp các dự đoán từ tất cả các cây.

XGBoost rất hiệu quả vì nó liên tục cải thiện mô hình bằng cách tập trung vào những gì mô hình trước đó làm chưa tốt**.**

2.1.1.3 Support Vector Machine (SVM)

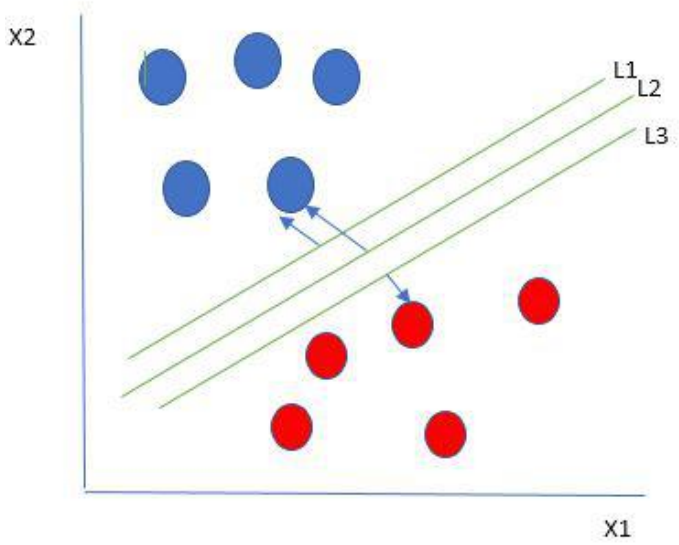
Support Vector Machine (SVM) là một thuật toán học máy được giám sát được sử dụng cho các tác vụ phân loại và hồi quy. Nó cố gắng tìm ra ranh giới tốt nhất được gọi là Hyperlane phân tách các lớp khác nhau trong dữ liệu. Nó rất hữu ích khi bạn muốn phân loại nhị phân. Mục tiêu chính của SVM là tối đa hóa biên độ giữa hai lớp. Biên độ càng lớn thì mô hình càng hoạt động trên dữ liệu mới và chưa thấy. [13]



Hình 3: Giới thiệu về SVM

Cách hoạt động của Thuật toán Support Vector Machine (SVM) [13]

Ý tưởng chính của thuật toán SVM là tìm một siêu phẳng (hyperplane) tốt nhất để phân tách hai lớp dữ liệu bằng cách tối đa hóa khoảng cách (margin) giữa chúng. Margin là khoảng cách từ siêu phẳng đến các điểm dữ liệu gần nhất thuộc mỗi lớp, gọi là support vectors (vector hỗ trợ).



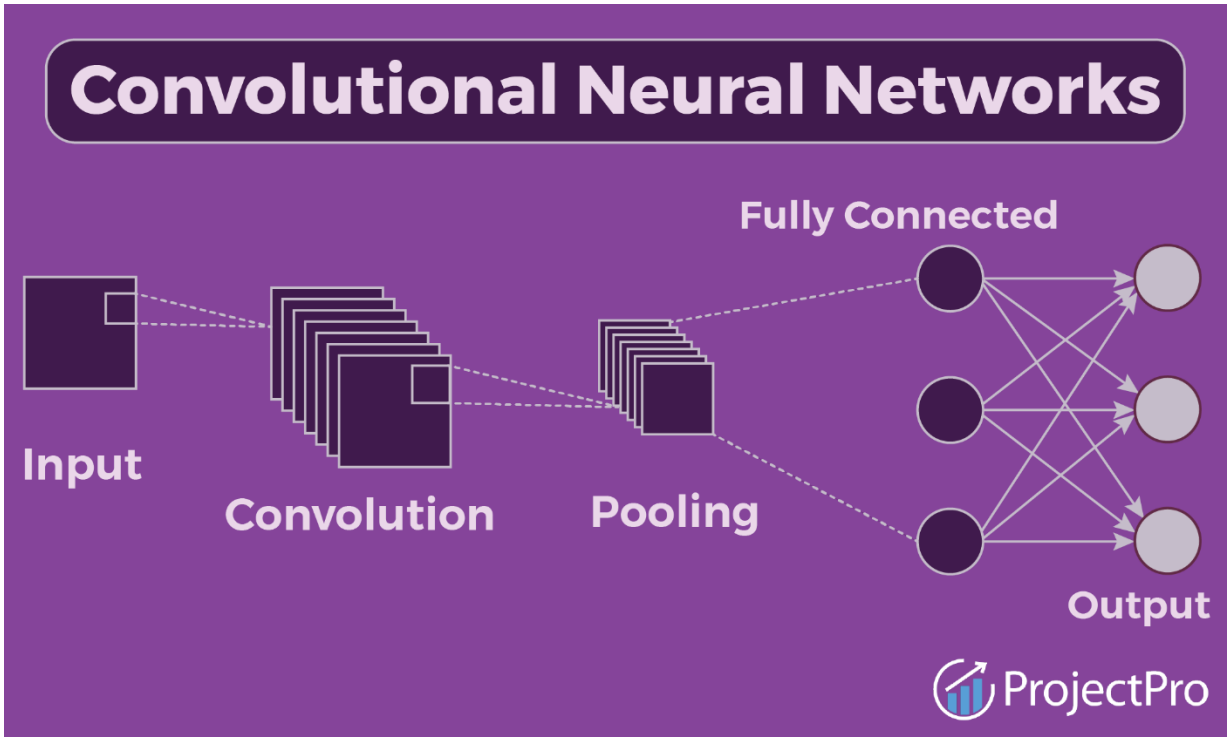
Hình 4: Cách hoạt động của SVM

2.1.2 Deep Learning

2.1.2.1 CNN (Convolutional Neural Network)

CNN hay Mạng Nơ-ron Tích Chập, là một loại mạng nơ-ron nhân tạo thuộc nhóm học sâu (deep learning), được thiết kế đặc biệt để xử lý dữ liệu có cấu trúc lưới, chẳng hạn như hình ảnh, video, hoặc dữ liệu âm thanh. CNN nổi bật trong các nhiệm vụ thị giác máy tính nhờ khả năng tự động học các đặc trưng không gian từ dữ liệu đầu vào, từ các đặc trưng cơ bản như cạnh và góc đến các đặc trưng phức tạp hơn như khuôn mặt hoặc đối tượng.

Nguồn gốc của CNN được lấy cảm hứng từ cách hệ thống thị giác của con người hoạt động, đặc biệt là cách vỏ não thị giác xử lý thông tin hình ảnh. Nghiên cứu của Hubel và Wiesel vào năm 1968 về các trường tiếp nhận trong vỏ não thị giác của mèo đã đặt nền tảng cho ý tưởng này, dẫn đến sự phát triển của các mô hình như neocognitron của Fukushima vào năm 1980 và sau đó là LeNet-5 của Yann LeCun vào những năm 1990, áp dụng cho nhận dạng chữ viết tay.



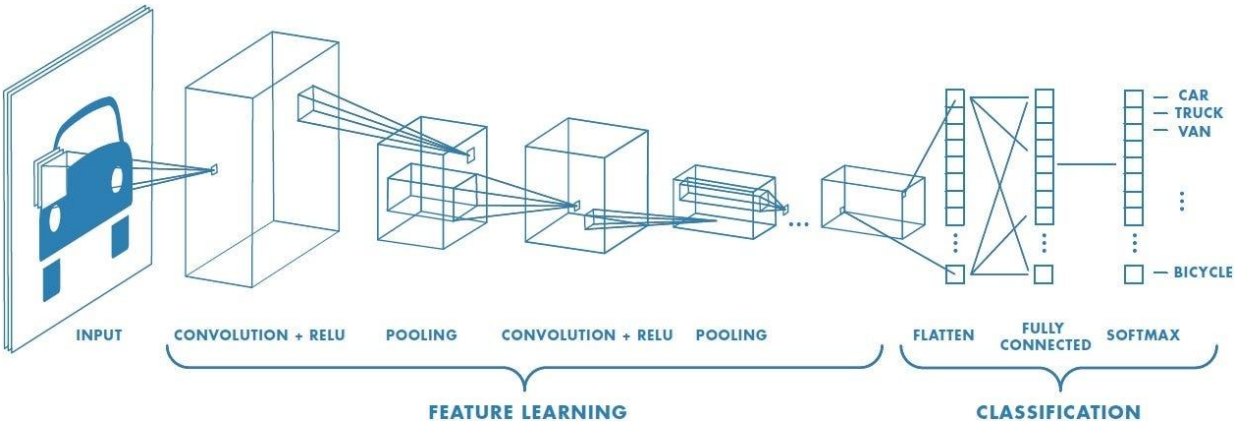
Hình 5: Các lớp cơ bản của CNN

Cách hoạt động cơ bản

- **Lớp tích chập:** Các bộ lọc nhỏ (filters) được áp dụng lên hình ảnh để tạo ra bản đồ đặc trưng, trích xuất các đặc trưng như cạnh, góc, hoặc mẫu phức tạp.
- **Lớp pooling:** Giảm kích thước bản đồ đặc trưng, chẳng hạn bằng cách lấy giá trị lớn nhất (max pooling), giúp giảm số lượng tham số và tăng tính bất biến với vị trí.
- **Lớp fully connected:** Kết nối tất cả neuron để phân loại hoặc dự đoán dựa trên các đặc trưng đã trích xuất, thường sử dụng hàm softmax.

Các thành phần bổ sung

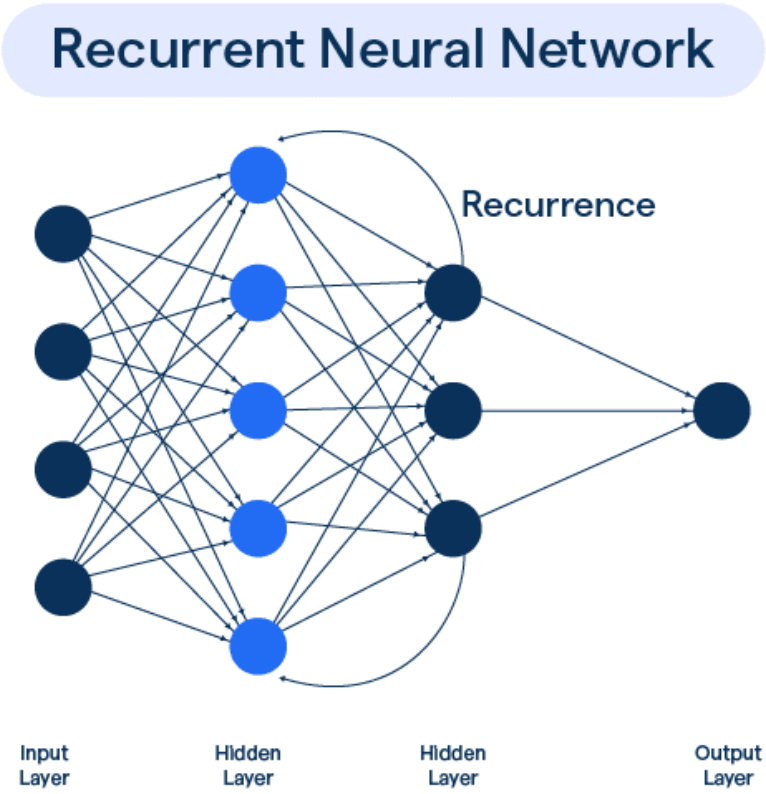
- **Lớp ReLU:** Được áp dụng sau mỗi lớp tích chập để giới thiệu tính phi tuyến, giúp mạng học được các đặc trưng phức tạp hơn.
- **Lớp Loss:** Sử dụng các hàm mất mát như Softmax cho các lớp phân loại độc quyền (mutually exclusive), Sigmoid cross-entropy cho các xác suất độc lập, hoặc Euclidean loss cho các nhãn giá trị thực.



Hình 6: Cách hoạt động cơ bản của CNN

2.1.2.2 RNN (Recurrent Neural Network) & LSTM

RNN hay Mạng Nơ-ron Hồi Quy, là một loại mạng nơ-ron nhân tạo được thiết kế đặc biệt để xử lý dữ liệu có tính chuỗi, như văn bản, âm thanh, hoặc chuỗi thời gian. Điểm đặc biệt của RNN là nó có các kết nối phản hồi (loops) cho phép thông tin được truyền từ các bước trước đến các bước sau. Điều này giúp mạng có khả năng "nhớ" các thông tin từ các bước trước, làm cho nó phù hợp với các nhiệm vụ mà thứ tự của dữ liệu có ý nghĩa, như dự đoán từ tiếp theo trong một câu hoặc phân tích chuỗi thời gian. RNN được chú ý nhiều trong xử lý ngôn ngữ tự nhiên nhờ khả năng xử lý thông tin tuần tự.



Hình 7: Giới thiệu về RNN

LSTM (Long Short-Term Memory) là một biến thể cải tiến của RNN, được thiết kế để giải quyết vấn đề phụ thuộc dài hạn mà RNN thông thường gặp phải. LSTM có cấu trúc phức tạp hơn với các cổng điều khiển (gates) để quản lý việc lưu trữ và truy xuất thông tin trong bộ nhớ. Nhờ đó, LSTM có khả năng nhớ thông tin trong một khoảng thời gian dài mà không gặp vấn đề vanishing gradient. LSTM được giới thiệu bởi Hochreiter & Schmidhuber (1997) và đã được cải tiến bởi nhiều nhà nghiên cứu. Chúng hoạt động hiệu quả trong nhiều bài toán như nhận dạng ngôn ngữ, dịch máy, hoặc phân tích chuỗi thời gian.

Hình 8: Giới thiệu về LSTM

Ưu điểm của LSTM so với RNN

- **Khả năng nhớ dài hạn**: LSTM có thể lưu trữ và truy xuất thông tin từ các bước xa trong quá khứ nhờ vào cấu trúc cổng và trạng thái ô.
- **Giảm vanishing gradient**: Các cổng điều khiển giúp kiểm soát dòng chảy của gradient, giảm thiểu vấn đề gradient biến mất.
- **Ứng dụng rộng rãi**: LSTM được sử dụng trong nhiều lĩnh vực như dịch máy, phân tích cảm xúc, dự đoán chuỗi thời gian

2.2 Các bước huấn luyện mô hình

2.2.1 Chuẩn bị dataset

Dataset dùng trong đồ án này Internet Firewall Dataset nguồn từ Fatih Ertam, Firat University, Turkey. Có tổng cộng 12 đặc trưng và 4 nhãn.

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Action	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received
0	57222	53	54587	53	allow	177	94	83	2	30	1	1
1	56258	3389	56258	3389	allow	4768	1600	3168	19	17	10	9
2	6881	50321	43265	50321	allow	238	118	120	2	1199	1	1
3	50553	3389	50553	3389	allow	3327	1438	1889	15	17	8	7
4	50002	443	45848	443	allow	25358	6778	18580	31	16	13	18
5	51465	443	39975	443	allow	3961	1595	2366	21	16	12	9
6	60513	47094	45469	47094	allow	320	140	180	6	7	3	3
7	50049	443	21285	443	allow	7912	3269	4643	23	96	12	11
8	52244	58774	2211	58774	allow	70	70	0	1	5	1	0
9	50627	443	16215	443	allow	8256	1674	6582	31	75	15	16

Hình 9: Dataset

2.2.2 Phân tích dữ liệu

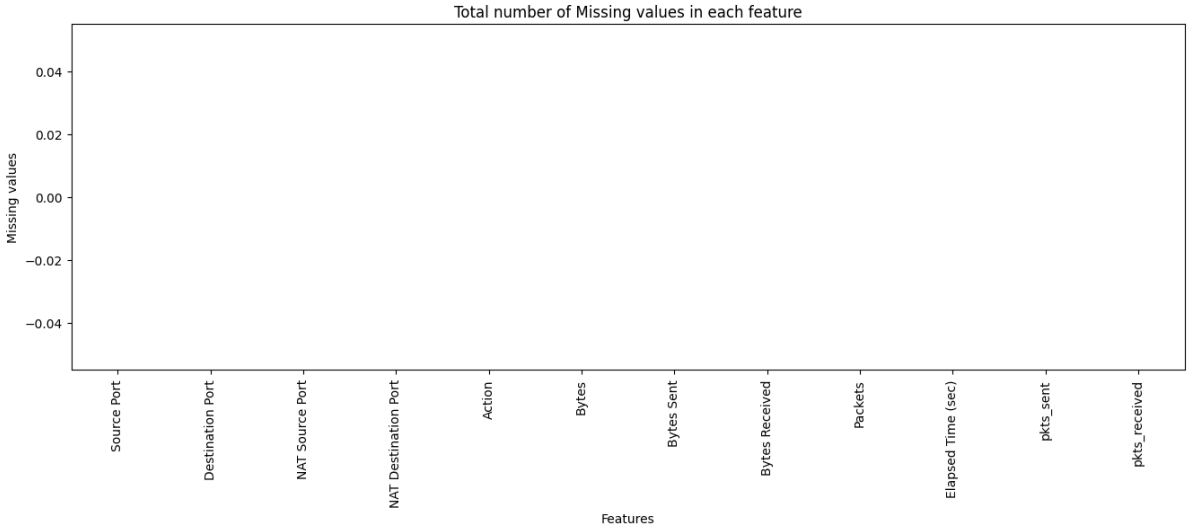
- Dataset ban đầu có tổng cộng 65532 dòng và 12 cột

Số lượng dòng và cột

```
[3]  print("This Dataset has {} rows and {} columns".format(df.shape[0], df.shape[1]))

     This Dataset has 65532 rows and 12 columns
```

Hình 10: Số dòng và số cột

- Thông tin về số lượng giá trị null trong 12 cột và kiểu dữ liệu của nó



Hình 11: Số lượng giá trị null trong 12 cột

Thông tin về null và kiểu dữ liệu

```
[4]  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65532 entries, 0 to 65531
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Source Port           65532 non-null  int64
 1   Destination Port      65532 non-null  int64
 2   NAT Source Port       65532 non-null  int64
 3   NAT Destination Port  65532 non-null  int64
 4   Action                65532 non-null  object
 5   Bytes                 65532 non-null  int64
 6   Bytes Sent            65532 non-null  int64
 7   Bytes Received        65532 non-null  int64
 8   Packets               65532 non-null  int64
 9   Elapsed Time (sec)    65532 non-null  int64
 10  pkts_sent             65532 non-null  int64
 11  pkts_received         65532 non-null  int64
dtypes: int64(11), object(1)
memory usage: 6.0+ MB
```

Hình 12: Số lượng null và kiểu dữ liệu từng cột

- Xem các nhãn và số lượng phân phối nhãn. Có tổng cộng 4 nhãn là allow, deny, drop, reset-both; ta thấy mất cân bằng dữ liệu nghiêm trọng.

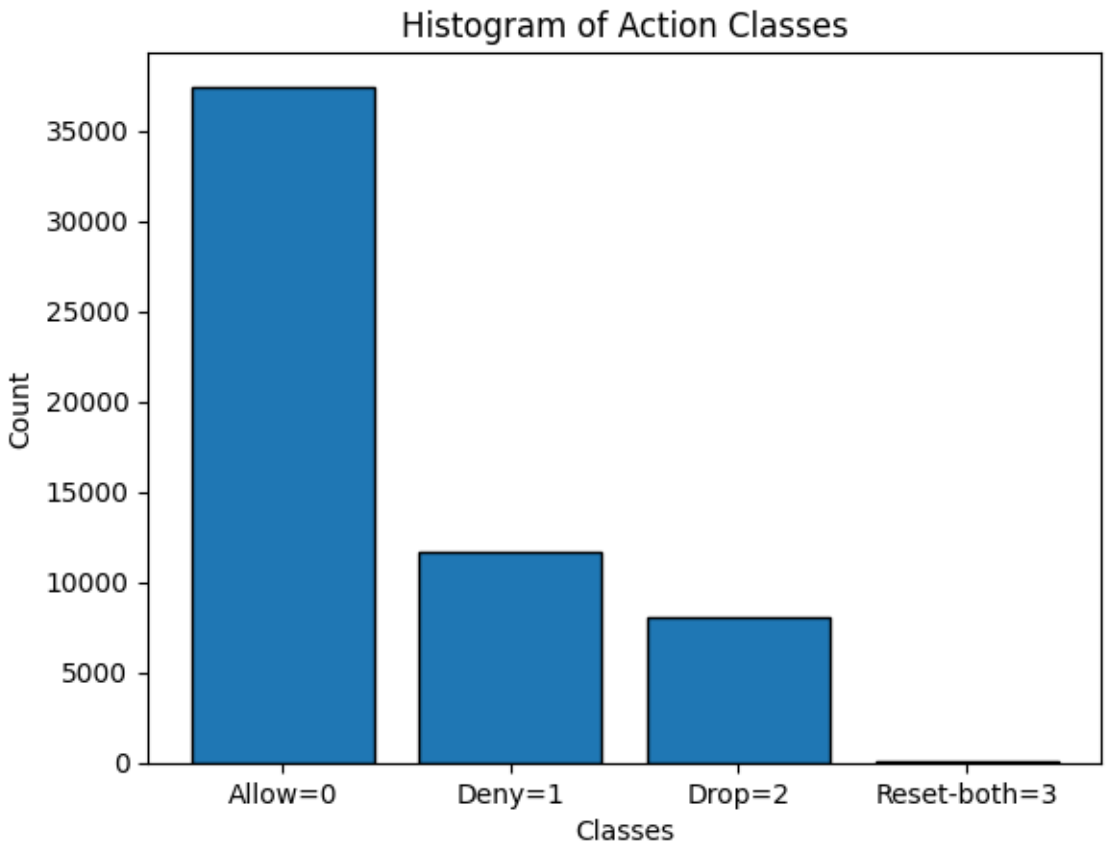
Xem phân phối nhãn

```
[6] df['Action'].value_counts()
```

Action	count
allow	37640
deny	14987
drop	12851
reset-both	54

dtype: int64

Hình 13: Phân phối nhãn



Hình 14: Biểu đồ phân phối nhãn

2.2.3 Tiền xử lý dữ liệu

- Xóa các dòng dữ liệu trùng lặp, sau khi xóa số dòng giảm xuống còn 57170 dòng

Xóa các dòng trùng lặp

```
[10] df.drop_duplicates(inplace=True, keep='first')
```

Hiển thị lại số dòng và cột

```
[11] print("This Dataset has {} rows and {} columns".format(df.shape[0], df.shape[1]))
```

```
This Dataset has 57170 rows and 12 columns
```

Hình 15: Xóa dữ liệu trùng lặp

- Chuyển đổi nhãn thành số

```
# Chuyển đổi nhãn thành số
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)  # Chuyển 'allow', 'deny', 'drop', 'reset-both' thành số

# Hiển thị ánh xạ giữa nhãn gốc và số
for i, label in enumerate(label_encoder.classes_):
    print(f"{label} => {i}")
```

```
allow => 0
deny => 1
drop => 2
reset-both => 3
```

Hình 16: Ánh xạ nhãn sang số

2.2.4 Chuẩn bị dữ liệu trước khi huấn luyện mô hình

- Chia tập train và train test với tỉ lệ 80/20, stratify=y đảm bảo tỉ lệ các nhãn trong tập train, test được giữ nguyên, random_state được dùng để đảm bảo tính tái lập.

Chia tập train, test

```
[49] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

Hình 17: Chia tập train, test

- Sử dụng SMOTE để xử lý mất cân bằng dữ liệu: SMOTE (Synthetic Minority Over-sampling Technique) là một kỹ thuật xử lý mất cân bằng dữ liệu bằng cách tạo ra các mẫu giả (synthetic samples) cho lớp thiểu số.

Sử dụng SMOTE để xử lý mất cân bằng dữ liệu

```
# Dùng SMOTE để cân bằng dữ liệu
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

Hình 18: Sử dụng SMOTE

- Số lượng phân phối nhãn được cân bằng sau khi sử dụng SMOTE đều là 29951

Số lượng mỗi nhãn sau khi dữ dụng SMOTE

```
[51] # Kiểm tra số lượng các nhãn trong dữ liệu sau khi sử dụng SMOTE
     print(Counter(y_resampled))
```

```
Counter({np.int64(1): 29951, np.int64(0): 29951, np.int64(2): 29951, np.int64(3): 29951})
```

Hình 19: Số lượng nhãn sau khi xử lý mất cân bằng

2.2.5 Huấn luyện mô hình Machine Learning

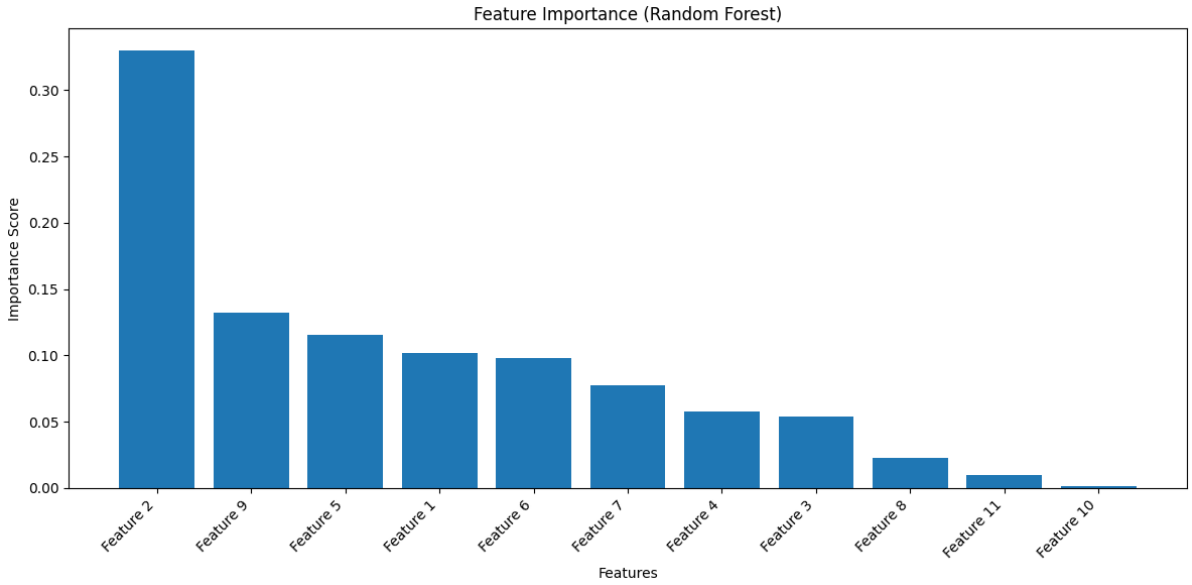
2.2.5.1 Random Forest

- Huấn luyện mô hình Random Forest

```
# Huấn luyện Random Forest
rf_model = RandomForestClassifier(class_weight='balanced', random_state=42)
rf_model.fit(X_resampled, y_resampled)
rf_pred = rf_model.predict(X_test)
```

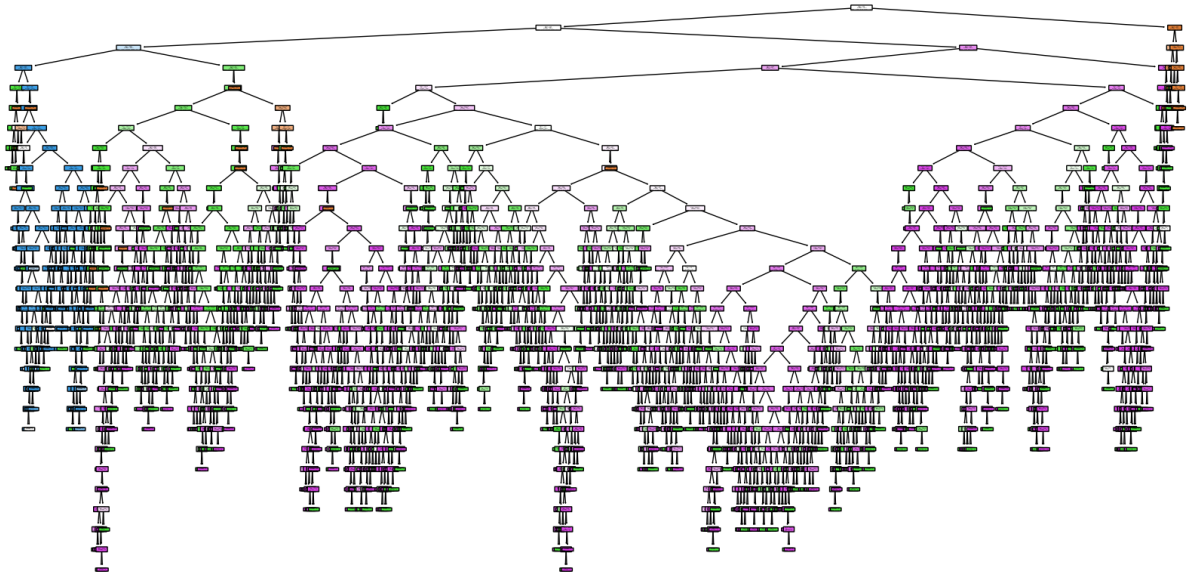
Hình 20: Huấn luyện mô hình RF

- Sắp xếp các đặc trưng quan trọng ảnh hưởng đến việc đưa ra quyết định



Hình 21: Biểu đồ đặc trưng quan trọng

- Vẽ cây quyết định đầu tiên trong mô hình RF



Hình 22: Cây quyết định đầu tiên trong RF

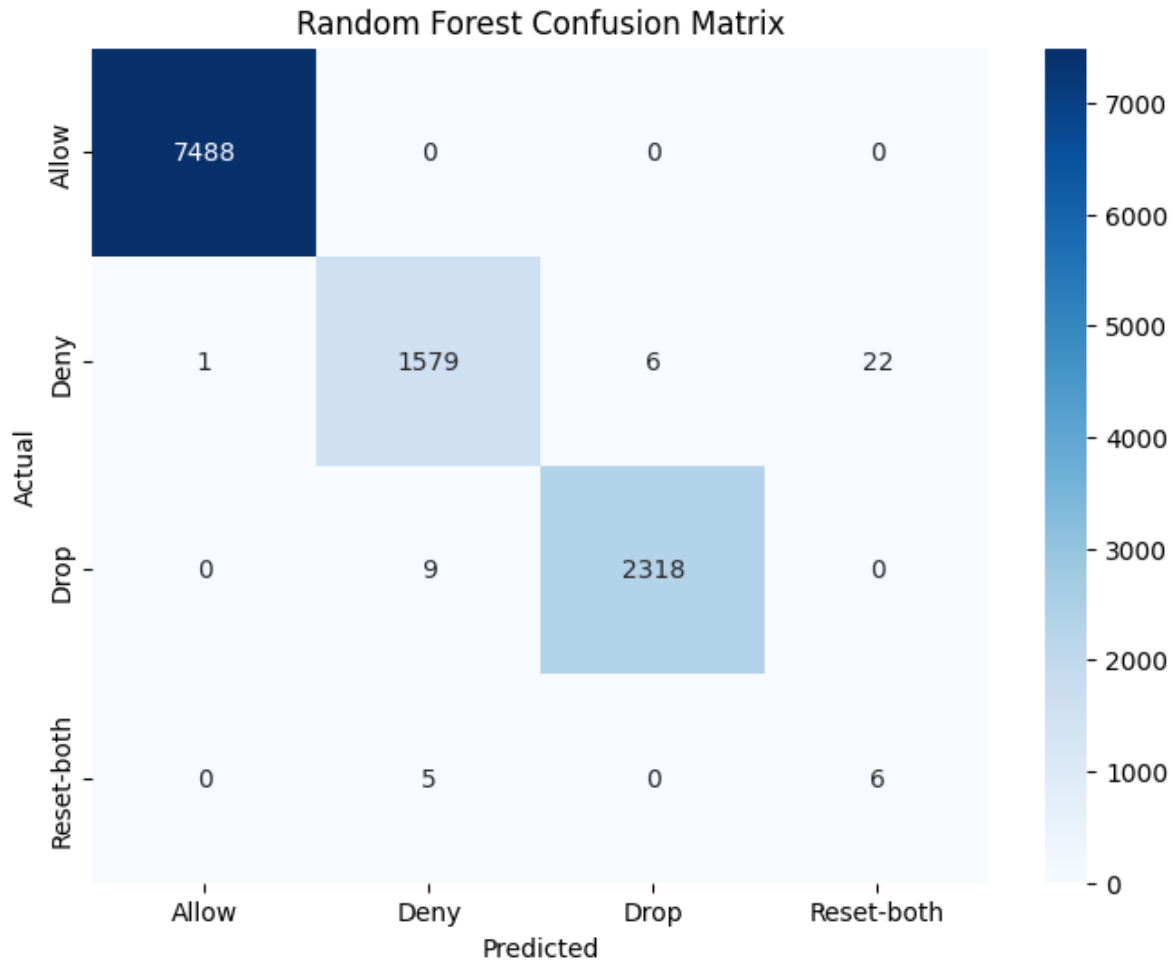
- Đánh giá mô hình qua các điểm như Accurancy, F1, Prescision, Recall (Tham số average=macro: tính chỉ số cho từng lớp riêng biệt, sau đó lấy trung bình không trọng số)

```
# Tính các chỉ số
rf_accuracy = accuracy_score(y_test, rf_pred)
rf_f1 = f1_score(y_test, rf_pred, average='macro')
rf_precision = precision_score(y_test, rf_pred, average='macro')
rf_recall = recall_score(y_test, rf_pred, average='macro')

# In kết quả
print('\n🔍 Random Forest Metrics:')
print(f'Accuracy:  {rf_accuracy:.4f}')
print(f'F1 Score:  {rf_f1:.4f}')
print(f'Precision: {rf_precision:.4f}')
print(f'Recall:    {rf_recall:.4f}')
```

```
🔍 Random Forest Metrics:
Accuracy:  0.9962
F1 Score:  0.8227
Precision: 0.8007
Recall:    0.8809
```

Hình 23: Đánh giá chỉ số của RF

- Ma trận nhầm lẫn (Confusion Matrix) của RF



Hình 24: Ma trận nhầm lẫn của RF

2.2.5.2 Xgboost

- Huấn luyện mô hình XGBoost

```
xgb_model = XGBClassifier(eval_metric='mlogloss', random_state=42)
xgb_model.fit(X_resampled, y_resampled)
xgb_pred = xgb_model.predict(X_test)
```

Hình 25: Huấn luyện mô hình XGBoost

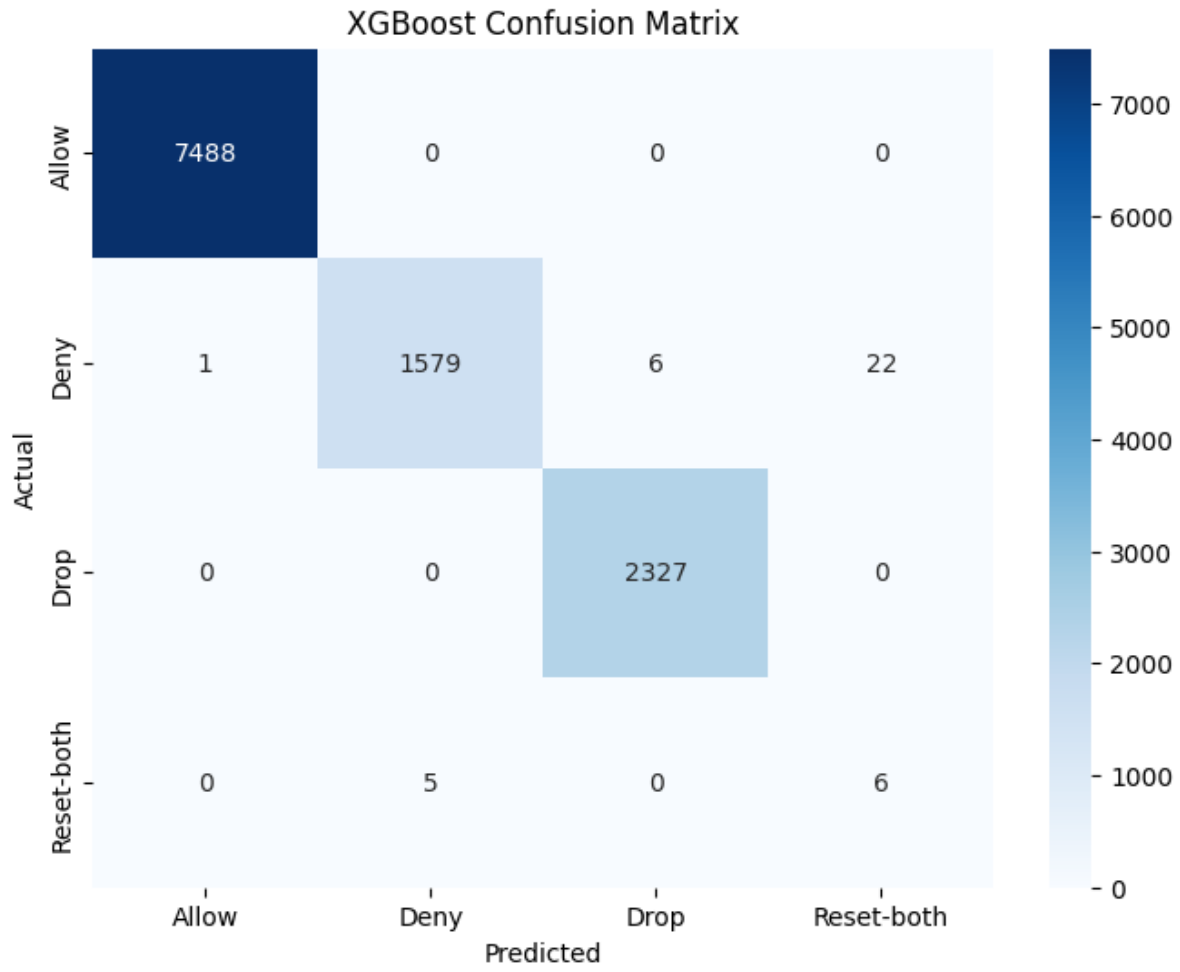
- Đánh giá mô hình giống như RF

```python
# Tính các chỉ số
xgb_accuracy = accuracy_score(y_test, xgb_pred)
xgb_f1 = f1_score(y_test, xgb_pred, average='macro')
xgb_precision = precision_score(y_test, xgb_pred, average='macro')
xgb_recall = recall_score(y_test, xgb_pred, average='macro')

# In kết quả
print('\n🔍 XGBoost Metrics:')
print(f'Accuracy:  {xgb_accuracy:.4f}')
print(f'F1 Score:  {xgb_f1:.4f}')
print(f'Precision: {xgb_precision:.4f}')
print(f'Recall:    {xgb_recall:.4f}')
```

```
🔍 XGBoost Metrics:
Accuracy:  0.9970
F1 Score:  0.8239
Precision: 0.8021
Recall:    0.8819
```

Hình 26: Đánh giá các chỉ số của XGBoost

- Ma trận nhầm lẫn cho XGBoost



Hình 27: Ma trận nhầm lẫn của XGBoost

2.2.5.3 SVM

- Cần phải chuẩn hóa dữ liệu trước khi sử dụng SVM

```
# Do SVM nhạy cảm với dữ liệu lớn và không scale ==> chuẩn hóa dữ liệu
scaler = StandardScaler()
X_resampled_scaled = scaler.fit_transform(X_resampled)
X_test_scaled = scaler.transform(X_test)
```

Hình 28: Chuẩn hóa dữ liệu

- Huấn luyện mô hình LinearSVC (mô hình SVM tuyến tính dùng trong bài toán phân loại) và CalibratedClassifierCV: một wrapper dùng để hiệu chỉnh đầu ra của mô hình

```
# Fit LinearSVC với xác suất thông qua CalibratedClassifierCV
base_svc = LinearSVC(class_weight='balanced', max_iter=10000, random_state=42)
svm_model = CalibratedClassifierCV(base_svc, cv=3)
svm_model.fit(X_resampled_scaled, y_resampled)
svm_pred = svm_model.predict(X_test_scaled)
```

Hình 29: Huấn luyện LinearSVC

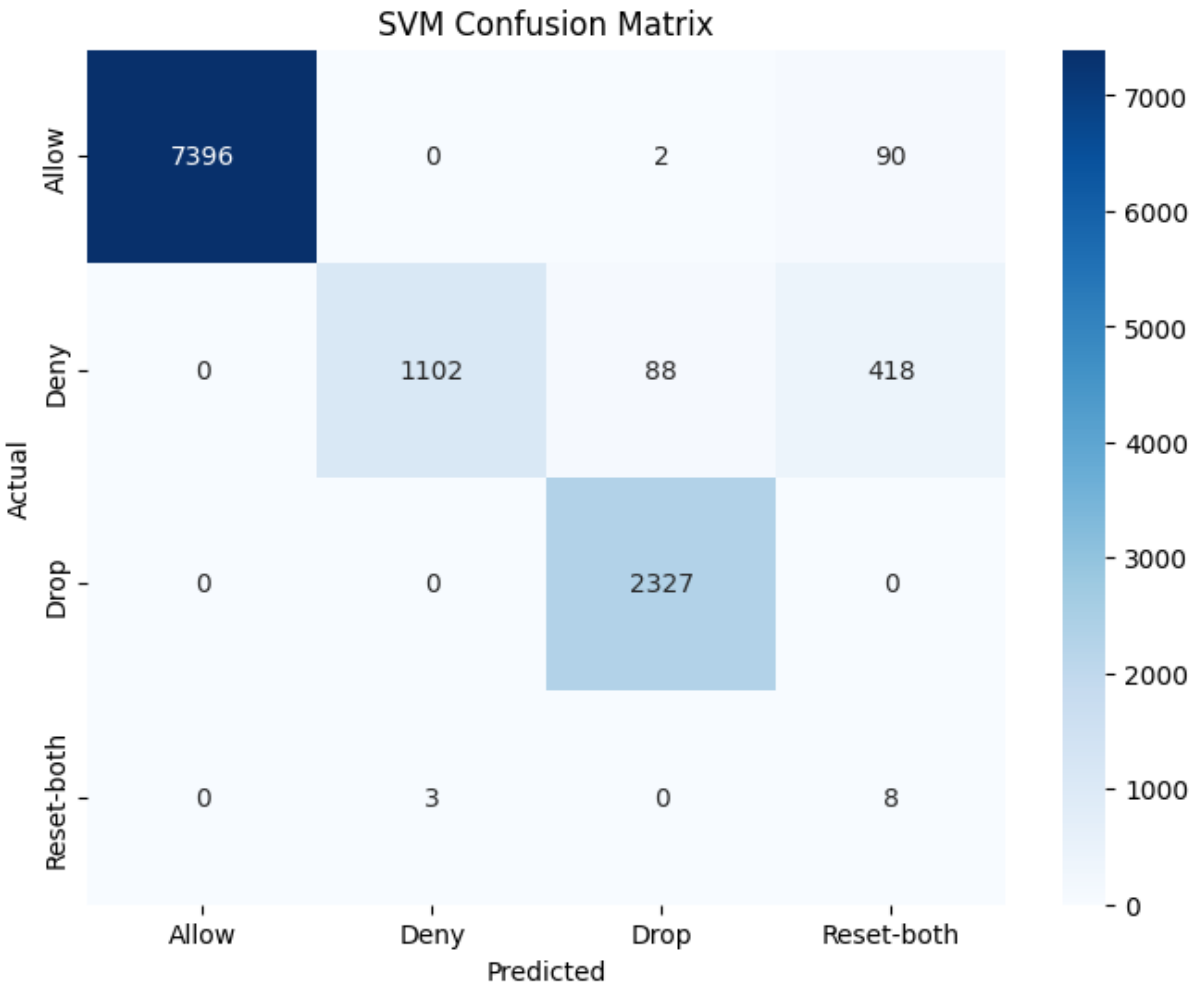
- Đánh giá mô hình cũng giống với RF và XGBoost

```
# Tính các chỉ số
svm_accuracy = accuracy_score(y_test, svm_pred)
svm_f1 = f1_score(y_test, svm_pred, average='macro')
svm_precision = precision_score(y_test, svm_pred, average='macro')
svm_recall = recall_score(y_test, svm_pred, average='macro')

# In kết quả
print('\n🔍 SVM Metrics:')
print(f'Accuracy:  {svm_accuracy:.4f}')
print(f'F1 Score:  {svm_f1:.4f}')
print(f'Precision: {svm_precision:.4f}')
print(f'Recall:    {svm_recall:.4f}')
```

```
🔍 SVM Metrics:
Accuracy:  0.9474
F1 Score:  0.7044
Precision: 0.7439
Recall:    0.8501
```

Hình 30: Đánh giá các chỉ số của SVM

- Ma trận nhầm lẫn cho SVM



Hình 31: Ma trận nhầm lẫn của SVM

2.2.5.4 So sánh các mô hình ML

- Bảng tổng hợp các chỉ số đánh giá (metrics) của 3 mô hình Machine Learning: **Random Forest**, **XGBoost**, và **SVM**

Model	Accuracy	F1 Score	Precision	Recall
Random Forest	0.9962	0.8227	0.8007	0.8809
XGBoost	0.9970	0.8239	0.8021	0.8819
SVM	0.9474	0.7044	0.7439	0.8501

Bảng 2: Bảng tổng hợp chỉ số đánh giá cho mô hình ML

  - Accuracy (Độ chính xác):
    - XGBoost đạt Accuracy cao nhất (0.9970), theo sát là Random Forest (0.9962).
    - SVM thấp hơn rõ rệt (0.9474).
  - F1 Score (Độ cân bằng Precision và Recall):
    - XGBoost dẫn đầu nhẹ (0.8239) so với Random Forest (0.8227).

        - SVM có F1 thấp hơn nhiều (0.7044), cho thấy mô hình chưa cân bằng tốt Precision và Recall.
    - Precision (Độ chính xác khi dự đoán dương tính) & Recall (Khả năng tìm đúng các mẫu dương tính):
        - XGBoost có Precision và Recall đều cao hơn hoặc ngang với Random Forest.
        - SVM lại có Precision thấp nhất (0.7439) nhưng Recall khá cao (0.8501), nghĩa là mô hình này có xu hướng ít bỏ sót nhưng gây ra nhiều báo động giả hơn.

XGBoost là mô hình tốt nhất tổng thể trong cả 4 tiêu chí: Accuracy, F1 Score, Precision và Recall. Random Forest cũng rất mạnh, chỉ thua XGBoost một chút. SVM yếu hơn đáng kể, đặc biệt là về Accuracy và F1 Score.

- Bảng so sánh về Confusion Matrix

Mô hình	Ưu điểm	Hạn chế
Random Forest	Rất chính xác với "Allow", "Drop"; dự đoán tốt toàn cục.	Một số nhầm lẫn nhỏ ở "Deny" và "Reset-both".
XGBoost	Chính xác nhất, cải thiện so với Random Forest ở "Drop".	Vẫn có chút nhầm lẫn nhỏ như RF ở "Deny" và "Reset-both".
SVM	Dự đoán "Drop" tốt	Nhầm lẫn nghiêm trọng ở "Deny", "Reset-both", "Allow". Accuracy thấp hơn đáng kể.

Bảng 3: Bảng so sánh về ma trận nhầm lẫn cho mô hình ML

- So sánh 3 mô hình trên chỉ số AUC **(**Area Under the Curve) cho các mô hình phân loại đa lớp

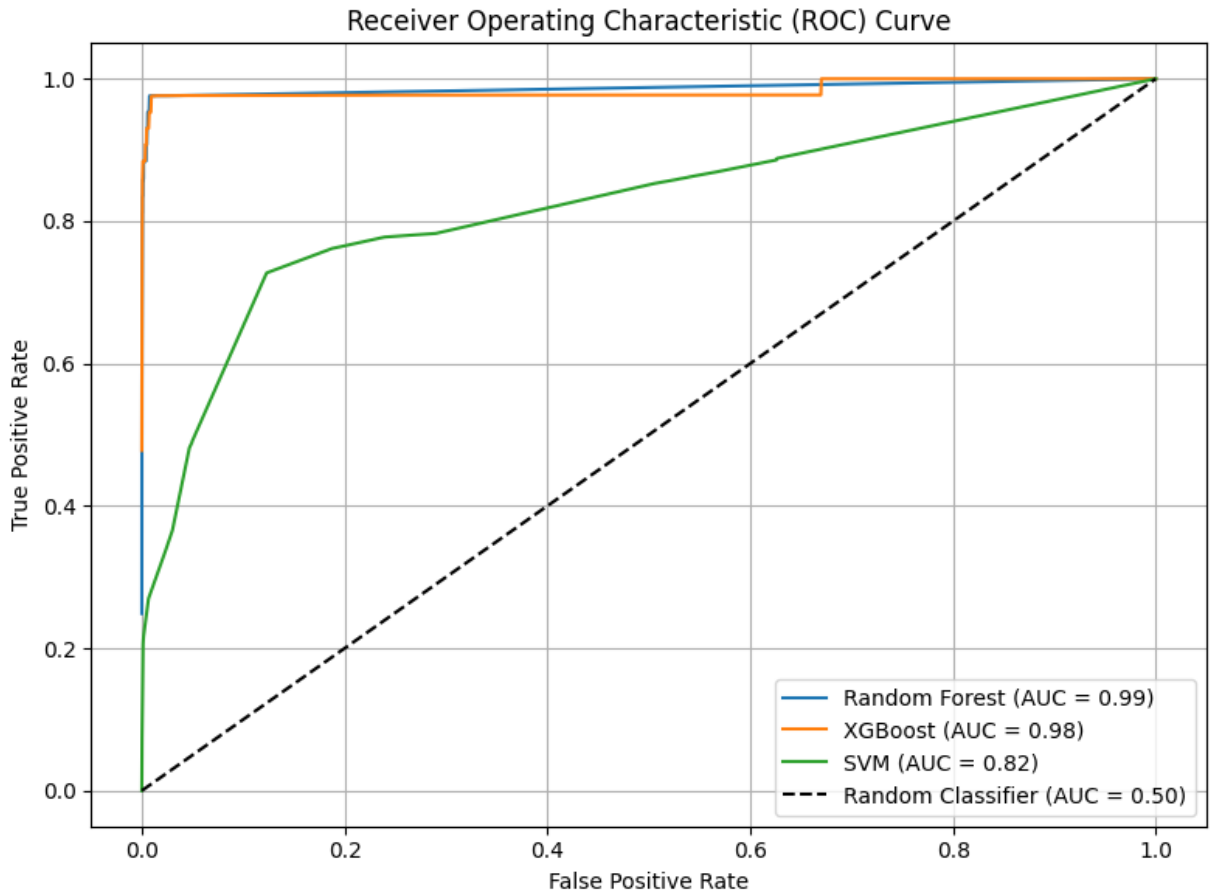
```python
# Giả sử nhãn là 0, 1, 2, 3
y_test_bin = label_binarize(y_test, classes=[0, 1, 2, 3])

# Predict probability
rf_proba = rf_model.predict_proba(X_test)
xgb_proba = xgb_model.predict_proba(X_test)
svm_proba = svm_model.predict_proba(X_test)

# Tính AUC macro
auc_rf = roc_auc_score(y_test_bin, rf_proba, multi_class='ovr', average='macro')
auc_xgb = roc_auc_score(y_test_bin, xgb_proba, multi_class='ovr', average='macro')
auc_svm = roc_auc_score(y_test_bin, svm_proba, multi_class='ovr', average='macro')


print("Random Forest AUC (macro):", auc_rf)
print("XGBoost AUC (macro):", auc_xgb)
print("LinearSVC AUC (macro):", auc_svm)
```

```
Random Forest AUC (macro): 0.9870832540444934
XGBoost AUC (macro): 0.9838882000123323
LinearSVC AUC (macro): 0.8234084336107579
```

Hình 32: Chỉ số AUC (macro) của các mô hình ML

- Đường cong ROC macro cho các mô hình Random Forest, XGBoost, SVM

c



Hình 33: Đường cong ROC cho 3 mô hình ML

- AUC (ROC macro):
  - Random Forest đạt AUC cao nhất (0.9871), cho thấy khả năng phân biệt giữa các lớp rất tốt.
  - XGBoost xếp thứ hai sát nút (0.9839).
  - SVM kém hơn hẳn (0.8234), xác nhận điều thấy rõ trên đồ thị ROC: đường cong thấp hơn nhiều so với 2 mô hình còn lại.

- Qua tổng thể đánh giá thì mô hình RF và XGBoost có độ chính xác và khả năng dự đoán đúng tốt hơn so với SVM. Nếu ưu tiên hiệu suất tổng thể (accuracy, F1, AUC) thì XGBoost là lựa chọn tốt nhất, với Random Forest là lựa chọn gần tương đương.

2.2.6 Huấn luyện mô hình Deep Learning

2.2.6.1 CNN

- Chuẩn hóa Standard và Reshape X_train và mã hóa one-pot y_train

Standar Scale, Reshape X_train và Mã hóa one-pot y_train

```
[99] scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X_resampled)

     # Reshape cho RNN hoặc CNN: (samples, time_steps, features)
     # Với tabular, giả định mỗi sample là 1 chuỗi với n đặc trưng
     X_reshaped = X_scaled.reshape((X_scaled.shape[0], X_scaled.shape[1], 1))  # time_steps = số đặc trưng

     #Xử lý mã hóa one-hot
     y_cat = to_categorical(y_resampled)
```

Hình 34: Chuẩn hóa và reshape và mã hóa one-pot

- Tạo mô hình và huấn luyện
  - cnn_model = Sequential(): khởi tạo mô hình Sequential.
  - cnn_model.add(Input(shape=(X_reshaped.shape[1], 1))): Đầu vào có kích thước (sequence_length, 1)
  - Convolutional Layer: cnn_model.add(Conv1D(64, kernel_size=3, activation='relu'))
    - Lớp tích chập 1 chiều: áp dụng 64 bộ lọc kích thước 3.
    - Mỗi bộ lọc học các mẫu liên tiếp trong chuỗi.
    - Hàm kích hoạt: ReLU.
  - Global Max Pooling: cnn_model.add(GlobalMaxPooling1D())
    - Lấy giá trị lớn nhất trong mỗi feature map.
    - Rút gọn dữ liệu thành vector 1 chiều đại diện cho toàn bộ chuỗi.
    - Giảm số chiều, giảm tính phụ thuộc vào độ dài chuỗi đầu vào.
  - Fully Connected Layers:
    - cnn_model.add(Dense(32, activation='relu')): Lớp Dense 32 node với ReLU để học đặc trưng cao cấp.
    - cnn_model.add(Dense(y_cat.shape[1], activation='softmax')): Lớp cuối cùng: số node bằng số lớp (y_cat.shape[1]), dùng softmax để phân loại đa lớp.
  - Compile mô hình: cnn_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    - Loss: categorical_crossentropy phù hợp với phân loại đa lớp.
    - Optimizer: adam – tối ưu hóa hiệu quả và phổ biến.
    - Metric: theo dõi độ chính xác (accuracy) trong quá trình huấn luyện.
  - Huấn luyện mô hình: cnn_model.fit(X_reshaped, y_cat, epochs=15, batch_size=64, validation_split=0.2)
    - Dữ liệu đầu vào: X_reshaped là đầu vào 3 chiều.
    - Nhãn: y_cat là dạng one-hot encoding.
    - Số epoch: 15: toàn bộ tập huấn luyện sẽ được đưa qua mô hình 15 lần.
    - Batch size: 64: dữ liệu huấn luyện sẽ được chia thành từng nhóm (mini-batches) gồm 64 mẫu mỗi nhóm.
    - Dùng 20% dữ liệu để đánh giá trên tập validation.

```python
from tensorflow.keras.layers import Input, Conv1D, GlobalMaxPooling1D

cnn_model = Sequential()
cnn_model.add(Input(shape=(X_reshaped.shape[1], 1)))
cnn_model.add(Conv1D(64, kernel_size=3, activation='relu'))
cnn_model.add(GlobalMaxPooling1D())
cnn_model.add(Dense(32, activation='relu'))
cnn_model.add(Dense(y_cat.shape[1], activation='softmax'))

cnn_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
cnn_model.fit(X_reshaped, y_cat, epochs=15, batch_size=64, validation_split=0.2)
```

Hình 35: Huấn luyện mô hình CNN

- Quá trình huấn luyện mô hình CNN

```
Epoch 1/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8298 - loss: 0.5300 - val_accuracy: 0.0838 - val_loss: 1.6513
Epoch 2/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9250 - loss: 0.2286 - val_accuracy: 0.1592 - val_loss: 1.4625
Epoch 3/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9292 - loss: 0.2139 - val_accuracy: 0.1689 - val_loss: 1.2967
Epoch 4/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9302 - loss: 0.2080 - val_accuracy: 0.2298 - val_loss: 1.7374
Epoch 5/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9347 - loss: 0.1950 - val_accuracy: 0.2102 - val_loss: 1.1313
Epoch 6/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.9341 - loss: 0.1927 - val_accuracy: 0.2186 - val_loss: 1.4259
Epoch 7/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9354 - loss: 0.1894 - val_accuracy: 0.2380 - val_loss: 1.1325
Epoch 8/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9368 - loss: 0.1834 - val_accuracy: 0.3694 - val_loss: 1.0482
Epoch 9/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9380 - loss: 0.1790 - val_accuracy: 0.2128 - val_loss: 1.1723
Epoch 10/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9391 - loss: 0.1752 - val_accuracy: 0.3476 - val_loss: 1.1041
Epoch 11/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9400 - loss: 0.1742 - val_accuracy: 0.2066 - val_loss: 1.3121
Epoch 12/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9407 - loss: 0.1677 - val_accuracy: 0.4074 - val_loss: 1.0316
Epoch 13/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.9430 - loss: 0.1615 - val_accuracy: 0.4171 - val_loss: 0.9928
Epoch 14/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9451 - loss: 0.1573 - val_accuracy: 0.5746 - val_loss: 0.9075
Epoch 15/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.9452 - loss: 0.1559 - val_accuracy: 0.4547 - val_loss: 1.0423
```

Hình 36: Quá trình huấn luyện CNN

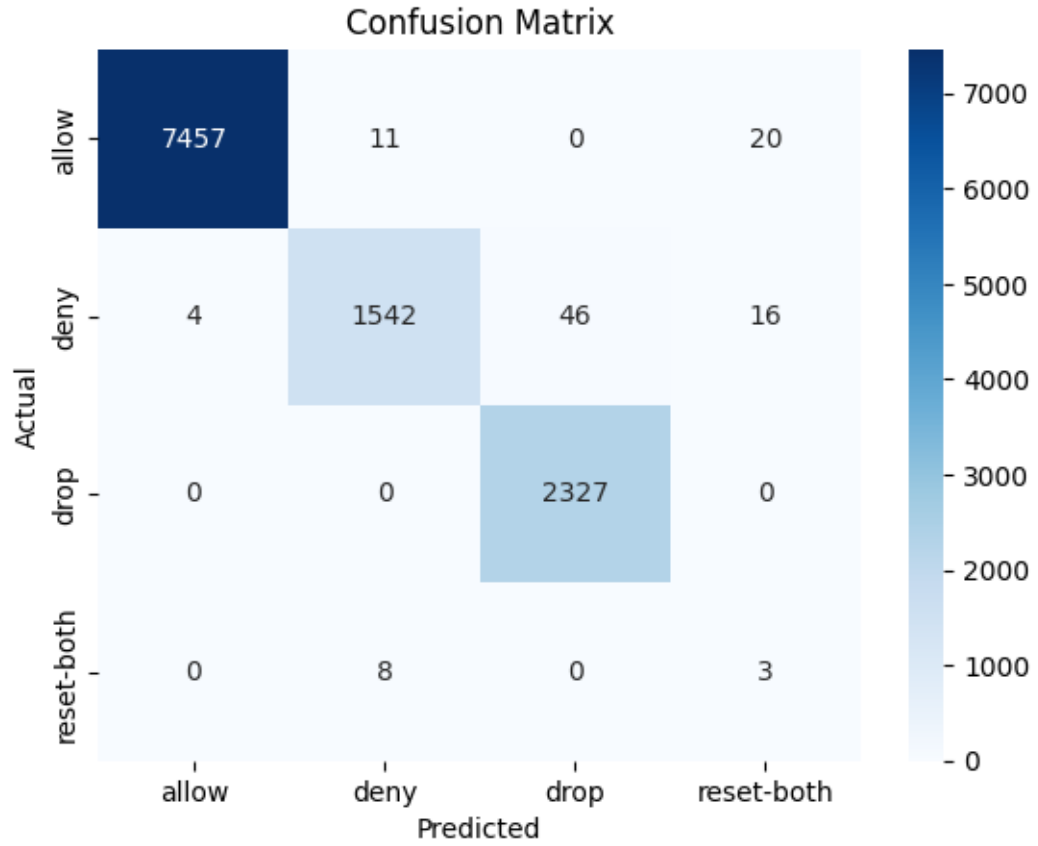
- Đánh giá phân loại nhãn

Classification Report:

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	7488
deny	0.99	0.96	0.97	1608
drop	0.98	1.00	0.99	2327
reset-both	0.08	0.27	0.12	11
accuracy			0.99	11434
macro avg	0.76	0.81	0.77	11434
weighted avg	0.99	0.99	0.99	11434

Hình 37: Báo cáo phân loại của CNN

- Ma trận nhầm lẫn của CNN



Hình 38: Ma trận nhầm lẫn của CNN

2.2.6.2 LSTM

- Bước đầu vẫn phải chuẩn hóa Standard và Reshape X_train và mã hóa one-pot y_train như CNN
- Tạo và huấn luyện mô hình LSTM

- lstm_model = Sequential(): khởi tạo mô hình tuyến tính.
- lstm_model.add(LSTM(64, input_shape=(X_reshaped.shape[1], 1))): Thêm một lớp LSTM với 64 đơn vị ẩn (hidden units), input_shape=(time_steps, 1):
  - time_steps = X_reshaped.shape[1]: số bước thời gian (chiều dài chuỗi).
  - 1: số đặc trưng (features) tại mỗi bước
- Fully Connected Layers:
  - lstm_model.add(Dense(32, activation='relu')): Lớp Dense 32 node với ReLU giúp trích xuất đặc trưng phi tuyến.
  - lstm_model.add(Dense(y_cat.shape[1], activation='softmax')): Lớp cuối cùng: số node bằng số lớp (y_cat.shape[1]), dùng softmax để phân loại đa lớp chuyển đầu ra thành xác suất.
- Compile mô hình: lstm_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  - Loss: categorical_crossentropy phù hợp với phân loại đa lớp.
  - Optimizer: adam – tối ưu hóa hiệu quả và phổ biến.
  - Metric: theo dõi độ chính xác (accuracy) trong quá trình huấn luyện.
- Huấn luyện mô hình: lstm_model.fit(X_reshaped, y_cat, epochs=15, batch_size=64, validation_split=0.2)
  - Dữ liệu đầu vào: X_reshaped là đầu vào 3 chiều.
  - Nhãn: y_cat là dạng one-hot encoding.
  - Số epoch: 15: toàn bộ tập huấn luyện sẽ được đưa qua mô hình 15 lần.
  - Batch size: 64: dữ liệu huấn luyện sẽ được chia thành từng nhóm (mini-batches) gồm 64 mẫu mỗi nhóm.
  - Dùng 20% dữ liệu để đánh giá trên tập validation.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

lstm_model = Sequential()
lstm_model.add(LSTM(64, input_shape=(X_reshaped.shape[1], 1)))
lstm_model.add(Dense(32, activation='relu'))
lstm_model.add(Dense(y_cat.shape[1], activation='softmax'))

lstm_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
lstm_model.fit(X_reshaped, y_cat, epochs=15, batch_size=64, validation_split=0.2)
```

Hình 39: Huấn luyện mô hình LSTM

- Quá trình huấn luyện mô hình LSTM

```
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 18s 10ms/step - accuracy: 0.8209 - loss: 0.5230 - val_accuracy: 0.0995 - val_loss: 1.5099
Epoch 2/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9233 - loss: 0.2299 - val_accuracy: 0.2115 - val_loss: 1.2798
Epoch 3/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9317 - loss: 0.2041 - val_accuracy: 0.2895 - val_loss: 1.2191
Epoch 4/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.9357 - loss: 0.1874 - val_accuracy: 0.2441 - val_loss: 1.2614
Epoch 5/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 17s 11ms/step - accuracy: 0.9384 - loss: 0.1782 - val_accuracy: 0.2948 - val_loss: 1.3017
Epoch 6/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 20s 11ms/step - accuracy: 0.9371 - loss: 0.1789 - val_accuracy: 0.2530 - val_loss: 1.2962
Epoch 7/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 19s 10ms/step - accuracy: 0.9387 - loss: 0.1781 - val_accuracy: 0.3624 - val_loss: 1.1979
Epoch 8/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.9357 - loss: 0.1820 - val_accuracy: 0.1967 - val_loss: 1.4572
Epoch 9/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.9375 - loss: 0.1783 - val_accuracy: 0.2578 - val_loss: 1.3740
Epoch 10/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 20s 11ms/step - accuracy: 0.9403 - loss: 0.1692 - val_accuracy: 0.2374 - val_loss: 1.2788
Epoch 11/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9411 - loss: 0.1654 - val_accuracy: 0.3227 - val_loss: 1.0398
Epoch 12/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 16s 10ms/step - accuracy: 0.9428 - loss: 0.1653 - val_accuracy: 0.4903 - val_loss: 0.9511
Epoch 13/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9441 - loss: 0.1538 - val_accuracy: 0.3517 - val_loss: 1.1025
Epoch 14/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - accuracy: 0.9465 - loss: 0.1505 - val_accuracy: 0.4918 - val_loss: 0.9727
Epoch 15/15
1498/1498 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9439 - loss: 0.1569 - val_accuracy: 0.5347 - val_loss: 0.8500
```

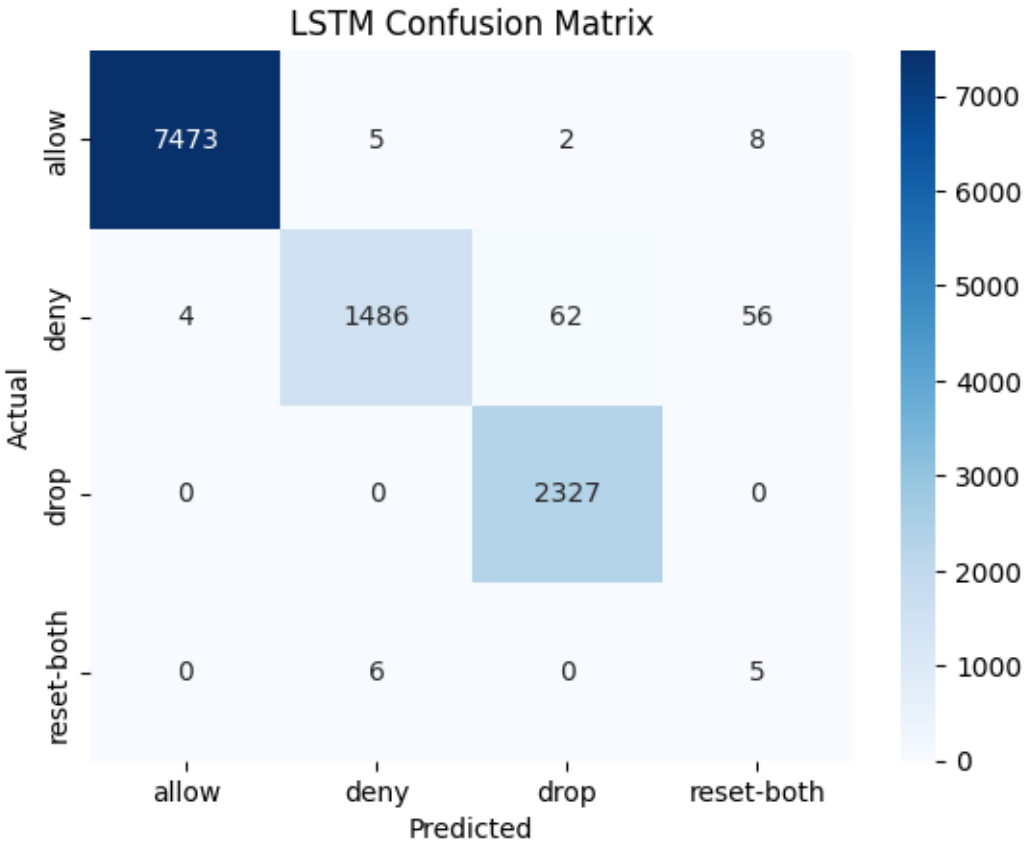
Hình 40: Quá trình huấn luyện LSTM

- Đánh giá phân loại nhãn

```
Classification Report:
              precision    recall  f1-score   support

       allow       1.00      1.00      1.00      7488
        deny       0.99      0.92      0.96      1608
        drop       0.97      1.00      0.99      2327
  reset-both       0.07      0.45      0.12        11

    accuracy                           0.99     11434
   macro avg       0.76      0.84      0.77     11434
weighted avg       0.99      0.99      0.99     11434
```

Hình 41: Báo cáo phân loại của LSTM

- Ma trận nhầm lẫn của LSTM



Hình 42: Ma trận nhầm lẫn của LSTM

2.2.6.3 So sánh hai mô hình DL

- Bảng so sánh hai mô hình CNN và LSTM dựa trên báo cáo phân loại

Metric	**Class**	**CNN**	**LSTM**
Precision	allow	1.00	1.00
	deny	0.99	0.99
	drop	0.98	0.97
	reset-both	0.08	0.07
Recall	allow	1.00	1.00
	deny	0.96	0.92
	drop	1.00	1.00
	reset-both	0.27	0.45
F1-Score	allow	1.00	1.00
	deny	0.97	0.96
	drop	0.99	0.99

	reset-both	0.12	0.12
Accuracy	All classes	0.99	0.99
Macro Avg	All classes	0.76 / 0.81 / 0.77	0.76 / 0.84 / 0.77
Weighted Avg	All classes	0.99 / 0.99 / 0.99	0.99 / 0.99 / 0.99

Bảng 4: Bảng so sánh báo cáo phân loại cho mô hình DL

- Cả hai mô hình đều có độ chính xác tổng thể (accuracy) rất cao: 0.99.
- LSTM cải thiện rõ rệt recall cho lớp hiếm reset-both (0.45 so với 0.27 của CNN), tuy precision vẫn thấp.
- CNN có precision cao hơn một chút với lớp drop và deny, nhưng recall thấp hơn ở lớp reset-both.
- Macro avg cho recall của LSTM (0.84) cao hơn CNN (0.81), điều này phản ánh mô hình LSTM xử lý các lớp không cân bằng tốt hơn một chút.

- Bảng so sánh về Confusion Matrix

Tiêu chí	**LSTM**	**CNN**
Allow	Dự đoán chính xác hơn một chút	CNN có nhiều nhầm lẫn hơn (Allow → Deny, Reset-both)
Deny	Nhầm lẫn đáng kể sang "Drop" và "Reset-both"	Dự đoán Deny tốt hơn (cao hơn ở đúng cột)
Drop	Dự đoán hoàn hảo	Dự đoán hoàn hảo
Reset-both	LSTM nhầm lẫn nhiều hơn	CNN nhầm ít hơn

Bảng 5: Bảng so sánh ma trận nhầm lẫn cho mô hình DL

- LSTM hoạt động ổn định hơn ở lớp "Allow", ít sai sót hơn CNN.
- CNN lại phân loại "Deny" và "Reset-both" tốt hơn so với LSTM.
- Cả hai đều phân loại "Drop" rất tốt, không có lỗi.
- LSTM có vấn đề với nhầm lẫn "Deny → Drop/Reset-both", trong khi CNN giữ được tập trung hơn vào các lớp cụ thể.

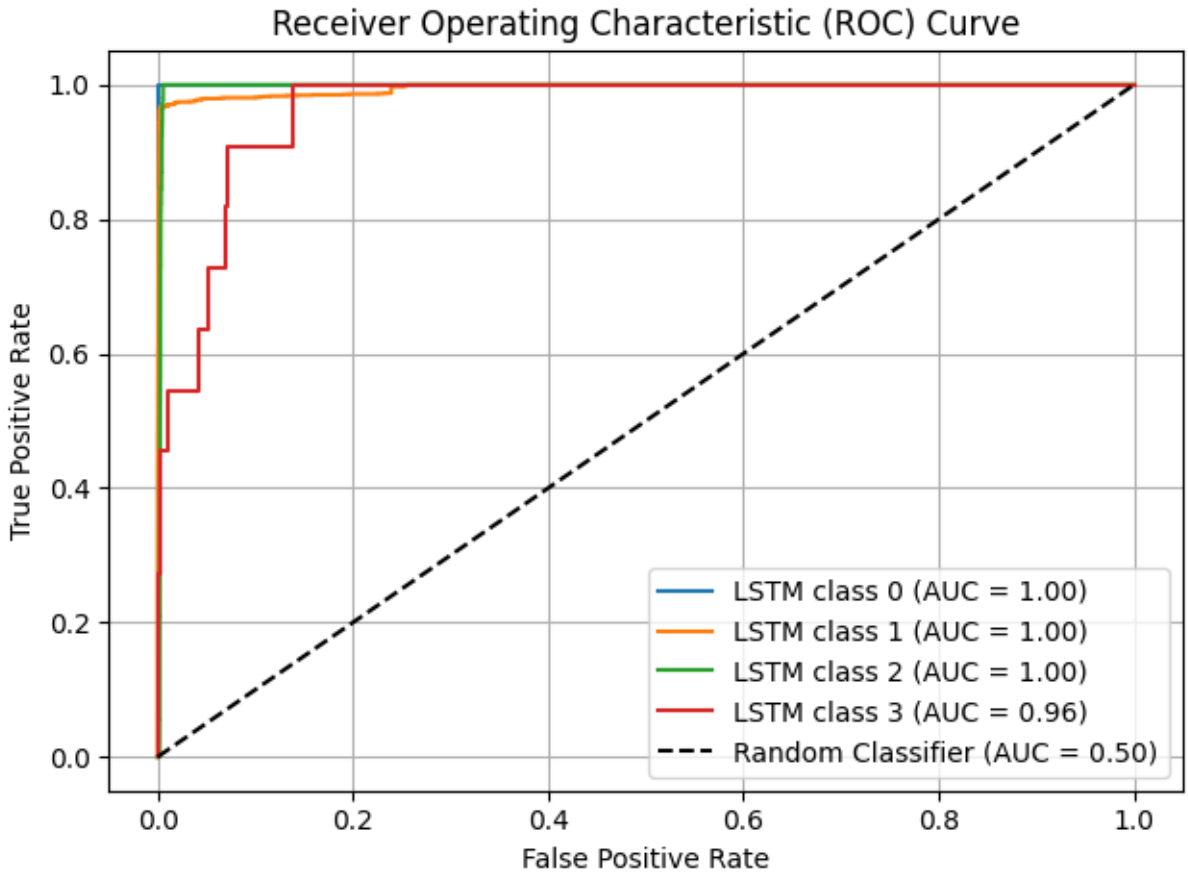
- Tính AUC với One vs Rest

```python
# Tính AUC cho mô hình LSTM
lstm_auc = roc_auc_score(y_test, lstm_pred_proba, multi_class='ovr')

# Tính AUC cho mô hình CNN
cnn_auc = roc_auc_score(y_test, cnn_pred_proba, multi_class='ovr')

# In kết quả AUC
print(f'AUC của mô hình LSTM: {lstm_auc}')
print(f'AUC của mô hình CNN: {cnn_auc}')
```

```
AUC của mô hình LSTM: 0.989156940503221
AUC của mô hình CNN: 0.9904174429191559
```

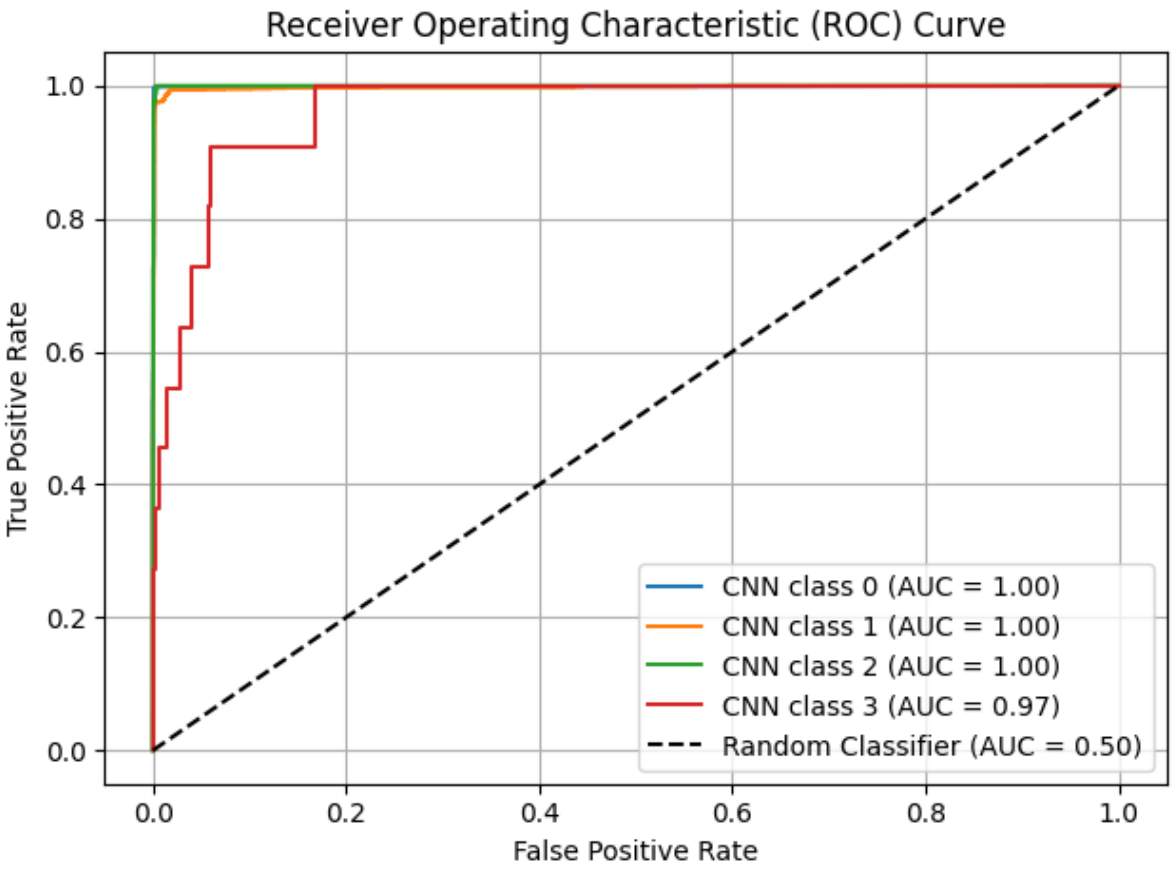
Hình 43: Điểm AUC (One to Rest) của mô hình DL

- Biểu đồ ROC cho phân loại từng nhãn của LSTM



Hình 44: Biểu đồ ROC cho phân loại từng nhãn của LSTM

- Biểu đồ ROC cho phân loại từng nhãn của CNN



Hình 45: Biểu đồ ROC cho phân loại nhãn của CNN

  - Cả hai mô hình đều có AUC rất cao → năng lực phân biệt class tốt.
  - CNN vẫn tốt hơn một chút về mặt phân biệt class khó (class 3).
  - Mức độ khái quát hóa (generalization) của CNN nhỉnh hơn.
- CNN thể hiện tốt hơn LSTM cả về độ chính xác phân loại và khả năng phân biệt qua ROC-AUC, đặc biệt ở các class ít xuất hiện như reset-both và deny. Nhưng nhìn chung cà hai đều khá tốt và ổn định không có sự chênh lệch quá lớn.

CHƯƠNG 3 TRIỂN KHAI HỆ THỐNG

3.1 Giới thiệu

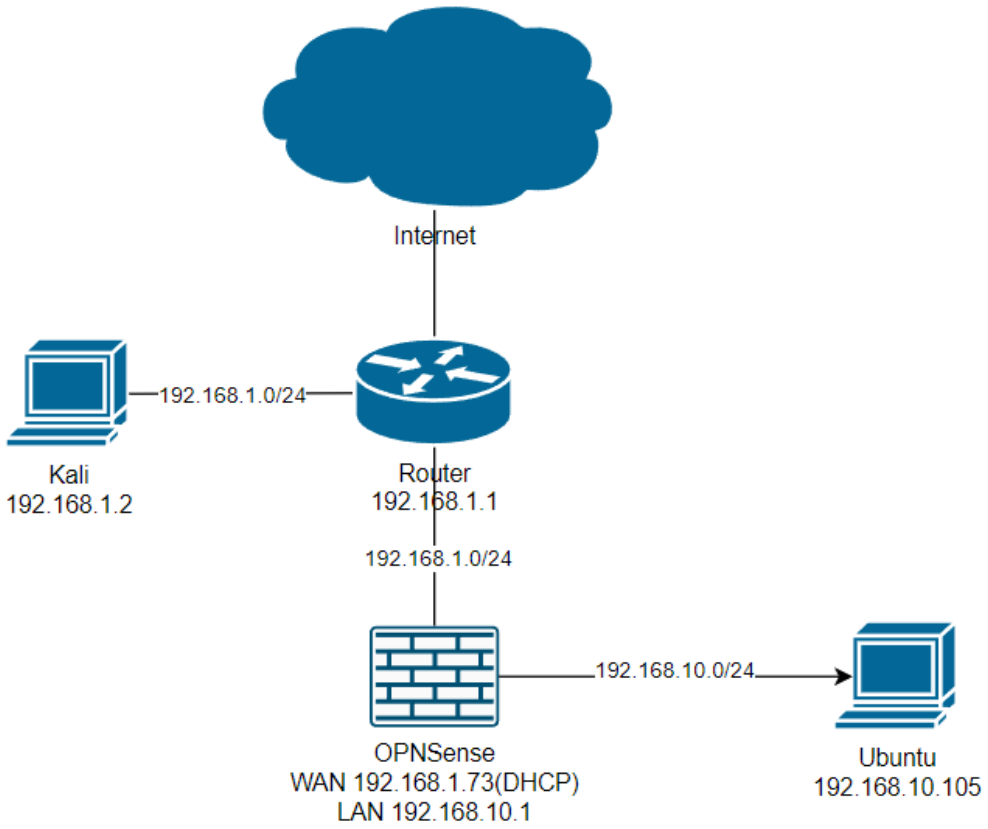
Mục tiêu của chương này là trình bày quá trình thiết kế, triển khai và đánh giá hệ thống phát hiện bất thường trong lưu lượng mạng, ứng dụng mô hình học sâu. Hệ thống được xây dựng trên nền tảng firewall mã nguồn mở OPNsense, với khả năng thu thập log real-time và dự đoán các hành vi bất thường dựa trên tập dữ liệu mạng thực tế.

3.2 Mô hình hệ thống

Hệ thống được triển khai trong môi trường ảo hóa gồm ba máy chính:

Thành phần	Vai trò	IP Address
OPNsense CE	Firewall chính, thu thập dữ liệu log từ state table	WAN: 192.168.1.73 LAN: 192.168.10.1
Ubuntu Server	Phân tích log và chạy mô hình CNN (.h5), host ứng dụng DVWA	192.168.10.105
Kali Linux	Máy tấn công thử nghiệm từ WAN	192.168.1.2

Bảng 6: Bảng thành phần trong hệ thống



Hình 46: Mô hình mạng

- Mô tả luồng xử lý:

- OPNsense ghi nhận các kết nối vào state table, bao gồm đầy đủ thông tin NAT, trạng thái phiên và thống kê lưu lượng.
- Hai cronjob được cấu hình trên OPNsense:
  - Ghi log state table định kỳ vào /var/log/state_table.log.
  - Tự động gửi file log sang Ubuntu qua SSH key (1 phút/lần).
- Ubuntu xử lý log, trích xuất 11 đặc trưng quan trọng, đưa vào mô hình CNN để phân loại.

```
# or /usr/local/etc/cron.d and follow the same format as
# /etc/crontab, see the crontab(5) manual page.
SHELL=/bin/sh
PATH=/etc:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
REQUESTS_CA_BUNDLE=/usr/local/etc/ssl/cert.pem
#minute hour    mday    month   wday    command
1       *       *       *       *       (/usr/local/sbin/configctl -d syslog archive) > /dev/null
2       *       *       *       *       (/usr/local/sbin/expiretable -v -t 3600 sshlockout) > /dev/null
3       *       *       *       *       (/usr/local/sbin/expiretable -v -t 3600 virusprot) > /dev/null
*/4     *       *       *       *       (/usr/local/sbin/ping_hosts.sh) > /dev/null
0       22      *       *       *       (/usr/local/sbin/configctl -d firmware changelog cron) > /dev/null
*       *       *       *       *       (/usr/local/bin/flock -n -E 0 -o /tmp/updaterrd.lock /usr/local/opnsense/scripts/health/updaterrd.php) > /dev/null
1       3       1       *       *       (/usr/local/sbin/configctl -d filter schedule bogons) > /dev/null
*       *       *       *       *       (/usr/local/bin/flock -n -E 0 -o /tmp/filter_update_tables.lock /usr/local/opnsense/scripts/filter/update_tables.py) > /dev/null
*       *       *       *       *       /usr/local/bin/state_dump.sh
*       *       *       *       *       /usr/local/bin/send_log.sh
```

Hình 47: Các cronjob trong crontab

```
root@OPNsense:~ # nano /usr/local/bin/state_dump.sh
  GNU nano 8.4
#!/bin/sh
/sbin/pfctl -s state -vv > /var/log/state_table.log
```

Hình 48: bash script để dump thông tin từ state_table của opnsense

```
root@OPNsense:~ # nano /usr/local/bin/send_log.sh
  GNU nano 8.4
#!/bin/sh
TEMP_LOG="/tmp/state_table.log"
#echo "Starting copy at $(date)" >> /var/log/transfer.log
cp /var/log/state_table.log "$TEMP_LOG"
#echo "Copy done at $(date)" >> /var/log/transfer.log
DEST_TEMP="/home/client/ai/state_table.tmp"
DEST_LOG="/home/client/ai/state_table.log"
#echo "Starting scp at $(date)" >> /var/log/transfer.log
scp -i /root/.ssh/id_ed25519 "$TEMP_LOG" client@192.168.10.105:"$DEST_TEMP"
#echo "Scp done at $(date)" >> /var/log/transfer.log
#echo "Starting mv at $(date)" >> /var/log/transfer.log
ssh -i /root/.ssh/id_ed25519 client@192.168.10.105 "mv -f $DEST_TEMP $DEST_LOG"
#echo "Mv done at $(date)" >> /var/log/transfer.log
rm -f "$TEMP_LOG"
echo "Cleanup done at $(date)" >> /var/log/transfer.log
```

Hình 49: bash script gửi state_table.log tới máy ubuntu chịu trách nhiệm chạy mô hình thông qua ssh có private key

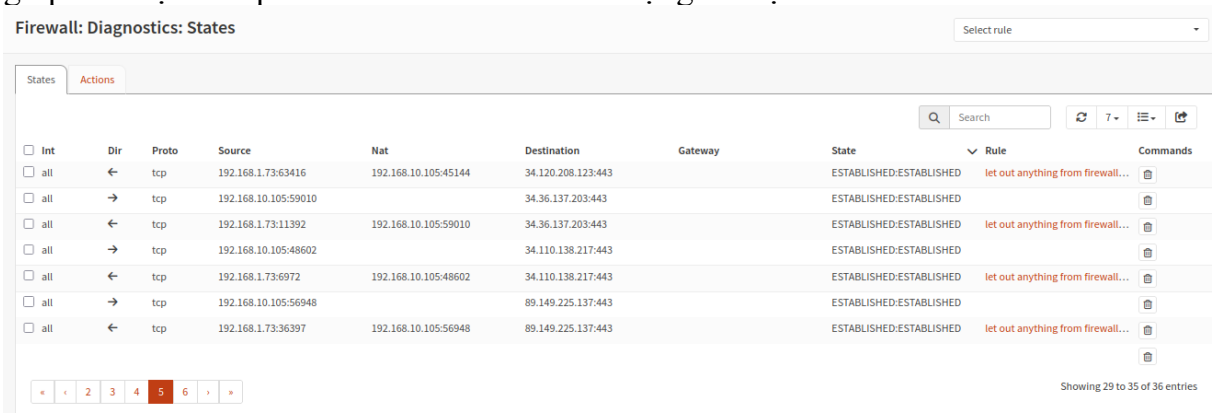
```
   origif: em2
all udp 192.168.1.73:42870 (192.168.10.105:32985) -> 8.8.8.8:53       MULTIPLE:SINGLE
   age 00:00:16, expires in 00:00:14, 2:2 pkts, 126:228 bytes, rule 62, allow-opts
   id: 7e043e6800000000 creatorid: 4305683a route-to: 192.168.1.1@em0
   origif: em0
all udp 192.168.1.73:28780 (192.168.1.73:57763) -> 137.184.40.180:53       MULTIPLE:SINGLE
   age 00:00:11, expires in 00:00:19, 1:1 pkts, 80:236 bytes, rule 62, allow-opts
   id: 7f043e6800000000 creatorid: 4305683a route-to: 192.168.1.1@em0
   origif: em0
all udp 192.168.1.73:54051 (192.168.1.73:26523) -> 212.25.19.23:53       MULTIPLE:SINGLE
   age 00:00:11, expires in 00:00:20, 1:1 pkts, 80:236 bytes, rule 62, allow-opts
   id: 80043e6800000000 creatorid: 4305683a route-to: 192.168.1.1@em0
   origif: em0
all udp 192.168.1.73:32338 (192.168.1.73:53249) -> 45.33.123.43:53       MULTIPLE:SINGLE
   age 00:00:10, expires in 00:00:20, 1:1 pkts, 80:236 bytes, rule 62, allow-opts
   id: 81043e6800000000 creatorid: 4305683a route-to: 192.168.1.1@em0
   origif: em0
all udp 192.168.1.73:54658 (192.168.1.73:19593) -> 46.227.203.69:53       MULTIPLE:SINGLE
   age 00:00:04, expires in 00:00:27, 1:1 pkts, 80:236 bytes, rule 62, allow-opts
   id: 82043e6800000000 creatorid: 4305683a route-to: 192.168.1.1@em0
   origif: em0
client@huynhlongUbuntu:~/ai$
```

Hình 50: Nội dung trong 1 file state_table.log

- **Lý do chọn dump state table**:
  Ban đầu thử nghiệm sử dụng NetFlow v9 để thu thập log. Tuy nhiên, khi triển khai thực tế trên OPNsense Community Edition (CE), nhóm nhận thấy một hạn chế nghiêm trọng:
  - NetFlow v9 không hỗ trợ thông tin NAT trong phiên bản CE. Các trường như nat_src_port, nat_dst_port – rất quan trọng để nhận biết các flow bất thường – không được ghi nhận.
  - Ngoài ra, NetFlow v9 cũng có xu hướng đếm gói tin theo cả chiều vào và ra, dẫn đến trùng lặp hoặc gây sai lệch số liệu trong phân tích.

  Vì vậy, nhóm quyết định sử dụng state table nội tại của OPNsense, vì:
  - Cung cấp đầy đủ các thông tin NAT (source/destination port, NAT mapping),
  - Ghi nhận chính xác số lượng packet, byte truyền nhận, thời gian phiên,
  - Cho phép cập nhật real-time bằng cronjob đơn giản.

  Cách tiếp cận này không cần nâng cấp lên phiên bản thương mại (Business Edition), giúp tiết kiệm chi phí mà vẫn đảm bảo chất lượng dữ liệu đầu vào cho mô hình AI.



Hình 51: State table trên giao diện WEB GUI của OPNSense

3.3 Phát hiện bất thường

3.3.1 Tích hợp mô hình vào hệ thống

Trên Ubuntu Server, nhóm đã huấn luyện sẵn mô hình CNN dựa trên tập dữ liệu gồm 11 đặc trưng đầu vào như: Source Port, Destination Port, NAT Port, Total Bytes, Elapsed Time, Packet Count, v.v.

Quy trình tích hợp:

- Mỗi phút, hệ thống nhận file log từ OPNsense.
- Script Python xử lý file log, trích xuất đặc trưng và đưa vào mô hình .h5.
- Nếu kết quả phân loại cho thấy bất thường → gửi thông báo chi tiết (IP, port, hành vi) đến nhóm giám sát qua.

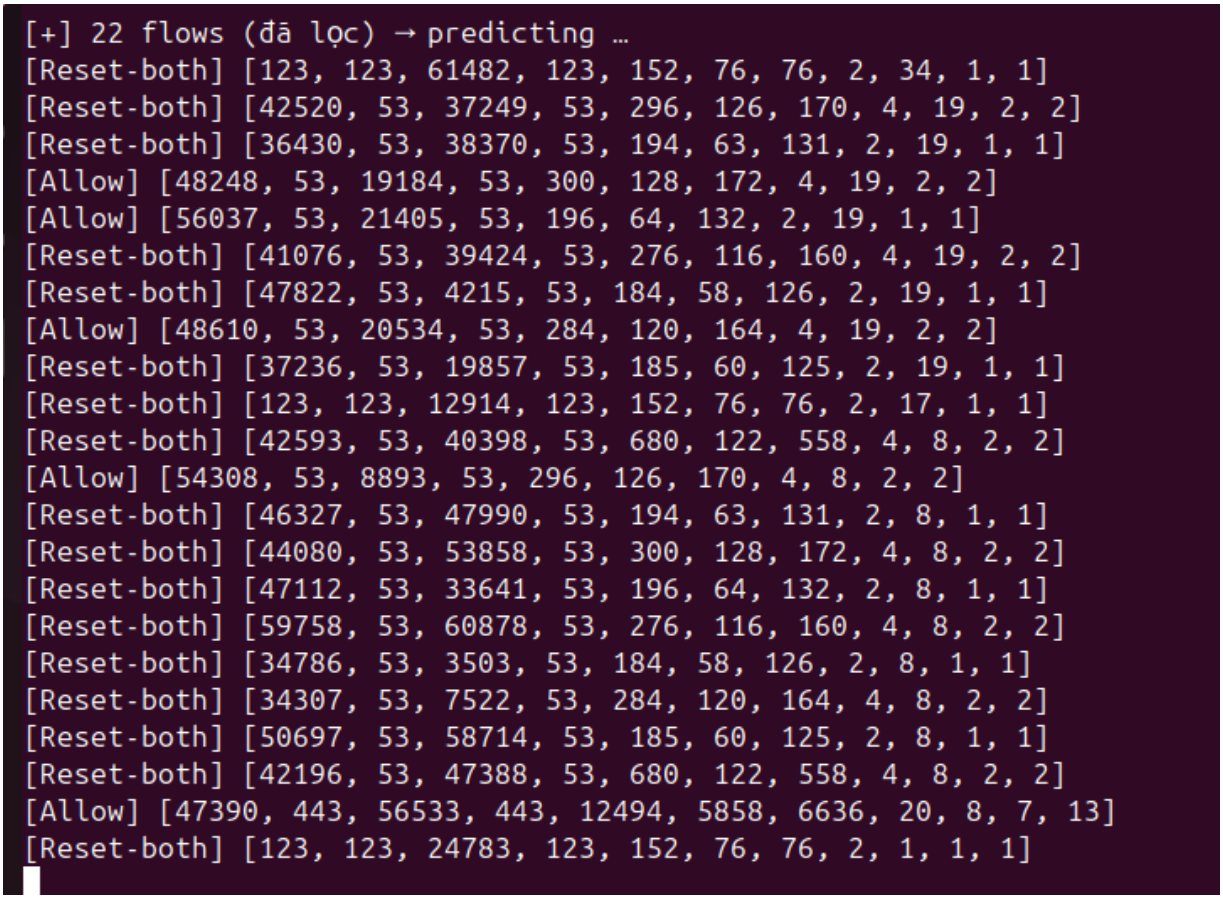
3.3.2 Kết quả thực nghiệm

Trong khuôn khổ đồ án, hệ thống đã được triển khai thành công và thực hiện phân tích dữ liệu log thu thập từ bảng trạng thái kết nối (state table) trên OPNsense. Tuy nhiên, do giới hạn về nguồn dữ liệu, hệ thống hiện tại chưa có khả năng phát hiện các hành vi tấn công mạng một cách đầy đủ. Các kết quả đạt được có thể tóm gọn như sau:

Hạng mục	Kết quả thực tế
Dự đoán nhãn	Mô hình CNN phân loại các kết nối ở mức độ đơn giản giữa hai nhãn chính: allow và reset-both, tương ứng với các flow đã được thiết lập thành công và đã kết thúc (FIN hoặc RESET).
Nguồn dữ liệu	Dữ liệu được trích xuất từ **state table**, chỉ ghi nhận các kết nối **đã được firewall cho phép** (trạng thái ESTABLISHED, MULTIPLE, SINGLE). Không ghi nhận các packet bị chặn (drop).
Tự động hóa	Việc dump log và phân tích diễn ra định kỳ mỗi phút nhờ cronjob, đảm bảo luồng xử lý liên tục.

Bảng 7: Bảng kết quả thực nghiệm

Ghi chú: Vì chỉ sử dụng state table nên hệ thống hiện tại không có khả năng phát hiện các hành vi tấn công như brute-force, scan, hoặc xâm nhập, vốn thường bị firewall chặn và không tạo ra state hợp lệ.



Hình 52: Kết quả chạy mô hình

3.3.3 Hạn chế

Một số khó khăn:

- Chưa dùng filterlog: Hệ thống chỉ sử dụng state table, không thu thập dữ liệu từ filterlog, dẫn đến thiếu thông tin về lưu lượng bị chặn (block/drop), hạn chế khả năng phát hiện mối đe dọa.
- Độ trễ đồng bộ: Do phụ thuộc cronjob 1 phút → chưa đạt mức real-time 100%.
- Hạn chế của firewall cộng đồng: OPNsense CE thiếu các tính năng nâng cao như deep packet inspection và hỗ trợ chuyên nghiệp, gây khó khăn khi mở rộng hệ thống.

3.3.4 Đề xuất cải thiện

- Tích hợp filterlog: Thu thập dữ liệu lưu lượng bị chặn để cải thiện khả năng phát hiện mối đe dọa.
- Sử dụng firewall thương mại: Các phiên bản như pfSense Plus hoặc Sophos XGS Firewall cung cấp tính năng nâng cao và hỗ trợ chuyên nghiệp.
- Nâng cấp phần cứng: Tăng hiệu suất xử lý log và đồng bộ thời gian thực.
- Tối ưu hóa đồng bộ: Phát triển công cụ giảm độ trễ trong xử lý dữ liệu.

KẾT LUẬN

Trong bối cảnh chuyển đổi số diễn ra mạnh mẽ và hệ thống thông tin ngày càng trở nên phức tạp, việc đảm bảo an toàn cho hệ thống mạng và dữ liệu doanh nghiệp không còn là lựa chọn, mà đã trở thành một yêu cầu tất yếu. Một trong những nguồn dữ liệu quan trọng giúp phát hiện sớm các nguy cơ an ninh – nhưng cũng là một thách thức lớn về mặt phân tích – chính là logfile. Với khối lượng dữ liệu khổng lồ, tính chất không đồng nhất và liên tục sinh ra theo thời gian thực, logfile vừa là một kho báu, vừa là một mê cung khó giải mã nếu không có công cụ đủ mạnh.

Đề tài "Hệ thống giám sát và phân tích logfile server dựa trên Học máy" được triển khai nhằm khai thác tiềm năng của các phương pháp Học máy hiện đại trong việc phân tích và phát hiện các bất thường trong log dữ liệu hệ thống. Qua quá trình nghiên cứu và thực nghiệm, hệ thống không chỉ chứng minh được khả năng tự động hóa cao mà còn cho thấy hiệu quả vượt trội so với các phương pháp truyền thống.

Bằng cách áp dụng các kỹ thuật tiền xử lý như loại bỏ nhiễu, chuẩn hóa và biểu diễn đặc trưng (TF-IDF, Word2Vec...), log dữ liệu được chuyển đổi từ dạng thô sang dạng có thể hiểu được bởi máy học. Các mô hình như Random Forest, XGBoost, SVM, và đặc biệt là LSTM hay CNN đã được thử nghiệm và cho thấy khả năng học tập mạnh mẽ, phát hiện chính xác các hành vi bất thường, từ đó hỗ trợ cảnh báo sớm những rủi ro tiềm ẩn trong hệ thống.

Không chỉ dừng lại ở việc phát hiện, hệ thống còn đóng vai trò như một "người lính gác thầm lặng", âm thầm theo dõi nhịp đập của hệ thống và cảnh báo khi có những xáo trộn bất thường. Chính nhờ sự kết hợp giữa trí tuệ nhân tạo và kỹ thuật giám sát, hệ thống trở thành một công cụ đắc lực giúp nâng cao năng lực phòng thủ chủ động trong môi trường mạng đầy biến động ngày nay.

Nhìn lại toàn bộ quá trình, có thể khẳng định rằng: việc áp dụng Học máy vào giám sát logfile không chỉ là một xu hướng công nghệ, mà còn là bước tiến tất yếu trong hành trình hướng tới an ninh mạng thông minh và tự thích ứng. Dù còn nhiều thách thức như tối ưu hiệu năng, xử lý thời gian thực, hay bảo mật chính dữ liệu đầu vào, nhưng kết quả đạt được đã mở ra một hướng đi mới, tiềm năng và đầy hứa hẹn cho tương lai.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] I. Novogroder, "lakeLS," Data Preprocessing in Machine Learning: Steps & Best Practices, 30 April 2024. [Online]. Available: https://lakefs.io/blog/data-preprocessing-in-machine-learning/. [Accessed 30 May 2025].

[2] L. Mitton, "splunk," Log Data 101: What It Is & Why It Matters, 31 August 2023. [Online]. Available: https://www.splunk.com/en_us/blog/learn/log-data.html. [Accessed 30 May 2025].

[3] S. Logic, "sumo logic," What is a log file?, [Online]. Available: https://www.sumologic.com/glossary/log-file. [Accessed 30 May 2025].

[4] T. Sigma, "Team Sigma," Quiet Noisy Data: How Data Smoothing Provides Sharper Analysis, [Online]. Available: https://www.sigmacomputing.com/blog/data-smoothing-cleaning. [Accessed 30 May 2025].

[5] H. Tekgul, "arize," Tokenization and Tokenizers for Machine Learning, 02 February 2023. [Online]. Available: https://arize.com/blog-course/tokenization/. [Accessed 30 May 2025].

[6] CodeSignal, "codesignal," Removing Stop Words and Stemming in Text, [Online]. Available: https://codesignal.com/learn/courses/collecting-and-preparing-textual-data-for-classification/lessons/removing-stop-words-and-stemming-in-text-preprocessing. [Accessed 30 May 2025].

[7] Geeksforgeeks, "geeksforgeeks," Removing Stop Words and Stemming in Text, 07 February 2025. [Online]. Available: https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/. [Accessed 30 May 2025].

[8] NebulaGraph, "nebula-graph," What are graph embeddings ?, 21 February 2024. [Online]. Available: https://www.nebula-graph.io/posts/graph-embeddings. [Accessed 30 May 2025].

[9] Sruthi, "analyticsvidhya," Random Forest Algorithm in Machine Learning, 01 May 2025. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/. [Accessed 30 May 2025].

[10] Geeksforgeeks, "geeksforgeeks," Random Forest Algorithm in Machine Learning, 22 May 2025. [Online]. Available: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/. [Accessed 30 May 2025].

[11] A. Tyagi, "analyticsvidhya," What is XGBoost Algorithm?, 23 April 2025. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/. [Accessed 30 May 2025].

[12] Geeksforgeeks, "geeksforgeeks," XGBoost, 23 May 2025. [Online]. Available: https://www.geeksforgeeks.org/xgboost/. [Accessed 30 May 2025].

[13] Geeksforgeeks, "geeksforgeeks," Support Vector Machine (SVM) Algorithm, 28 May 2025. [Online]. Available: https://www.geeksforgeeks.org/support-vector-machine-algorithm/. [Accessed 30 May 2025].