|  |
| --- |
| BAN CƠ YẾU CHÍNH PHỦ  **HỌC VIỆN KỸ THUẬT MẬT MÃ**  ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯  Logo HvKTMM |
| ĐỒ ÁN TỐT NGHIỆP  **PHÁT HIỆN MÃ ĐỘC ANDROID DỰA TRÊN MÔ HÌNH HỌC SÂU KẾT HỢP** |
| Ngành: Công nghệ thông tin  Mã số: 7.48.02.01  *Sinh viên thực hiện*:  **Nguyễn Ngọc Minh**  MSSV: CT050134    *Người hướng dẫn*:  **ThS. Nguyễn Đức Thắng**  Công ty TNHH NEC Việt Nam |
| **Hà Nội, 2025** |

# MỤC LỤC

[MỤC LỤC i](#_Toc200402257)

[DANH MỤC KÝ TỰ VIẾT TẮT iii](#_Toc200402258)

[DANH MỤC BẢNG iv](#_Toc200402259)

[DANH MỤC HÌNH VẼ v](#_Toc200402260)

[LỜI CẢM ƠN vi](#_Toc200402261)

[LỜI CAM ĐOAN vii](#_Toc200402262)

[LỜI NÓI ĐẦU 1](#_Toc200402263)

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 2](#_Toc200402264)

[1.1. Hiện trạng 2](#_Toc200402265)

[1.2. Một số khái niệm 4](#_Toc200402266)

[1.2.1. Hệ điều hành Android 4](#_Toc200402267)

[1.2.2. Mã độc Android 6](#_Toc200402268)

[1.3. Các phương pháp phát hiện mã độc 11](#_Toc200402269)

[1.4. Kết luận chương 12](#_Toc200402270)

[CHƯƠNG 2: TÌM HIỂU MÔ HÌNH HỌC SÂU KẾT HỢP 13](#_Toc200402271)

[2.1. Kiến thức nền tảng 13](#_Toc200402272)

[2.1.1. Tổng quan về học máy 13](#_Toc200402273)

[2.1.2. Tổng quan về học sâu 16](#_Toc200402274)

[2.1.3. Mạng học sâu nhiều lớp MLP 18](#_Toc200402275)

[2.2. Thuật toán di truyền 20](#_Toc200402276)

[2.2.1. Nguyên lý hoạt động 20](#_Toc200402277)

[2.2.2. Ưu điểm và hạn chế 23](#_Toc200402278)

[2.3. Mô hình học sâu kết hợp thuật toán di truyền 24](#_Toc200402279)

[2.3.1. Mục tiêu 24](#_Toc200402280)

[2.3.2. Mô hình GA-MLP 24](#_Toc200402281)

[2.4. Kết luận chương 26](#_Toc200402282)

[CHƯƠNG 3: THỰC NGHIỆM 27](#_Toc200402283)

[3.1. Sơ đồ tổng thể phân loại mã độc 27](#_Toc200402284)

[3.2. Bộ dữ liệu và đặc trưng 27](#_Toc200402285)

[3.2.1. Bộ dữ liệu 27](#_Toc200402286)

[3.2.2. Đặc trưng 27](#_Toc200402287)

[3.2.3. Phương pháp trích xuất đặc trưng 28](#_Toc200402288)

[3.3. Kịch bản thực nghiệm 31](#_Toc200402289)

[3.3.1. Thiết lập môi trường 31](#_Toc200402290)

[3.3.2. Các tiêu chí đánh giá 31](#_Toc200402291)

[3.3.3. Các kịch bản thử nghiệm 33](#_Toc200402292)

[3.4. Kết quả và đánh giá 38](#_Toc200402293)

[3.4.1. Kết quả 38](#_Toc200402294)

[3.4.2. Đánh giá 50](#_Toc200402295)

[3.5. Chương trình thực nghiệm 51](#_Toc200402296)

[3.6. Kết luận chương 54](#_Toc200402297)

[KẾT LUẬN 55](#_Toc200402298)

[TÀI LIỆU THAM KHẢO 56](#_Toc200402299)

# DANH MỤC KÝ TỰ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **Từ viết tắt** | **Nghĩa tiếng Anh** | **Nghĩa tiếng Việt** |
| AI | Artificial Intelligence | Trí tuệ nhân tạo |
| DL | Deep Learning | Học sâu |
| LSTM | Long Short Term Memory | Mạng bộ nhớ dài hạn–ngắn hạn |
| ML | Machine Learning | Học máy |
| ACO | Ant Colony Algorithm | Thuật toán đàn kiến |
| GA | Genetic Algorithm | Thuật toán di truyền |
| RNN | Recurrent Neural Network | Mạng nơ–ron hồi quy |
| KNN | K-nearest neighbors | Giải thuật k hàng xóm gần nhất |
| SL | Supervised Learning | Học có giám sát |
| RF | Random Forest | Rừng ngẫu nhiên |
| MLP | Multi-layer Perceptron | Perceptron nhiều lớp |
| HAL | Hardware Abstraction Layer | Lớp trừu tượng phần cứng |
| ReLU | Rectified Linear Unit | Đơn vị tuyến tính chỉnh lưu |
| CFG | Control Flow Graph | Biểu đồ luồng điều khiển |
| UL | Unsupervised Learning | Học không giám sát |
| PUA | Potentially Unwanted Application | Ứng dụng không mong muốn |

# DANH MỤC BẢNG

[Bảng 3.1: Số lượng mẫu các nhóm phân loại 27](#_Toc200402301)

[Bảng 3.2: Cấu trúc ma trận nhầm lần 33](#_Toc200402302)

[Bảng 3.3: Báo cáo phân loại mô hình GA-MLP 39](#_Toc200402303)

[Bảng 3.4: Báo cáo phân loại của mô hình ACO-MLP 40](#_Toc200402304)

[Bảng 3.5: Báo cáo phân loại của mô hình MLP 41](#_Toc200402305)

[Bảng 3.6: Báo cáo phân loại của các mô hình học máy 42](#_Toc200402306)

[Bảng 3.7: Báo cáo phân loại mô hình GA-MLP 45](#_Toc200402307)

[Bảng 3.8: Báo cáo phân loại mô hình ACO-MLP 46](#_Toc200402308)

[Bảng 3.9: Báo cáo phân loại mô hình MLP 47](#_Toc200402309)

[Bảng 3.10: Báo cáo phân loại của mô hình học máy 49](#_Toc200402310)

# DANH MỤC HÌNH VẼ

[Hình 1.1: Thống kê lượng mã độc và PUA qua các năm 3](#_Toc200402313)

[Hình 1.2: Kiến trúc hệ điều hành Android 5](#_Toc200402314)

[Hình 1.3: Sơ đồ tổng quan về phương pháp phát hiện dựa trên chữ ký 11](#_Toc200402315)

[Hình 2.1: Ứng dụng của Học máy trong các lĩnh vực khác nhau 14](#_Toc200402316)

[Hình 2.2 Mạng nơ-ron nhiều lớp MLP 19](#_Toc200402317)

[Hình 2.3: Sơ đồ tổng quan thuật toán di truyền 22](#_Toc200402318)

[Hình 2.4: Kiến trúc mô hình GA-MLP 25](#_Toc200402319)

[Hình 3.1: Sơ đồ tổng quát quy trình thực nghiệm 27](#_Toc200402320)

[Hình 3.2: Công cụ AndroPyTool 29](#_Toc200402321)

[Hình 3.3: Biểu đồ tiến hóa thích nghi 39](#_Toc200402322)

[Hình 3.4: Ma trận nhầm lẫn mô hình GA-MLP 40](#_Toc200402323)

[Hình 3.5: Ma trận nhầm lẫn của mô hình ACO-MLP 41](#_Toc200402324)

[Hình 3.6: Ma trận nhầm lẫn của mô hình MLP 42](#_Toc200402325)

[Hình 3.7: Ma trận nhầm lẫn mô hình SVM 43](#_Toc200402326)

[Hình 3.8: Ma trận nhầm lẫn mô hình Random Forest 43](#_Toc200402327)

[Hình 3.9: Biểu đồ tiến hóa thích nghi 45](#_Toc200402328)

[Hình 3.10: Ma trận nhầm lẫn mô hình GA-MLP 46](#_Toc200402329)

[Hình 3.11: Ma trận nhầm lẫn của mô hình ACO-MLP 47](#_Toc200402330)

[Hình 3.12: Ma trận nhầm lẫn của mô hình MLP 48](#_Toc200402331)

[Hình 3.13: Ma trận nhầm lẫn mô hình SVM 49](#_Toc200402332)

[Hình 3.14: Ma trận nhầm lẫn mô hình Random Forest 50](#_Toc200402333)

[Hình 3.15: Màn hình ứng dụng lúc khởi chạy 52](#_Toc200402334)

[Hình 3.16: Ứng dụng khi đang phân tích tệp apk 52](#_Toc200402335)

[Hình 3.17: Ứng dụng hiển thị kết quả 53](#_Toc200402336)

[Hình 3.18: Ứng dụng hiển thị kết quả (tiếp) 53](#_Toc200402337)

# LỜI CẢM ƠN

Trong quá trình thực hiện đồ án tốt nghiệp “Phát hiện mã độc Android dựa trên mô hình học sâu kết hợp”, em đã nhận được sự giúp đỡ tận tình của cán bộ hướng dẫn là ThS. Nguyễn Đức Thắng – Công ty TNHH NEC Việt Nam. Em xin chân thành cảm ơn Thầy đã hết lòng giúp đỡ, hướng dẫn, tạo điều kiện để em có thể hoàn thành đề tài một cách tốt nhất.

Mặc dù đã có nhiều cố gắng, nỗ lực; bài báo cáo vẫn có những thiếu sót do thời gian thực hiện và kiến thức em còn hạn chế. Em rất mong nhận được sự góp ý của quý Thầy Cô để kiến thức của chúng em được hoàn thiện hơn.

Em xin chân thành cảm ơn!

*Hà Nội, Ngày … tháng … năm 2025.*

|  |
| --- |
| **SINH VIÊN THỰC HIỆN ĐỒ ÁN** |

# LỜI CAM ĐOAN

Em xin cam đoan bản đồ án này do em tự nghiên cứu dưới sự hướng dẫn của thầy ThS. Nguyễn Đức Thắng.

Để hoàn thành đồ án này, em chỉ sử dụng những tài liệu đã ghi trong mục tài liệu tham khảo, ngoài ra không sử dụng bất cứ tài liệu nào khác mà không được ghi.

Nếu sai, em xin chịu mọi hình thức kỷ luật theo quy định của Học viện.

*Hà Nội, Ngày … tháng … năm 2025.*

|  |
| --- |
| **Sinh viên thực hiện**  *(Ký và ghi rõ họ tên)* |

# LỜI NÓI ĐẦU

Trong những năm gần đây, sự bùng nổ của các thiết bị di động đã làm thay đổi mạnh mẽ cách con người tiếp cận và sử dụng công nghệ.Trong đó, Android với đặc điểm là một hệ điều hành mã nguồn mở, dễ tùy biến, kho ứng dụng phong phú và khả năng tương thích với nhiều thiết bị đã nhanh chóng chiếm lĩnh phần lớn thị phần trên toàn cầu. Tuy nhiên, chính sự phổ biến và mở rộng nhanh chóng của nền tảng này cũng đã biến nó trở thành mục tiêu hàng đầu của các cuộc tấn công mạng, với sự gia tăng đáng kể về số lượng và mức độ tinh vi của các loại mã độc.

Các cuộc tấn công bằng mã độc trên thiết bị Android không chỉ đe dọa đến quyền riêng tư cá nhân mà còn gây ảnh hưởng nghiêm trọng đến an ninh thông tin, dữ liệu tài chính và thậm chí là an ninh quốc gia. Trong bối cảnh đó, việc phát hiện và phòng chống mã độc Android đã và đang trở thành một thách thức lớn đối với các nhà nghiên cứu, chuyên gia an ninh mạng cũng như các tổ chức bảo mật trên toàn thế giới.

Hiện nay có nhiều phương pháp đã được áp dụng để phát hiện mã độc Android, các phương pháp truyền thống như dựa trên chữ ký hay các mẫu hành vi sẵn có bộc lộ những hạn chế, đặc biệt là trước sự thay đổi nhanh chóng và phức tạp của các kỹ thuật che giấu và biến đổi mã độc. Trong khi đó, các mô hình học sâu đã được chứng minh với hiệu quả vượt trội trong việc phát hiện các mẫu phức tạp và không dễ nhận biết trong dữ liệu. Tuy vậy, các mô hình học sâu thường đòi hỏi khối lượng dữ liệu lớn, cấu trúc mạng phù hợp.

Đề tài “Phát hiện mã độc Android bằng mô hình học sâu kết hợp” được thực hiện nhằm khai thác sức mạnh của mạng học sâu nhiều lớp (MLP) kết hợp với thuật toán di truyền GA trong việc tối ưu hóa quá trình phát hiện mã độc. Bằng cách kết hợp này, mô hình đề xuất không chỉ nâng cao độ chính xác mà còn giúp cải thiện hiệu suất xử lý và khả năng khái quát hóa.

Nội dung đề tài được chia thành ba chương chính như sau:

* **Chương 1. Cơ sở lý thuyết**

Trình bày cơ sở lý thuyết và các phương pháp phát hiện mã độc hiện nay.

* **Chương 2. Mô hình học sâu kết hợp thuật toán di truyền**

Mô tả mô hình học sâu kết hợp GA và cách áp dụng vào bài toán.

* **Chương 3. Thực nghiệm**

Trình bày quá trình thực nghiệm, đánh giá kết quả và chương trình thực nghiệm.

# CƠ SỞ LÝ THUYẾT

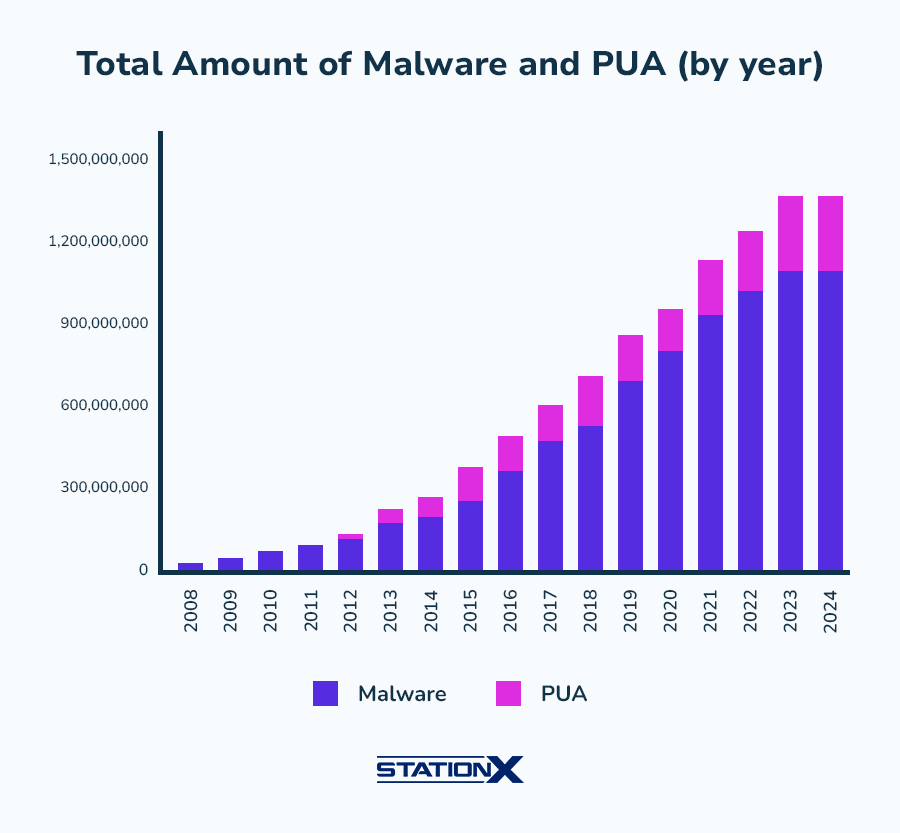
## Hiện trạng

Xã hội ngày càng phụ thuộc vào máy tính để thực hiện các công việc hàng ngày, dù là ở nơi làm việc hay trong cuộc sống cá nhân. Trong khi hầu hết máy tính được sản xuất đi kèm với phần mềm từ nhà sản xuất, chúng thường được thiết kế để cho phép cài đặt các chương trình từ bên thứ ba để mở rộng thêm các tính năng sẵn có. Phát triển phần mềm bên thứ ba đã gia tăng nhanh chóng trong các thập kỷ gần đây và đã tạo ra các ứng dụng bao phủ hầu hết mọi mặt trong cuộc sống của chúng ta bao gồm sức khỏe, tài chính, gia đình, quản lý kinh doanh.

Việc sử dụng các thiết bị tin học đã tăng đáng kể cùng với sự phát triển của thiết bị di động. Cụ thể hơn, sự trỗi dậy của điện thoại thông minh đã khiến chúng trở thành một vật dụng bỏ túi không thể thiếu của hầu hết mọi người. Điện thoại thông minh cung cấp đa dạng các dịch vụ vượt xa khả năng chỉ liên lạc trên những thiết bị thế hệ trước. Đặc biệt, nhiều cảm biến phần cứng hiện được bao gồm trong thiết bị và khả năng cài đặt ứng dụng bên thứ ba đã mở rộng chức năng của chúng thành một trợ lý cá nhân có thể ghi chú, quản lý tin nhắn và danh bạ của người dùng, theo dõi vị trí và lộ trình di chuyển, cung cấp thông tin sức khỏe và một số các dịch vụ khác. Có nhiều nhà cung cấp thiết bị di động và một số hệ điều hành nhắm đến các thiết bị như vậy. Tính đến quý I năm 2025, Android giữ vị thế là hệ điều hành di động hàng đầu thế giới với thị phần khoảng 71,88%.

Việc tạo ra và áp dụng các ứng dụng bên thứ ba đã giúp nhiều tác vụ trở nên dễ dàng hơn nhưng đã mang đến một phương thức tấn công mới cho những kẻ muốn phá hoại các quy trình thông thường của họ vì mục đích xấu, chẳng hạn như để kiếm lợi nhuận hoặc gây hại cho người dùng thiết bị. Đặc biệt, do có rất nhiều chức năng trong các thiết bị di động và được sử dụng trong cuộc sống hàng ngày, chúng là kho tàng thông tin cá nhân nhạy cảm và là bộ điều khiển của vô số các hành động nhạy cảm có thể được thực hiện thay cho người dùng. Các dịch vụ được cung cấp bởi ứng dụng bên thứ ba có thể gây ra thiệt hại lớn cho người dùng nếu chúng bị kẻ xấu phá hoại. Ví dụ, ứng dụng ngân hàng có thể bị khai thác để đánh cắp tiền từ người dùng hoặc chiếm quyền kiểm soát thiết bị của người dùng.

Thực tế, số lượng mã độc được phát hiện hàng năm đã gia tăng đáng kể, phản ánh xu hướng tấn công mạng ngày càng phổ biến và nghiêm trọng. Hình sau đây minh họa lượng gia tăng đáng kể của mã độc và ứng dụng không mong muốn qua những năm gần đây.



Hình 1.1: Thống kê lượng mã độc và PUA qua các năm

Do đó, việc kiểm tra và thẩm định phần mềm bên thứ ba trở nên cần thiết để đảm bảo rằng chúng không chứa lỗ hổng có thể bị khai thác để cho phép các cuộc tấn công như vậy. Đặc biệt, các chợ ứng dụng thúc đẩy và tạo điều kiện thuận lợi cho việc cài đặt phần mềm bên thứ ba đang cố gắng loại bỏ phần mềm độc hại khỏi các dịch vụ của họ để duy trì tính bảo mật cho người dùng. Cửa hàng Google Play, nền tảng phân phối chính của các ứng dụng trên hệ điều hành Android, tất cả phải được kiểm tra bảo mật nghiêm ngặt trước khi xuất hiện trên Google Play. Google Play Protect quét 200 tỷ ứng dụng mỗi ngày để đảm bảo mọi thứ đều chính xác. Công cụ này cho phép Google phản ứng nhanh chóng với các mối đe dọa và hạn chế số lượng thiết bị và người dùng bị ảnh hưởng bởi chúng.

Trong bối cảnh đó, các phương pháp phát hiện mã độc truyền thống, chủ yếu dựa vào chữ ký hoặc luật định, đang dần bộc lộ hạn chế, đặc biệt là về khả năng phát hiện sớm và khả năng thích ứng với các mẫu tấn công mới. Đây là lúc các công nghệ mới, đặc biệt là trí tuệ nhân tạo, học máy và học sâu được nghiên cứu và ứng dụng nhằm nâng cao hiệu quả phát hiện mã độc.

Mặc dù các mô hình học sâu cho thấy nhiều hứa hẹn trong việc nhận diện các hành vi bất thường của ứng dụng, song việc tối ưu hóa mô hình để đạt hiệu quả cao vẫn là một thách thức. Do đó việc kết hợp học sâu với các phương pháp tối ưu như thuật toán di truyền GA đang nổi lên như một giải pháp tiềm năng để cải thiện hiệu quả phân loại và phát hiện mã độc trên Andorid.

## Một số khái niệm

### Hệ điều hành Android

Hệ điều hành Android là một hệ điều hành nguồn mở, được xây dựng dựa trên Linux được tạo ra dành cho nhiều thiết bị và kiểu dáng.

* Nhân Linux

Nền tảng của Android là nhân Linux. Ví dụ: Android Runtime (ART) dựa vào nhân hệ điều hành Linux để thực hiện các chức năng cơ bản như phân luồng và quản lý bộ nhớ cấp thấp. Việc sử dụng nhân hệ điều hành Linux cho phép Android tận dụng các tính năng bảo mật chính và cho phép nhà sản xuất thiết bị phát triển trình điều khiển phần cứng cho một hệ điều hành đã được biết tới.

* Lớp trừu tượng phần cứng (HAL)

Tầng trừu tượng phần cứng (HAL) cung cấp các giao diện tiêu chuẩn để cho phép các tính năng phần cứng của thiết bị tiếp cận khung API Java cấp cao hơn. HAL bao gồm nhiều mô-đun thư viện, mỗi mô-đun trong số đó triển khai một giao diện cho một loại thành phần phần cứng cụ thể, chẳng hạn như mô-đun camera hoặc Bluetooth. Khi khung API thực hiện lệnh gọi để truy cập vào phần cứng của thiết bị, hệ thống Android sẽ tải mô-đun thư viện cho thành phần phần cứng đó.

* Môi trường thời gian chạy Android

Đối với các thiết bị chạt Android phiên bản 5.0 (API cấp 21) trở lên, mỗi ứng dụng chạy theo quy trình riêng với phiên bản Android Runtime (ART) riêng. ART được viết để chạy nhiều máy ảo trên các thiết bị có bộ nhớ thấp bằng cách thực thi các tệp dịnh dạng có thể thực thi Dalvik (DEX), một định dạng mã byte được thiết kế riêng cho Android được tối ưu hóa để đạt được mức sử dụng bộ nhớ tối thiểu. Các công cụ tạo bản dựng, chẳng hạn như d8, biên dịch nguồn Java thành mã byte DEX, có thể chạy trên nền tảng Android.

* Thư viện C/C++ gốc

Nhiều thành phần và dịch vụ cốt lõi của hệ thống Android (chẳng hạn như ART và HAL) được xây dựng từ mã gốc cần có thư viện gốc viết bằng C và C++. Nền tảng Android cung cấp các khung API Java để hiển thị chức năng của một số thư viện gốc trong số này cho ứng dụng. Ví dụ: ta có thể truy cập vào OpenGL ES thông qua Java OpenGL API của khung Android để thêm tính năng hỗ trợ vẽ và chỉnh sửa đồ họa 2D và 3D trong ứng dụng.



Hình 1.2: Kiến trúc hệ điều hành Android

* Ứng dụng hệ thống

Android đi kèm với một tập hợp các ứng dụng cốt lỗi dành cho email, nhắn tin SMS, lịch, duyệt Internet, danh bạ và nhiều tính năng khác. Trong số các ứng dụng mà người dùng chọn cài đặt, các ứng dụng đi kèm nền tảng sẽ không có trạng thái đặc biệt nào. Vì vậy, ứng dụng bên thứ ba có thể trở thành trình duyệt web, tin nhắn SMS hoặc thậm chí là bàn phím mặc định của người dùng. Có một số trường hợp ngoại lệ, chẳng hạn như ứng dụng Cài đặt của hệ thống.

Ứng dụng hệ thống vừa có chức năng là ứng dụng cho người dùng, vừa cung cấp những chức năng chính mà nhà phát triển có thể truy cập trên ứng dụng của họ. Ví dụ: nếu muốn ứng dụng của mình gửi tin nhắn SMS, ta không cần phải tự xây dựng chức năng đó. Thay vào đó, có thể gọi bất kỳ ứng dụng SMS nào đã được cài đặt để gửi tin nhắn tới người nhận được chỉ định.

* Khung API Java

Ta có thể sử dụng toàn bộ các tính năng của hệ điều hành Android thông qua các API được viết bằng ngôn ngữ Java. Các API này tạo thành những thành phần cần thiết để tạo ứng dụng Android bằng cách đơn giản hóa việc sử dụng lại các thành phần và dịch vụ cốt lõi của hệ thống, theo mô-đun, bao gồm những thành phần sau:

* + Một hệ thống chế độ xem phong phú và dễ mở rộng mà bạn có thể sử dụng để xây dựng giao diện người dùng của ứng dụng, bao gồm danh sách, lưới, hộp văn bản, nút và thậm chí là một trình duyệt web có thể nhúng.
  + Trình quản lý tài nguyên, cấp quyền truy cập vào các tài nguyên không phải mã, chẳng hạn như các chuỗi đã được bản địa hóa, tệp đồ họa và tệp bố cục.
  + Trình quản lý thông báo cho phép tất cả ứng dụng hiển thị cảnh báo tùy chỉnh trên thanh trạng thái.
  + Trình quản lý hoạt động quản lý vòng đời của các ứng dụng và cung cấp một ngăn xếp lui điều hướng thông thường.
  + Trình cung cấp nội dung cho phép ứng dụng truy cập vào dữ liệu từ các ứng dụng khác, chẳng hạn như ứng dụng Danh bạ hoặc chia sẻ dữ liệu của riêng họ.

### Mã độc Android

* Các loại mã độc

Mã độc nhắm vào thiết bị Android chủ yếu hướng đến việc khai thác các dịch vụ được cung cấp bởi phần cứng thiết bị, mặc dù một số mã độc (như mã độc tống tiền) tương tự như những dạng được thấy trong các hệ thống máy tính truyền thống. Dự án Android Malware Genome đã thực hiện một phân tích thủ công toàn diện về mã độc Android, thu thập và phân tích 1200 mẫu mã độc, phân loại chúng thành 49 họ mã độc khác nhau. Bộ dữ liệu này được sử dụng rộng rãi trong nghiên cứu về bảo mật Android để đánh giá hiệu quả của các công cụ phát hiện và phân tích mã độc. Một ví dụ khác là VirusTotal bao gồm các ứng dụng độc hại đã được gửi và quét bởi nhiều các công cụ chống virus hiện có.

Sau đây là một số loại mã độc phổ biến:

* + Vi-rút: Một vi-rút máy tính là một loại phần mềm độc hại được thiết kế để tự sao chép và lây lan. Nó có thể tự gắn vào các chương trình hoặc tệp hợp pháp mà không bị người dùng biết đến hoặc đồng thuận. Vi-rút máy tính là một dạng thức của mối đe dọa mạng và được coi là có hại cho cả người dùng cá nhân và tổ chức.
  + Sâu (Worm): Sâu máy tính là một loại phần mềm độc hại lây lan qua hệ thống và mạng máy tính, thường khai thác lỗ hổng để tự sao chép. Không giống như vi-rút, sâu máy tính không cần phải tự gắn vào các tệp hoặc chương trình hiện có để lây lan. Chúng có thể thực thi và lan truyền độc lập, khiến chúng đặc biệt hiệu quả trong việc phát tán nhanh chóng. Sâu là mối đe dọa về an ninh mạng đáng kể do khả năng gây ra thiệt hại nhanh chóng và lan rộng.
  + Mã độc tống tiền (Ransomware): Mã độc tống tiền là một dạng phần mềm độc hại mã hóa các tệp của nạn nhân hoặc khóa chúng khỏi hệ thống của họ, khiến dữ liệu của họ không thể truy cập được. Sau đó, kẻ tấn công yêu cầu thanh toán tiền chuộc, thường là bằng tiền điện tử, để đổi lấy việc cung cấp khóa giải mã hoặc khôi phục quyền truy cập vào dữ liệu bị xâm phạm. Các cuộc tấn công bằng mã độc tống tiền đã trở nên ngày càng phổ biến và gây gián đoạn trong những năm gần đây, ảnh hưởng đến cá nhân, doanh nghiệp và thậm chí cả cơ sở hạ tầng quan trọng.
  + Phần mềm gián điệp (Spyware): Spyware là một kiểu phần mềm độc hại bí mật thu thập thông tin từ máy tính hoặc thiết bị của người dùng mà không có sự nhận thức hoặc đồng thuận của họ. Nó được thiết kế để theo dõi và thu thập dữ liệu về hoạt động của người dùng, bao gồm hành vi online, thông tin cá nhân, thói quen duyệt web… Thông tin được thu thập này sẽ được gửi đến một máy chủ từ xa được kiểm soát bởi kẻ tấn công.
  + Addisplay: đây là một dạng phần mềm quảng cáo, chuyên hiển thị các quảng cáo không mong muốn trên thiết bị người dùng, thường là dưới dạng pop-up, biểu ngữ hoặc chuyển hướng trình duyệt đến các trang web quảng cáo. Addisplay có thể được cài đặt cùng với các phần mềm hợp pháp hoặc thông qua ứng dụng giả mạo. Mục tiêu chính của loại mã độc này là tạo doanh thu từ quảng cáo cho kẻ tấn công hoặc nhà phát triển không minh bạch. Mặc dù một số adware có thể chỉ gây phiền toái, nhưng nhiều trường hợp lại xâm phạm quyền riêng tư, giảm hiệu năng thiết bị, hoặc thậm chí hoạt động như một backdoor để tải xuống mã độc nguy hiểm hơn.
  + Riskware (Phần mềm rủi ro): chỉ những chương trình hợp pháp nhưng có thể bị khai thác để gây hại hoặc thực hiện hành vi trái phép. Riskware không phải lúc nào cũng được tạo ra với mục đích xấu nhưng khi rơi vào tay kẻ tấn công, nó có thể bị lạm dụng để phá vỡ bảo mật hệ thống, thu thập dữ liệu hoặc can thiệp vào các chức năng quan trọng của hệ điều hành. Ví dụ về loại phần mềm này bao gồm phần mềm điều hành, công cụ chụp màn hình, phần mềm quản lý hệ thống….Việc sử dụng và cài đặt loại phần mềm này cần được kiểm soát nghiêm ngặt, đặc biệt trong môi trường doanh nghiệp hoặc tổ chức.
  + SPR: Đây là thuật ngữ dùng để chỉ những phần mềm hoặc hành vi có thể gây ra rủi ro đến bảo mật hoặc quyền riêng tư của người dùng, dù chúng không phải là mã độc theo nghĩa truyền thống. SPR thường bao gồm các ứng dụng hoặc tiện ích có hành vi như:
    - Thu thập dữ liệu nhạy cảm mà không có thông báo rõ ràng
    - Gửi thông tin người dùng về máy chủ bên ngoài
    - Truy cập camera, micro, danh bạ mà không có lý do chính đáng
    - Tạo điều kiện cho bên thứ ba thu thập thông tin cá nhân
* Các kỹ thuật che giấu mã độc

Để né tránh sự phát hiện của các hệ thống phòng chống mã độc, các nhà phát triển mã độc Android thường sử dụng nhiều kỹ thuật che giấu nhằm làm cho mã độc trở nên khó bị nhận biết, khó phân tích hoăc khó truy vết. Những kỹ thuật này không chỉ giúp mã độc vượt qua các hệ thống phát hiện truyền thống mà còn làm phức tạp hóa quá trình phân tích của các nhà nghiên cứu bảo mật. Dưới đây là một số kỹ thuật phổ biến thường được sử dụng trong phát triển mã độc trên nền tảng này:

* + Mã hóa và làm rối mã nguồn (Code Obfuscation)

Đây là kỹ thuật phổ biến nhất được sử dụng nhằm làm cho mã nguồn trở nên khó hiểu hoặc không thể đọc được đối với con người hoặc các công cụ phân tích mã. Các kỹ thuật obfuscation bao gồm:

* + - Đổi tên biến, hàm và lớp thành các chuỗi vô nghĩa (ví dụ: a.b.c() thay vì getUserLocation()).
    - Chèn mã thừa (junk code) hoặc các đoạn mã không ảnh hưởng đến chức năng nhưng gây nhiễu cho quá trình phân tích.
    - Làm rối luồng điều khiển khiến luồng thực thi chương trình trở nên phức tạp.
    - Sử dụng các kỹ thuật mã hóa dữ liệu hoặc mã hóa chuỗi để che giấu thông tin nhạy cảm như URL, số điện thoại, từ khóa độc hại…
  + Tải và thực thi mã động

Mã độc có thể không chứa mã độc hại ngay trong tệp APK gốc, mà thay vào đó nó sẽ tải mã độc từ một máy chủ bên ngoài khi được cài đặt và kích hoạt. Điều này giúp mã độc tránh được quá trình phân tích tĩnh vì mã thực thi không tồn tại ở thời điểm quét.

* + Sử dụng kỹ thuật chống phân tích

Mã độc Android thường tích hợp các cơ chế nhằm phát hiện và tránh né môi trường phân tích như sandbox, máy ảo hoặc trình giả lập. Một số kỹ thuật bao gồm:

* + - Phát hiện tên thiết bị, IMEI, hoặc các thông tin hệ thống thường được gán cố định trong emulator.
    - Kiểm tra độ trễ CPU, bộ nhớ và hành vi người dùng giả tạo.
    - Không kích hoạt hành vi độc hại trừ khi chạy trên thiết bị thật.
  + Kỹ thuật phân mảnh

Một số mã độc chia nhỏ chức năng độc hại thành nhiều phần và phân bố rải rác trong mã nguồn hoặc mô-đun phụ. Điều này khiến cho việc phát hiện hành vi độc hại trở nên khó khăn hơn nếu chỉ phân tích từng thành phần riêng lẻ.

* + Sử dụng kỹ thuật native

Thay vì chỉ sử dụng mã Java, mã độc có thể sử dụng thư viện native (viết bằng C/C++) thông qua JNI. Do mã native khó phân tích hơn và ít bị các công cụ phát hiện tĩnh kiểm tra, đây là một kỹ thuật phổ biến để thực hiện các hành vi như keylogging, đánh cắp dữ liệu.

* + Kỹ thuật thay đổi hành vi theo thời gian

Mã độc có thể chỉ kích hoạt sau một khoảng thời gian nhất định kể từ khi cài đặt, hoặc vào các thời điểm cụ thể, làm cho việc phát hiện hành vi độc hại trong giai đoạn đầu trở nên khó khăn.

Việc nắm bắt được đặc điểm và cơ chế hoạt động của các kỹ thuật che giấu là bước chuẩn bị quan trọng để xây dựng các mô hình phát hiện mã độc có tính khái quát và chính xác cao.

* Các phương pháp phân tích

Phân tích mã độc bao gồm quá trình phân tích và hiểu phần mềm độc hại đó để xác định chức năng, hành vi và tác động tiềm ẩn của nó. Phân tích phần mềm độc hại cung cấp góc nhìn rõ ràng về phần mềm độc hại trong mã và ý nghĩa của chuỗi byte được kẻ tấn công sử dụng để đánh cắp thông tin hoặc thay đổi mã gốc. Trong quá trình phân tích, các đặc trưng chính khác nhau của mã được trích xuất cung cấp thông tin và chức năng của các hành động của mã độc vào hệ thống.

* + Phân tích tĩnh

Phân tích tĩnh là phương thức kiểm tra mã mà không cần thực thi chương trình và tạo ra kết quả bằng cách kiểm tra cấu trúc mã, trình tự các câu lệnh và cách các biến được xử lý trong các lệnh gọi hàm khác nhau.

Ưu điểm chính của phân tích tĩnh là toàn bộ mã đều được phân tích. Một quy trình phân tích tĩnh bắt đầu bằng cách biểu diễn mã ứng dụng đã phân tích cho một số mô hình trừu tượng (ví dụ: biểu đồ luồng điều khiển hoặc sơ đồ lớp UML) dựa trên mục đích để phân tích. Các mô hình trừu tượng đó thực sự cung cấp một giao diện đơn giản hóa để hỗ trợ các phân tích máy khách cấp cao hơn như phân tích vết (taint analysis). Một ví dụ khác chẳng hạn như giá trị của các biến (ví dụ: được truyền từ các giá trị hằng số) tại các câu lệnh khác nhau của CFG cũng có thể được thu thập để cho phép phân tích tĩnh hỗ trợ xác minh chuyên sâu hơn, ví dụ: thông qua phân tích luồng dữ liệu.

Nhược điểm chính của phân tích tĩnh là sự đánh đổi giữa tính không chính xác và hiệu suất, vì một số thông tin chỉ có thể được biết chính xác khi chạy chương trình. Hơn nữa, việc mô hình hóa việc thực thi ứng dụng với độ chính xác cao đòi hỏi nhiều tài nguyên. Sự đánh đổi giữa hiệu suất, độ chính xác và tính hoàn chỉnh phải thực hiện sẽ phụ thuộc vào mục tiêu cuối cùng và các ràng buộc vận hành của nó.

* + Phân tích động

Khi chạy ứng dụng Android ở môi trường thực tế hay môi trường ảo chẳng hạn như sandbox, các tính năng hành vi thời gian chạy được gọi là đặc trưng động, phương pháp phân tích tương ứng là phân tích động. Đặc biệt, đối với ứng dụng Android, đối tượng của phân tích động bao gồm lời gọi hệ thống, lời gọi API, lưu lượng mạng và dữ liệu CPU. Vì công cụ phân tích động có thể truy cập vào thông tin thời gian chạy từ việc thực thi ứng dụng, do vậy nó thường chính xác hơn phân tích tĩnh. Nó phân tích các chỉ thị và đường dẫn được biết là có thể thực thi và truy cập trong ứng dụng. Ví dụ như trên hệ điều hành Android, kỹ thuật fuzzing được sử dụng để tìm kiếm lỗ hổng phần mềm giúp cải thiện mức bao phủ kiểm thử, ít lỗi bảo mật và chất lượng mã cao hơn.

Điểm yếu chính của phân tích động là chỉ có mã đã được thực thi mới có thể được phân tích. Việc có thể tự động khám phá toàn bộ mã trong ứng dụng có thể khó khăn. Mã chưa được khám phá có thể dẫn tới kết quả False Negative, khi phân tích bỏ sót hành vi hoặc hành động do ứng dụng thực hiện và có thể nhận được kết quả không đầy đủ. Hơn nữa, các kỹ thuật che giấu mã độc như mã hóa, nén hoặc tải mã động có thể khiến hành vi độc hại không được kích hoạt trong quá trình phân tích, làm giảm hiệu quả của phương pháp này. Do đó để đạt hiệu quả tối ưu, phân tích động thường được kết hợp với phân tích tĩnh trong các hệ thống phát hiện mã độc hiện đại.

* + Phương pháp phân tích kết hợp

Phân tích kết hợp là phương pháp tận dụng điểm mạnh của cả phân tích tĩnh và phân tích động để cải thiện độ chính xác và khả năng phát hiện hành vi độc hại của mã. Trong phương pháp này, phân tích tĩnh được sử dụng để quét và trích xuất thông tin từ toàn bộ mã nguồn hoặc mã bytecode mà không cần thực thi, từ đó xác định các điểm nghi ngờ hoăc khu vực mã có thể chứa hành vi nguy hiểm. Sau đó, phân tích động sẽ được triển khai nhắm vào các điểm này trong môi trường giám sát thời gian thực để xác nhận hành vi và đánh giá tác động thực tế.

Việc kết hợp này không chỉ giúp phát hiện được các hành vi độc hại tiềm ẩn mà còn giảm thiểu nguy cơ bỏ sót do mã chưa được thực thi hoặc bị che giấu thông qua kỹ thuật trốn tránh phân tích. Các hệ thống phát hiện mã độc tiên tiến hiện nay thường áp dụng mô hình phân tích kết hợp để cân bằng giữa độ phủ, độ chính xác và hiệu suất. Ngoài ra, việc tích hợp học máy vào phân tích kết hợp còn giúp tự đông hóa quá trình nhận diện mẫu mã độc mới, từ đó nâng cao khả năng phát hiện sớm các mối đe dọa chưa từng được biết đến.

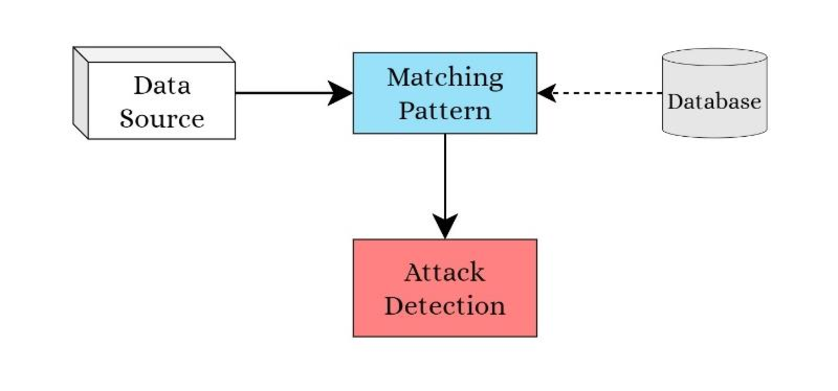
## Các phương pháp phát hiện mã độc

Có nhiều phương pháp được phát triển để phát hiện mã độc trên Android, mỗi phương pháp có ưu nhược điểm riêng và có thể được chia thành các nhóm như sau:

* Phát hiện dựa trên chữ ký (Signature-based Detection)

Phương pháp này dựa trên việc so sánh các đoạn mã trong ứng dụng với cơ sở dữ liệu chứa các chữ ký mã độc đã biết. Đây là kỹ thuật truyền thống và được sử dụng phổ biến trong các phần mềm diệt vi-rút. Nếu có sự trùng khớp, ứng dụng sẽ bị đánh dấu là độc hại. Ví dụ, so sánh khớp mã SHA-256 của tệp APK với danh sách đen, tìm kiếm chuỗi lệnh hoăc API độc hại như sendTextMessage, exec(“su”).

* + Ưu điểm: Nhanh, hiệu quả cao đối với các mã độc đã được nhận biết.
  + Hạn chế: Không phát hiện được mã độc mới (zero-day), mã độc biến thể.



Hình 1.3: Sơ đồ tổng quan về phương pháp phát hiện dựa trên chữ ký

* Phát hiện dựa trên hành vi (Behavior-based Detection)

Phương pháp này theo dõi hành vi thời gian thực hoặc mô phỏng của ứng dụng khi chạy trên thiết bị hoặc môi trường giả lập (sandbox) để phát hiện hành vi đáng ngờ, nguy hiểm. Phân tích hành vi có thể được thực hiện trên thiết bị thật hoặc trong môi trường ảo sandbox.

Các hành vi thường được giám sát như:

* + Truy cập vào danh bạ tin nhắn, định vị mà không có sự đồng ý của người dùng.
  + Gửi tin nhắn SMS tới đầu số tính phí cao.
  + Tải và cài đặt mã độc từ máy chủ bên ngoài.
  + Cố gắng chiếm quyền kiểm soát của hệ thống.

Ưu điểm: Có khả năng phát hiện các mối đe dọa chưa được biết tới.

Hạn chế: Có thể bị mã độc qua mặt nếu hành vi bị ẩn, chi phí tính toán cao, yêu cầu tài nguyên lớn.

* Phát hiện dựa trên học máy (Machine Learning-based Detection)

Phát hiện mã độc bằng học máy là hướng tiếp cận hiện đại và đang ngày càng phổ biến. Phương pháp này hoạt động theo quy trình sau đây:

* + Thu thập dữ liệu từ các ứng dụng
  + Trích xuất đặc trưng từ mã (chẳng hạn như lời gọi API, quyền truy cập,…)
  + Huấn luyện mô hình học máy
  + Sử dụng mô hình được huấn luyện để phân loại, phát hiện mã độc

Ưu điểm: Tự động phát hiện cả mã độc mới và biến thể, khả năng tổng quát hóa tốt.

Hạn chế: Phụ thuộc vào chất lượng dữ liệu huấn luyện, nhạy cảm với dữ liệu nhiễu.

Mỗi phương pháp phát hiện mã độc đều có vai trò và phạm vi ứng dụng khác nhau. Trong bối cảnh mã độc ngày càng tinh vi, đặc biệt là với các kỹ thuật che giấu và biến thể liên tục, việc kết hợp, lựa chọn phương pháp phù hợp ngày càng quan trọng. Đặc biệt, học máy đang trở thành công cụ trung tâm, có khả năng tự động hóa và tổng quát tốt, giúp tăng cường độ chính xác và khả năng thích ứng trong việc phát hiện mã độc Android hiện đại.

## Kết luận chương

Chương 1 đã trình bày tổng quan về các kiến thức nền tảng liên qua đến đề tài gồm hệ điều hành Android, các khái niệm cơ bản về mã độc Android. Từ đó chương đã phân tích đặc điểm, cách thức lây nhiễm và hành vi nguy hiểm của mã độc trên Android, tạo nền tảng để hiểu rõ hơn về các kỹ thuật phát hiện ngày nay. Các phương pháp phát hiện mã độc như dựa trên chữ ký, hành vi, học máy cũng đã được trình bày. Mỗi phương pháp đều có những ưu điểm và hạn chế riêng, trong đó phương pháp dựa trên học máy đang nổi lên như một hướng tiếp cận hiệu quả nhờ khả năng học tự động và tổng quát hóa tốt. Chương tiếp theo sẽ trình bày các kiến thức liên quan đến học máy và đề xuất mô hình phát hiện mã độc phù hợp.

# TÌM HIỂU MÔ HÌNH HỌC SÂU KẾT HỢP

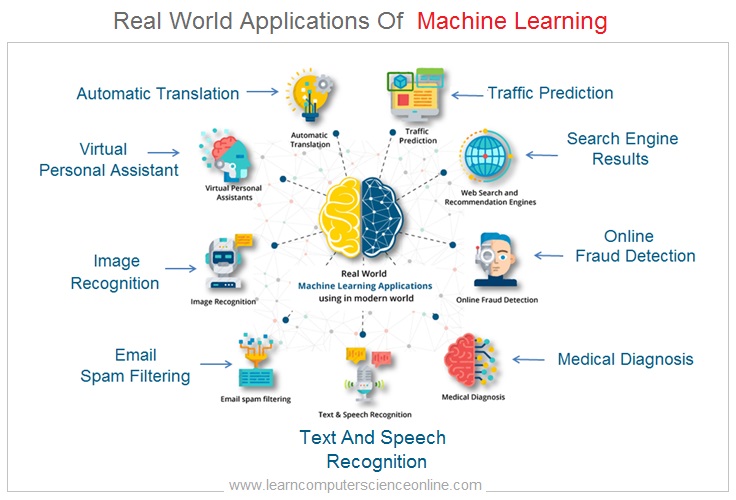
## Kiến thức nền tảng

### Tổng quan về học máy

Học máy là một nhánh của trí tuệ nhân tạo, được định nghĩa rộng rãi là khả năng của máy tính có thể nhận diện các mẫu và cải thiện hiệu suất của chúng theo thời gian mà không cần phải được lập trình cho mọi tình huống xảy ra. Thay vì tuân theo các quy tắc cứng nhắc, hệ thống học máy sẽ phân tích dữ liệu, đưa ra dự đoán và điều chỉnh cách tiếp cận của chúng dựa vào quá trình học. Nhờ đó, máy tính có thể thực hiện các tác vụ mà trước đây đòi hỏi trực giác của con người, như nhận diện đối tượng trong ảnh, hiểu ngôn ngữ nói hoặc phát hiện giao dịch gian lận. Vào năm 1959, Arthur Samuel, một trong những người tiên phong trong ngành khoa học máy tính đã định nghĩa học máy là “lĩnh vực nghiên cứu cung cấp cho máy tính khả năng học mà không cần được lập trình rõ ràng”. Ý tưởng này sau đó được củng cố bởi Herbert Simon, người được coi là cha đẻ của trí tuệ nhân tạo, giải thích rằng học máy về cơ bản là cải thiện hiệu suất thông qua kinh nghiệm, giống như con người trở nên giỏi hơn trong các nhiệm vụ thông qua luyện tập.

Học máy bắt đầu với dữ liệu – chữ số, ảnh, hoặc văn bản, ví dụ như giao dịch ngân hàng, ảnh người, dữ liệu chuỗi thời gian từ cảm biến hoặc báo cáo bán hàng. Dữ liệu được thu thập và chuẩn bị để sử dung làm dữ liệu đào tạo hoặc thông tin mà mô hình học máy sẽ được đào tạo. Càng nhiều dữ liệu, chương trình càng tốt. Từ đây, người lập trình sẽ chọn mô hình học máy để sử dụng, cung cấp dữ liệu và để mô hình máy tính tự huấn luyện để tìm ra các mẫu hoặc đưa ra dự đoán. Theo thời gian, người lập trình cũng có thể điều chỉnh mô hình, bao gồm thay đổi các tham số của nó, để giúp đưa ra kết quả chính xác hơn. Một số dữ liệu được giữ lại từ dữ liệu huấn luyện để sử dụng làm dữ liệu đánh giá, kiểm tra độ chính xác của mô hình học máy khi gặp dữ liệu mới. Kết quả là một mô hình có thể được sử dụng trong tương lai với các tập dữ liệu khác nhau.

Sau khi mô hình được huấn luyện, nó có thể được áp dụng vào các tình huống thực tế nhằm đưa ra dự đoán hoặc quyết định tự đông. Nhờ tính linh hoạt, học máy đã và đang được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau như y tế, tài chính, thương mại điện tử, giao thông,…Việc khai thác khả năng phân tích và dự đoán từ dữ liệu tối ưu hóa quy trình, giảm chi phí và nâng cao hiệu quả hoạt động trong hầu hết các ngành công nghiệp hiện đại.



Hình 2.1: Ứng dụng của Học máy trong các lĩnh vực khác nhau

Tùy theo cách dữ liệu được cung cấp và mục tiêu học, học máy được chia thành ba nhóm chính bao gồm:

* + - Mô hình học máy có giám sát (Supervised Learning) được đào tạo bằng cách các tập dữ liệu được gắn nhãn, cho phép mô hình học và phát triển chính xác hơn theo thời gian. Ví dụ, một thuật toán sẽ được đào tạo bằng hình ảnh về mèo và các thứ khác, tất cả đều được gắn nhãn bởi con người, và máy sẽ tự học cách nhận dạng hình ảnh mèo. Học máy có giám sát là loại phổ biến nhất được sử dụng hiện nay.
    - Trong học máy không giám sát (Unsupervised Learning), một chương trình tìm kiếm các mẫu trong dữ liệu không được gắn nhãn. Học máy không giám sát có thể tìm thấy các mẫu hoặc xu hướng mà mọi người không tìm kiếm một cách rõ ràng. Ví dụ, một chương trình học máy không giám sát có thể xem qua dữ liệu bán hàng trực tuyến và xác định các loại khách hàng khác nhau thực hiện mua hàng.
    - Học máy tăng cường (Reinforcement Learning) là kiểu học dựa trên thử và sai. Thay vì học trên một tập dữ liệu cố định, nhận phản hồi thông qua cơ chế thưởng và phạt. Theo thời gian, nó tinh chỉnh các chiến lược của mình để tối đa hóa các kết quả tích cực. Một chiếc xe tự lái là một ví dụ điển hình. Nó không có một bộ hướng dẫn từ trước cho mọi tình huống có thể xảy ra trên đường. Thay vào đó, nó học bằng cách thử các hành động khác nhau, chẳng hạn như tăng tốc, phanh hoặc rẽ và quan sát kết quả. Khi một hành động đưa nó gần hơn với việc lái xe an toàn và hiệu quả, nó được củng cố như một lựa chọn tốt. Học tăng cường được sử dụng rộng rãi trong ngành robot, dự đoán thị trường chứng khoán và tối ưu hóa logistics.
* Một số thuật toán phổ biến trong lĩnh vực học máy như:
  + K-nearest neighbors (KNN): Thuật toán KNN dựa trên khoảng cách giữa các điểm dữ liệu để phân loại hoặc hồi quy. Một điểm dữ liệu mới được gán nhãn dựa trên đa số nhãn của các điểm gần nhất trong tập luyện.
  + Cây quyết định (DT) và Rừng ngẫu nhiên (RF): Cây quyết định là mô hình đơn giản, trực quan giúp phân loại dữ liệu bằng cách đặt ra các câu hỏi nhị phân theo đặc trưng. Rừng ngẫu nhiên là tổ hợp nhiều Cây quyết định để tăng độ chính xác và giảm hiện tượng overfitting.
  + SVM (Support Vector Machine): Đây là một trong những thuật toán phân loại mạnh mẽ, hoạt động hiệu quả trong không gian đa chiều. Mô hình tìm siêu phẳng (hyperplane) tối ưu để phân tách các lớp dữ liệu sao cho khoảng cách tới các điểm gần nhất của hai lớp là lớn nhất. SVM cũng có thể sử dụng kernel để xử lý các bài toán phi tuyến.
  + Hồi quy tuyến tính (Linear Regression): Hồi quy tuyến tính là thuật toán đơn giản nhưng hiệu quả, dùng để dự đoán một giá trị liên tục dựa trên mối quan hệ tuyến tính giữa các biến đầu vào và đầu ra. Đây là nền tảng cho nhiều mô hình hồi quy khác phức tạp hơn.
  + Hồi quy logistic (Logistic Regression): Khác với hồi quy tuyến tính, hồi quy logistic dùng để giải quyết các bài toán phân loại nhị phân. Mô hình sử dụng hàm sigmoid để ánh xạ đầu ra thành xác suất thuộc về một lớp cụ thể.
* Ứng dụng của học máy trong lĩnh vực an ninh mạng

Trong bối cảnh các mối đe dọa an ninh mạng ngày càng gia tăng về cả số lượng lẫn mức độ tinh vi, các phương pháp truyền thống như sử dụng chữ ký (signature-based) hay quy tắc thủ công (rule-based) không còn đủ hiệu quả để phát hiện các cuộc tấn công mới, chưa từng biết đến (zero-day attacks). Chính vì vậy, học máy ngày càng đóng vai trò quan trọng trong việc xây dựng các hệ thống phát hiện và phòng chống tấn công mạng hiện đại. Một trong những ứng dụng nổi bật của học máy trong lĩnh vực này là phân loại mã độc. Dựa trên việc học từ dữ liệu hành vi hoặc đặc trưng của các ứng dụng, các mô hình học máy có thể phát hiện phần mềm độc hại với độ chính xác cao, kể cả khi chưa từng thấy các mẫu trước đó. Bên cạnh đó học máy còn được ứng dụng trong nhiều lĩnh vực khác của an ninh mạng như:

* + Phát hiện xâm nhập (Intrusion Detection Systems -IDS): sử dụng mô hình học máy để nhận biết hành vi bất thường trong lưu lượng mạng.
  + Phát hiện tấn công từ chối dịch vụ (DDoS): giúp phân biệt các mẫu lưu lượng hợp lệ và tấn công.
  + Phân loại email spam/phishing: mô hình học máy được sử dụng để phân tích nội dung và phát hiện thư rác hoặc giả mạo.

### Tổng quan về học sâu

* Khái niệm

Học sâu (Deep Learning) được xem là một nhánh mở rộng của học máy, sử dụng mạng nơ-ron nhiều lớp, được gọi là mạng nơ-ron sâu (deep neural networks), để mô phỏng sức mạnh ra quyết định phức tạp của bộ não con người. Dựa trên mạng nơ-ron nhân tạo, học sâu cho phép các hệ thống máy học có thể tự động trích xuất đặc trưng và học từ dữ liệu ở mức độ trừu tượng cao, đặc biệt hiệu quả trong các bài toán như nhận diện hình ảnh, xử lý ngôn ngữ tự nhiên hay nhận dạng giọng nói.

Bộ não của con người truyền cảm hứng cho mạng nơ-ron nhân tạo. Chúng có thể giải quyết nhiều vấn đề bằng các kỹ thuật như xử lý ngôn ngữ tự nhiên, nhận dạng hình ảnh và nhận dạng giọng nói.

* Sự bùng nổ của học sâu

Trong vòng một thập niên trở lại đây, học sâu đã trải qua một sự phát triển bùng nổ và trở thành công nghệ cốt lõi trong nhiều ứng dụng trí tuệ nhân tạo hiện đại. Sự khởi đầu của làn sóng này có thể được đánh dấu bằng chiến thắng của mô hình mạng nơ-ron tích chập AlexNet tại cuộc thi ImageNet năm 2012 – nơi mạng học sâu vượt trội so với các phương pháp học máy truyền thống trong nhiệm vụ nhận diện hình ảnh quy mô lớn. Sự bùng nổ của học sâu trong thập kỷ qua được thúc đẩy bởi ba yếu tố chính: (1) sự gia tăng nhanh chóng của dữ liệu lớn, cung cấp nguồn thông tin dồi dào để huấn luyện các mô hình phức tạp; (2) tiến bộ vượt bậc về phần cứng, đặc biệt là GPU và TPU, với khả năng xử lý song song vượt trội, giúp rút ngắn thời gian huấn luyện; và (3) những cải tiến về thuật toán, bao gồm các kỹ thuật tối ưu hóa hiện đại như Adam, RMSprop và kiến trúc mạng tiên tiến như ResNet hay Transformer.

* Một số thuật ngữ liên quan
* Hàm kích hoạt

Hàm kích hoạt là thành phần không thể thiếu trong mạng nơ-ron nhân tạo, giúp mô hình học được các quan hệ phi tuyến phức tạp. Một số hàm kích hoạt phổ biến như:

* + - Sigmoid:

Hàm có đầu ra nằm trong (0, 1), dùng cho phân loại nhị phân.

* + - Tanh:

Hàm có đầu ra nằm trong (-1, 1)

* + - ReLU:

Hàm đơn giản, hiệu quả, rất phổ biến.

* + - Softmax:

Dùng ở lớp đầu ra trong các bài toán phân loại đa lớp.

* Backpropagation

Đây là thuật toán học có giám sát quan trọng nhất được sử dụng để huấn luyện mạng nơ-ron. Mục tiêu của quá trình này là tối ưu trọng số và bias trong mạng sao cho sai số giữa dự đoán và nhãn thực tế là nhỏ nhất.

Cơ chế hoạt động:

* + - Lan truyền xuôi (forward pass): Dữ liệu được truyền từ lớp đầu vào qua các lớp ẩn đến lớp đầu ra để tính dự đoán.
    - Tính toán lỗi: So sánh đầu ra dự đoán với nhãn thật bằng một hàm mất mát.
    - Lan truyền ngược (backward pass): Tính toán đạo hàm của lỗi theo từng trọng số, áp dụng quy tắc chuỗi để phân bố ngược từ lớp cuối cùng về đầu vào.
    - Cập nhật trọng số: Dùng các thuật toán tối ưu (SGD, Adam) để điều chỉnh trọng số theo hướng giảm lỗi.
  + Dropout

Đây là một kỹ thuật regularization mạnh mẽ giúp mạng nơ-ron tránh bị quá khớp (overfitting), giúp mô hình tổng quát hơn trên dữ liệu mới.

Nguyên lý:

* + - Trong mỗi lần huấn luyện, một tỷ lệ ngẫu nhiên các nơ-ron trong mỗi lớp (thường từ 20% đến 50%) sẽ bị tạm thời “tắt”
    - Điều này buộc mạng phải học các biểu diễn phân tán và độc lập, tránh việc phụ thuộc vào một nhóm nơ-ron cụ thể.
* Các mô hình học sâu phổ biến
  + Mạng nơ-ron đa lớp MLP: là dạng cơ bản nhất của mạng nơ-ron, bao gồm các lớp đầu vào, ẩn và đầu ra, được kết nối đầy đủ với nhau.
  + Mạng nơ-ron tích chập CNN: được thiết kế đặc biệt để xử lý dữ liệu có cấu trúc lưới như hình ảnh, với các lớp tích chập giúp phát hiện các đặc trưng không gian.
  + Mạng nơ-ron hồi tiếp RNN: phù hợp với dữ liệu chuỗi như văn bản hoặc tín hiệu thời gian, với khả năng lưu trữ thông tin theo thời gian.

Ngoài ra, các kiến trúc hiện đại như Transformer với cơ chế attention đang ngày càng trở nên phổ biến, đặc biệt trong các tác vụ xử lý ngôn ngữ tự nhiên và thị giác máy tính.

* Ưu điểm và ứng dụng

Học sâu đã chứng minh hiệu quả vượt trội trong nhiều lĩnh vực như:

* + Nhận diện hình ảnh và video: Phân loại đối tượng, nhận diện khuôn mặt, phân đoạn hình ảnh.
  + Xử lý ngôn ngữ tự nhiên: Dịch thuật, phân tích cảm xúc, chatbot
  + Y tế: Phát hiện bệnh từ hình ảnh y khoa, phân tích gen
  + Tài chính: Dự đoán xu hướng thị trường, phát hiện giao dịch gian lận
  + Giao thông và tự động hóa: Xe tự lái, tối ưu hóa lộ trình

Nhờ vào khả năng học đặc trưng tự động và xử lý dữ liệu quy mô lớn, học sâu đã trở thành công nghệ nền tảng trong nhiều hệ thống trí tuệ nhân tạo hiện đại, và tiếp tục là lĩnh vực nghiên cứu sôi động trong thời gian tới.

### Mạng học sâu nhiều lớp MLP

Mỗi đơn vị nơ-ron trong mạng có thể được biểu diễn bởi công thức sau:

Trong đó:

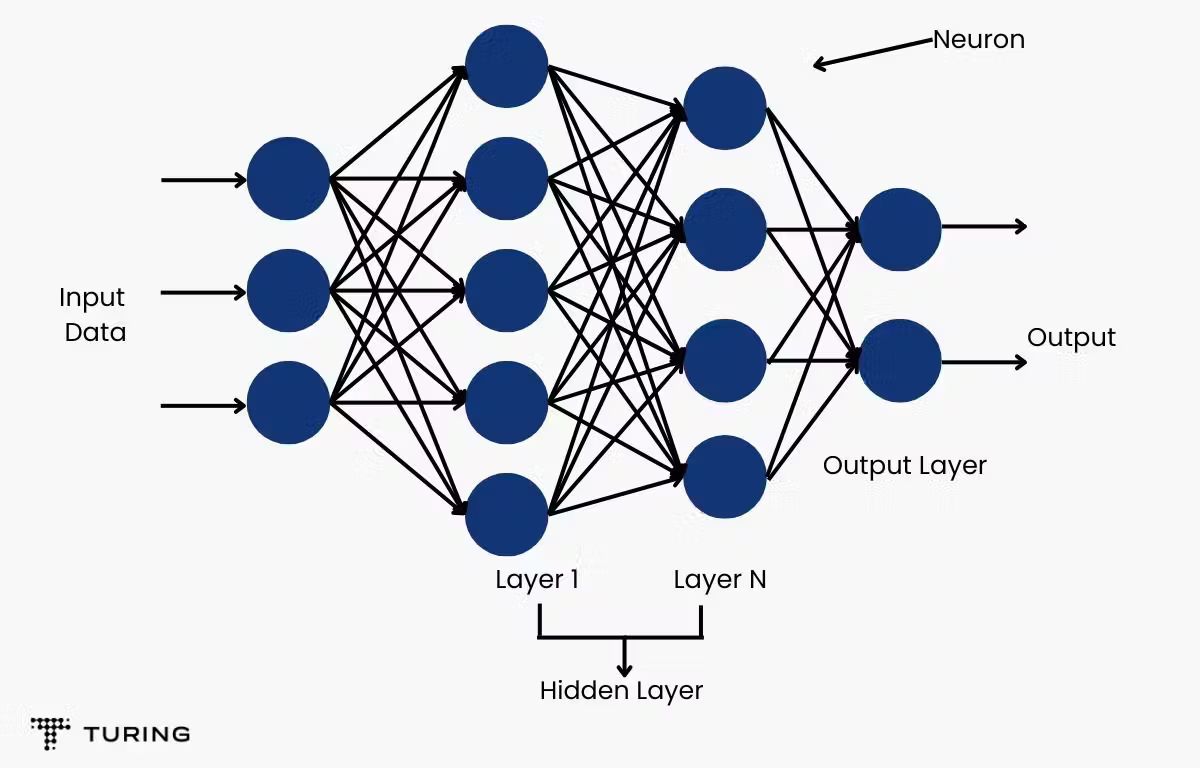
* + - * xj là đầu vào thứ j
      * wj là trọng số liên kết từ đầu vào xj
      * b là bias (hệ số dịch)
      * 𝜙 là hàm kích hoạt phi tuyến
      * a là đầu ra (kích hoạt) của nơ-ron.

Một vài ví dụ về các đơn vị xử lý này là:

* + Hồi quy tuyến tính sử dụng mô hình tuyến tính, do đó 𝜙(z) = z
  + Trong phân loại nhị phân tuyến tính, 𝜙 là một ngưỡng cứng tại 0.
  + Hồi quy logistic, 𝜙 là một hàm logistic 𝜎(z) = 1/(1+ )

Việc xếp chồng nhiều lớp nơ-ron có chức năng phi tuyến giúp MLP học được các ánh xạ phi tuyến giữa đầu vào và đầu ra. Nếu không có hàm kích hoạt phi tuyến ở giữa các lớp, mạng sẽ chỉ tương đương một phép biến đổi tuyến tính đơn giản, bất kể có bao nhiêu lớp. Mỗi đơn vị thực hiện chức năng rất đơn giản và rập khuôn, nhưng khi kết hợp lại, chúng có thể thực hiện một số phép tính rất hữu ích. Mạng nơ-ron truyền thẳng (FNN) trong đó các đơn vị được sắp xếp thành một đồ thị mà không có bất kỳ chu kỳ nào, do đó, tất cả các phép tính có thể thực hiện tuần tự. Điều này trái ngược với mạng nơ-ron hồi quy (RNN), trong đó đồ thị có thể có các chu kỳ, do đó quá trình xử lý có thể tự truyền vào chính nó.

Mạng Perceptron nhiều lớp MLP là một loại mạng nơ-ron nhân tạo thuộc nhóm mạng truyền thẳng FNN. Đây là một trong những mô hình học sâu cơ bản nhất, bao gồm nhiều lớp tuyến tính kết hợp với hàm kích hoạt phi tuyến giữa các lớp, cho phép mạng học được các ánh xạ phi tuyến phức tạp giữa đầu vào và đầu ra.



Hình 2.2 Mạng nơ-ron nhiều lớp MLP

Ở đây, các nơ-ron được sắp xếp thành một tập hợp các lớp, mỗi lớp chứa một số lượng nhất định các đơn vị nơ-ron. Mỗi đơn vị trong một lớp được kết nối với mọi đơn vị trong lớp tiếp theo; ta nói rằng mạng được kết nối đầy đủ. Lớp đầu tiên là lớp đầu vào và các đơn vị của nó lấy các giá trị của đặc trưng đầu vào. Lớp cuối cùng là lớp đầu ra và nó có một đơn vị đối với cho mỗi giá trị tại đầu ra của mạng nơ-ron ( tức là k đơn vị trong trường hợp phân loại k lớp). Tất cả các lớp ở giữa những lớp này được gọi là lớp ẩn, vì chúng ta không biết trước những đơn vị này tính toán những gì và điều này cần được khám phá trong quá trình học. Các đơn vị trong các lớp nêu trên lần lượt được gọi là đơn vị đầu vào, đơn vị đầu ra và đơn vị ẩn. Chiều sâu (depth) của mạng là số lượng các lớp, trong khi chiều rộng (width) là số lượng nơ-ron trong mỗi lớp. “Học sâu” đề cập đến khả năng huấn luyện các mô hình với nhiều lớp ẩn – điều từng là thách thức trong quá khứ do hạn chế về dữ liệu và tính toán.

## Thuật toán di truyền

### Nguyên lý hoạt động

Thuật toán di truyền (Genetic Algorithm – GA) là một loại thuật toán tối ưu hóa, nghĩa là chúng được sử dụng để tìm ra giải pháp tối ưu cho một bài toán nhất định, tối đa hóa hoặc tối thiểu hóa một hàm cụ thể. Thuật toán di truyền đại diện cho một nhánh của lĩnh vực nghiên cứu gọi là tính toán tiến hóa (evolutionary computation), trong đó chúng mô phỏng các quá trình sinh học của quá trình sinh sản và chọn lọc tự nhiên để giải quyết các giải pháp phù hợp nhất. Giống như trong quá trình tiến hóa, nhiều quy trình của thuật toán di truyền là ngẫu nhiên, tuy nhiên kỹ thuật tối ưu hóa này cho phép người ta thiết lập mức độ ngẫu nhiên và mức độ kiểm soát. Thuật toán này mạnh mẽ và hiệu quả hơn nhiều so với các thuật toán tìm kiếm ngẫu nhiên và tìm kiếm đầy đủ, nhưng không yêu cầu thông tin bổ sung về bài toán đã cho. Tính năng này cho phép tìm kiếm giải pháp cho các bài toán mà các phương pháp tối ưu hóa khác không thể xử lý do tính thiếu liên tục, đạo hàm hoặc không xác định rõ cấu trúc toán học.

Vì thuật toán di truyền được thiết kế để mô phỏng quá trình tự nhiên do đó hầu hết các thuật ngữ được mượn từ ngành sinh học. Cấu trúc và thành phần chính của thuật toán di truyền là:

* + - Hàm thích nghi (fitness function)
    - Nhiễm sắc thể (chromosome)
    - Quần thể (population)
    - Toán tử lựa chọn (selection operator)
    - Lai ghép (crossover)
    - Đột biến (Mutation)

Hàm thích nghi là hàm mà thuật toán đang cố tối ưu hóa. Từ “thích nghi” được lấy từ thuyết tiến hóa. Nó được sử dụng ở đây vì hàm thích nghi kiểm tra và định lượng mức độ phù hợp của từng giải pháp tiềm năng. Hàm thích nghi là một trong những thành phần quan trọng nhất của thuật toán.

Thuật ngữ nhiễm sắc thể dùng để chỉ biểu diễn (mã hóa) của một cá thể, tức là một giải pháp tiềm năng cho một vấn đề mà thuật toán di truyền đang cố gắng giải quyết. Mỗi giải pháp tiềm năng được mã hóa dưới dạng một mảng các giá trị tham số, một quá trình cũng được tìm thấy trong các thuật toán tối ưu khác. Nếu một vấn đề có *Npar* chiều, thì thông thường mỗi nhiễm sắc thể được mã hóa dưới dạng một mảng *Npar* phần tử

trong đó mỗi *pi* là một giá trị cụ thể của tham số thứ i. Người tạo ra thuật toán di truyền phải đưa ra cách dịch không gian mẫu của các giải pháp tiềm năng thành nhiễm sắc thể. Một cách tiếp cận là chuyển đổi từng giá trị tham số thành một chuỗi bit ( chuỗi 0 và 1), sau đó nối các tham số đầu cuối giống như các gen trong một chuỗi DNA để tạo ra nhiễm sắc thể. Theo truyền thống, nhiễm sắc thể thường được mã hóa theo cách này, và nó vẫn là một phương pháp phù hợp cho các không gian giải pháp rời rạc.

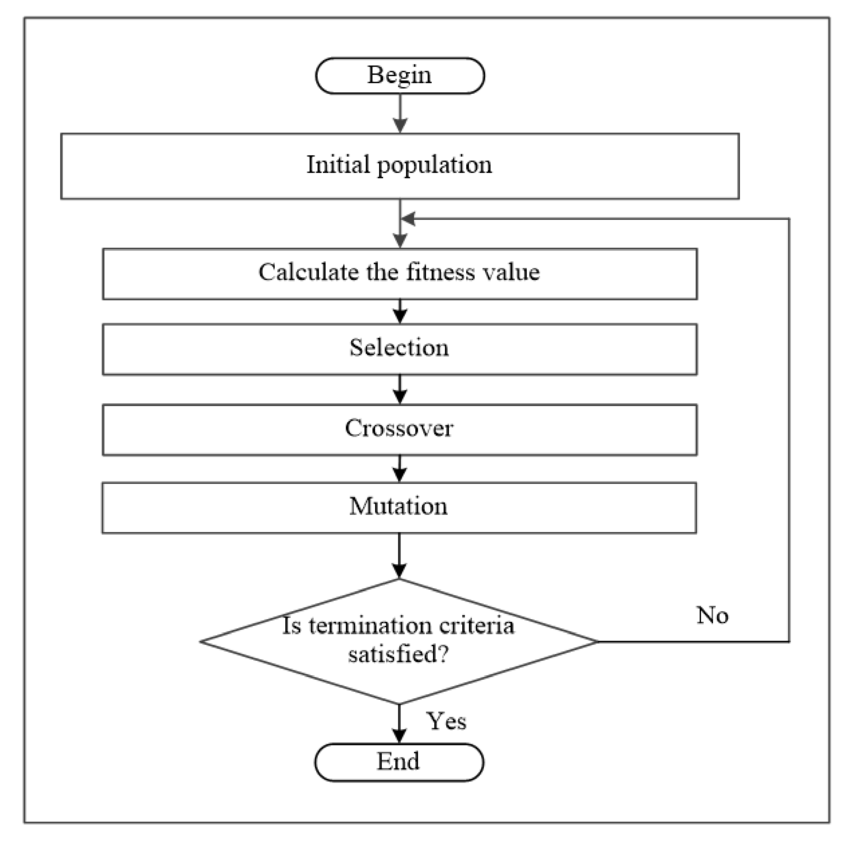
Thuật toán di truyền khởi đầu bằng một nhóm nhiễm sắc thể được chọn ngẫu nhiên, đóng vai trò là thế hệ đầu tiên (quần thể ban đầu). Sau đó, mỗi cá thể trong quần thể được đánh giá bằng hàm thích nghi để kiểm tra mức độ giải quyết vấn đề trước mắt của nó.

Bây giờ, toán từ lựa chọn sẽ chọn một vài nhiễm sắc thể để sinh sản dựa trên phân phối xác suất do người dùng xác định. Cá thể càng phù hợp thì càng có khả năng được lựa chọn. Ví dụ, nếu *f* là một hàm thích nghi không âm, xác suất nhiễm sắc thể *C53* được chọn để sinh sản có thể là

Chú ý rằng toán tử lựa chọn có thể chọn những nhiễm sắc thể có sự thay thế, do đó cùng một nhiễm sắc thể có thể được chọn nhiều lần. Toán tử lai ghép giống với sự lai ghép sinh học và tái tổ hợp nhiễm sắc thể trong giảm phân tế bào. Toán tử này hoán đổi một chuỗi con của hai nhiễm sắc thể đã chọn để tạo ra hai thế hệ con. Ví dụ, nếu nhiễm sắc thể cha mẹ là [11010111001000] và [01011101010010] được trao đổi chéo sau bit thứ tư thì [01010111001000] và [11011101010010] sẽ là con của chúng. Toán tử đột biến ngẫu nhiên đảo ngược từng bit riêng lẻ trong nhiễm sắc thể mới (biến 0 thành 1 và ngược lại). Thông thường, đột biến xảy ra với xác suất rất thấp, chẳng hạn như 0.001. Một số thuật toán triển khai toán tử đột biến trước các toán tử chọn lọc và toán tử lai ghép. Thoạt nhìn, toán tử đột biến có vẻ không cần thiết. Trên thực tế, nó đóng vai trò quan trọng, ngay cả khi nó là thứ yếu so với vai trò của chọn lọc và lai ghép. Chọn lọc và lai ghép duy trì thông tin di truyền của nhiễm sắc thể thích nghi hơn, nhưng các nhiễm sắc thể này chỉ thích nghi hơn so với thế hệ hiện tại. Nói cách khác, thuật toán có thể bị kẹt ở mức tối ưu cục bộ trước khi tìm ra mức tối ưu toàn cục. Toán tử đột biến giúp bảo vệ chống lại vấn đề này bằng cách duy trì tính đa dạng trong quần thể, nhưng nó cũng có thể khiến thuật toán hội tụ chậm hơn.

Thông thường, quá trình chọn lọc, lai ghép và đột biến sẽ tiếp tục cho đến khi số lượng con cháu bằng với quần thể ban đầu, do đó thế hệ thứ hai được tạo thành hoàn toàn từ con cháu mới và thế hệ đầu tiền được thay thế hoàn toàn. Bây giờ, thế hệ thứ hai sẽ được kiểm tra bởi hàm thích nghi, và chu kỳ lặp lại. Một thực hành phổ biến là ghi lại cá thể có độ thích nghi cao nhất (cùng với giá trị độ thích nghi của nó) từ mỗi thế hệ hoặc cá thể “tốt nhất hiện tại”.

Thuật toán di truyền được lặp lại cho đến khi giá trị độ phù hợp của cá thể “tốt nhất hiện tại” ổn định và không thay đổi trong nhiều thế hệ. Điều này có nghĩa là thuật toán đã hội tụ đến một hoặc nhiều giải pháp. Toàn bộ quá trình lắp lại được gọi là một lần chạt. Vào cuối mỗi lần chạy, thường có ít nhất một cá thể có độ thích nghi cao với bài toán. Tùy thuộc vào thuật toán được triển khai, cá thể được chọn làm nghiệm cuối cùng có thể là cá thể tốt nhất trong toàn bộ quá trình chạy hoặc chỉ là cá thể tốt nhất ở thế hệ cuối cùng.



Hình 2.3: Sơ đồ tổng quan thuật toán di truyền

Hiệu suất của thuật toán di truyền phụ thuộc rất nhiều vào phương pháp được sử dụng để mã hóa các giải pháp tiềm năng thành nhiễm sắc thể và “tiêu chí cụ thể để thành công”, hoặc hàm thích nghi thực sự đang đo lường điều gì. Các chi tiết quan trọng khác là xác suất lai ghép, đột biến, kích thước quần thể và số lần lặp lại. Những giá trị này có thể được điều chỉnh sau khi đánh giá hiệu suất của thuật toán trong một vài lần chạy thử.

### Ưu điểm và hạn chế

* Ưu điểm

Thuật toán di truyền có nhiều ưu điểm hơn so với các phương thức truyền thống. Không giống như các phương pháp tìm kiếm đơn nghiệm dựa như leo đồi (hill-climbing), thuật toán di truyền tiến hóa từ một quần thể các cá thể giải pháp thay vì một giá trị duy nhất. Điều này làm giảm đáng kể khả năng tìm thấy một giải pháp tối ưu cục bộ thay vì tối ưu toàn cục. Thuật toán di truyền không yêu cầu thông tin đạo hàm hay liên tục của hàm mục tiêu. GA chỉ cần giá trị hàm thích nghi để đánh giá mức độ tốt của mỗi cá thể, do đó nó có thể áp dụng cho các bài toán không trơn, phi tuyến, không khả vi hoặc có không gian tìm kiếm bị nhiễu.

Ngoài ra, GA còn có các lợi ích nổi bật sau:

* + - Tính linh hoạt cao: dễ dàng áp dụng cho các bài toán tối ưu rời rạc hoặc liên tục, bài toán nhiều mục tiêu, ràng buộc phức tạp.
    - Khả năng song song hóa: do các cá thể trong quần thể có thể được đánh giá độc lập, GA phù hợp với môi trường xử lý song song.
    - Khả năng mở rộng và kết hợp: thuật toán có thể tích hợp với các kỹ thuật khác như học sâu, fuzzy logic hoặc tìm kiếm cục bộ (local search) để nâng cao hiệu suất.
* Hạn chế

Tuy nhiên, thuật toán di truyền cũng tồn tại một số hạn chế nhất định:

* + - Chi phí tính toán cao: đối với các bài toán phức tạp, việc đánh giá hàm thích nghi có thể rất tốn kém về thời gian và tài nguyên, đặc biệt khi phải mô phỏng các giải pháp. Điều này khiến GA không phù hợp cho các ứng dụng yêu cầu thời gian thực hoặc tài nguyên hạn chế.
    - Hội tụ chậm và dễ mắc vào hội tụ sớm: GA có thể cần nhiều thế hệ để hội tụ và đôi khi hội tụ vào các nghiệm cục bộ không tối ưu nếu không được thiết lập đúng (ví dụ: xác suất đột biến quá thấp, đa dạng quần thể giảm đi).
    - Phụ thuộc vào tham số: hiệu suất GA phụ thuộc nhiều vào cách chọn các tham số như kích thước quần thể, xác suất lai ghép, xác suất đột biến, số vòng lặp…Việc lựa chọn này yêu cầu kiến thức chuyên môn và thử nghiệm thực tế.
    - Khó khăn trong thiết kế hàm thích nghi: lựa chọn đúng hàm thích nghi ảnh hưởng trực tiếp đến hiệu quả tìm kiếm. Một hàm thiết kế không tốt có thể dẫn tới hướng tìm kiếm sai lệch hoặc không có khả năng phân biệt giữa các cá thể.

## Mô hình học sâu kết hợp thuật toán di truyền

### Mục tiêu

Học sâu đã chứng minh hiệu quả vượt trội trong nhiều bài toán phân loại. Mô hình MLP là một trong những mô hình cơ bản nhưng có tiềm năng mạnh mẽ khi được huấn luyện với tập dữ liệu đủ lớn và cấu hình mô hình hợp lý. Tuy đơn giản về cấu trúc, nhưng hiệu năng của MLP phụ thuộc rất lớn vào cấu trúc mạng và các siêu tham số, bao gồm số lớp ẩn, số lượng nơ-ron mỗi lớp, tỷ lệ dropout, tốc độ học…Việc tinh chỉnh các tham số này thông qua thử nghiệm thủ công là quá trình tốn thời gian, dễ bị sai lệch do trực giác hoặc thiếu kinh nghiệm, và không đảm bảo tìm ra được mô hình có hiệu quả tốt.

Do đó, mục tiêu chính đề xuất trong đề tài này là đề xuất một mô hình GA-MLP, trong đó thuật toán di truyền được tích hợp để tự động hóa việc tối ưu cấu trúc và siêu tham số của mạng MLP. GA sẽ tìm kiếm trong không gian siêu tham số các cấu hình có độ chính xác phân loại cao nhất thông qua quá trình tiến hóa.

Cụ thể, nghiên cứu này hướng đến các mục tiêu sau:

* + - Tự động hóa quá trình thiết kế và điều chỉnh mạng MLP.
    - Tối ưu hóa các siêu tham số quan trọng nhằm đạt độ chính xác phân loại cao trên tập dữ liệu mã độc Android.
    - Cải thiện khả năng khái quát hóa của mô hình, tránh overfitting.
    - Khai thác tính linh hoạt và mạnh mẽ của GA trong việc tìm kiếm không gian tham số lớn và phi tuyến tính.

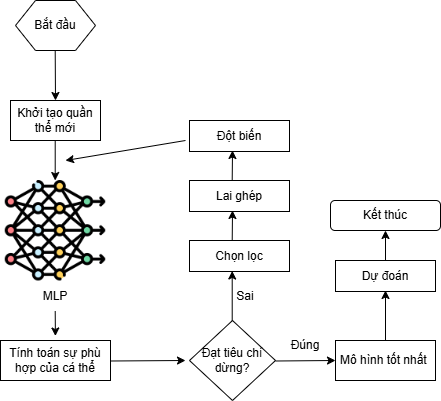
Bằng cách kết hợp mô hình học sâu MLP với thuật toán di truyền, nghiên cứu kỳ vọng xây dựng được một mô hình phát hiện mã độc có độ chính xác cao, ổn định và có khả năng mở rộng sang các bài toán tương tự trong các lĩnh vực khác.

### Mô hình GA-MLP

Mô hình GA-MLP được xây dựng nhằm tối ưu hóa hiệu quả học của mạng MLP bằng cách tự động điều chỉnh các siêu tham số. Quá trình này không những giúp cải thiện độ chính xác của mô hình, mà còn giảm thiểu thời gian thử sai khi lựa chon thủ công các tham số mạng.

Mô hình hoạt động theo cơ chế hai tầng: tầng tối ưu và tầng huấn luyện. Ở tầng tối ưu, thuật toán di truyền thực hiện tìm kiếm các tổ hợp siêu tham số phù hợp nhất thông qua quá trình tiến hóa. Tầng huấn luyện chịu trách nhiệm xây dựng và đánh giá các mô hình tương ứng với từng cá thể trong quần thể hiện tại.

Cấu trúc tổng quan của quá trình này được minh họa trong hình sau đây:



Hình 2.4: Kiến trúc mô hình GA-MLP

Các thành phần chính trong kiến trúc bao gồm:

* + - Tập dữ liệu đầu vào:

Dữ liệu tĩnh huấn luyện được chuẩn hóa và gán nhãn gồm các đặc trưng tĩnh được trích xuất từ ứng dụng Android. Tập dữ liệu được chia thành hai phần chính: tập huấn luyện, xác thực và kiểm tra, trong đó tập xác thực đóng vai trò quan trọng để đánh giá độ khái quát hóa và kiểm soát overfitting trong quá trình tiến hóa.

* + - Thuật toán di truyền:

GA khởi tạo một quần thể các cá thể ngẫu nhiên. Mỗi cá thể được mã hóa thành một nhiễm sắc thể chứa các siêu tham số:

* + - * Số lớp ẩn (num\_layers)
      * Số lượng nơ-ron ở mỗi lớp (neurons)
      * Tỷ lệ dropout của từng lớp (dropout\_rates)
      * Tốc độ học (learning\_rate)
      * Kích thước batch (batch\_size)
    - Xây dựng và huấn luyện MLP:

Từ mỗi cá thể, một mạng MLP được khởi tạo tương ứng và huấn luyện trên tập huấn luyện. Độ chính xác trên tập xác thực được sử dụng làm giá trị độ thích nghi (fitness value) để đánh giá cá thể.

* + - Chọn lọc – Lai ghép – Đột biến:

GA chọn lọc các cá thể có độ thích nghi cao để tạo thế hệ sau. Quá trình lai ghép và đột biến được áp dụng để tạo ra các nhiễm sắc thể con, nhằm đảm bảo sự đa dạng và khả năng khám phá của thuật toán.

* + - Tạo thế hệ mới và lặp lại:

Chu trình tiến hóa được lặp lại cho đến khi đạt số thế hệ tối đa hoặc giá trị thích nghi không cải thiện đáng kể. Cá thể có độ thích nghi cao nhất trong toàn bộ quá trình sẽ được chọn làm mô hình cuối cùng.

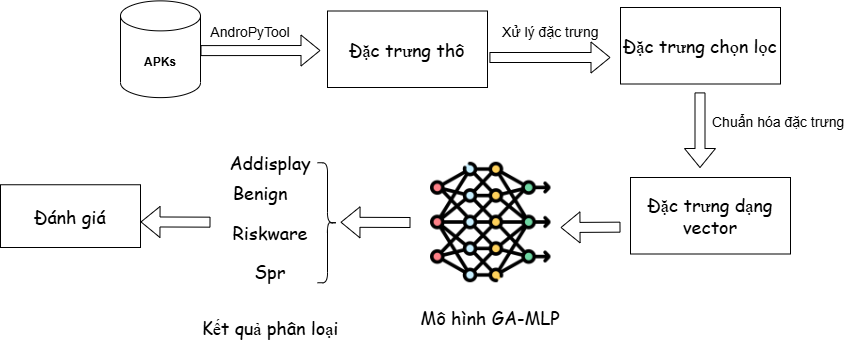
## Kết luận chương

Trong chương này, các kiến thức nền tảng phục vụ cho việc xây dựng mô hình học sâu kết hợp thuật toán di truyền đã được trình bày một cách hệ thống. Trước tiên, chương đã điểm qua các khái niệm về học máy và học sâu, đặt nền móng cho việc lựa chọn mô hình mạng nơ-ron nhiều lớp (MLP) làm công cụ chính trong bài toán phân loại. Tiếp theo, thuật toán di truyền, một phương pháp tối ưu hóa dựa trên cơ chế chọn lọc tự nhiên trong tiến hóa sinh học, đã được phân tích chi tiết. Các bước chính như khởi tạo, chọn lọc, lai ghép và đột biến được trình bày rõ ràng, cùng với những ưu điểm như khả năng khám phá không gian tham số rộng và tránh mắc kẹt tại các cực trị cục bộ.

Phần cuối của chương giới thiệu mô hình học sâu kết hợp với thuật toán di truyền. Cụ thể, mô hình sử dụng mạng nơ-ron nhiều lớp (MLP) để phân loại, trong khi thuật toán di truyền được áp dụng để tối ưu kiến trúc mạng và các siêu tham số của mạng học sâu. Việc tích hợp này không chỉ giúp nâng cao hiệu suất phân loại mà còn tự động hóa quá trình thiết kế mô hình – một quá trình thường đòi hỏi nhiều công sức và kinh nghiệm chuyên môn.

# THỰC NGHIỆM

## Sơ đồ tổng thể phân loại mã độc



Hình 3.1: Sơ đồ tổng quát quy trình thực nghiệm

## Bộ dữ liệu và đặc trưng

### Bộ dữ liệu

Đề tài thu thập và sử dụng bộ dữ liệu gồm các ứng dụng được phân loại thành các nhóm: Addisplay, Riskware, Benign, Spr. Mỗi nhóm chứa một số lượng mẫu nhất định, cụ thể như bảng dưới đây.

|  |  |
| --- | --- |
| Nhóm | Số lượng mẫu |
| Benign | 6000 |
| Addisplay | 3500 |
| Riskware | 3500 |
| Spr | 3500 |

Bảng 3.1: Số lượng mẫu các nhóm phân loại

### Đặc trưng

Đặc trưng là những thuộc tính quan trọng dùng để mô tả và phân biệt các mẫu trong bộ dữ liệu. Trong nghiên cứu này, các đặc trưng được trích xuất từ mã tĩnh của ứng dụng Android, chủ yếu thuộc các nhóm sau:

* + - Permissions: Là các quyền mà ứng dụng yêu cầu để truy cập vào các chức năng hệ thống Android, chẳng hạn như quyền truy cập mạng, đọc/ghi bộ nhớ ngoài, đặc biệt là các hành vi xâm phạm quyền riêng tư hoặc gây hại cho hệ thống.
    - Intents: Là cơ chế cho phép ứng dụng tương tác với hệ điều hành hoặc các ứng dụng khác. Các hành động như android.intent.action.MAIN hoặc android.intent.action.CREATE\_SHORTCUT có thể tiết lộ mục đích và cách thức hoạt động của ứng dụng.
    - Services: Là các thành phần chạy ngầm trong hệ thống Android, thường được sử dụng để thực hiện các tác vụ nền như kết nối mạng, gửi dữ liệu, hoặc tải về nội dung. Các dịch vụ có thể bị lợi dụng để duy trì hoạt động liên tục hoặc cài đặt mã độc bổ sung mà không cần tương tác từ người dùng.
    - API Calls: Ứng dụng Android sử dụng API để tương tác với hệ thống. Một số API có thể là dấu hiệu cho các hành vi nguy hiểm, như đọc danh bạ, gửi tin nhắn, hoặc truy xuất thông tin định vị. Do đó, tần suất và loại API được gọi có thể phản ánh rõ mức độ nguy hiểm của ứng dụng.
    - Receivers: Là các thành phần giúp ứng dụng tự động phản ứng với các sự kiện hệ thống, như BOOT\_COMPLETED (ứng dụng tự chạy khi khởi động thiết bị), SMS\_RECEIVED (nghe lén hoặc đánh cắp tin nhắn), CONNECTIVITY\_CHANGE (phát hiện khi có kết nối Internet để truyền dữ liệu). Việc ứng dụng đăng ký nhận các broadcast này có thể là dấu hiệu của hành vi độc hại.

Các đặc trưng trên giúp mô hình học sâu phân biệt giữa ứng dụng bình thường và ứng dụng độc hại dựa trên hành vi tiềm ẩn, là cơ sở quan trọng trong giai đoạn huấn luyện và đánh giá mô hình.

### Phương pháp trích xuất đặc trưng

Quá trình trích xuất đặc trưng từ bộ dữ liệu bao gồm các bước sau:

#### Tiền xử lý:

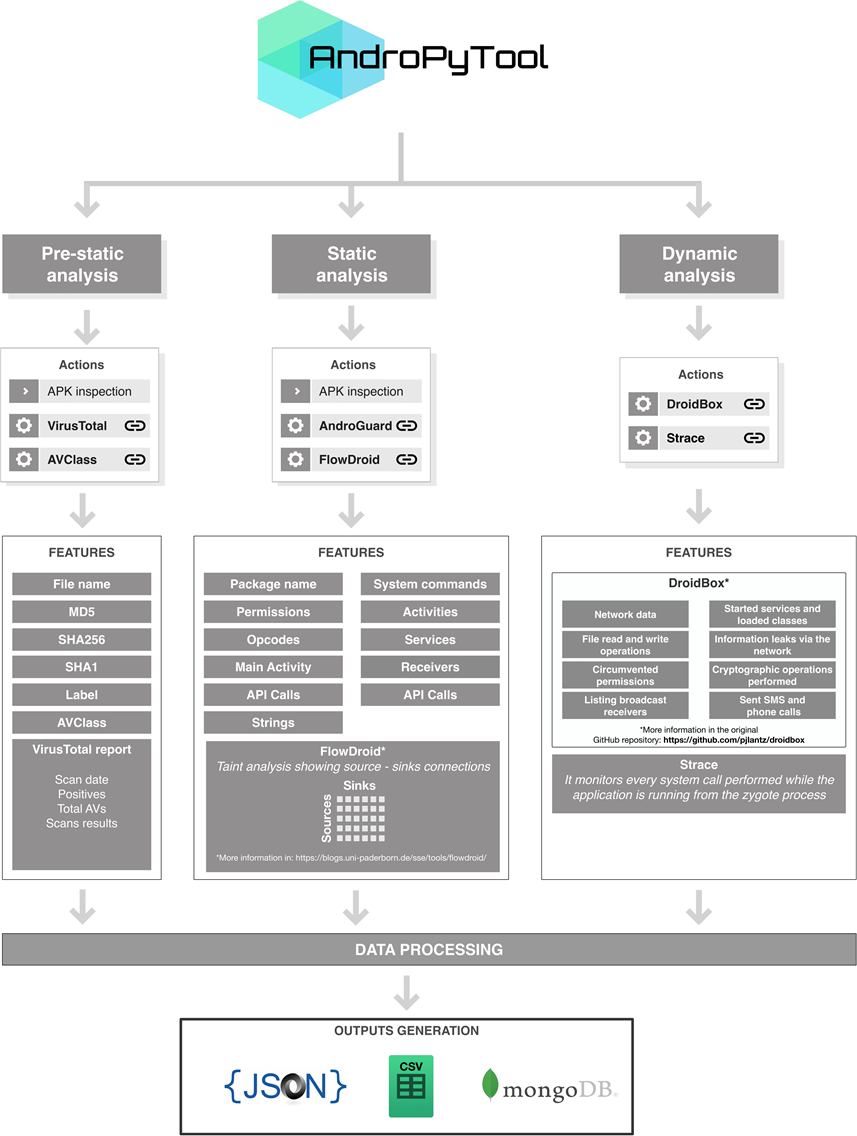
Trong quá trình này, các đặc trưng của ứng dụng Android sẽ được trích xuất bằng cách sử dụng công cụ AndroidPyTool. Đây là một công cụ mạnh mẽ dùng để trích xuất tự động các đặc trưng tĩnh và động của ứng dụng Android. Nó tích hợp một số công cụ phân tích ứng dụng nổi tiếng như DroidBox, FlowDroid, Strace và VirusTotal.

Trong nghiên cứu này, phiên bản với Docker của AndroidPyTool được sử dụng. Sau khi cài đặt Docker, công cụ được tải từ Docker Hub bằng lệnh:

Để trích xuất các đặc trưng tĩnh, thực hiện lệnh sau:

docker run --volume=<PATH>:/apks alexmyg/andropytool -s /apks -f -fw

Trong đó, <PATH> là đường dẫn chứa các tệp apk sẽ được phân tích. Kết quả được lưu vào thư mục Features\_files dưới dạng các tệp JSON, mỗi tệp chứa một thông tin đặc trưng của một nhóm ứng dụng.



Hình 3.2: Công cụ AndroPyTool

#### Phân tích các tệp JSON

Các tệp JSON chứa thông tin đặc trưng được đọc và xử lý lần lượt. Mỗi tệp JSON đại diện cho một nhóm ứng dụng. Để xác định nhóm của từng ứng dụng, tên của tệp JSON được sử dụng để phân loại ứng dụng vào các nhóm tương ứng.

Từ mỗi tệp JSON, các đặc trưng tĩnh được trích xuất bao gồm: Permissions, Intents, API Calls, Services, Receivers.

#### Lọc các đặc trưng phổ biến

Sau khi tổng hợp tất cả các đặc trưng từ các ứng dụng, top 500 phổ biến nhất từ mỗi nhóm đặc trưng (dựa trên tần suất xuất hiện) được lựa chọn để đảm bảo tính đại diện và giảm chiều dữ liệu đầu vào.

#### Gán nhãn ứng dụng

Mỗi ứng dụng được ánh xạ vào một nhãn số phục vụ cho bài toán phân loại bốn lớp. Cụ thể:

* + - Addisplay: nhãn 0
    - Benign: nhãn 1
    - Riskware: nhãn 2
    - Spr: nhãn 3

#### Chuẩn hóa đặc trưng

Sau khi xác định các đặc trưng phổ biến nhất, mỗi ứng dụng sẽ được biểu diễn dưới dạng các đặc trưng nhị phân. Mỗi đặc trưng sẽ tạo thành một côt trong bảng dữ liệu: Nếu đặc trưng xuất hiện trong ứng dụng sẽ được gán giá trị 1, còn không xuất hiện sẽ gán giá trị 0.

Tên cột được đặt theo quy ước như sau:

* + - perm\_<permission\_name>
    - action\_<action\_name>
    - service\_<service\_name>
    - api\_<api\_name>
    - receiver\_<receiver\_name>.

Ví dụ, nếu một ứng dụng có quyền INTERNET, cột tương ứng trong bảng dữ liệu sẽ là perm\_INTERNET. Nếu ứng dụng không yêu cầu quyền này, giá trị trong cột sẽ là 0, ngược lại nếu có thì giá trị sẽ là 1.

Việc chuẩn hóa tên các đặc trưng theo cách này không chỉ giúp quản lý dữ liệu dễ dàng hơn mà còn đảm bảo tính nhất quán và thuận tiện khi sử dụng trong các mô hình học máy hoặc các công cụ phân tích.

#### Kết quả và xuất dữ liệu

Sau khi trích xuất và chuẩn hóa đặc trưng, dữ liệu cuối cùng được tổ chức thành một bảng gồm:

* + - app\_id: Mã định danh ứng dụng
    - category: nhóm ứng dụng
    - label: nhãn số của nhóm ứng dụng
    - các đặc trưng nhị phân: mỗi đặc trưng (permission, intent, api call, service, receiver) được biểu diễn dưới dạng các cột nhị phân, thể hiện sự hiện diện hay vắng mặt của đặc trưng trong ứng dụng.

Bảng dữ liệu sau đó được lưu dưới dạng tệp CSV, sử dụng cho việc huấn luyện mô hình học sâu.

## Kịch bản thực nghiệm

### Thiết lập môi trường

Các mô hình được sử dụng trong kịch bản thử nghiệm được huấn luyện và đánh giá trong môi trường thực nghiệm như sau:

* + - Nền tảng: Google Colab
    - GPU sử dụng: NVIDIA A100 40GB
    - Hệ điều hành: Linux (Ubuntu-based, mặc định của Colab)
    - Ngôn ngữ lập trình: Python 3.10
    - Thư viện chính:
      * Học máy & học sâu: *Scikit-learn, TensorFlow, Keras*
      * Xử lý dữ liệu: *NumPy, Pandas*
      * Trực quan hóa: *Matplotlib, Seaborn*
      * Tối ưu hóa tiến hóa: *PyGad*

### Các tiêu chí đánh giá

Hiệu suất của các mô hình phân loại được đánh giá theo các tiêu chí sau:

#### Độ chính xác (accuracy):

Độ chính xác là tỷ lệ giữa số lượng mẫu được phân loại đúng và tổng số mẫu trong tập kiểm tra. Độ chính xác được tính theo công thức:

Trong đó:

* + - TP (True Positive): Số mẫu dự đoán đúng thuộc lớp dương.
    - TN (True Negative): Số mẫu dự đoán đúng thuộc lớp âm.
    - FP (False Positive): Mẫu thực tế không thuộc lớp cần dự đoán nhưng mô hình lại dự đoán sai là thuộc lớp đó.
    - FN (False Negative): Mẫu thực tế thuộc lớp cần phát hiện, nhưng mô hình lại dự đoán sai là không thuộc lớp đó.

#### Báo cáo phân loại (Classification Report):

Đây là một báo cáo tổng quan về hiệu suất của mô hình phân loại, thường được sử dụng để đánh giá chi tiết các chỉ số như độ chính xác (precision), độ phủ (recall), điểm f1 (f1-score) của mô hình trên từng lớp trong bài toán phân loại. Nó giúp người dùng có cái nhìn sâu sắc về cách mô hình hoạt động đối với từng lớp trong dữ liệu, thay vì chỉ dựa vào một chỉ số tổng quát như độ chính xác (accuracy). Cụ thể như sau:

* + Độ chính xác (Precision):

Chỉ số này phàn ánh tỷ lệ dự đoán đúng trong số các dự đoán thuộc lớp đó, giúp hiểu được độ tin cậy của mô hình khi dự đoán một lớp nhất định.

Công thức:

|  |  |
| --- | --- |
|  |  |

Trong đó:

* + - TP (True Positive): Số lượng mẫu thực sự thuộc về lớp và được mô hình dự đoán là lớp đó.
    - FP (False Positive): Số lượng mẫu không thuộc lớp nhưng lại được mô hình phân loại sai thành lớp đó.
* Độ phủ (Recall):

Cho biết mô hình có khả năng phát hiện được bao nhiêu mẫu của lớp đó. Một độ phủ cao nghĩa là mô hình ít bỏ sót các mẫu thuộc lớp đó.

Công thức:

|  |  |
| --- | --- |
|  |  |

* Điểm f1 (f1-score):

Là trung bình điều hòa của độ chính xác và độ phủ, giúp cân bằng giữa chúng khi muốn tối ưu hóa cả độ chính xác và độ phủ. Điểm f1 đặc biệt hữu ích trong trường hợp mất cân bằng giữa các lớp.

Công thức:

#### Ma trận nhầm lẫn (Confusion Matrix)

Đây là một công cụ quan trọng trong việc đánh giá hiệu suất của các mô hình phân loại, đặc biệt là trong các bài toán phân loại nhiều lớp. Nó cung cấp cái nhìn chi tiết về các lỗi mà mô hình có thể gặp phải khi phân loại các mẫu vào các lớp khác nhau. Ma trận thể hiện số lượng các mẫu mà mô hình phân loại đúng và sai cho từng lớp cụ thể, giúp phân tích và hiểu rõ hơn về các loại lỗi mà mô hình gặp phải.

Ví dụ, ma trận nhầm lẫn cho mô hình với bài toán phân loại bốn lớp với các nhãn tương ứng:

* + - Addisplay: nhãn 0
    - Benign: nhãn 1
    - Riskware: nhãn 2
    - Spr: nhãn 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Predicted 0 | Predicted 1 | Predicted 2 | Predicted 3 |
| Actual 0 | 50 | 5 | 10 | 2 |
| Actual 1 | 3 | 60 | 8 | 4 |
| Actual 2 | 2 | 6 | 70 | 5 |
| Actual 3 | 1 | 4 | 3 | 80 |

Bảng 3.2: Cấu trúc ma trận nhầm lần

### Các kịch bản thử nghiệm

#### Kịch bản 1: Phát hiện mã độc

Trong kịch bản phát hiện mã độc, các mô hình như SVM, RF, MLP, GA-MLP được sử dụng nhằm so sánh độ hiệu quả và chính xác trong việc phát hiện mã độc.

Bộ dữ liệu: Tổng số mẫu: 12000 ứng dụng

* + - * Addisplay: 2000
      * Bengin: 6000
      * Riskware: 2000
      * Spr: 2000

Tiền xử lý nhãn:

* Benign: nhãn 0
* Malware (gộp từ Addisplay, Riskware, Spr): nhãn 1
* Kịch bản 1.1: Mô hình GA-MLP
* Không gian tìm kiếm của bộ mã hóa (chromosome):

Mỗi cá thể trong quần thể được mã hóa dưới dạng một nhiễm sắc thể (chromosome) chứa các gene đại diện cho các siêu tham số của mạng MLP. Không gian tìm kiếm cụ thể như sau:

* + - Số lớp ẩn: từ 1 đến 4
    - Số lượng nơ-ron mỗi lớp: chọn từ [16, 32, 64, 128, 256]
    - Tỷ lệ dropout mỗi lớp: trong khoảng 0.1, 0.5
    - Learning rate: chọn từ [0.0005, 0.001, 0.005]
    - Batch size: chọn từ [16, 32, 64, 128]

Ví dụ về một cấu trúc một cá thể như sau:

{

‘num\_layers’: 3,

‘neurons’: [64, 128, 64],

‘dropout\_rates’: [0.3, 0.3, 0.2],

‘learning\_rate’: 0.001,

‘batch\_size’: 64

}

* Cấu hình thuật toán di truyền:
  + - Kích thước quần thể: 20 cá thể
    - Số thế hệ: 15
    - Tỷ lệ đột biến: 0.3
    - Elitism: giữ lại 3 cá thể tốt nhất qua mỗi thế hệ
    - Chiến lược chọn lọc: Tournament Selection (chọn ngẫu nhiên một nhóm nhỏ và lấy cá thể tốt nhất)
    - Lai ghép: các tham số cấu trúc được lai giữa hai cá thể cha mẹ
    - Đột biến: một số gene (tham số) có thể thay đổi ngẫu nhiên theo xác suất
  + Cơ chế đánh giá:
    - Mỗi cá thể (tương ứng một kiến trúc MLP cụ thể) được huấn luyện trên tập huấn luyện và được theo dõi quá trình học thông qua tập xác thực riêng biệt.
    - Giá trị độ chính xác trên tập xác thực được dùng làm giá trị độ thích nghi này cho thuật toán di truyền.
    - Early stopping được sử dụng trong quá trình huấn luyện nhằm tránh overfitting.
    - Sau khi thuật toán di truyền kết thúc, mô hình tốt nhất được đánh giá trên tập kiểm thử để tính toán độ chính xác thực tế của mô hình.
* Kịch bản 1.2: Mô hình ACO-MLP

Mô hình MLP kết hợp thuật toán đàn kiến có cấu hình như sau:

* + Cấu hình thuật toán tối ưu:
    - Số lượng kiến (n\_ants): 10
    - Số vòng lặp (n\_iterations): 10
    - evaporation\_rate: 0.3
    - Thông số ảnh hưởng pheromone 𝛼: 1.0
    - Thông số ảnh hưởng pheromone 𝛽: 2.0
  + Không gian tìm kiếm cấu hình mạng MLP:
    - Số lớp ẩn (num\_layers): chọn từ [1, 2, 3, 4]
    - Số lượng nơ-ron mỗi lớp: chọn từ [16, 32, 64, 128, 256]
    - Tỷ lệ dropout: chọn từ [0.1, 0.15, …, 0.5] với bước 0.05
    - Learning rate: chọn từ [0.0005, 0.001, 0.005, 0.01]
    - Batch size: chọn từ [16, 32, 64, 128]
  + Cấu hình huấn luyện:
    - Hàm mất mát: binary\_crossentropy
    - Hàm kích hoạt lớp đầu ra: sigmoid
    - Thuật toán tối ưu: Adam
    - Số epoch tối đa:15
    - Early stopping: dừng sớm khi val\_loss không cải thiện sau 3 epoch
* Kịch bản 1.3: MLP
* Mô hình được thiết kế với cấu trúc như sau:
  + - Layer 1: Dense 128 nơ-ron, ReLU, Batch Normalization, tỷ lệ dropout 0.3
    - Layer 2: Dense 64 nơ-ron, ReLU, Batch Normalization, tỷ lệ dropout 0.2
    - Layer 3: Dense 32 nơ-ron, ReLU, tỷ lệ dropout 0.2
    - Output Layer: 1 nơ-ron, sigmoid
* Cấu hình huấn luyện:
  + - Hàm mất mát: binary\_crossentropy
    - Thuật toán tối ưu: Adam, learning rate 0.001
    - Batch size: 32
    - Epochs: 45
    - Early stopping: theo dõi val\_loss, dừng nếu không cải thiện sau 5 epoch liên tiếp.
* Kịch bản 1.4: Mô hình học máy

Ở kịch bản thử nghiệm này, có 2 mô hình học máy được sử dụng là SVM và Random Forest.

* + Mô hình SVM được sử dụng với cấu hình như sau:
    - Kernel: rbf
    - C (Regularization parameter): 1.0 – tham số điều chỉnh độ chĩnh xác huấn luyện và độ khái quát
    - Gamma: scale
    - probability: true
    - random\_state: 42
  + Mô hình Random Forest được sử dụng với cấu hình sau:
    - n\_estimators: 100
    - max\_depth: 10
    - random\_state: 42

#### Kịch bản 2: Phân loại mã độc

Trong kịch bản phân loại mã độc, các mô hình học máy và học sâu được triển khai nhằm so sánh hiệu quả và độ chính xác trong việc phân loại các mẫu vào bốn nhóm cụ thể: Addisplay, Benign, Riskware và Spr.

Bộ dữ liệu gồm các tệp apk được phân bố đều theo từng nhóm:

* + - Addisplay: 3500
    - Benign: 3500
    - Riskware: 3500
    - Spr: 3500

Các nhãn được ánh xạ như sau:

* + - Addisplay: nhãn 0
    - Benign: nhãn 1
    - Riskware: nhãn 2
    - Spr: nhãn 3
* Kịch bản 2.1: Mô hình GA-MLP
  + Không gian tìm kiếm của cá thể:
    - Số lớp ẩn: chọn ngẫu nhiên từ 3 đến 6
    - Số lượng nơ-ron mỗi lớp: chọn từ tập giá trị [64, 128, 256, 512, 1024]
    - Tỷ lệ dropout ở mỗi lớp: chọn trong khoảng [0.2, 0.6]
    - Learning rate: chọn từ [0.0001, 0.0005, 0.001, 0.005, 0.01]
    - Batch size: chọn từ [32, 64, 128, 256]

Lớp đầu ra gồm 4 nơ-ron (tương ứng 4 lớp), sử dụng hàm kích hoạt softmax. Các lớp ẩn sử dụng ReLU, có thêm BatchNormalization và Dropout.

* + Cấu hình thuật toán di truyền
    - Kích thước quần thể: 20 cá thể
    - Số thế hệ: 15
    - Tỷ lệ đột biến: 0.3
    - Elitism: giữ nguyên 4 cá thể tốt nhất từ thế hệ trước
    - Cơ chế chọn lọc: Tournament Selection với 3 cá thể/tournament
    - Lai ghép: chọn ngẫu nhiên từng gene từ hai cá thể cha mẹ
    - Đột biến: một tham số (gene) có thể thay đổi ngẫu nhiên nếu rơi vào xác suất đột biến
  + Cơ chế đánh giá:
    - Mỗi cá thể (tương ứng một kiến trúc MLP cụ thể) được huấn luyện trên tập huấn luyện và được theo dõi quá trình học thông qua tập xác thực riêng biệt.
    - Giá trị độ chính xác trên tập xác thực được dùng làm giá trị độ thích nghi này cho thuật toán di truyền.
    - Early stopping được sử dụng trong quá trình huấn luyện nhằm tránh overfitting.
    - Sau khi thuật toán di truyền kết thúc, mô hình tốt nhất được đánh giá trên tập kiểm thử để tính toán độ chính xác thực tế của mô hình.
* Kịch bản 2.2: Mô hình ACO-MLP

Mô hình kết hợp MLP với thuật toán đàn kiến có cấu hình như sau:

* + Cấu hình tìm kiếm:
    - Số lượng kiến (ants): 10
    - Số vòng lặp (iterations): 10
    - evaporation\_rate: 0.3
    - Thông số pheromone 𝛼: 1.0
    - Thông số heuristic 𝛽: 2.0
  + Không gian tìm kiếm kiến trúc:
    - Số lớp ẩn: chọn từ [2, 3, 4, 5]
    - Số nơ-ron mỗi lớp: chọn từ [32, 64, 128, 256, 512]
    - Tỷ lệ dropout: chọn từ [0.2, 0.3, 0.4, 0.5, 0.6]
    - Learning rate: chọn từ [0.0001, 0.0005, 0.001, 0.005]
    - Batch size: chọn từ [32, 64, 128, 256]
  + Cấu hình huấn luyện
    - Hàm mất mát: categorical\_crossentropy
    - Thuật toán tối ưu: Adam
    - Số epoch: 15
    - Lớp đầu ra: Dense 4 nơ-ron, softmax
* Kịch bản 2.3: Mô hình MLP
  + Mô hình được thiết kế với cấu trúc các lớp như sau:
    - Layer 1: Dense 128 nơ-ron, ReLU, Batch Normalization, tỷ lệ dropout 0.3
    - Layer 2: Dense 64 nơ-ron, ReLU, Batch Normalization, tỷ lệ dropout 0.2
    - Layer 3: Dense 32 nơ-ron, ReLU, tỷ lệ dropout 0.2
    - Output Layer: Dense 4 nơ-ron, softmax
  + Cấu hình huấn luyện:
    - Hàm mất mát: categorical\_crossentropy
    - Thuật toán tối ưu: Adam
    - Learning rate: 0.001
    - Batch size: 32
    - Epochs: 45
    - Early stopping: theo dõi val\_loss, dừng nếu không cải thiện sau 5 epoch liên tiếp
* Kịch bản 2.4: Mô hình học máy

Hai mô hình học máy SVM và Random Forest sẽ được áp dụng ở kịch bản này.

* + Mô hình SVM có cấu hình như sau:
    - Kernel: rbf
    - C (Regularization parameter): 10.0
    - Gamma: auto
    - Probability: True
    - class\_weight: ‘balanced’
    - random\_state: 42
  + Mô hình Random Forest
    - n\_estimators: 100
    - max\_depth: 10
    - random\_state: 42

## Kết quả và đánh giá

### Kết quả

Sau khi tiến hành thực nghiệm các kịch bản khác nhau, kết quả thu được như sau:

#### Kịch bản 1: Phát hiện mã độc

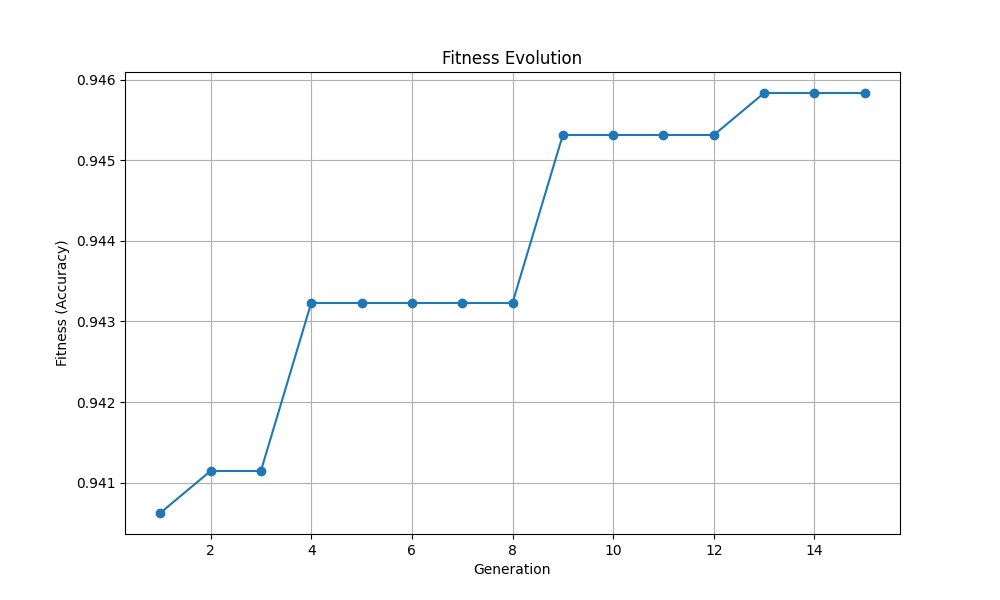
* Kịch bản 1.1: Mô hình GA-MLP
  + Quá trình cải thiện độ thích nghi qua các thế hệ
    - Ở đầu quá trình (thế hệ đầu), độ thích nghi có xu hướng tăng lên hoặc duy trì ở mức cao.
    - Giá trị độ thích nghi cao nhất qua các thế hệ dao động quanh khoảng 0.94 – 0.9458.
    - Sự cải thiện tuy không lớn nhưng cho thấy thuật toán đã ổn định và tìm được cấu hình tối ưu gần cực trị toàn cục.
  + Xu hướng tham số của các cá thể tốt
    - Số lớp: Hầu hết cá thể tốt chỉ sử dụng 1 lớp, số lớp nhiều hơn (3 lớp) không cải thiện đáng kể hoặc thậm chí thấp hơn.
    - Số nơ-ron: Các cấu hình nơ-ron phổ biến ở cá thể tốt là tập hợp gồm các giá trị như 32, 16, 256 nhưng đa phần nơ-ron tập trung vào các con số nhỏ và trung bình, ít dùng nơ-ron quá lớn cho các lớp chính.
    - Tỷ lệ dropout: Thường nằm trong khoảng từ 0.13 đến 0.43, phổ biến nhất là ~0.2-0.3, có thể thấy dropout đều được dùng để tránh overfitting nhưng không quá cao.
    - Learning rate: Hầu hêt được giữ cố định ở 0.0005, có thể do GA không tìm thấy learning rate khác hiệu quả hơn hoặc giới hạn trong phạm vi này.
    - Batch\_size: Chủ yếu là 32, một số ít cá thể batch size 128, nhưng cá thể tốt nhất đều dùng batch size 32.
  + Sự đa dạng và khám phá
    - Mặc dù có nhiều cá thể với các cấu hình khác nhau, tuy nhiên sự đa dạng tham số trong các cá thể tốt có xu hướng khá hạn chế.
    - Sự lặp lại các cá thể với các tham số nơ-ron và dropout tương tự nhau (ví dụ nhiều cá thể có nơ-ron [32, 32, 16, 32]) cho thấy GA có thể bị hội tụ sớm, tập trung vào khu vực tham số nhất định.
    - Learning rate và batch size ít đa dạng trong các cá thể tốt, cho thấy phạm vi khám phá các tham số này có thể bị giới hạn hoặc không đủ mở rộng.
  + Kết quả cuối cùng

Thuật toán GA hoàn thành quá trình tối ưu kiến trúc MLP với mô hình tốt nhất tìm thấy sau 15 thế hệ có độ chính xác là 94.46%. Cấu hình tốt nhất gồm 1 lớp ẩn với số lượng nơ-ron lần lượt là 32, 32, 16 và 32 trên các lớp. Tỷ lệ dropout ở các lớp lần lượt xấp xỉ 0.135, 0.304, 0.203, 0.330. Learning rate cố định ở mức 0.0005 và batch size là 32.

Mô hình đạt độ chính xác 94.46% trên tổng số 2400 mẫu là kết quả rất ấn tượng, chứng tỏ mô hình có khả năng phân biệt hiệu quả giữa ứng dụng bình thường và ứng dụng độc hại.

Chỉ số Precision và Recall giữa hai lớp được duy trì ở mức tương đương, điều này phản ánh sự cân bằng tốt của mô hình.

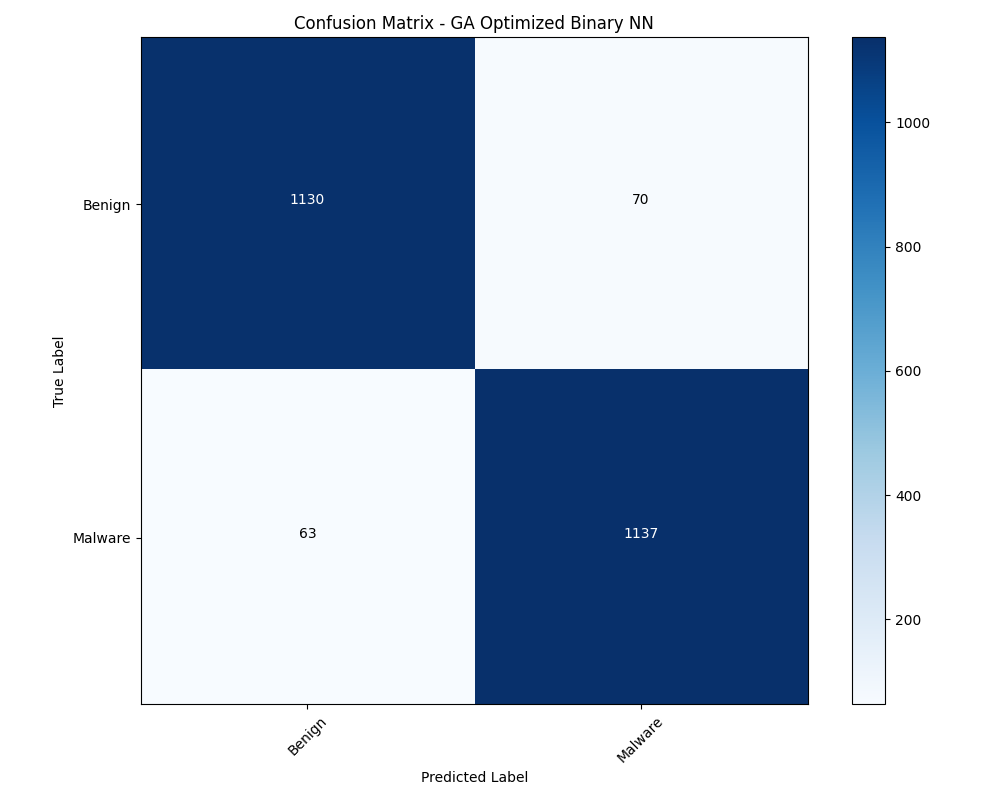
Sự đồng đều giữa Macro average và Weighted average cho thấy mô hình không chỉ hiệu quả trong tổng thể mà còn duy trì hiệu suất cân bằng giữa các lớp.



Hình 3.3: Biểu đồ tiến hóa thích nghi

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Benign | 0.95 | 0.94 | 0.94 | 1200 |
| Malware | 0.94 | 0.95 | 0.94 | 1200 |
| Accuracy |  |  | 0.94 | 2400 |
| Macro avg | 0.94 | 0.94 | 0.94 | 2400 |
| Weighted avg | 0.94 | 0.94 | 0.94 | 2400 |

Bảng 3.3: Báo cáo phân loại mô hình GA-MLP



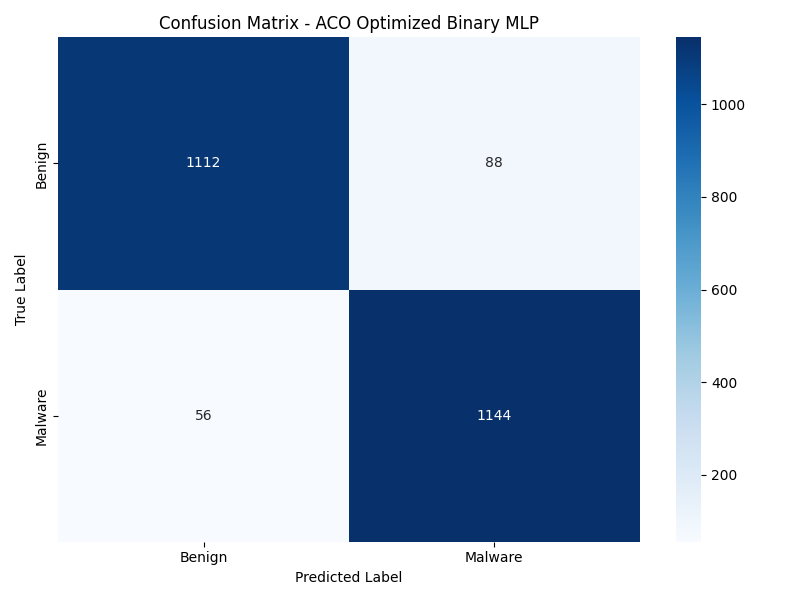
Hình 3.4: Ma trận nhầm lẫn mô hình GA-MLP

* Kịch bản 1.2: Mô hình ACO-MLP

Nhận xét: Mô hình cũng cho ra độ chính xác cao 94%, hiệu năng tốt trong bài toán phát hiện mã độc.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Benign | 0.95 | 0.93 | 0.94 | 1200 |
| Malware | 0.93 | 0.95 | 0.94 | 1200 |
| Accuracy |  |  | 0.94 | 2400 |
| Macro avg | 0.94 | 0.94 | 0.94 | 2400 |
| Weighted avg | 0.94 | 0.94 | 0.94 | 2400 |

Bảng 3.4: Báo cáo phân loại của mô hình ACO-MLP



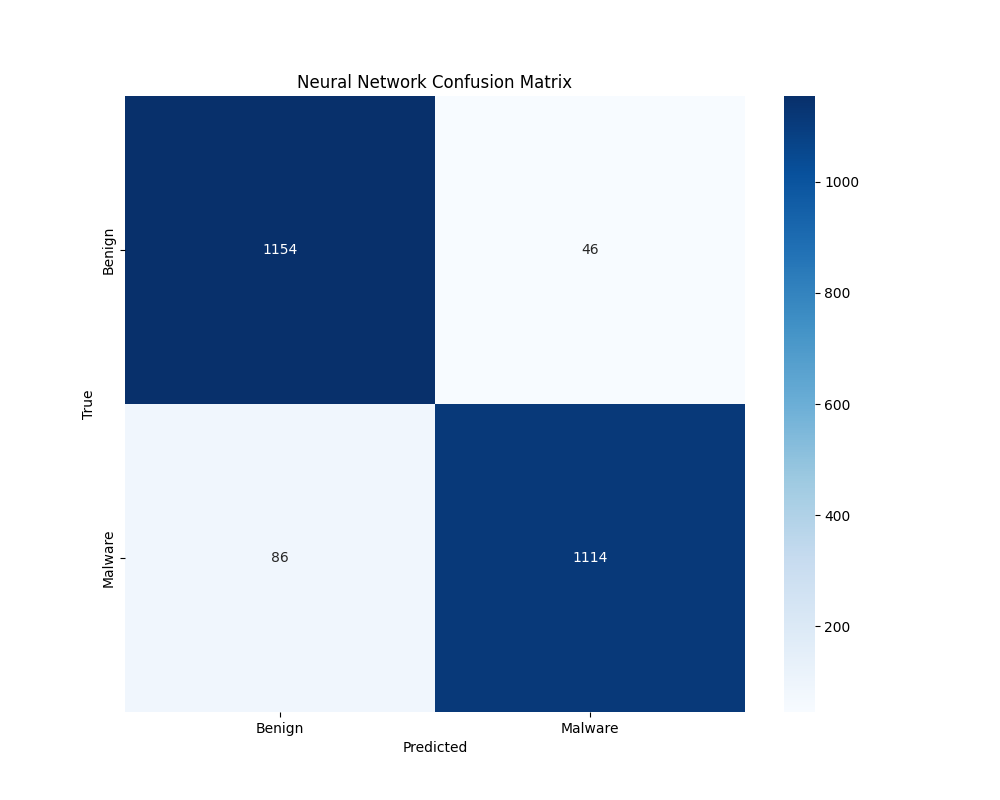
Hình 3.5: Ma trận nhầm lẫn của mô hình ACO-MLP

* Kịch bản 1.3: Mô hình MLP

Nhận xét: Mô hình này cho ra độ chính xác cao 94.5%, cho thấy khả năng phân loại dữ liệu rất tốt và gần như tương đương với hiệu suất của mô hình GA-MLP đã được tối ưu.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Benign | 0.93 | 0.96 | 0.95 | 1200 |
| Malware | 0.96 | 0.93 | 0.94 | 1200 |
| Accuracy |  |  | 0.94 | 2400 |
| Macro avg | 0.95 | 0.95 | 0.94 | 2400 |
| Weighted avg | 0.95 | 0.94 | 0.94 | 2400 |

Bảng 3.5: Báo cáo phân loại của mô hình MLP



Hình 3.6: Ma trận nhầm lẫn của mô hình MLP

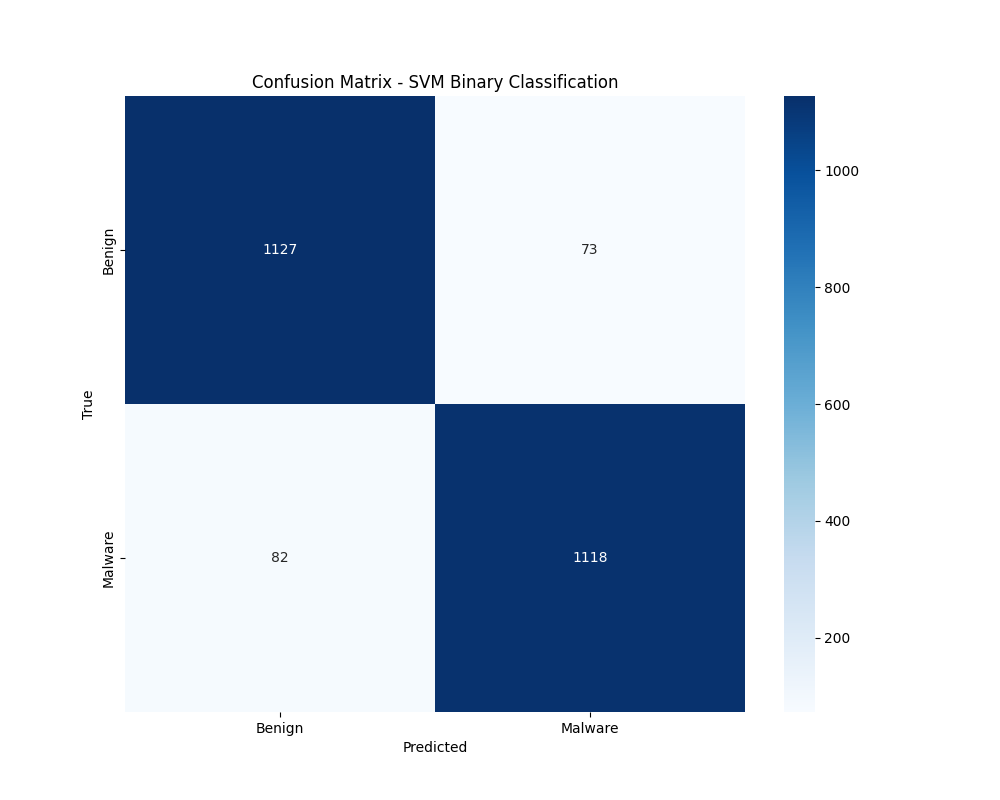
* Kịch bản 1.4: Mô hình học máy

Nhận xét:

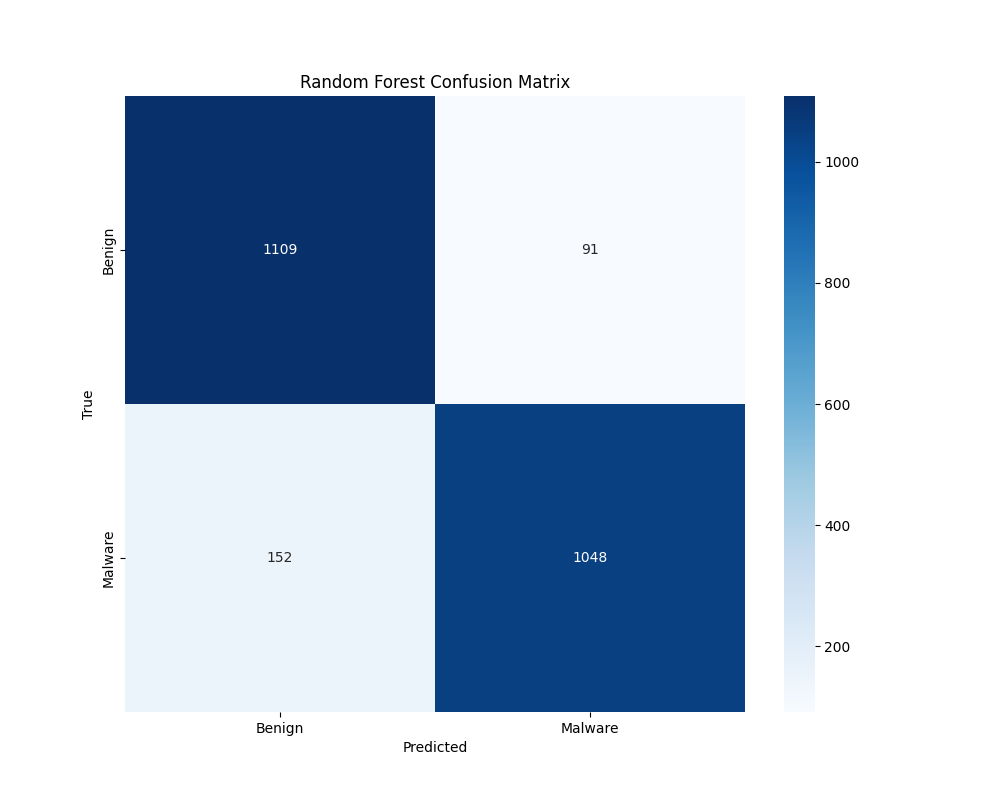
* + - Mô hình SVM đạt độ chính xác cao 93.54%, ổn định, gần như tương đương với các mô hình học sâu
    - RF đạt hiệu quả tốt 89.88%, kém hơn SVM và các mô hình học sâu

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Precision | Recall | F1-score | Support |
| SVM | Benign | 0.93 | 0.94 | 0.94 | 1200 |
| Malware | 0.94 | 0.93 | 0.94 | 1200 |
| Accuracy |  |  | 0.94 | 2400 |
| Macro avg | 0.94 | 0.94 | 0.94 | 2400 |
| Weighted avg | 0.94 | 0.94 | 0.94 | 2400 |
| RF | Benign | 0.88 | 0.92 | 0.90 | 1200 |
| Malware | 0.92 | 0.87 | 0.90 | 1200 |
| Accuracy |  |  | 0.90 | 2400 |
| Macro avg | 0.90 | 0.90 | 0.90 | 2400 |
| Weighted avg | 0.90 | 0.90 | 0.90 | 2400 |

Bảng 3.6: Báo cáo phân loại của các mô hình học máy



Hình 3.7: Ma trận nhầm lẫn mô hình SVM



Hình 3.8: Ma trận nhầm lẫn mô hình Random Forest

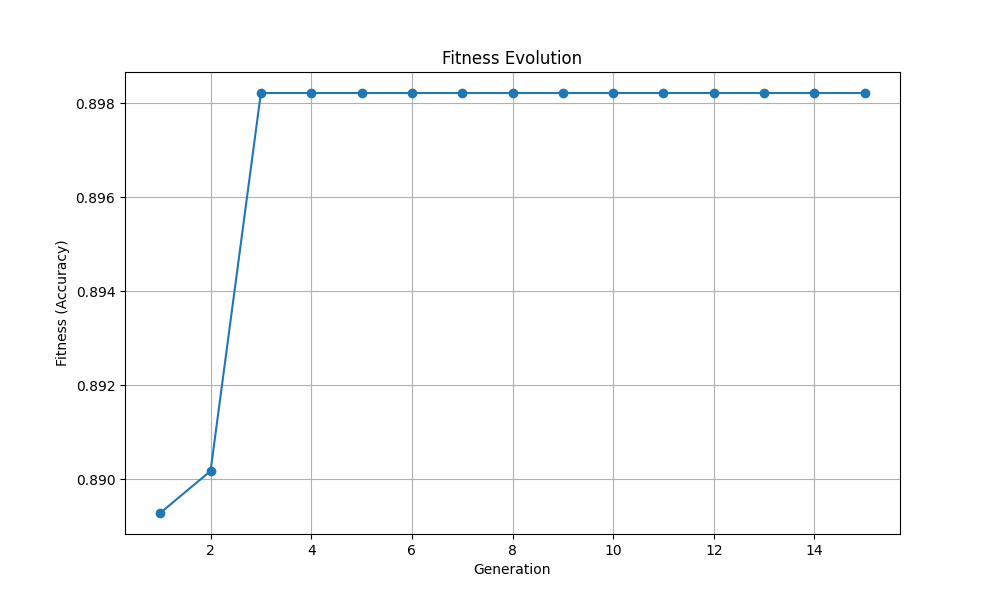
#### Kịch bản 2: Phân loại mã độc

* Kịch bản 2.1: Mô hình GA-MLP
  + Quá trình cải thiện độ thích nghi qua các thế hệ
    - Có sự cải thiện rõ rệt từ thế hệ đầu đến thế hệ cuối.
  + Xu hướng tham số của các cá thể tốt
    - Số lớp tốt nhất thường là 3 lớp (có một số cá thể 4 hoặc 6 lớp nhưng ít được chọn).
    - Tỷ lệ dropout thường dao động từ 0.3 đến 0.55, cho thấy dropout trung bình khá cao được ưu tiên để tránh overfitting.
    - Learning rate cố định ở 0.0005 và batch size thường là 256
  + Sự đa dạng và khám phá
    - Thuật toán vẫn giữ đa dạng tham số như số lớp, số nơ-ron, dropout cho phép khám phá không gian tham số rộng
    - Qua từng thế hệ, các cá thể có sự thay đổi nhẹ ở dropout hoặc nơ-ron.
  + Kết quả cuối cùng

Thuật toán GA hoàn thành với độ chính xác tốt nhất đạt 88.71%, với cấu hình tối ưu được tìm thấy sau 15 thế hệ.

Cấu hình tốt nhất được tìm thấy như sau:

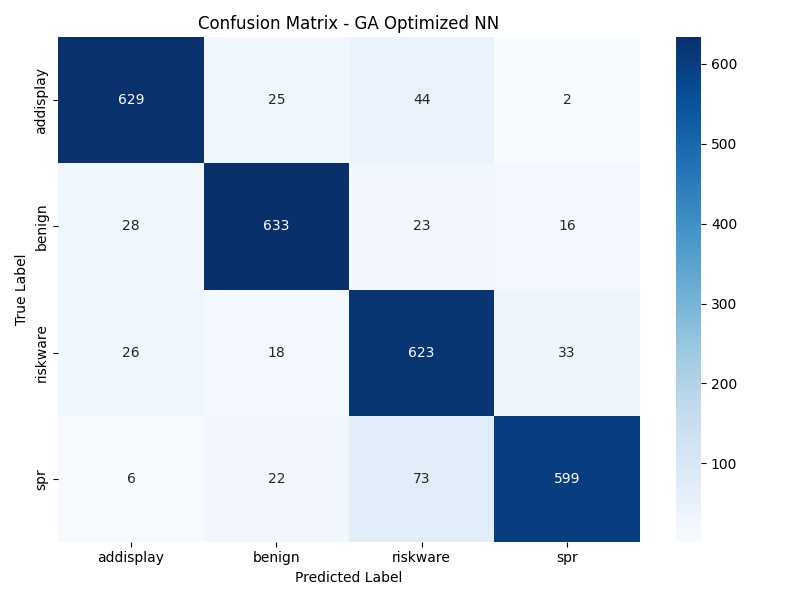
* + - * 3 lớp ẩn
      * Số lượng nơ-ron lần lượt là 512, 128, 512, 1024, 128 và 64 nơ-ron trên các lớp.
      * Tỷ lệ dropout ở các lớp dao động lần lượt xấp xỉ 0.477, 0.341, 0.394, 0.326, 0.368, 0.425
      * Learning rate: 0.0005
      * Batch size: 256.
  + Nhận xét:
    - Độ chính xác đạt 88.71% trên tập kiểm tra gồm 2800 mẫu, là một kết quả tốt cho bài toán phân loại 4 lớp.
    - Marco và Weighted avg cho Precision, Recall, F1-score đều ở mức 0.89 cho thấy mô hình có hiệu năng đồng đều trên các lớp, có khả năng tổng quát hóa tốt với phân phối dữ liệu khá cân bằng.
    - Riskware là lớp yếu nhất, với Precision thấp nhất (0.82). Có thể do đặc trưng giữa Riskware và các nhóm khác không rõ ràng, cần xem lại đặc trưng hoặc bổ sung thêm đặc trưng phân biệt tốt hơn.
    - Recall thấp hơn Precision ở lớp Spr, có thể do mô hình bỏ sót một số mẫu Spr. Do đó cần cân nhắc xử lý mất cân bằng dữ liệu hoăc điều chỉnh hàm mất mát phù hợp hơn.



Hình 3.9: Biểu đồ tiến hóa thích nghi

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Addisplay | 0.91 | 0.90 | 0.91 | 700 |
| Benign | 0.91 | 0.90 | 0.91 | 700 |
| Riskware | 0.82 | 0.89 | 0.85 | 700 |
| Spr | 0.92 | 0.86 | 0.89 | 700 |
| Accuracy |  |  | 0.89 | 2800 |
| Macro avg | 0.89 | 0.89 | 0.89 | 2800 |
| Weighted avg | 0.89 | 0.89 | 0.89 | 2800 |

Bảng 3.7: Báo cáo phân loại mô hình GA-MLP



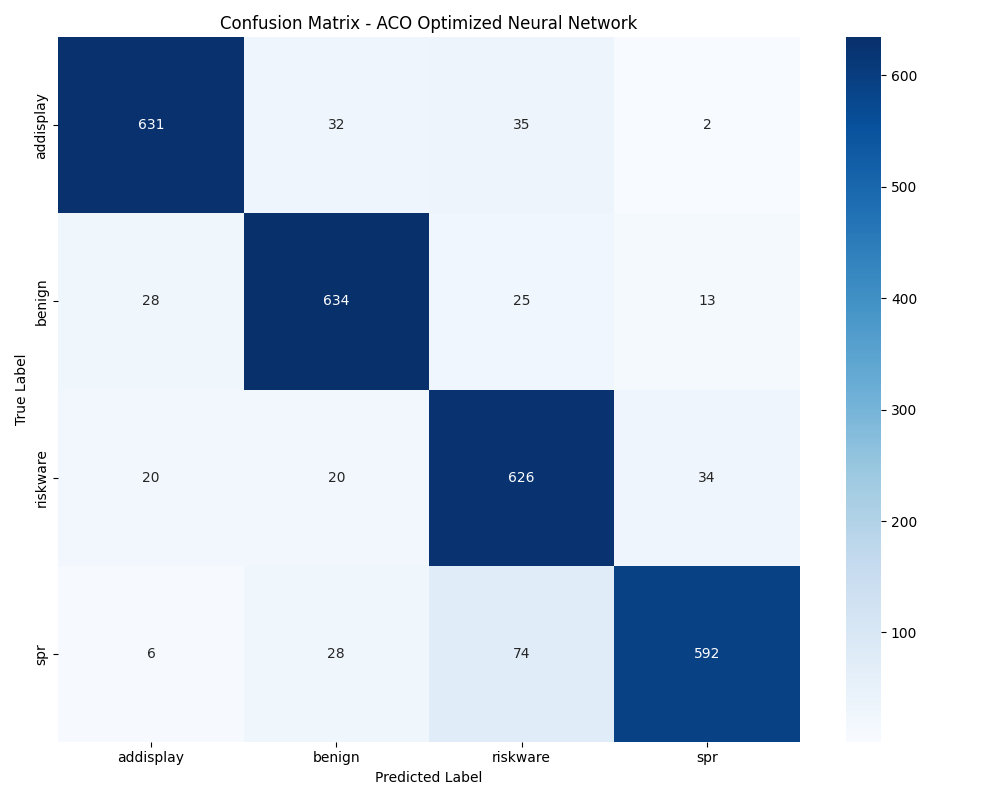
Hình 3.10: Ma trận nhầm lẫn mô hình GA-MLP

* Kịch bản 2.2: Mô hình ACO-MLP

Nhận xét: Mô hình đạt độ chính xác 88.68%, là mức tương đối cao. Các chỉ số đều đạt 0.89 ở mức macro average và weighted average cho thấy mô hình hoạt động ổn định và nhất quán giữa các lớp.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Addisplay | 0.92 | 0.90 | 0.91 | 700 |
| Benign | 0.89 | 0.91 | 0.90 | 700 |
| Riskware | 0.82 | 0.89 | 0.86 | 700 |
| Spr | 0.92 | 0.85 | 0.88 | 700 |
| Accuracy |  |  | 0.89 | 2800 |
| Macro avg | 0.89 | 0.89 | 0.89 | 2800 |
| Weighted avg | 0.89 | 0.89 | 0.89 | 2800 |

Bảng 3.8: Báo cáo phân loại mô hình ACO-MLP



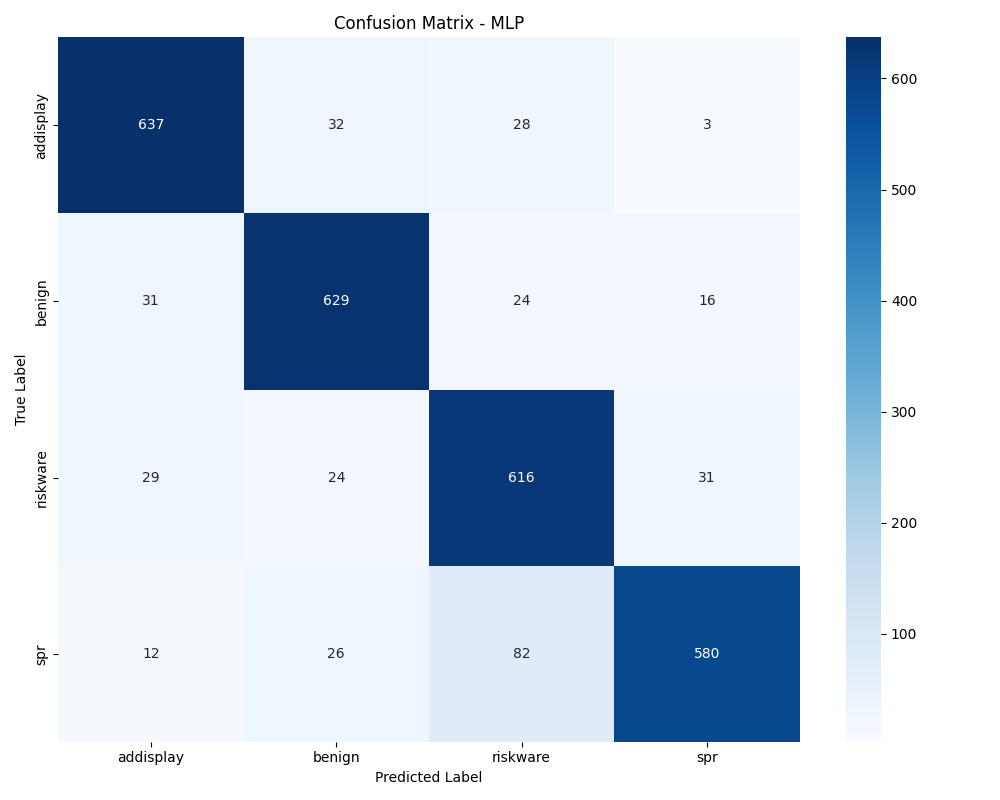
Hình 3.11: Ma trận nhầm lẫn của mô hình ACO-MLP

* Kịch bản 2.3: Mô hình MLP

Nhận xét: Độ chính xác của mô hình đạt 87.93%

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Addisplay | 0.90 | 0.91 | 0.90 | 700 |
| Benign | 0.88 | 0.90 | 0.89 | 700 |
| Riskware | 0.82 | 0.88 | 0.85 | 700 |
| Spr | 0.92 | 0.83 | 0.87 | 700 |
| Accuracy |  |  | 0.88 | 2800 |
| Macro avg | 0.88 | 0.88 | 0.88 | 2800 |
| Weighted avg | 0.88 | 0.88 | 0.88 | 2800 |

Bảng 3.9: Báo cáo phân loại mô hình MLP



Hình 3.12: Ma trận nhầm lẫn của mô hình MLP

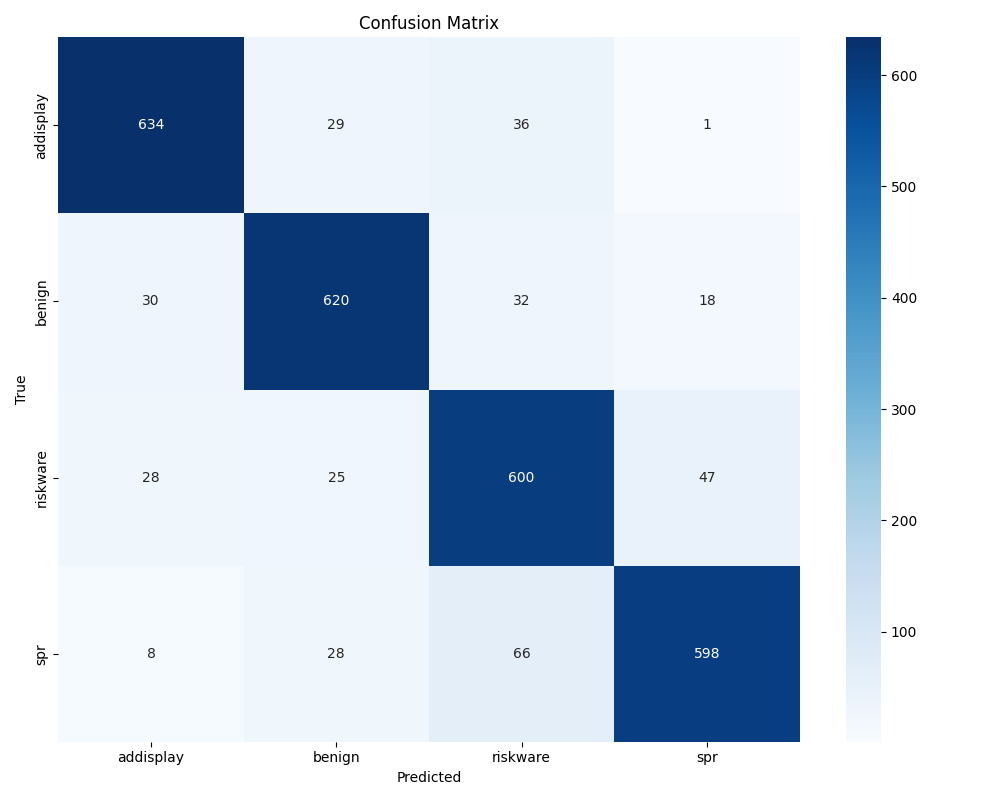
* Kịch bản 2.4: Mô hình học máy

Nhận xét:

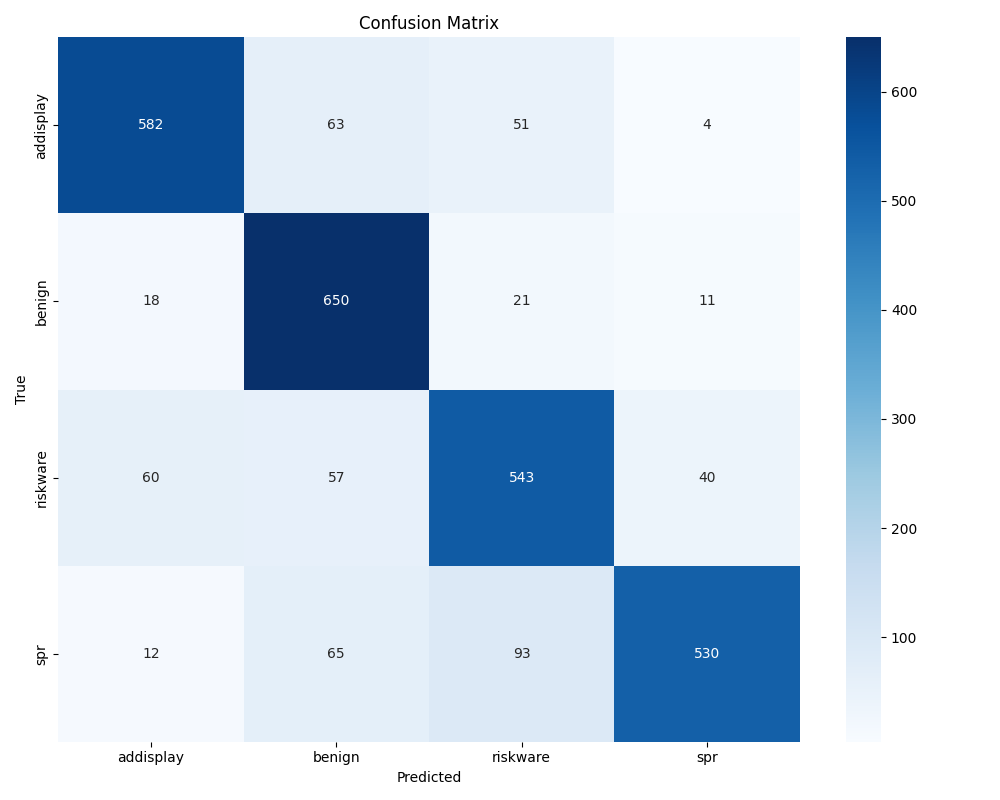
* + - Mô hình SVM đạt độ chính xác 87.57%, RF đạt 82.32%.
    - SVM thể hiện sự cân bằng giữa các lớp, hiệu suất cao và ổn định giữa các lớp, hiệu suất cao.
    - RF cho thấy độ chính xác kém hơn, đặc biệt là mất cân bằng precision, recall giữa các lớp, dễ gây lỗi nghiêm trọng trong ứng dụng thực tế nếu không điều chỉnh tốt.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Precision | Recall | F1-score | Support |
| SVM | Addisplay | 0.91 | 0.91 | 0.91 | 700 |
| Benign | 0.88 | 0.89 | 0.88 | 700 |
| Riskware | 0.82 | 0.86 | 0.84 | 700 |
| Spr | 0.90 | 0.85 | 0.88 | 700 |
| Accuracy |  |  | 0.88 | 2800 |
| Macro avg | 0.88 | 0.88 | 0.88 | 2800 |
| Weighted avg | 0.88 | 0.88 | 0.88 | 2800 |
| RF | Addisplay | 0.87 | 0.83 | 0.85 | 700 |
| Benign | 0.78 | 0.93 | 0.85 | 700 |
| Riskware | 0.77 | 0.78 | 0.77 | 700 |
| Spr | 0.91 | 0.76 | 0.82 | 700 |
| Accuracy |  |  | 0.82 | 2400 |
| Macro avg | 0.83 | 0.82 | 0.82 | 2400 |
| Weighted avg | 0.83 | 0.82 | 0.82 | 2400 |

Bảng 3.10: Báo cáo phân loại của mô hình học máy



Hình 3.13: Ma trận nhầm lẫn mô hình SVM



Hình 3.14: Ma trận nhầm lẫn mô hình Random Forest

### Đánh giá

Dựa trên kết quả thực nghiệm được trình bày ở trên, có thể đưa ra một số nhận xét và đánh giá tổng quan về hiệu quả của các mô hình trong các kịch bản thử nghiệm khác nhau như sau:

* + Đối với bài toán phát hiện mã độc:
    - Ở kịch bản này, cả ba mô hình học sâu (GA-MLP, ACO-MLP và MLP) đều đạt độ chính xác rất cao, dao động quanh mức 94%. Cụ thể, mô hình MLP đạt độ chính xác cao nhất là 94.5%, nhỉnh hơn GA-MLP (94.46%) và ACO-MLP (94%). GA-MLP chỉ thấp hơn MLP một chút, cho thấy thuật toán di truyền đã tìm được cấu hình gần tối ưu. Tuy nhiên, sự chênh lệch giữa các mô hình này là không đáng kể, cho thấy cả ba đều có khả năng nhận diện tốt giữa hai lớp Benign và Malware.
    - So với các mô hình học máy truyền thống như SVM và Random Forest, các mô hình học sâu đều cho khả năng vượt trội hơn. SVM đạt độ chính xác 93.54%, trong khi Random Forest chỉ đạt 89.88%. Điều này cho thấy việc sử dụng các mô hình học sâu, đặc biệt là khi kết hợp với thuật toán tối ưu như GA hoặc ACO, có thể mang lại hiệu quả tốt hơn trong việc phát hiện mã độc.
  + Với bài toán phân loại mã độc:
    - Đối với bài toán phân loại chi tiết các loại mã độc, độ chính xác của các mô hình nhìn chung giảm so với bài toán phát hiện, do độ phức tạp và số lớp phân loại tăng lên. Trong các mô hình học sâu, GA-MLP đạt kết quả tốt nhất với độ chính xác là 88.71%, mô hình này cũng thể hiện hiệu năng ổn định hơn với các chỉ số precision, recall và f1-score tương đối cân bằng ở cả bốn lớp.
    - Các mô hình học sâu khác như ACO-MLP và MLP đạt độ chính xác lần lượt là 88.68% và 87.93% cũng cho thấy hiệu năng ổn định và đồng đều.
    - Các mô hình học máy truyền thống một lần nữa cho kết quả thấp hơn, trong đó SVM đạt độ chính xác 87.57%, còn Random Forest chỉ đạt 82.32%. Đặc biệt, Random Forest cho thấy sự mất cân bằng rõ rệt giữa các lớp, thể hiện qua các chỉ số precision và recall không đồng đều, đặc biệt đối với lớp Benign và Spr.

Những kết quả trên cho thấy mô hình MLP cơ bản đã đạt hiệu quả khá tốt trong cả hai bài toán. Tuy nhiên, việc tích hợp các thuật toán tối ưu như GA hay ACO giúp cải thiện hiệu suất tổng thể của hệ thống, đặc biệt trong việc tăng cường khả năng tìm kiếm và tự động tối ưu trọng số trong quá trình huấn luyện mạng học sâu.

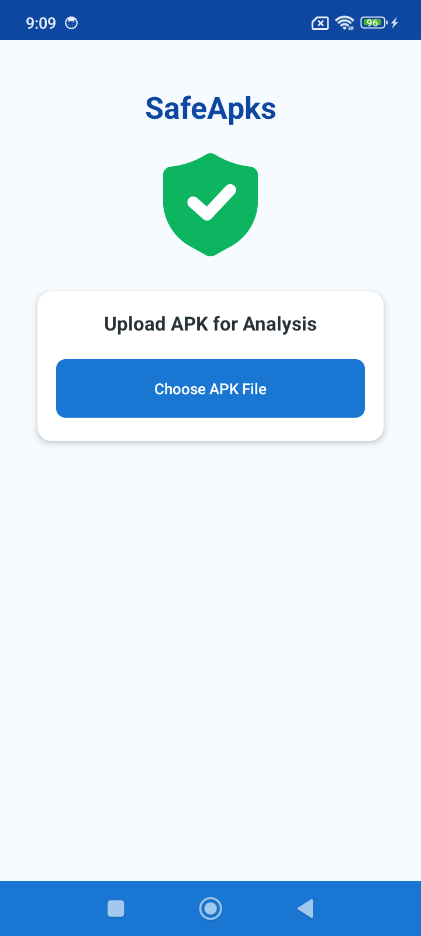
Mô hình GA-MLP nổi bật hơn khi đạt kết quả cao và ổn định ở cả hai kịch bản phát hiện và phân loại mã độc. Điều này khẳng định rằng việc sử dụng thuật toán di truyền để tối ưu hóa tham số cho mạng học sâu là hướng đi hiệu quả trong các bài toán phân tích và phát hiện mã độc trên nền tảng Android.

## Chương trình thực nghiệm

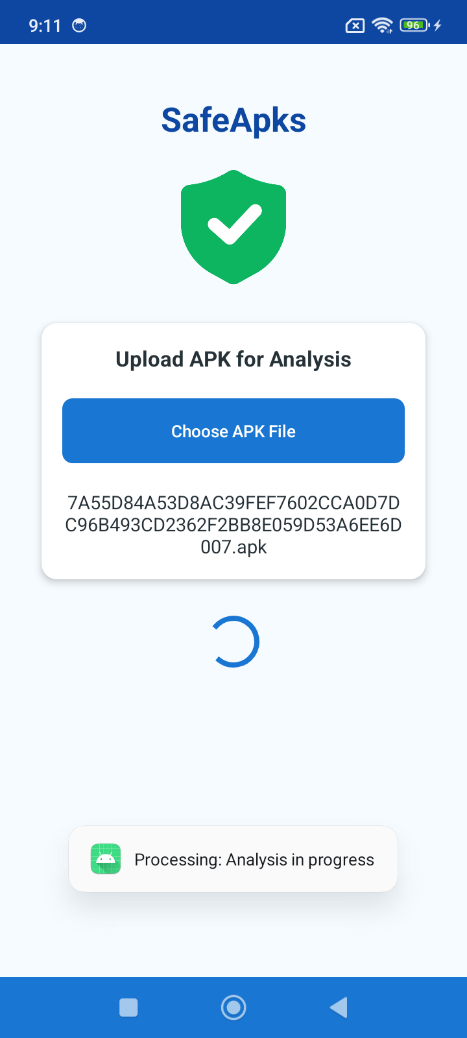
Sau khi được huấn luyện, mô hình GA-MLP theo kịch bản phân loại mã độc được tích hợp vào một ứng dụng di động đơn giản nhằm thực hiện phân loại các tệp apk. Ứng dụng cho phép người dùng chọn tệp apk từ thiết bị cần phân tích, sau đó hiển thị kết quả phân loại dựa trên mô hình đã huấn luyện.

Quy trình sử dụng được thiết kế đơn giản và thuận tiện: người dùng chọn tệp apk từ bộ nhớ thiết bị, sau đó hệ thống sẽ trích xuất đặc trưng tĩnh, đưa vào mô hình GA-MLP đã được huấn luyện và hiển thị kết quả phân loại. Kết quả trả về bao gồm phân loại ứng dụng, mức cảnh báo tương ứng, cùng với các thông tin kỹ thuật liên quan như mã băm MD5 và SHA-256 của tệp.

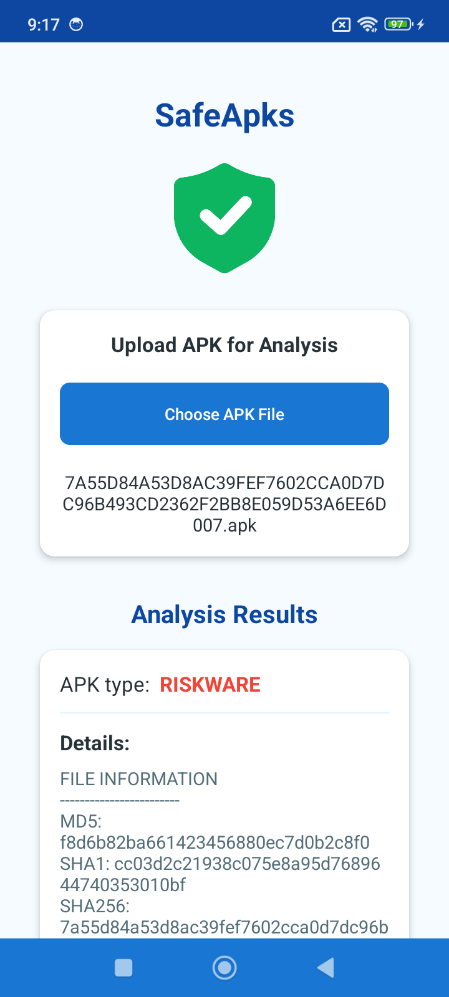
Dưới đây là một số hình ảnh giao diện ứng dụng minh họa quá trình hoạt động:



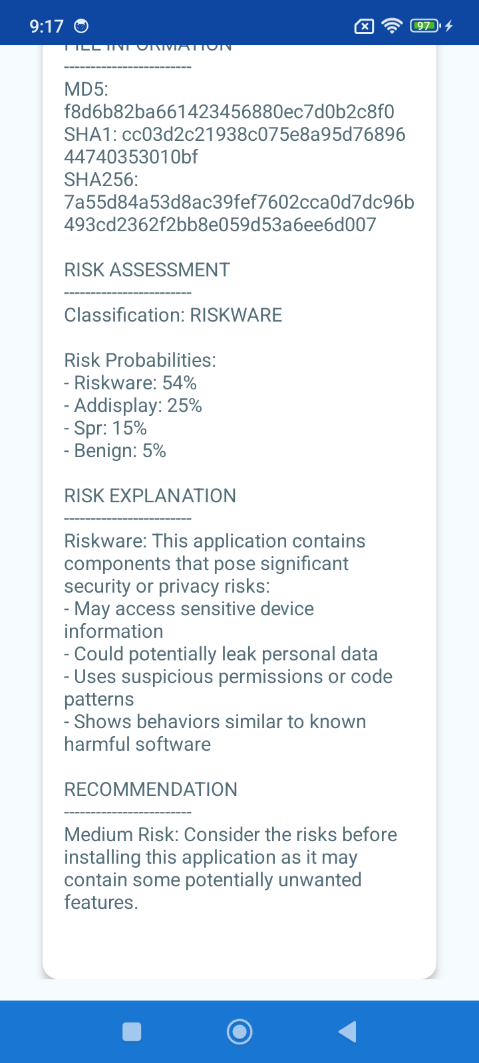
Hình 3.15: Màn hình ứng dụng lúc khởi chạy



Hình 3.16: Ứng dụng khi đang phân tích tệp apk



Hình 3.17: Ứng dụng hiển thị kết quả



Hình 3.18: Ứng dụng hiển thị kết quả (tiếp)

## Kết luận chương

Trong chương này, các thí nghiệm đã được tiến hành nhằm đánh giá hiệu quả của mô hình GA-MLP. Kết quả thực nghiệm cho thấy mô hình này đạt độ chính xác cao và ổn định, vượt trội so với các mô hình học máy truyền thống như SVM, Random Forest đồng thời cho hiệu năng tương đương hoặc nhỉnh hơn so với các mô hình học sâu khác như ACO-MLP và MLP.

Cụ thể, mô hình GA-MLP không chỉ đạt kết quả tốt trong bài toán phát hiện mã độc mà còn thể hiện ở khả năng phân loại đa lớp với độ chính xác 89.32%. Việc tích hợp mô hình vào một ứng dụng di động thực nghiệm cũng chứng minh tính khả thi trong triển khai thực tế.

Một điểm mạnh nổi bật của GA-MLP là khả năng tự động tìm kiếm cấu hình mạng tối ưu bao gồm số lớp ẩn, số lượng nơ-ron mỗi lớp, tỷ lệ dropout, tốc độ học và kích thước batch. GA đảm bảo sự cân bằng giữa việc khai thác các tham số tốt hiện tại và khám phá không gian siêu tham số mới, từ đó giúp mô hình tránh hiện tượng overfitting và đạt được độ khái quát tốt hơn.

Ngoài ra, chương cũng đã trình bày việc triển khai mô hình vào ứng dụng di động, cho thấy tính khả thi của việc tích hợp mô hình học sâu vào môi trường thực tế, với khả năng phát hiện mã độc trực tiếp từ thiết bị của người dùng. Kết quả này mở ra tiềm năng ứng dụng lớn trong lĩnh vực bảo mật di động, đặc biệt trong bối cảnh số lượng phần mềm độc hại trên Android ngày càng gia tăng và tinh vi hơn.

# KẾT LUẬN

Mô hình phát hiện mã độc Android dựa trên mô hình học sâu kết hợp với thuật toán di truyền để tối ưu hóa cấu trúc và các siêu tham số của mô hình đã được nghiên cứu và triển khai. Mục tiêu chính là xây dựng một hệ thống phát hiện mã độc có độ chính xác cao, khả năng tổng quát hóa tốt, đồng thời giảm thiểu chi phí thử nghiệm thủ công trong quá trình lựa chọn kiến trúc mạng.

Thông qua quá trình thực nghiệm, mô hình đề xuất đã cho thấy hiệu quả rõ rệt trong việc phát hiện mã độc, đặc biệt là: độ chính xác cao, tự động hóa, khả năng mở rộng.

Bên cạnh những kết quả đạt được, đồ án vẫn còn một số hạn chế nhất định như: thời gian huấn luyện tương đối dài do sử dụng GA kết hợp mô hình học sâu, mô hình mới đang dừng lại ở phân tích đặc trưng tĩnh, chưa bao gồm dữ liệu động.

Trong thời gian tới, nghiên cứu có thể được mở rộng theo các hướng sau:

* Tích hợp đặc trưng động từ hành vi thời gian thực của ứng dụng để nâng cao độ chính xác.
* Kết hợp nhiều thuật toán tối ưu như PSO, Bayesian Optimization để cải thiện hiệu suất tìm kiếm siêu tham số.
* Triển khai mô hình trong môi trường thực tế, ví dụ như tích hợp vào hệ thống bảo mật của Android hoặc ứng dụng kiểm tra APK.
* Nghiên cứu giảm thiểu chi phí tính toán.

# TÀI LIỆU THAM KHẢO

[1] Sherif, A. (2025) Mobile OS Market Share Worldwide 2009-2025, Statista. Available at: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009 (Accessed: 11 April 2025).

[2] Smith, G. (2024) +65 malware statistics for 2025, StationX. Available at: https://www.stationx.net/malware-statistics/ (Accessed: 25 May 2025).

[3] Play protect  |  google for developers (no date a) Google. Available at: https://developers.google.com/android/play-protect (Accessed: 25 May 2025).

[4] Li, L. et al. (2017) ‘Static Analysis of Android Apps: A systematic literature review’, Information and Software Technology, 88, pp. 67–95. doi:10.1016/j.infsof.2017.04.001.

[5] Martín, A., Lara-Cabrera, R., & Camacho, D. (2018). A new tool for static and dynamic Android malware analysis. In Data Science and Knowledge Engineering for Sensing Decision Support (pp. 509-516). World Scientific.

[6] Martín, A., Lara-Cabrera, R., & Camacho, D. (2018). Android malware detection through hybrid features fusion and ensemble classifiers: the AndroPyTool framework and the OmniDroid dataset. Information Fusion. DOI: 10.1016/j.inffus.2018.12.006

[7] What is machine learning? key concepts and real-world uses (2025) iSchool. Available at: https://ischool.syracuse.edu/what-is-machine-learning/ (Accessed: 25 May 2025).

[8] Brown, S. (2021) Machine Learning, explained, MIT Sloan. Available at: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained (Accessed: 25 May 2025).

[9] Carr, J., 2014. An introduction to genetic algorithms. Senior Project, 1(40), p.7.