**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**
**KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

**PHAN NGỌC YẾN NHI**
**HOÀNG THANH LÂM**

**KHÓA LUẬN TỐT NGHIỆP**

# PHƯƠNG PHÁP TẠO BIẾN THỂ MÃ ĐỘC WINDOWS DỰA TRÊN HỌC TĂNG CƯỜNG CÓ KIỂM CHỨNG CHỨC NĂNG BẰNG SO SÁNH TƯƠNG ĐỒNG NHỊ PHÂN

## A METHOD OF WINDOWS MALWARE MUTATION USING REINFORCEMENT LEARNING WITH BINARY CODE SIMILARITY DETECTION-BASED FUNCTIONALITY VALIDATION

**CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN**

**TP. HỒ CHÍ MINH, 2024**

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

**PHAN NGỌC YẾN NHI – 20521717**

**HOÀNG THANH LÂM – 20521513**

**KHÓA LUẬN TỐT NGHIỆP**

# PHƯƠNG PHÁP TẠO BIẾN THỂ MÃ ĐỘC WINDOWS DỰA TRÊN HỌC TĂNG CƯỜNG CÓ KIỂM CHỨNG CHỨC NĂNG BẰNG SO SÁNH TƯƠNG ĐỒNG NHỊ PHÂN

## A METHOD OF WINDOWS MALWARE MUTATION USING REINFORCEMENT LEARNING WITH BINARY CODE SIMILARITY DETECTION-BASED FUNCTIONALITY VALIDATION

**CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN**

**GIẢNG VIÊN HƯỚNG DẪN**

**THS. TRẦN THỊ DUNG**

**THS. ĐỖ THỊ THU HIỀN**

**TP. HỒ CHÍ MINH, 2024**

## THÔNG TIN HỘI ĐỒNG CHẤM KHÓA LUẬN TỐT NGHIỆP

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số …… ngày …….. của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

# LỜI CẢM ƠN

Để hoàn thành khóa luận tốt nghiệp này, chúng tôi xin gửi lời cảm ơn chân thành đến Ban giám hiệu Trường Đại học Công nghệ Thông tin – Đại học Quốc Gia Thành Phố Hồ Chí Minh vì đã tạo điều kiện học tập, nghiên cứu tốt nhất. Cảm ơn quý thầy cô giảng dạy tại trường nói chung và Khoa Mạng máy tính & Truyền thông nói riêng vì đã truyền đạt những kiến thức chuyên môn bổ ích, những kinh nghiệm thực tế quý báu trong suốt quá trình học tập và rèn luyện tại trường.

Chúng tôi xin gửi lời tri ân và biết ơn đến ThS. Trần Thị Dung đã trực tiếp quan tâm, đồng hành trong suốt quá trình thực hiện đề tài. Đặc biệt, chúng tôi xin gửi lời cảm ơn trân trọng nhất đến ThS. Đỗ Thị Thu Hiền, là người đã định hướng, dẫn dắt và đồng hành cùng chúng tôi từ những ngày đầu hình thành ý tưởng cho khoá luận này.

Bên cạnh đó, với tình cảm sâu sắc và chân thành, chúng tôi cũng xin cảm ơn các thầy cô, anh chị, các bạn đang công tác tại Phòng thí nghiệm An toàn thông tin - InSecLab vì đã luôn tạo điều kiện về cơ sở vật chất, luôn sẵn sàng nhiệt tình hỗ trợ chúng tôi về chuyên môn lẫn kinh nghiệm trong các hoạt động nghiên cứu và thực hiện khoá luận.

Cuối cùng, do kiến thức chuyên môn còn hạn chế nên khóa luận chắc chắn không tránh khỏi những thiếu sót. Rất mong nhận được nhận xét, ý kiến đóng góp, phê bình từ quý thầy cô trong hội đồng để khóa luận được hoàn thiện hơn.

**Nhóm thực hiện.**

# MỤC LỤC

# DANH MỤC HÌNH ẢNH

# DANH MỤC BẢNG

# DANH MỤC TỪ VIẾT TẮT

| Thuật ngữ | Mô tả |
|---|---|
| RL | Reinforcement Learning |
| PE | Portable Executable |
| DiDQN | Double Q-Learning |
| CNN | Convolutional Neural Network |
| ARBE | Append Random Byte |
| ARI | Append Randomly Named Library with Random Function Name |
| ARS | Append Randomly Named Section |
| RS | Remove Signature from Certificate Table of the DataDirectory |
| Relu | Rectified Linear Unit |
| R | Reward |
| DistDQN | Distributaional DQN |
| MalConv | Malware Convolutaion |
| IR | Intermediate Representation |
| DL | Deep learning |
| ER | Evasion rate |
| ERFP | Evasive Rate with Functionality |
| TP | True Positive |
| TN | True Negative |
| FP | False Positive |
| FN | False Negative |

| | |
|---|---|
| RAM | Random Access Memory |

# TÓM TẮT KHOÁ LUẬN

Trong thời gian gần đây, sự gia tăng cả về số lượng và độ phức tạp của các loại phần mềm độc hại đã đặt ra một thách thức đáng kể trong việc phát hiện và ngăn chặn chúng. Sự phát triển này phản ánh sự tiến bộ của các kỹ thuật tấn công, khiến cho các hệ thống bảo mật phải ngày càng tiến xa hơn để đối phó. Nổi bật hơn hết, kỹ thuật học máy và học sâu đã được áp dụng rộng rãi trong việc xây dựng các hệ thống phát hiện phần mềm độc hại. Tuy nhiên, điều đáng chú ý là các phương pháp này cũng dễ bị tấn công bởi các kỹ thuật tinh vi của phần mềm độc hại, khiến cho chúng có thể tránh được các trình phát hiện hoặc gây ra các kết quả giả mạo.

Nghiên cứu hiện nay đang tập trung vào việc phát triển các biến thể của phần mềm độc hại, nhằm mục đích vượt qua các hệ thống phát hiện dựa trên học máy và học sâu. Điều này bao gồm việc tạo ra các biến thể mới của phần mềm độc hại hoặc sửa đổi các biến thể hiện có để tránh bị phát hiện.

Trong ngữ cảnh này, hệ điều hành Windows đang trở thành một mục tiêu phổ biến của các tác nhân tấn công, và việc phát triển các giải pháp phát hiện phần mềm độc hại hiệu quả trở nên càng quan trọng hơn bao giờ hết.

Tuy nhiên, việc tạo ra các biến thể của phần mềm độc hại không chỉ đòi hỏi sự cân nhắc về tính năng mà còn về khả năng tránh được phát hiện. Đây cũng chính là lý do, trong khoá luận này, chúng tôi hướng tới việc xây đựng mô hình kết hợp phương pháp học tăng cường để tạo ra các biến thể mã độc và các mô hình học máy để kiểm chứng chức năng của mã độc. Cuối cùng, chúng tôi đưa ra các đánh giá về hiệu suất và đề xuất định hướng phát triển đề tài này trong tương lai.

# CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

## 1.1. Lý do chọn đề tài

Hiện nay, với sự phát triển vượt bậc của kỹ thuật công nghệ hiện đại, tốc độ gia tăng của các loại mã độc đã chạm đến ngưỡng cao nhất so với những năm gần đây. Các mã độc không chỉ tăng lên về số lượng mà còn đa dạng về cách tấn công và mức độ nguy hiểm. Chúng có khả năng biến đổi để qua mặt các bộ phát hiện một cách dễ dàng. Điều này đã đặt ra thách thức rất lớn cho các nhà nghiên cứu về mã độc trong việc cải tiến và tìm ra những phương pháp mới để ngăn chặn và phát hiện chúng một cách hiệu quả và nhanh chóng.

Học tăng cường (Reinforcement Learning) là một mô hình được sử dụng rộng rãi và phổ biến để tạo các biến thể mới của mã độc. Với khả năng tự học từ môi trường và tối ưu hóa hành vi thông qua các phản hồi, RL có thể tạo ra những biến thể mã độc phức tạp và khó phát hiện hơn .

Một trong những vấn đề quan trọng khi tạo biến thể mã độc là làm sao để có thể xác định biến thể nào còn hoạt động theo đúng chức năng của nó. Nhiều nghiên cứu hiện nay đã đề xuất các phương pháp để xác định xem mẫu nào còn hoạt động bằng cách đưa chúng vào các trình phát hiện mã độc. Tuy nhiên, việc này chưa đủ để đảm bảo chức năng của các biến thể vẫn chưa được kiểm chứng một cách toàn diện.

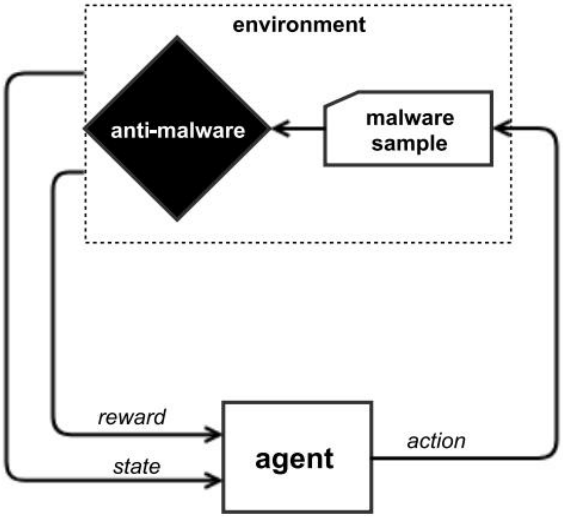
Chức năng của các biến thể vẫn chưa được kiểm chứng. Đó là vấn đề chúng tôi muốn giải quyết trong khoá luận. Song song với việc tạo biến thể mã độc bằng RL, chúng tôi đề xuất một phương pháp so sánh tương đồng mã nhị phân của các tệp nhị phân và sử dụng học máy để kiểm chứng sự giống và khác nhau giữa chúng. Phương pháp này không chỉ giúp tạo ra các biến thể mã độc một cách hiệu quả mà còn đảm bảo rằng các biến thể đó vẫn duy trì được các chức năng cần thiết.

## 1.2. Nghiên cứu liên quan

Đã có một số nghiên cứu tập trung vào việc tạo biến thể mã độc với nhiều hướng tiếp cận khác nhau.

Ví dụ, nghiên cứu [1] đề xuất một phương pháp khám phá đặc trưng biến đổi mã độc dựa trên thuật toán Monte Carlo Tree Search (MCTS), với một tập luật tùy biến để hạn chế các biến đổi không hợp lệ hoặc làm hỏng mã độc. Đồng thời, nghiên cứu phân tích kịch bản tấn công xám trong đó kẻ tấn công không biết thuật toán và quyết định của bộ phân loại mục tiêu, nhưng biết các đặc trưng được sử dụng trong huấn luyện. Kẻ tấn công huấn luyện một mô hình thay thế cục bộ để xác nhận các biến đổi đặc trưng. Tác giả cũng thực hiện so sánh với phương pháp khác, cho thấy phương pháp đề xuất đã tìm ra nhiều biến thể phần mềm độc hại hơn và có tỷ lệ né tránh cao hơn so với phương pháp tìm kiếm ngẫu nhiên (Random Search). Tuy vậy, nghiên cứu chỉ ra phần mềm độc hại vẫn duy trì chức năng độc hại sau khi biến đổi, nhưng không kiểm tra điều này trên các mẫu thực tế. Điều này có thể làm giảm tính thực tế của kết quả.

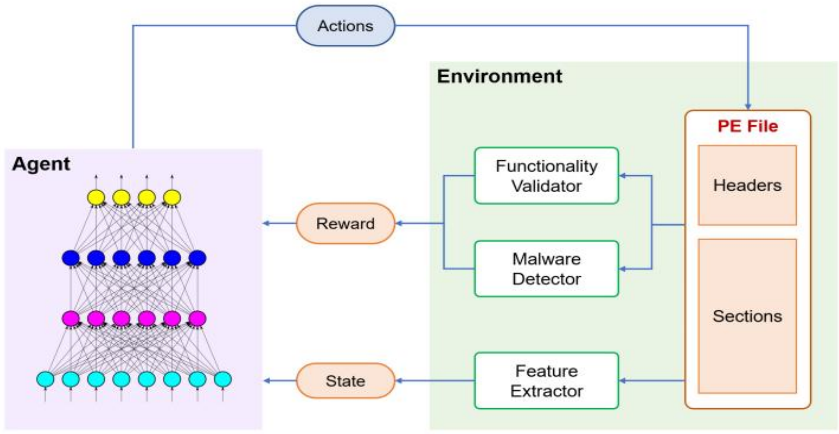
Trong một nghiên cứu khác [2], tác giả áp dụng pr để tạo ra các biến thể mã độc có thể né tránh một mô hình học máy mà không cần biết cấu trúc hay đặc trưng của mô hình (**Hình 1.1**). Phương pháp này chỉ cần có khả năng truy vấn mô hình để nhận được nhãn xấu hay tốt cho một tệp PE bất kỳ. Trong mô hình đề xuất, tác nhân học tăng cường có thể áp dụng chính sách hành động của mình để tấn công các mẫu mã độc mới chưa từng thấy trong quá trình huấn luyện. Ngoài ra, tác giả cũng khẳng định rằng các biến thể mã độc được tạo ra bằng cách thay đổi tệp nhị phân vẫn giữ được định dạng và chức năng của tệp PE gốc. Tác giả cung cấp mã nguồn mở cho môi trường tấn công dưới dạng OpenAI gym, cho phép các nhà nghiên cứu khác sử dụng, thích ứng và cải tiến phương pháp của họ.




*Hình 1.1. Mô hình RL được đề xuất trong [2].*

Bên cạnh các kết quả tiềm năng, nghiên cứu này vẫn có tỷ lệ né tránh thấp hơn so với các phương pháp tấn công khác có nhiều thông tin hơn về mô hình bị tấn công. Mặt khác, một số biến thể phần mềm độc hại bị phá vỡ định dạng PE hoặc chức năng do các vấn đề về phân tích cú pháp hoặc mã hóa. Đồng thời, một số biến thể phần mềm độc hại có thể để lại dấu vết đặc trưng cho mô hình học máy, làm giảm hiệu quả của việc huấn luyện lại.

Ở nghiên cứu [3], nhóm tác giả đã đề xuất một phương pháp tiên tiến để biến đổi các phần mềm độc hại trên hệ điều hành Windows, sử dụng học tăng cường và tập trung vào việc bảo toàn chức năng của chúng **(Hình 1.2**). Phương pháp này áp dụng thuật toán Double Q-Learning (DiDQN), đưa ra không gian hành động với 10 biến thể khác nhau có thể áp dụng cho các tệp mã độc PE, và sử dụng hệ thống tính điểm đa chiều để đảm bảo chất lượng biến thể. Đồng thời, cơ chế kiểm chứng chức năng được đề xuất bằng cách thực thi lại các tệp đã biến đổi trên môi trường máy ảo Windows, nhằm đảm bảo sự bảo toàn của các chức năng ban đầu.




*Hình 1.2. Mô hình tạo đột biến phần mềm độc hại dựa trên RL có kiểm chứng chức năng.*

Kết quả đánh giá của nghiên cứu này cho thấy các tác nhân được huấn luyện với DiDQN trên tập dữ liệu EMBER đã tạo ra các biến thể đạt được kết quả tiềm năng. Tuy nhiên, nghiên cứu cũng nhấn mạnh rằng việc kiểm chứng chức năng vẫn còn là một thách thức, với phần trọng tâm hiện tại chủ yếu là tính thực thi mà chưa xác định rõ các chức năng cụ thể của các biến thể mới.

## 1.3. Tiềm năng của RL

Trong các nghiên cứu về tạo biến thể phần mềm độc hại, một trong những phương pháp nổi bật được sử dụng là học tăng cường. RL là một lĩnh vực của trí tuệ nhân tạo mà các hệ thống tự động học cách ra quyết định để đạt được một mục tiêu hoặc tối ưu hóa một hàm phần thưởng trong một môi trường.

Sử dụng RL trong việc tạo ra các biến thể của phần mềm độc hại có thể giúp tạo ra các mẫu độc hại có khả năng vượt qua các hệ thống phát hiện dựa trên học máy/học sâu. Tuy nhiên, việc áp dụng RL cũng đặt ra các thách thức về tính hiệu quả và tính thực tiễn của các biến thể được tạo ra. Việc kết hợp RL vào quy trình tạo ra biến thể phần mềm độc hại là một lĩnh vực đầy triển vọng nhưng cũng đòi hỏi sự hiểu biết sâu sắc về cả vấn đề phần mềm độc hại và các thuật toán RL. Điều này cũng đặt ra một cơ hội để mở rộng kiến thức và phát triển các giải pháp tiên tiến hơn trong cuộc chiến chống lại các loại mã độc cải tiến.

## 1.4. Tính khoa học và tính mới của đề tài

Tuy có nhiều ưu điểm, nhưng việc áp dụng RL vào tạo biến thể mã độc cũng có điểm hạn chế là chưa đảm bảo được việc giữ nguyên các chức năng chính của bản gốc khi tạo ra các biến thể. Dù vẫn có khả năng tạo biến thể mã độc Windows nhưng vẫn chưa thể chứng minh tính chính xác hoặc kiểm chứng chức năng một cách đầy đủ. Việc đảm bảo các chức năng của biến thể có thể hoạt động tương tự bản gốc vì vậy đang là vấn đề cần được giải quyết, do đó đề tài của nhóm chúng em hướng đến giải quyết một phần vấn đề này.

Một giải pháp tiềm năng là áp dụng các phương pháp so sánh tương đồng nhị phân trong các mô hình học tăng cường, đóng vai trò so sánh độ tương đồng giữa mã độc gốc và biến thể đã được tạo, từ đó đảm bảo tính chính xác và kiểm chứng chức năng của các biến thể phần mềm độc hại. Để làm được điều này, đã có một phương pháp so sánh tương đồng nhị phân đáng chú ý là áp dụng thuật toán prov2vex [4] để chuyển đổi các biến thể dưới dạng nhị phân thành các vector đại diện, kết hợp và biến đổi các vector này thành ảnh, sau đó thực hiện đánh giá tương đồng bằng CNN [4] . Tuy nhiên, để tối ưu hoá thời gian chạy của thuật toán prov2vex, chúng tôi đã thực hiện một số thay đổi trong quy trình thực hiện của thuật toán, nhằm giảm thời gian chạy xuống mức có thể và thay vì dùng các thuật toán Cosin và Jaccard [5] để tính toán độ tương đồng giữa các tệp như đã làm ở đồ án chuyên ngành, chúng tôi dùng CNN để kiểm chứng vì nhận ra hai thuật toán này không còn phù hợp khi đã có những thay đổi trong phương pháp prov2vex.

## 1.5. Mục tiêu

- Sử dụng phương pháp học tăng cường để xây dựng mô hình huấn luyện nhằm tạo ra các mã độc mới dựa trên tập các mã độc gốc có sẵn.
- Đảm bảo các mã độc mới được tạo ra có thể vượt qua được bộ phát hiện mã độc (Malwares detector).

- Nếu mã độc sau khi veto qua bộ phát hiện tiếp tục vượt qua được bộ kiểm chứng chức năng sử dụng so sánh nhị phân thì thành công trong việc tạo ra một biến thể mã độc hoàn chỉnh.
- Thời gian trích xuất vector cho bộ kiểm chứng sẽ tối ưu hơn, nhưng vẫn đảm bảo hiệu quả.

## 1.6. Đối tượng

- Mã độc Windows và các biến thể mã độc.
- Mô hình học tăng cường để tạo biến thể mã độc.
- Mô hình kiểm chứng chức năng bằng so sánh nhị phân có sử dụng CNN
- Mô hình học máy phát hiện mã độc.

## 1.7. Phạm vi

Sử dụng mô hình học tăng cường tạo để tạo ra các biến thể mã độc PE bằng cách thay đổi các tệp gốc, và thực hiện đánh giá thông qua mô-đun kiểm chứng chức năng và bộ phát hiện mã độc.

## 1.8. Phương pháp nghiên cứu

Tìm hiểu về cách hoạt động của học tăng cường trong việc tạo ra biến thể mã độc, song song với việc tìm hiểu phương pháp trích xuất vector từ tệp thực thi và dùng các mô hình học máy cả cho mô dun kiểm chứng chức năng và phát hiện mã độc. Tìm cách tối ưu các thuật toán sử dụng và tạo bộ dữ liệu phù hợp cho mô hình học máy để có thể thu được hiệu suất cao. Tiếp đến, chúng tôi tiến hành thực nghiệm, đánh giá kết quả, đưa ra nhận xét và đề xuất hướng đi cho đề tài trong tương lai.

## 1.9. Cấu trúc khoá luận tốt nghiệp

Khoá luận được tổ chức trong 5 chương sau:

• Chương 1: TỔNG QUAN ĐỀ TÀI VÀ TÌNH HÌNH NGHIÊN CỨU

• Chương 2: CƠ SỞ LÝ THUYẾT

• Chương 3: PHƯƠNG PHÁP THỰC HIỆN

• Chương 4: HIỆN THỰC, ĐÁNH GIÁ VÀ THẢO LUẬN

• Chương 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

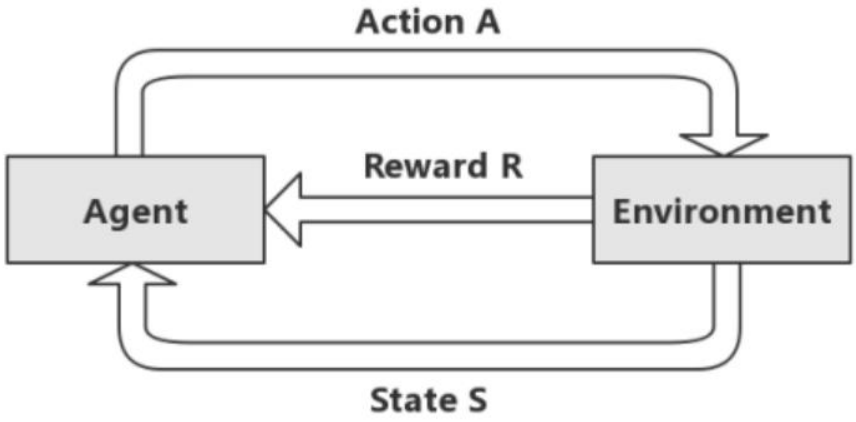
# CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

## 2.1. Học tăng cường

### 2.1.1. Tổng quan về mô hình học tăng cường

Học tăng cường (RL) là một lĩnh vực quan trọng, có tiềm năng ứng dụng cao của trí tuệ nhân tạo, tập trung vào việc xây dựng và nghiên cứu các thuật toán mà một tác nhân tự động học cách ra quyết định hành động trong một môi trường để nhằm tối đa hóa phần thưởng nhận được, từ đó đạt được mục tiêu huấn luyện. Trong RL, tác tử tương tác với môi trường bằng cách đưa ra quyết định hoặc thực hiện các hành động và nhận được phản hồi từ môi trường dưới dạng phần thưởng đại diện cho kết quả từ quyết định hoặc hành động mà tác tử đưa ra. Mục tiêu của RL là tối ưu việc đưa ra các quyết định hoặc hành động mà tác nhân có thể thực hiện nhằm tối đa hóa phần thưởng mà nó thu được trong quá trình tương tác với môi trường.

### 2.1.2. Các thành phần chính




*Hình 2.1. Kiến trúc chung của học tăng cường.*

Các thành phần cơ bản trong RL (**Hình 2.1**) bao gồm:

- Agent (Tác nhân): Là thực thể hoặc chương trình máy tính có khả năng thực hiện các hành động đã đề ra sẵn trong môi trường. Nó có mục tiêu là tối đa hóa tổng phần thưởng mà nó nhận được thông qua quá trình tương tác với môi trường.

- Environment (Môi trường): là hệ thống bên ngoài mà tác nhân tương tác. Nó có thể là bất kỳ hình thức nào, từ trò chơi máy tính đến robot thực tế hoặc các hệ thống tự động.

- State (Trạng thái): là biểu diễn của môi trường tại một thời điểm cụ thể. Sau khi tác nhân đã thực hiện hành động cụ thể, trạng thái sẽ cung cấp thông tin về tình hình hiện tại của môi trường mà tác nhân sử dụng để ra quyết định hành động đã làm.

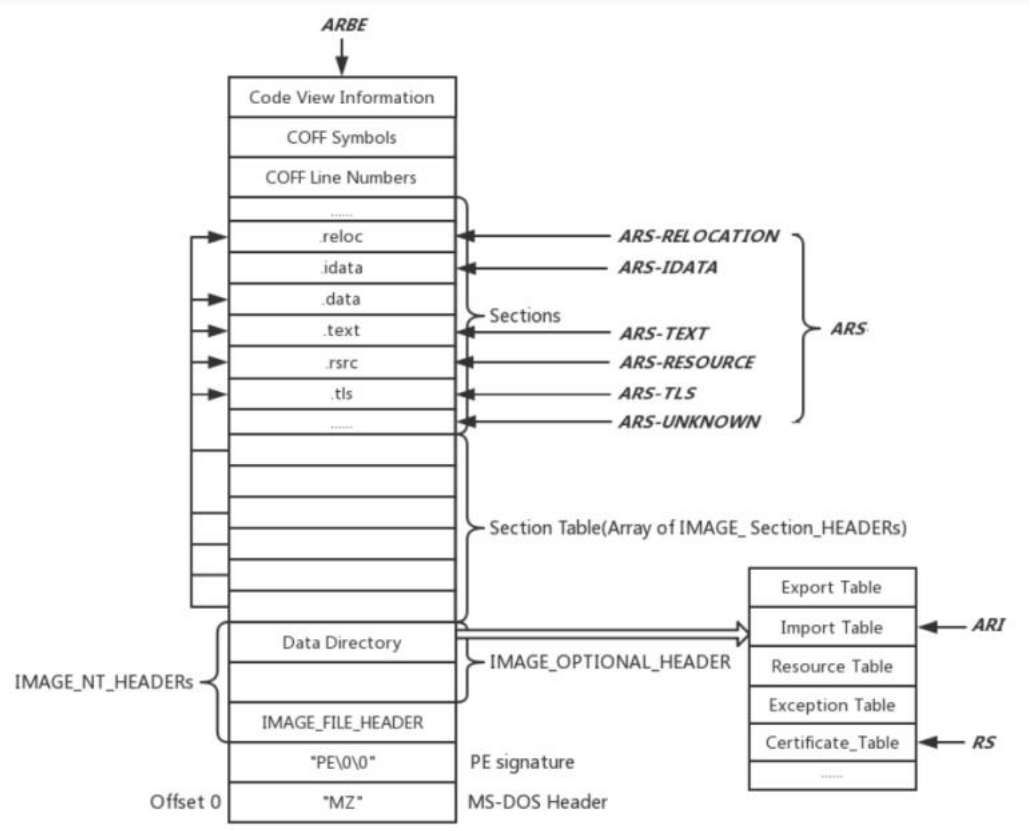
- Action (Hành động): là các hành động mà tác nhân có thể thực hiện trong môi trường. Hành động này thường được lựa chọn từ một tập hợp các hành động có thể thực hiện tại mỗi trạng thái.

- Reward (Phần thưởng): là phản hồi từ môi trường cho mỗi hành động của tác nhân và hướng dẫn tác nhân học từ kinh nghiệm. Nếu tác nhân có những hành động đúng và hiệu quả thì sẽ được thưởng, ngược lại nếu tác nhân có những hành động xấu thì sẽ bị phạt.

- Policy (Chính sách): các tác nhân trong môi trường phải tuân theo chính sách là chuỗi những quy tắc được đặt ra để chọn ra hành động phù hợp tại mỗi trạng thái. Chính sách có thể được biểu diễn dưới dạng một bảng quyết định hoặc một hàm giá trị.

### 2.1.3. Các action có thể thực hiện trên file PE

Trong ngữ cảnh của việc sử dụng RL để xử lý vấn đề tránh bị phát hiện do ảnh hưởng của các phần mềm độc hại, có một số hành động có thể thực hiện trên file PE [6] như **Hình 2.2.**




*Hình 2.2.Cấu trúc file PE và những hành động dùng để biến đổi nội dung của file.*

- ARBE : Thêm các byte ngẫu nhiên vào cuối của tệp PE. Việc này có thể làm thay đổi kích thước của tệp một cách ngẫu nhiên mà không ảnh hưởng đến chức năng của nó.

- ARI : Thêm một thư viện mới với tên và tên hàm ngẫu nhiên vào bảng địa chỉ nhập của tệp PE, có thể làm cho tệp trở nên phức tạp hơn và khó phát hiện hơn.

- ARS : Thêm một phần mới với tên ngẫu nhiên vào bảng phần của tệp PE. Phần này có thể chứa dữ liệu không mong muốn hoặc mã độc hại.

- RS : Loại bỏ chữ ký số từ bảng chứng chỉ của tệp PE, từ đó có thể gây ra thay đổi trong dữ liệu chứng chỉ và làm cho tệp trở nên khó phát hiện hơn.

### 2.1.4. Ứng dụng

RL được áp dụng rộng rãi trong ứng dụng khoa học và công nghệ trong dời sống hiện nay như:

- Phát triển hệ thống tự động lái xe, từ xe tự lái trong công nghiệp ô tô đến các phương tiện tự lái như xe buýt và xe điện tự lái. RL giúp hệ thống lái xe tự động học từ môi trường xung quanh và tự điều chỉnh hành vi lái xe với mục đích đảm bảo an toàn và hiệu quả.

- Robotics và Automation: áp dụng trong nhiều ứng dụng robot như robot dọn dẹp nhà cửa, robot giao hàng, robot y tế và robot sản xuất. RL giúp robot học và tự lựa chọn và điều chỉnh hành vi của mình dựa trên môi trường và mục tiêu cụ thể.

- Chăm sóc sức khỏe: RL được áp dụng trong nhiều mặt của lĩnh vực y học cũng như về chăm sóc sức khỏe để tối ưu hóa quy trình chuẩn đoán và điều trị bệnh, lập lịch hóa các cuộc hẹn bác sĩ và dự đoán các kết quả của bệnh lý.

## 2.2. Học máy

### 2.2.1. Tổng quan về mô hình học máy

Mô hình học máy là một lĩnh vực thuộc trí tuệ nhân tạo (AI) nơi mọi thành phần của kiến trúc hệ thống được xây dựng theo mục đích riêng nên có thể học một cách tự động các dữ liệu và cải thiện hiệu suất mà không cần phải được theo dõi một cách thủ công từ con người. Mục tiêu chính của việc sử dụng học máy là phát triển khả năng dự đoán hoặc phân loại dữ liệu mới dựa vào kinh nghiệm đã học từ bộ huấn luyện.

### 2.2.2. Các thành phần chính

Một số khái niệm/thành phần cơ bản liên quan tới học máy:

– *Dữ liệu Huấn Luyện*: là dữ liệu mà mô hình sử dụng để học. Nó thường bao gồm các dữ liệu (dạng ảnh, văn bản, video...) và nhãn phân loại tương ứng.

- *Model*: Một mô hình là một biểu diễn toán học của quy luật mà mô hình cố gắng học từ dữ liệu. Mô hình có thể là một tập hợp các tham số được điều chỉnh trong quá trình huấn luyện.
- *Training*: còn gọi là huấn luyện, là quá trình tinh chỉnh các tham số của mô hình để mô hình có khả năng dự đoán đầu ra mong muốn từ đầu vào. Quá trình này thường bao gồm việc tối ưu hóa một hàm mất mát (loss function) nhằm cải thiện độ chính xác của mô hình.
- *Dự Đoán*: áp dụng mô hình đã huấn luyện để dự đoán kết quả với dữ liệu mới mà nó chưa từng thấy.
- *Hàm Mất Mát (Loss Function)*: Đây là một phép đo độ chệch giữa đầu ra dự đoán của mô hình và giá trị thực tế mong muốn. Mục tiêu là giảm thiểu giá trị của hàm mất mát trong quá trình huấn luyện.
- *Thuật Toán Học Máy*: Là các phương pháp và kỹ thuật được sử dụng để xây dựng và huấn luyện mô hình. Các thuật toán phổ biến bao gồm học giám sát (supervised learning), học không giám sát (unsupervised learning), và học tăng cường (reinforcement learning).
- *Overfitting* xảy ra khi mô hình học quá mức các đặc trưng và nhiễu của dữ liệu huấn luyện, làm cho mô hình trở nên quá phù hợp với dữ liệu này mà không thể áp dụng tôt cho dữ liệu mới. Khi một mô hình bị overfitting, nó thể hiện hiệu suất rất cao trên tập huấn luyện nhưng lại kém trên tập kiểm tra hoặc tập xác thực. Điều này xảy ra do mô hình đã học thuộc lòng các mẫu cụ thể của dữ liệu huấn luyện, bao gồm cả nhiễu, thay vì học các đặc trưng chung có thể áp dụng cho dữ liệu mới. Một số nguyên nhân chính của overfitting bao gồm việc sử dụng một mô hình quá phức tạp với quá nhiều tham số, thời gian huấn luyện quá dài, hoặc dữ liệu huấn luyện không đủ đa dạng.
- *Underfitting* xảy ra khi mô hình thiết kế quá đơn giản và không thể học được cấu trúc phức tạp của dữ liệu. Khi bị underfitting, hiệu suất của mô hình sẽ thấp trên cả trên tập huấn luyện và tập kiểm tra. Underfitting thường xảy ra

khi mô hình có quá ít tham số, sử dụng các kỹ thuật regularization quá mạnh, hoặc thời gian huấn luyện không đủ.

### 2.2.3. Giới thiệu mô hình Convolutional Neural Network(CNN)

#### *2.2.3.1. Mạng nơ ron tích chập (CNN)*

Mạng nơ ron tích chập (CNN) là một trong những mô hình học sâu hiện đại và thường được ưu tiên sử dụng cho các bài toán nhận diện đối tượng trong hình ảnh.

#### *2.2.3.2. Convolutional*

Convolutional có thể hiểu là một loại cửa sổ dạng trượt di chuyển trên một ma trận đầu vào và chứa các parameter (tham số) có khả năng tự điều chỉnh để trích xuất được những thông tin chính xác nhất mà không cần chọn đặc trưng.

#### *2.2.3.3. Các thành phần cơ bản của CNN*

- *Convolutional Layer*

Lớp này là thành phần quan trọng nhất của CNN, nó đảm nhiệm việc thực hiện các phép tính cần thiết để trích xuất đặc trưng từ dữ liệu. Các phép tính như stride, padding, filter map, feature map đều được thực hiện trong lớp này. Những yếu tố này đóng vai trò thiết yếu cho việc tối ưu hoá khả năng nhận diện và phân loại của CNN.

Trong đó:

- Filter map: là các ma trận ba chiều chứa các tham số, được sử dụng để áp (nhân) vào từng vùng cụ thể của ảnh.
- Stride: sự dịch chuyển filter map từ trái sang phải (theo pixel).
- Padding: là các giá trị 0 được thêm vào lớp input hay còn gọi là ma trận của hình ảnh, mục đích là để tăng kích thước của ảnh nhằm phù hợp với filter map mà không làm giảm chi tiết ảnh cũng như hiệu suất của mô hình.
- Feature map: kết quả thu được sau mỗi lần các filter map quét vào từng vùng ảnh.

- *Relu Laye*r (Rectified Linear Unit)

Còn có tên gọi khác là hàm kích hoạt, hàm này đóng vai trò tăng tốc cho quá trình training và làm nổi bật các đặc trưng sau khi filter map.

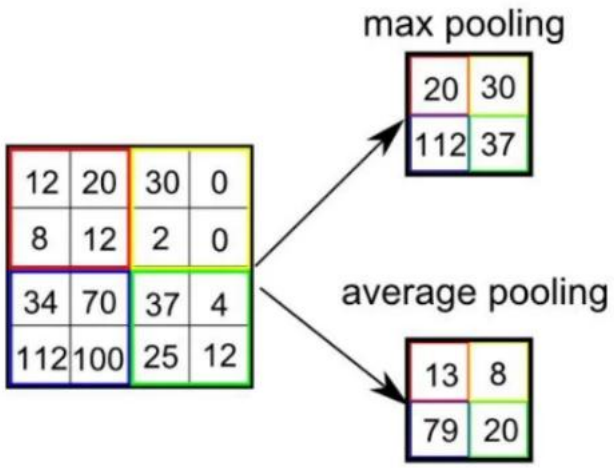
Công thức của hàm Relu:

$$f(x) = \max(0, x)$$

Kết quả của công thức là x nếu x > 0 và 0 nếu ngược lại, tức là làm cho các điểm ảnh trên ma trận luôn không âm, từ đó tăng tốc cho quá trình training.

- *Pooling Layer*

Thường dùng xen giữa các convolution layer, mục đích là làm giảm kích thước input hay ma trận hình ảnh, đồng thời vẫn giữ được các đặc trưng quan trọng (Hình 2.3).




*Hình 2.3. Pooling.*

Có 2 loại pooling :

– Max pooling: Chọn điểm ảnh có giá trị cao nhất trong vùng ảnh được quét qua

– Average: lấy giá trị trung bình của các điểm ảnh trong vùng ảnh được quét qua

– Fully Connected Layer: đây là phần có trách nhiệm tổng hợp kết quả sau khi hai lớp Convolutional và Pooling đã thực hiện phân tích input. Khi đến lớp này, mô hình sẽ có khả năng nhận diện các đặc trưng của ảnh một cách tốt nhất. Lớp này tạo sự kết nối đến tất cả các nơ-ron từ lớp trước, cho phép tạo ra các kết quả đầu ra phức tạp và đa dạng hơn.

### 2.2.4. Ứng dụng của việc áp dụng học máy

- Dự đoán và Dự báo: Học máy được sử dụng nhiều trong việc dự đoán kết quả sau khi đã học từ dữ liệu hiện tại. Ví dụ, trong tài chính, nó có thể được sử dụng để dự đoán giá chứng khoán, đề xuất các rủi ro về tín dụng, hay trong thời tiết để dự báo thời tiết.

- Xử lý Ngôn ngữ Tự nhiên: Việc áp dụng học máy giúp máy tính hiểu và đưa ra xử lý tốt hơn với ngôn ngữ con người. Ứng dụng bao gồm máy dịch, chatbot, phân loại văn bản, phân tích ý kiến, và tổng hợp tin tức.

- Thị giác máy tính: Giúp máy tính nhận diện, phân loại hoặc trích xuất dữ liệu từ hình ảnh và video. Chẳng hạn như nhận diện khuôn mặt trên các thiết bị điện tử thông minh, phân loại đối tượng, và xe tự lái.

- An toàn và Bảo mật: Học máy có thể được sử dụng để phát hiện, nhận diện các hành vi độc hại, như gian lận tín dụng, hay bảo vệ mạng máy tính khỏi các mối đe dọa không đáng có.

- Học máy trong Y tế : Học máy giúp trong việc chẩn đoán bệnh, dự đoán kết quả bệnh, và quản lý nguồn dữ liệu khổng lồ trong y tế bằng cách học thông tin từ các hình ảnh y khoa và dữ liệu gen.

## 2.3. Thuật toán Distributaional DQN (DistDQN)

### 2.3.1. Tìm hiểu về thuật toán DQN (Deep Q-Network)

- DQN [7] là một thuật toán trong học tăng cường giúp agent học và thực hiện nhiệm vụ bằng cách thử nhiều hành động và ghi nhớ các kết quả .

- DQN sử dụng mạng nơ-ron để dự đoán kết quả của mỗi hành động đối với một trạng thái nhất định. Kết quả của hành động thường thể hiện dưới dạng số và cho biết hành động đó "tốt" hay "xấu" dựa trên kinh nghiệm đã có.

### 2.3.2. Tìm hiểu về thuật toán Distributaional DQN (DistDQN)

- Là một phiên bản nâng nấp của DQN, DistDQN có một số điểm mới như sau:
    - Thay vì chỉ dự đoán một con số duy nhất cho mỗi hành động (ví dụ như giá trị trung bình), DistDQN dự đoán cả một dãy số (tất cả các giá trị có thể) để cho biết nhiều khả năng khác nhau của kết quả.
- Cách hoạt động của DistDQN:
    - DistDQN đưa ra nhiều kết quả dự đoán cho mỗi hành động thay vì chỉ một như DQN. Ví dụ như thay vì dự đoán điểm báo cáo khóa luận là 6 (giá trị trung bình), thì DistDQN sẽ dự đoán nhiều giá trị hơn, như 20% khả năng được 7 điểm, 30% khả năng được 8 điểm,…
- Ưu điểm khi sử dụng DistDQN:
    - Việc dự đoán nhiều giá trị giúp DistDQN nắm bắt được nhiều khả năng cũng như các rủi ro trong mỗi hành động mà agent thực hiện, thay vì chỉ dựa vào một kết quả duy nhất như ở DQN.
    - DistDQN có lợi thế trong việc thực hiện các tác vụ phức tạp, nơi mà các kết quả thay đổi liên tục đối với mỗi hành động khác nhau được thực hiện.

# CHƯƠNG 3. PHƯƠNG PHÁP THỰC HIỆN

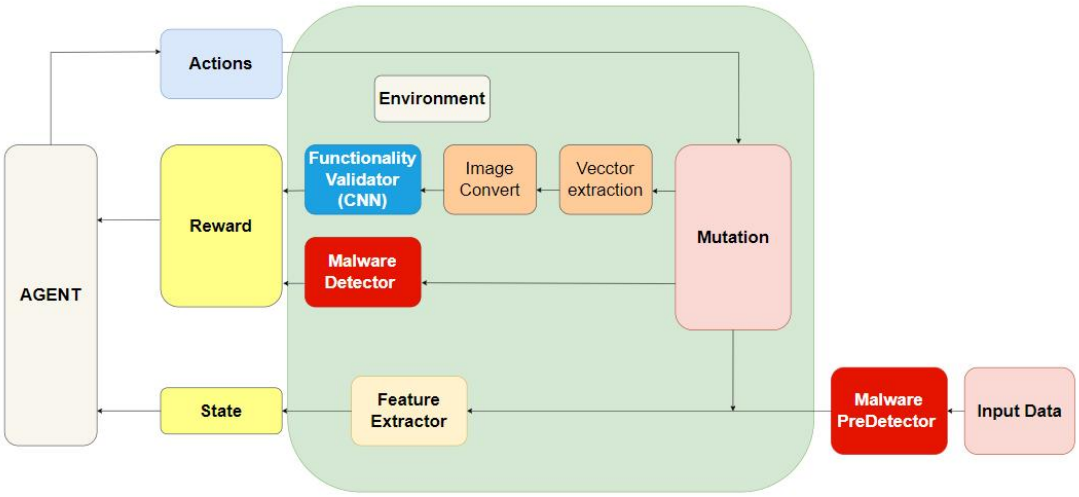
## 3.1. Mô hình RL đề xuất

### 3.1.1. Mục đích

Tương tự như với mọi mô hình RL, mục tiêu chính của mô hình RL mà nhóm đề xuất là huấn luyện agent nhằm tối đa hóa phần thưởng nhận được, cụ thể là thông qua việc ra quyết định áp dụng các hành động lên mã độc gốc một cách hợp lý, từ đó giải quyết được mục tiêu đặt của bài toán là tạo ra các biến thể mã độc nhưng vẫn đảm bảo giữ nguyên được chức năng thực thi.

### 3.1.2. Sơ lược các thành phần và chức năng trong mô hình RL

Mô hình RL để tạo biến thể mã độc được nhóm chúng tôi đề xuất thể hiện ở **Hình 3.1**, trong đó bao gồm nhiều thành phần, mỗi thành phần thực hiện một vai trò khác nhau trong việc biến đổi các tập tin PE gốc để tạo thành các biến thể PE mới.




*Hình 3.1. Tổng quan mô hình RL đề xuất.*

- Agent: hay còn gọi là tác tử (hoặc tác nhân), hoạt động trong môi trường được thiết lập sẵn và học cách để đưa ra quyết định thực hiện các hành động trong không gian hành động nhằm tạo ra các biến thể hoàn chỉnh để vượt qua 2 bộ kiểm tra.

- Môi trường (Environment): là nơi để agent học và tương tác để nhân và tối ưu các phần thưởng nhận được. Cụ thể trong môi trường bao gồm các thành phần sau:
    - Dữ liệu đầu vào (Input): các tập tin thực thi trên Windows (.exe) sẽ được trích xuất đặc trưng, sau đó được agent tương tác để tạo ra các biến thể thông qua các hành động được áp dụng.
    - Trạng thái (State): các đặc trưng của tệp mà agent cần biến để học và kết hợp với phần thưởng để xác định xem hành động nào để áp dụng lên input file hoặc biến thể.
    - Bộ phát hiện mã độc (Malware detector): phát hiện xem tệp tin thực thi có phải là mã độc không dựa trên ngưỡng dự đoán được thiết lập sẵn.
    - Bộ kiểm chứng chức năng (Functionality validator): do nhóm tự đề xuất và thiết kế nhằm kiểm chứng xem tập tin tạo ra có hoạt động đúng mục đích hay không.
    - Hành động (action): hành động từ không gian hành động mà agent lựa chọn áp dụng lên tệp tin cần tạo biến thể.
    - Phần thưởng (reward): là một con số thể hiện cho mức độ thành công/ thất bại của agent sau khi quyết định một hành động trong một trạng thái cụ thể.

## 3.2. Các thành phần cụ thể

### 3.2.1. Không gian hành động - Action Space

Danh sách các hành động (action) mà agent có thể chọn để áp dụng trong mô hình của nhóm được liệt kê trong **Bảng 3.1.**

*Bảng 3.1. Không gian hành động trong mô hình RL*

| STT | Hành động | Mô tả |
|---|---|---|
| 1 | imports_append | Thêm các functions (hàm) vào Import table |

| | | |
|---|---|---|
| | | trong phần header của tập tin PE. |
| 2 | section_rename | Đổi tên section bất kỳ trong tập tin PE. |
| 3 | section_add | Tạo thêm section mới trong tập tin PE, section này sẽ không được dùng khi thực thi. |
| 4 | section_append | Thêm các bytes ngẫu nhiên gây nhiễu vào cuối section bất kỳ trong tập tin PE. |
| 5 | remove_signature | Xóa chữ ký của file |
| 6 | remove_dedug | Sửa đổi/xóa thông tin debug trong phần Optional header của tập tin PE. |
| 7 | upx_pack | Nén tập tin với công cụ UPX. |
| 8 | upx_unpack | Giải nén tập tin với công cụ UPX. |
| 9 | break_optional _header_checksum | Sửa đổi checksum trong phần Optional header của tập tin PE. |
| 10 | overlay_append | Thêm các bytes ngẫu nhiên gây nhiễu vào cuối tập tin PE. |

### 3.2.2. Môi trường - Environment

Môi trường, hay Environment là nơi agent sẽ lựa chọn hành động và áp dụng lên các tập tin PE khi đang được huấn luyện. Trong môi trường này, các thành phần gồm bộ phát hiện mã độc và bộ kiểm chứng chức năng sẽ được triển khai và đưa ra kết quả đánh giá mức độ "thành công" trong quyết định của agent gọi là phần thưởng agent, nhằm cải thiện việc lựa chọn các hành động trong các vòng huấn luyện tiếp theo.

Các biến thể được tạo ra hoặc input file sau khi vượt qua bộ kiểm tra sẽ được nhận vào môi trường và được trích xuất các đặc đặc trưng ở dạng vector bởi *Feature Extractor*, các vector đặc trưng này được gọi là state (trạng thái) trong mô hình, thể hiện cho các đặc điểm và thuộc tính của các file dưới dạng vector, cụ thể

dựa theo các hành động trong không gian hành động thì các đặc trưng là Impor Table, header, section, checksum và debug.

Song song với Feature Extractor, các biến thể tạo ra còn được xử lý với mô-đun phát hiện mã độc và mô-đun kiểm chứng chức năng, với kết quả cho ra là *phần thưởng (reward)* để phản hồi về cho agent. Phần thưởng này được tính dựa trên kết quả nhận diện mã độc và mức độ bảo toàn chức năng cùng các trọng số khác nhau như **Công thức (3.1)**.

$$R = R_{det} * \omega_{det} + R_{dis} * \omega_{dis} + R_{func} * \omega_{func} \quad (3.1)$$

Trong đó:

$R_{det}$ và $\omega_{det}$: reward và trọng số tương ứng của bộ phát hiện mã độc,

$R_{func}$ và $\omega_{func}$: reward và trọng số tương ứng của mô-đun kiểm chứng chức năng.

$R_{dis}$ và $\omega_{dis}$: reward và trọng số tương ứng của distance, được tính dựa trên số hành động đã được áp dụng lên biến thể

Công thức tính $R_{dis}$:

$$R_{dis} = (R_{max}/t_{max}) * t \quad (3.2)$$

Trong đó:

$R_{max}$: giá trị reward tối đa được thiết lập sẵn

$t_{max}$: số hành động tối đa có thể áp dụng lên file

t: số hành động đã được áp dụng lên file

Ngoài ra, trong mô hình còn áp dụng các hình phạt (penalty) để giảm giá trị reward nhận được khi xảy ra trường hợp lặp lại cùng một hành động lên một file:

$$R = R * p \quad (3.3)$$

Trong đó:

R: phần thưởng nhận được từ môi trường sau khi xử lý

p: số lần lặp lại 1 hành động lên 1 file, với p = 0.8 khi lặp lại 1 lần và p = 0.6 khi lặp lại từ 2 lần trở lên

### 3.2.3. Agent

Trong mô hình RL của nhóm, agent được xây dựng với thuật toán Distributaional DQN (DistDQN), trình tối ưu hóa (Optimizer) Adam và Explorer Noisy Nets. Với thuật toán DistDQN, tốc độ huấn luyện và hiệu suất được cải thiện hơn so với DQN bằng cách phân phối công việc thành các tác vụ nhỏ hơn cho nhiều nhân xử lý. Trong khi đó, explorer Noisy Nets được sử dụng để giúp agent tối ưu khả năng lựa chọn các hành động trong không gian hành động (action space), tránh bị lặp lại việc lựa chọn một hành động cụ thể.

Cách thức hoạt động của agent:

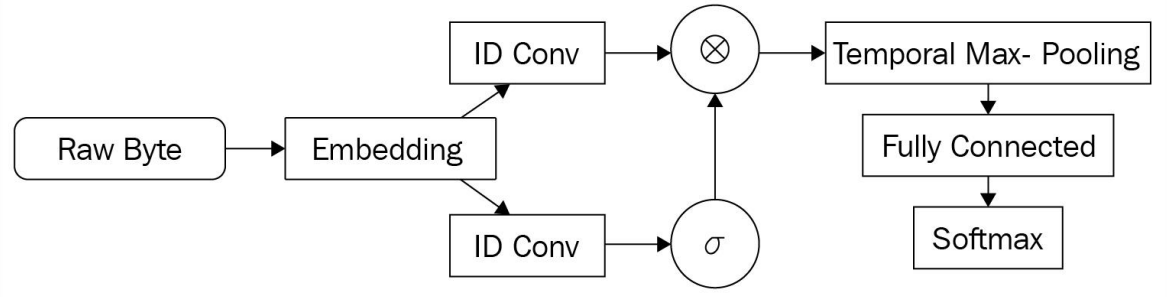
1. Agent nhận state và reward từ môi trường (với reward cho lần huấn luyện đầu tiên là 0).
2. Sử dụng thuật toán DistDQN để lựa chọn ra hành động phù hợp dựa trên state và reward.
3. Áp dụng hành động đã lựa chọn lên đoạn bytes tương ứng với file cần áp dụng, sau đó tạo ra một file thực thi mới.
4. Tiếp tục đưa file vừa tạo vào môi trường để tạo state và reward phục vụ cho lần huấn luyện tiếp theo.
5. Kiểm tra điều kiện dừng 1 tập huấn luyện (episode): nếu số lần huấn luyện (turn) của file đã đạt tốt đa hoặc file đồng thời vượt qua được malware detector và functionality validator thì dừng việc huấn luyện file đó, sau đó tiến hành thiết lập mới (reset) môi trường.
6. Trong trường hợp có lỗi khi thực hiện huấn luyện thì điều kiện dừng cũng sẽ được áp dụng.
7. Lặp lại bước (2) cho tới khi số tập huấn luyện (episode) đạt tới giá trị tối đa.

### 3.2.4. Mô-đun phát hiện mã độc MalConv

Mô-đun phát hiện mã độc (malware detector) được nhóm sử dụng trong mô hình là MalConv (Malware Convolutaion) dựa theo nghiên cứu [8] (**Hình 3.2**), một mô hình học sâu được sử dụng để phân loại các tệp tin độc hại (malware) mà không cần phải giải mã hoặc phân tích nội dung của chúng một cách chi tiết. Đây là một trong những mô hình tiên phong áp dụng học sâu vào lĩnh vực phát hiện phần mềm độc hại và đã chứng minh được tính hiệu quả cũng như độ tin cậy do đã được huấn luyện với dataset gồm hơn 100.000 file khác nhau (bao gồm cả mã độc và file bình thường).

Mô-đun MalConv áp dụng mạng nơ-ron tích chập CNN để huấn luyện agent phân loại tập tin độc hại:

1. Đầu tiên là Embbeding: thực hiện chuyển đổi input thành các vector với số chiều xác định trước.
2. Tiếp đến lớp Convolution Layer sẽ thực hiện học các đặc trưng từ vector vừa tạo sử dụng ReLu
3. Lớp pooling thực hiện giảm kích cho vector đặc trưng thông qua việc lọc lấy các giá trị thể hiện các thông tin được cho là quan trọng nhất
4. Các lớp Fully Connected: Đưa ra kết quả thể hiện dự đoán xác suất mẫu mã là malware hay không.



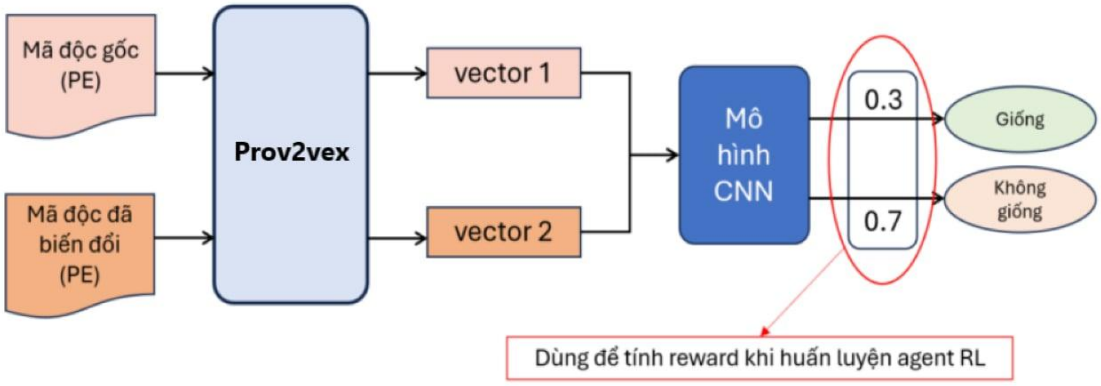

*Hình 3.2. Mô hình tổng quan bộ phát hiện mã độc MalConv*

- Sử dụng Mô-đun MalConv thực hiện dự đoán mã độc:
  1. Chuyển đổi tập tin cần dự đoán sang kiểu bytes

2. Tập tin đầu vào sau khi được chuyển đổi sẽ được đưa vào mô hình MalConv đã được huấn luyện để dự đoán
3. Kết quả trả về (trong mô hình của nhóm là score) sẽ được đem so sánh với ngưỡng (threshold) xác định trước, nếu score > threshold thì kết luận file là mã độc và ngược lại.
4. Tùy vào kết quả dự đoán thì reward tương ứng sẽ là $R_{det} = 10$ nếu không bị phát hiện và ngược lại $R_{det} = 0$ nếu file bị phát hiện là malware

### 3.2.5. Mô-đun kiểm chứng chức năng

**Hình 3.3** mô tả các bước xử lý của mô-đun kiểm chứng chức năng được đề xuất. Mô-đun này được thiết kế để nhận đầu vào là 2 tập tin PE cần so sánh sự tương đồng về mặt chức năng. Trong phương pháp đề xuất, mỗi tập tin PE sẽ được xử lý với Prov2vex để tạo các vector đặc trưng cho chúng, sau đó 2 vector đã trích xuất được sẽ được dùng làm đầu vào cho mô hình CNN. Đầu ra kỳ vọng của mô-đun kiểm chứng chức năng là tỉ lệ thể hiện mức độ tương đồng của 2 tập tin đầu vào, trong đó tỉ lệ này có thể được dùng để tính toán *phần thưởng (reward)* tương ứng nhằm phản hồi về cho agent.

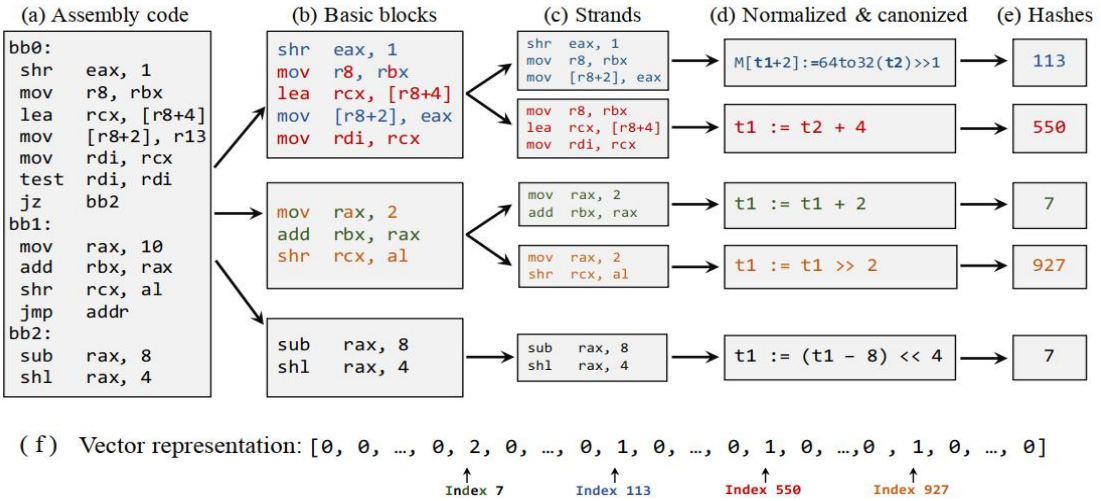



*Hình 3.3. Tổng quan mô-đun kiểm chứng chức năng.*

### *3.2.5.1. Kỹ thuật Prov2vex*

*a) Kỹ thuật Prov2vex*

Thuật toán Prov2vex giúp chuyển đổi các thủ tục hoặc đoạn mã thành các vector, được tham khảo từ [4], gồm các bước xử lý được biểu diễn ở **Hình 3.4.**




*Hình 3.4. Kỹ thuật Prov2vex khi chưa biến đổi.*

Như mô tả trong **Hình 3.4**, với đoạn mã hợp ngữ của một thủ tục (**Hình 3.4**a), kỹ thuật sẽ chuyển đổi nó thành một vector bao gồm năm bước như sau.

(1) Đầu tiên, thuật toán sẽ hiểu tệp đầu vào như mã assembly, sau đó chia mã assembly thành các khối cơ bản (**Hình 3.4**b). Một khối cơ bản được xác định dựa vào vị trí của các lệnh jmp trong mã nhị phân của thủ tục.

(2) Như trong **Hình 3.4**c, tiếp tục phân giải mỗi khối cơ bản thành các strand. Trong hình, các strand khác nhau có các màu sắc khác nhau, trong khi các lệnh thuộc về một số strand được đánh dấu bằng tất cả các màu liên quan.

(3) Tiếp theo, đưa các strand cú pháp khác nhau có cùng ý nghĩa ngữ nghĩa đến cùng một biểu diễn văn bản hay dạng chính tắc. Để đạt được mục đích này cần sử dụng kỹ thuật biến đổi khối mã thành các strand khác nhau và tập hợp các strand có ngữ nghĩa giống nhau theo từng nhóm và tối ưu hóa cơ sở mỗi strand. Như thể hiện trong **Hình 3.4**d, bước này thay đổi biểu diễn của các strand thành

một biểu diễn chuẩn, để các phép cộng liên tiếp được nhóm lại, nhân với lũy thừa của hai được thay thế bằng các dịch trái, các phép toán số học được sắp xếp lại thành một biểu diễn nhất quán, v.v.

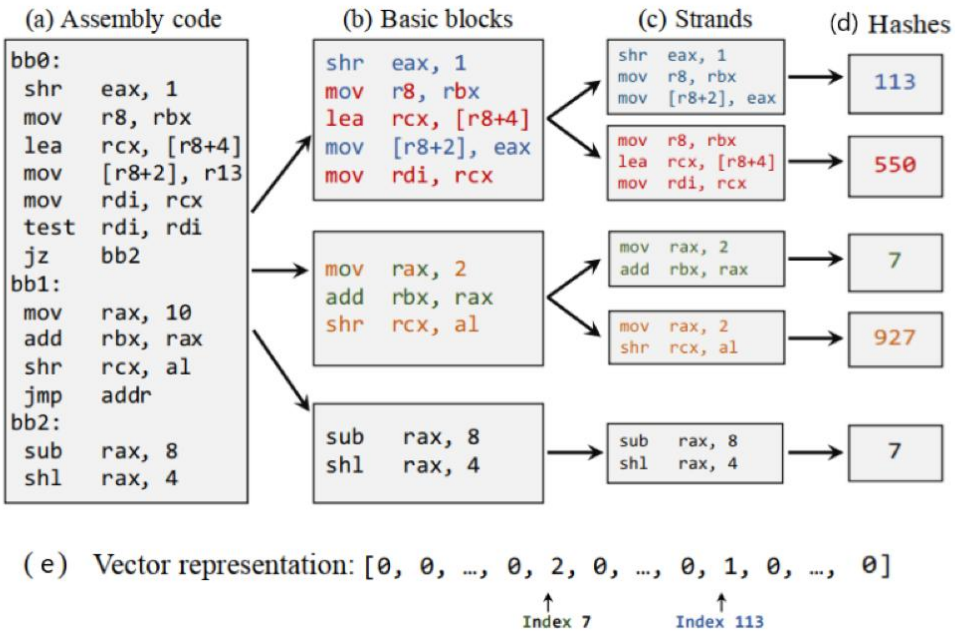
(4) Tác giả áp dụng hàm băm MD5 b-bit trên biểu diễn văn bản của các strand; do đó, chuyển đổi mỗi strand thành một số nguyên (**Hình 3.4**e) trong khoảng {0, 2b − 1}. Lưu ý rằng nếu b nhỏ thì xung đột có thể xảy ra, do đó các strand khác nhau có thể được ánh xạ vào cùng một giá trị băm. Trong quá trình thực nghiệm, tác giả nhận ra sử dụng MD5 10 bit mang lại kết quả băm tốt nhất, mang lại tính tối ưu cho biểu diễn vector của một thủ tục khi đưa vào mô hình mạng thần kinh.

(5) Cuối cùng, như mô tả trong **Hình 3.4**f, tác giả sử dụng tập hợp số nguyên kết quả từ các giá trị băm của mỗi biểu thức và đưa vào vector có độ dài $2^b$ trong đó mỗi giá trị của mảng băm tương ứng với số thứ tự trong vector, mỗi một giá trị băm xuất hiện tương ứng với giá trị 1 trong vector, và nếu có 2 giá trị băm giống nhau thì tương ứng với giá trị 2 trên vector ở vị trí của giá trị hàm băm và nhiều hơn nếu xuất hiện nhiều giá trị băm giống nhau như vậy. Ví dụ trong **Hình 3.4f**, trong phần băm có giá trị 113 thì khi biểu diễn ở dạng vector nó sẽ là "1" ở vị trí thứ 113, mặt khác, ta quan sát có 2 giá trị băm là "7" thì khi biểu diễn dạng vector, nó sẽ là "2" ở vị trí thứ 7. Do đó, các phần tử của vector có thể lớn hơn một, và tổng các phần tử của vector bằng số strand mà thủ tục tương ứng chứa.

Để triển khai thuật toán trên, tác giả sử dụng thư viện mã nguồn mở PyVEX. PyVEX chủ yếu đóng vai trò trong việc chuyển đổi mã máy thành VEX-IR. IR là một biểu diễn trung gian giữa mã nguồn và mã máy, giúp đơn giản hóa quá trình phân tích và tối ưu hóa mã máy. Tác giả sử dụng trình biên dịch nhị phân của nó để chuyển đổi mã hợp ngữ thành VEX-IR và cắt nó thành các strand (bước 2). Tác giả cũng tận dụng bộ tối ưu hóa VEX trên từng sợi để đưa chúng đến một biểu diễn chuẩn hóa (bước 3).

*b) Tối ưu Prov2vex*

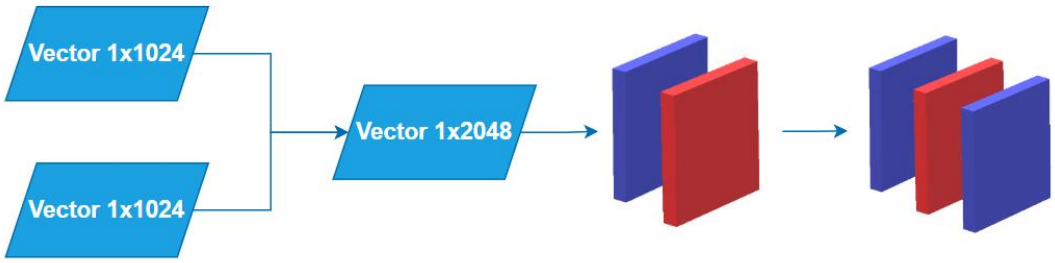
Qua quá trình hiện thực và thực nghiệm Prov2vex, chúng tôi nhận thấy thời gian để tạo được 1 vector của phương pháp gốc với đầu vào là một tệp thực thi PE là rất lớn (khoảng 10-50 phút/tệp), việc này có thể ảnh hưởng đến hiệu suất của toàn bộ mô hình RL. Do vậy, để rút ngắn thời gian tạo vector, chúng tôi đã lược bỏ bước thứ 3 trong các bước biến đổi, chúng tôi sẽ trực tiếp băm các strands mà không cần phải chuyển đổi chúng sang dạng biểu thức chính tắc với các bước còn lại không thay đổi, và vẫn giữ nguyên được định dạng kết quả đầu ra (**Hình 3.5**). Kết quả thực nghiệm cho thấy việc tùy chỉnh này không làm giảm đáng kể độ chính xác của mô-đun trong việc so sánh tương đồng chức năng của các tập tin.




*Hình 3.5. Kỹ thuật Prov2vex sau khi biến đổi.*

### *3.2.5.2. Tiền xử lý dữ liệu cho mô hình CNN*

Sau bước sử dụng kỹ thuật Prov2vex đã biến đổi, ta có được 2 vector biểu diễn đại diện cho 2 tập tin PE cần so sánh chứng năng. Để đưa được 2 vector này vào mô hình DL nhằm đánh giá độ tương đồng, nhóm đề xuất phương pháp gộp vector biểu diễn thành vector input duy nhất, được mô tả ở **Hình 3.6**.




*Hình 3.6. Tiền xử lý dữ liệu cho mô hình CNN.*

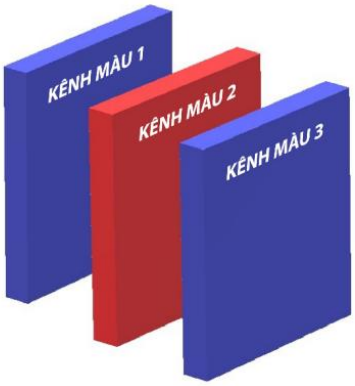
Cụ thể, với 2 vector ban đầu, chúng tôi tạo ảnh gồm 3 kênh từ 2 vector này.

**Bước 1.** 2 vector đầu ra của Prov2vex có cùng kích thước là 1.024 chiều, sẽ được gộp lại thành một vector lớn 2.048 chiều.

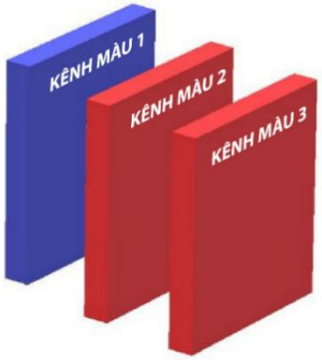
**Bước 2.** Vector lớn 2.048 chiều sau đó sẽ được chia thành 2 mảng đều nhau có kích thước 32x32.

**Bước 3.** 2 mảng sẽ được chia thành 2 kênh màu khác nhau: mảng đầu tiên sẽ thuộc kênh màu thứ nhất, mảng thứ 2 thuộc kênh màu thứ hai.

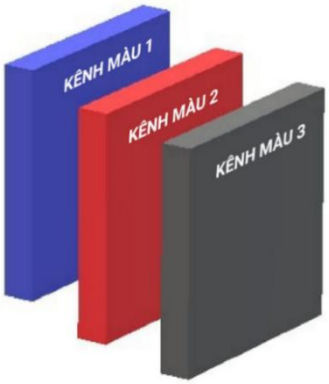
**Bước 4.** Do thư viện hỗ trợ chỉ xử lý ảnh có 1 kênh (ví dụ như ảnh xám), 3 kênh (ví dụ như ảnh màu BGR), hoặc 4 kênh (ví dự như ảnh màu BGRA) nên để tránh mất mát dữ liệu, một kênh thứ 3 được thêm vào ảnh để tạo thành ảnh có số kênh hợp lệ. Giá trị của các thành phần trong kênh thứ 3 này được tùy chỉnh với các trường hợp khác nhau (**Hình 3.7**) gồm trường hợp giống với kênh đầu tiên, giống kênh thứ 2, hoặc là các giá trị đồng nhất chẳng hạn như giá trị 0, để đánh giá được sự ảnh hưởng của kênh thứ 3 này với hiệu quả của phương pháp.




a) Hình ảnh có kênh màu thứ 3 trùng với kênh màu thứ nhất




b) Hình ảnh có kênh màu thứ 3 trùng với kênh màu thứ hai

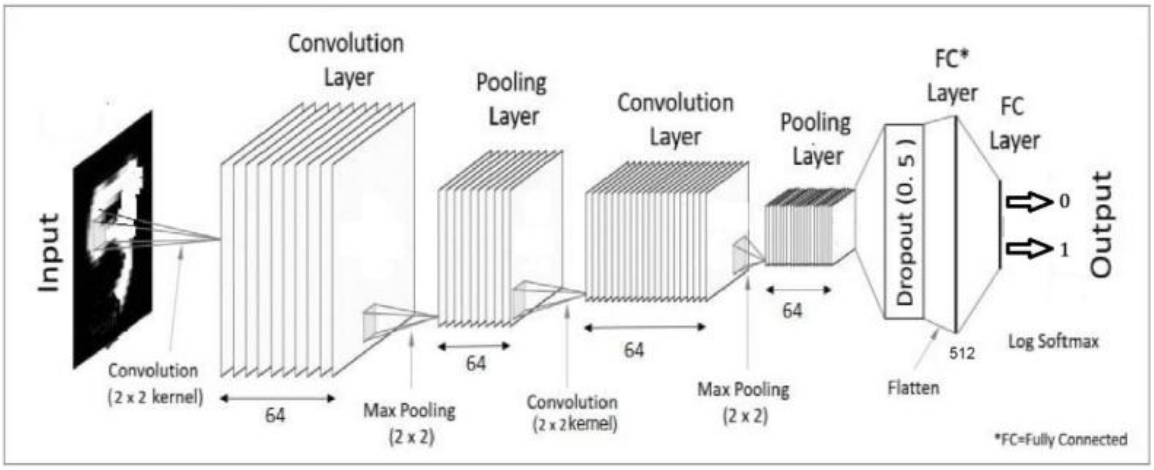



c) Hình ảnh có kênh màu thứ 3 là các giá trị 0

*Hình 3.7. Các trường hợp tạo ảnh gộp từ 2 vector đại diện.*

Ở giai đoạn huấn luyện mô hình DL cho chức năng phát hiện mã độc, ảnh được tạo sau đó sẽ được gán nhãn 1 (giống nhau) cho các vector của cùng một loại mã độc, ngược lại với nhãn 0 (khác nhau) cho các vector từ 2 loại mã độc khác nhau.

### *3.2.5.3. Xây dựng mô hình CNN cho mô-đun kiểm chứng chức năng*




*Hình 3.8. Mô hình CNN*

Mô hình CNN này bao gồm 8 lớp (1 lớp đầu vào, 1 lớp đầu ra và 6 lớp ẩn) với các chức năng hỗ trợ việc trích xuất và xử lý đặc trưng từ dữ liệu ảnh đầu vào nhằm phân loại hai kết quả (0 và 1) tương ứng với "không giống" và "giống" để đánh giá độ tương đồng giữa các file (**Hình 3.8**). Cụ thể các lớp của mô hình CNN:

- Bắt đầu với lớp đầu tiên, *lớp tích chập (Convolutional Layer)*, áp dụng 64 bộ lọc có kích thước 2x2 lên dữ liệu đầu vào là một ảnh có kích thước 32x32x3. Mỗi bộ lọc hoạt động như một máy dò đặc trưng, tìm kiếm các đặc điểm cục bộ như các cạnh hoặc đường viền trong hình ảnh. Lớp này sử dụng hàm ReLU hỗ trợ mô hình phân loại học được các đặc trưng phi tuyến và tăng khả năng biểu diễn của mạng. Đầu vào cho lớp này được xác định bởi input_shape, đảm bảo rằng dữ liệu được định dạng đúng cho việc xử lý.
- Tiếp theo là *lớp gộp (MaxPooling2D)* đầu tiên, với kích thước gộp 2x2. Chức năng của lớp này là giảm kích thước không gian của bản đồ đặc

trưng bằng cách chọn giá trị lớn nhất trong mỗi vùng 2x2. Quá trình này không chỉ giảm số lượng tham số tính toán mà còn giữ lại các đặc trưng quan trọng nhất, đồng thời giúp chống lại tình trạng overfitting bằng cách giảm độ phức tạp của mạng.

- *Lớp tích chập thứ hai*, giống với lớp đầu tiên, cũng sử dụng 64 bộ lọc có kích thước 2x2 và hàm kích hoạt ReLU. Chức năng của lớp này là tiếp tục trích xuất các đặc trưng phức tạp hơn nữa từ đầu ra của lớp gộp trước đó, giúp mô hình hiểu sâu hơn về các mẫu trong dữ liệu.
- *Lớp gộp thứ hai*, với cùng kích thước gộp 2x2, tiếp tục giảm kích thước không gian của bản đồ đặc trưng từ lớp tích chập thứ hai. Việc giảm kích thước này giúp mô hình tập trung vào các đặc trưng quan trọng hơn và giảm bớt độ phức tạp tính toán.
- Sau các lớp tích chập và gộp, mô hình sử dụng *lớp làm phẳng (Flatten)* để chuyển đổi các bản đồ đặc trưng 2D thành một vector 1D. Điều này là cần thiết vì các lớp tiếp theo trong mô hình là các lớp kết nối đầy đủ (Fully Connected Layers), yêu cầu dữ liệu đầu vào phải ở dạng vector.
- *Lớp Dropout* với tỷ lệ dropout là 0.5 được sử dụng tiếp theo. Lớp này ngẫu nhiên loại bỏ 50% số nơ-ron trong quá trình huấn luyện, giúp giảm sự phụ thuộc quá mức vào các nơ-ron cụ thể và do đó ngăn chặn hiện tượng overfitting. Dropout giúp mô hình trở nên mạnh mẽ hơn bằng cách học cách dựa vào tất cả các nơ-ron thay vì một số ít.
- Tiếp theo là *lớp kết nối đầy đủ (Dense)* với 512 nơ-ron và hàm kích hoạt ReLU. Lớp này học các đặc trưng phi tuyến từ dữ liệu đã được làm phẳng và dropout, cung cấp khả năng biểu diễn mạnh mẽ cho mô hình.
- Cuối cùng, *lớp đầu ra (Dense)* với 2 nơ-ron và hàm để đưa ra dự đoán cuối cùng là hàm softmax. Lớp này thực hiện phân loại nhị phân, với hai nhãn đầu ra của các lớp "không giống" và "giống". Chúng tôi sử dụng hàm softmax để đảm bảo rằng đầu ra của lớp này nằm trong khoảng từ 0 đến 1, phù hợp cho việc biểu diễn xác suất.

Tóm lại, mô hình CNN này được thiết kế để trích xuất và xử lý các đặc trưng từ hình ảnh đầu vào qua các lớp tích chập và gộp, sau đó chuyển đổi dữ liệu qua lớp làm phẳng và lớp dropout để ngăn chặn overfitting, cuối cùng đưa ra dự đoán phân loại thông qua các lớp kết nối đầy đủ. Kết quả đầu ra của mô hình là hai giá trị đại diện cho độ tương đồng giữa các file, giúp xác định liệu các file có giống nhau hay không. Nếu tệp có xác suất của nhãn nào lớn hơn thì sẽ được gán là nhãn đó, sau đó giá trị xác suất này sẽ được tính toán phù hợp cho phần thưởng (reward) của tác tử.

## 3.3. Luồng thực thi của mô hình RL

Cách thức hoạt động của mô hình RL dựa trên **Hình 3.1** như sau:

(1) Môi trường (environment) thực hiện lựa chọn 1 tập tin trong tập tin mã độc (PE) cần huấn luyện và thiết lập các tham số cơ bản cho môi trường

(2) Feature extractor tiến hành trích xuất vecotr đặc trưng từ tập tin và trả về state cho agent

(3) Agent nhận state và reward (ban đầu là 0) và tiến hành xác định hành động (action) sẽ sử dụng

(4) Sau khi action được áp dụng, tập tin thực thi mới được tạo ra, đồng thời tiếp tục được trích xuất đặc trưng để tạo state mới.

(5) Tập tin thực thi vừa tạo sẽ được bộ phát hiện mã độc phân loại và trả về kết quả cùng reward tương ứng

(6) Tiếp tục đưa file thực thi vào mô-đun kiểm chứng chức năng để kiểm chứng và trả về kết quả cùng reward tương ứng

(7) Dựa vào các reward ở bước (5), (6) để tiến hành tính reward tổng thể

(8) Kiểm tra điều kiện dừng khi huấn luyện trên 1 tập tin tức tập tin đó đã vượt qua được bộ phát hiện mã độc và mô-đun kiểm chứng chức năng nhận định là còn chức năng, hay đã đạt tới số lượt huấn luyện nhất định chưa, nếu thỏa mãn thì chuyển tiếp sang bước (9), ngược lại thì quay lại từ bước (3), sử dụng state và reward mới.

(9) Kiểm tra điều kiện dừng huấn luyện xem số tập huấn luyện đã đạt ngưỡng tối đa hay chưa, nếu không thỏa mãn điều kiện thì thực hiện lặp lại từ bước (1) với tập tin mới, ngược lại thì kết thúc quá trình huấn luyện.

# CHƯƠNG 4. THỰC NGHIỆM, ĐÁNH GIÁ VÀ THẢO LUẬN

## 4.1. Thực nghiệm

Trong phần này, nhóm sẽ tiến hành mô tả và thiết lập các thông số cho môi trường thực nghiệm trên máy ảo Windows 10 Home, 6GB RAM, 100GB bộ nhớ và mã nguồn được chạy trên Python phiên bản 3.11.9.

### 4.1.1. Bộ dữ liệu

#### *4.1.1.1. Thu thập dữ liệu*

*a) Huấn luyện và đánh giá mô hình CNN*

Chúng tôi trích xuất một phần từ tập dữ liệu Binkit [5] để làm dữ liệu huấn luyện cho mô hình CNN cho quá trình thực nghiệm và đánh giá khả năng nhận diện tương đồng của các chương trình. Tập dữ liệu Binkit bao gồm các tệp nhị phân từ các phiên bản phần mềm khác nhau và được chia thành 6 bộ nhỏ là NORMAL, SIZEOPT, NOINLINE, PIE, LTO và OBFUSCATION. Nhóm sẽ thực nghiệm với bộ Normal với 3 ứng dụng khác nhau với tổng cộng 195 tệp nhị phân với mỗi ứng dụng là 65 tệp được biên dịch với 8 kiến trúc khác nhau và 20 phiên bản trình biên dịch khác nhau, với 5 mức tối ưu hoá từ O0 đến O3, Os và Ofast.

*b) Huấn luyện và đánh giá mô hình RL*

Để thực nghiệm và đánh giá mô hình RL của nhóm, chúng tôi sử dụng bộ Dataset có tên "Malware Detection PE-Based Analysis Using Deep Learning Algorithm Dataset" [9] và bao gồm 5 loại virus khác nhau là Locker (300 files), Mediyes (1.450 files), Winwebsec (4.400 files), Zbot (2.100 files), Zeroaccess (690 files). Tập dữ liệu trên được thu thập từ 2 nguồn chính là https://virusshare.com và malicia-project.com cũng như đã được chứng thực bởi VirusTotal.

Trong quá trình thực nghiệm mô hình, nhóm trích xuất một phần của tập dữ liệu trên để sử dụng, trong đó: 1.000 files được sử dụng cho việc huấn luyện và 300 files cho việc đánh giá mô hình.

#### *4.1.1.2. Xử lý trích xuất vector đặc trưng*


*Hình 4.1. Trích xuất vector đặc trưng của các tập tin PE.*

Trong hoạt động việc huấn luyện và đánh giá mô hình CNN, việc trích xuất vector đặc trưng từ các tập tin PE là bước đầu tiên trong việc tiền xử lý dữ liệu. Trong đó, khi cần trích xuất vector đặc trưng, tập tin PE đầu vào sẽ được đưa vào kỹ thuật Prov2vex đã tối ưu để tạo thành vector có số chiều là 1.024. Mỗi vector đặc trưng sau đó sẽ được lưu dưới dạng tập tin txt để sử dụng trong các tác vụ xử lý sau này (**Hình 4.1**). Quy trình này được dùng cho cả quá trình huấn luyện riêng biệt mô-đun kiểm chứng chức năng và khi đã tích hợp vào mô hình RL để tạo biến thể mã độc.

### 4.1.2. Mô-đun phát hiện mã độc (Malware detector)

Để hiện thực bộ phát hiện mã độc trong mô hình RL tạo biến thể được đề xuất, nhóm tiến hành sử dụng 2 mô hình phát hiện mã độc khác nhau đã được huấn luyện trước là MalConv [3] và GradientBoosting [3] thay vì chỉ 1 loại duy nhất. Trong đó MalConv sẽ được sử dụng làm bộ phát hiện mã độc mục tiêu cho quá trình huấn luyện và bộ phát hiện mã độc còn lại sẽ được sử dụng để kiểm tra chất lượng của các biến thể đã tạo được từ mô hình RL.

Các bộ phát hiện này cho ra kết quả là một giá trị thể hiện mức độ độc hại của tập tin đầu vào cần kiểm tra. Giá trị này sẽ được so sánh với ngưỡng phát hiện để kết luận là tập tin độc hại hay lành tính. Dựa trên hiệu suất trong kết quả gần nhất của mỗi loại bộ phát hiện mã độc, nhóm sẽ thiết lập các giá trị ngưỡng khác nhau như trong **Bảng 4.1**.

*Bảng 4.1. Ngưỡng phát hiện tương ứng của các bộ phát hiện mã độc*

| Bộ phát hiện | Ngưỡng |
|---|---|
| MalConv | 0,9 |
| GradientBoosting | 0,9 |

### 4.1.3. Mô-đun kiểm chứng chức năng (Functionality validator)

Với bộ kiểm chứng chức năng đề xuất, chúng tôi thực hiện huấn luyện trên tập dữ liệu gồm hơn 25.000 tập tin. Bộ kiểm chứng chức năng sau khi được huấn luyện sẽ được tích hợp vào mô hình RL với đầu vào là 2 tập tin gồm 1 tập tin biến thể và 1 tập tin gốc tương ứng. Đầu ra sau khi xử lý sẽ là 1 mảng gồm 2 giá trị thể hiện cho mức độ tương đồng trong so sánh nhị phân đối với chức năng của 2 tập tin là giống hay khác nhau và kết quả sẽ là giá trị cao hơn. Ngoài ra, kết quả thu được sau khi kiểm chứng cũng sẽ được sử dụng để tính phần thưởng tương ứng với bộ kiểm chứng chức năng như **Công thức (4.1).**

$$R_{func} = \text{FUNCTIONALITY\_RESULT} * 10 \quad (4.1)$$

### 4.1.4. Trình tạo biến thể mã độc

Trong quá trình triển khai mô hình RL đề xuất, chúng tôi thiết lập giá trị cho các tham số sử dụng để thực hiện quá trình huấn luyện, được thể hiện ở **Bảng 4.2**. Ngoài ra nhóm có sử dụng một số thư viện liên quan để hỗ trợ quá trình xây dựng. Đầu tiên là chainnerl [10] thư viện này cung cấp nhiều thuật toán liên quan tới học sâu và quyết định lựa chọn thuật toán DiDQN để áp dụng cho agent. Thứ hai là thư viện LIEF [11], sử dụng để hỗ trợ quá trình trích xuất các đặc trưng từ tệp và chỉnh sửa file nhằm tạo ra các biến thể thông qua việc áp dụng các hành động được lựa chọn bởi agent lên file gốc.

*Bảng 4.2. Các tham số thiết lập trong mô hình RL*

| Tham số | Giá trị thiết lập | Mô tả |
|---|---|---|

| | | |
|---|---|---|
| episodes | 1.000 | Số tập huấn luyện agent trong môi trường |
| max_turns | 10 | Số hành động tối đa được áp dụng lên mỗi tập tin |
| adam_epsilon | 1e-2 | Giá trị nhiễu trong trình tối ưu Adam |
| detected_weight, distance_weight, functinality_weight | 0,33 | Trọng số của tương ứng của các reward |
| Tập dữ liệu huấn luyện | 1.000 | Số lượng tập tin dùng trong quá trình huấn luyện |
| Tập dữ liệu đánh giá | 300 | Số lượng tập tin dùng trong quá trình đánh giá mô hình |

## 4.2. Phương pháp đánh giá

### 4.2.1. Các kịch bản thực nghiệm

#### *4.2.1.1. Phần huấn luyện*

*a) Kịch bản 1: Đánh giá mô hình CNN trong so sánh tương đồng nhị phân*

Kịch bản này nhằm mục đích đánh giá mô hình CNN trong việc so sánh tương đồng của 2 tập tin nhị phân trước khi đưa vào mô hình RL. Để đánh giá hiệu quả của mô-đun kiểm chứng chức năng dựa trên mô hình CNN, chúng tôi sử dụng một tập dữ liệu gồm các tập tin nhị phân từ tập dữ liệu Binkit [5] ở phần 4.1.1.1. với thống kê như **Bảng 4.3**. Cụ thể, sau khi đã trích xuất các tệp thực thi này thành vector, chúng tôi kết hợp lần lượt hai vector từ 3 ứng dụng cppi, cflow và ccd2cue với nhau. Các tệp thực thi được tạo ra bằng cách biên dịch mã nguồn với các trình biên dịch, kiến trúc và tùy chọn tối ưu hóa như ở **Bảng 4.4**.

Chúng tôi dùng nguyên tắc cứ 2 tệp x và y sẽ được kết hợp theo các trường hợp (x, x), (x, y), (y, x), (y, y). Nói cách khác, cứ mỗi 2 tệp vector đầu vào sẽ tồn tại 4 tệp txt đầu ra, mỗi tệp chứa một vector đã được kết hợp với nhau. Từ đây, ta có cách tính số cặp vector được tạo ra với n vector đầu vào là $n^2$ với $\forall n \in Z^+$ và $n > 1$.

Đối với nhãn 1, sự kết hợp này diễn ra tuần tự với từng cặp tệp trong cùng một ứng dụng. Ngược lại đối với nhãn 0, sự kết hợp diễn ra ở 2 tệp khác nhau của hai ứng dụng khác nhau. Sau đó các tệp được kết hợp được chuyển thành dạng ảnh 3 kênh màu xám ở phần **3.2.5.2** với thống kê như **Bảng 4.5.**

*Bảng 4.3. Thông tin tập dữ liệu được trích từ Binkit được sử dụng*

| STT | Tên ứng dụng | Phiên bản | Số lượng tệp |
|---|---|---|---|
| 1 | Cppi | 1.18 | 65 |
| 2 | Cflow | 1.7 | 65 |
| 3 | Ccd2cue | 0.5 | 65 |

*Bảng 4.4. Các phiên bản trình biên dịch, kiến trúc và mức độ tối ưu hóa*

| Trình biên dịch | | Kiến trúc | Mức tối ưu hoá |
|---|---|---|---|
| gcc | clang | x86_32 | O0 |
| gcc-4.9.4 | clang-4.0.0 | x86_64 | O1 |
| gcc-5.5.0 | clang-5.0.2 | arm_32 | O2 |
| gcc-6.4.0 | clang-6.0.1 | arm_64 | O3 |
| gcc-6.5.0 | clang-7.0.1 | mips_32 | Os |
| gcc-7.3.0 | clang-8.0.0 | mips_64 | Ofast |
| gcc-8.2.0 | clang-9.0.1 | mipseb_32 | |
| gcc-8.5.0 | clang-10.0.1 | mipseb_64 | |
| gcc-9.4.0 | clang-11.0.1 | | |
| gcc-10.3.0 | clang-12.0.1 | | |
| gcc-11.2.0 | clang-13.0.0 | | |

*Bảng 4.5. Tập dữ liệu đánh giá mô-đun kiểm chứng chức năng*

| Nhãn | Huấn luyện | Kiểm tra/Điều chỉnh | Tổng |
|---|---|---|---|
| 0 (Không giống) | 7.605 | 2.535 | 12.675 |
| 1 (Giống) | 7.605 | 2.535 | 12.675 |

Để huấn luyện CNN với dữ liệu đã được chuẩn bị, chúng tôi tạp 3 bộ dữ liệu ảnh tương đương với 3 trường hợp tạo ảnh. Với mỗi bộ có số tệp ảnh là 25.350,

tổng 3 bộ dữ liệu ảnh là 76.050 tệp. Mỗi bộ được chia thành 2 nhãn 0 và 1 với số tệp cho nhãn 0 và nhãn 1 là 12.675. Số lượng tệp của mỗi nhãn sẽ được chia thành các tập Huấn luyện, Kiểm tra và Điều chỉnh theo tỉ lệ 60:20:20.

Chúng tôi thực hiện kiểm tra hiệu suất của CNN trong 3 trường hợp ảnh có kênh màu thứ 3 lần lượt là các giá trị 0, giống với kênh màu đầu tiên, giống với kênh màu thứ 2.

*b) Kịch bản 2: Huấn luyện mô hình RL trong trường hợp không có bộ kiểm chứng chức năng*

Kịch bản này thực hiện đánh giá mô hình RL đề xuất mà không có bộ kiểm chứng chức năng tham gia trong quá trình huấn luyện. Mục đích của kịch bản là nhằm so sánh và đánh giá hiệu suất của mô hình với trường hợp khi có sự tham gia của bộ phận kiểm chứng chức năng, từ đó đánh giá được tầm quan trọng của bộ kiểm chứng chức năng trong mô hình RL của nhóm.

Tập dữ liệu nhóm sử dụng trong kịch bản này gồm 1.000 file bao gồm 5 loại mã độc khác nhau với tên gọi và số lượng được thể hiện ở **Bảng 4.6.**

*Bảng 4.6. Tập dữ liệu dùng trong huấn luyện mô hình RL*

| **Loại mã độc** | **Số lượng tập tin** |
|---|---|
| Locker | 200 |
| Mediyes | 200 |
| Winwebsec | 200 |
| Zbot | 200 |
| Zeroaccess | 200 |

*c) Kịch bản 3: Huấn luyện mô hình RL trong trường hợp có bộ phận kiểm chứng chức năng*

Kịch bản này thực hiện đánh giá mô hình RL đề xuất khi có sự tham gia của bộ kiểm chứng chức năng trong quá trình huấn luyện. Hiệu suất của mô hình sau khi huấn luyện sẽ được đánh giá và so sánh với kết quả từ kịch bản không có sự tham gia của bộ kiểm chứng chức năng, từ đó đánh giá được hiệu suất cho tổng thể

mô hình huấn luyện cũng như tầm quan trọng và tác động của bộ kiểm chứng chức năng trong quá trình tạo biến thể mã độc.

Trong quá trình triển khai kịch bản, nhóm tiến hành sử dụng lại bộ dữ liệu đã được áp dụng cho kịch bản không có sự tham gia của bộ kiểm chứng chức năng. Với việc sử dụng cùng một tập dữ liệu sẽ giúp cho việc khi so sánh các kết quả giữa các kịch bản sẽ trở nên khách quan hơn.

#### *4.2.1.2. Phần đánh giá*

*a) Kịch bản 4: Đánh giá khả năng tạo biến thể của mô hình trong quá trình khi huấn luyện*

Kịch bản này chúng tôi thực hiện sử dụng các biến thể hoàn chỉnh mà mô hình đã tạo được từ kịch bản 2 và 3 cùng với các tệp mã độc gốc tương ứng để tiến hành đánh giá chúng thông qua bộ phát hiện mã độc Gradient Boosting với ngưỡng phát hiện được thiết lập như ở **Bảng 4.1**. Mục đích của kịch bản này là đánh giá mức độ hiệu quả của các phương pháp biến đổi đối với các bộ phát hiện mã độc khác nhau cũng như khả năng học hỏi của mô hình huấn luyện.

*b) Kịch bản 5: Đánh giá khả năng tạo biến thể của mô hình sau khi huấn luyện*

Kịch bản này nhóm chúng tôi sẽ thực hiện tạo các biến thể dựa trên 2 agent đã huấn luyện được ở kịch bản 2 và 3 với tập dữ liệu gồm 300 files bao gồm 5 loại malware với tên và số lượng được thể hiện ở **Bảng 4.7**. Sau khi tạo các biến thể, chúng tôi sẽ thực hiện đánh giá chúng cũng với 2 bộ phát hiện mã độc và các thông số như kịch bản 4, từ đó đánh giá hiệu suất của mô hình RL trong 2 trường hợp có và không sự tham gia của bộ kiểm chứng chức chức năng, từ đó đánh giá được tầm quan trọng của phương pháp kiểm chứng chức năng bằng so sánh tương đồng nhị phân mà nhóm đề xuất.

*Bảng 4.7. Tập dữ liệu dùng để đánh giá mô hình huấn luyện*

| **Loại mã độc** | **Số lượng (files)** |
|---|---|
| Locker | 60 |
| Mediyes | 60 |

| | |
|---|---|
| Winwebsec | 60 |
| Zbot | 60 |
| Zeroaccess | 60 |

### 4.2.2. Thông số đánh giá

Để đánh giá hiệu quả của mô hình kiểm chứng chức năng cũng như mô hình tạo biến thể, chúng tôi sử dụng các thông số như sau.

- **Accuracy**

Accuracy là chỉ số đánh giá hiệu suất của mô hình phân loại, phản ánh tỷ lệ dự đoán đúng trên tổng số dự đoán. Chỉ số này càng cao thì tỉ lệ dự đoán chính xác của mô hình phân loại càng cao. Với TP là Số trường hợp dương tính được dự đoán đúng, TN là Số trường hợp âm tính được dự đoán đúng, FP là Số lượng mẫu âm tính nhưng bị dự đoán sai thành dương tính, FN là Số lượng mẫu dương tính nhưng bị dự đoán sai thành âm tính, thông số Accuracy được tính như công thức (4.2).

$$Accuracy = \frac{TP + TN + FP + FN}{TP + TN} \quad (4.2)$$

Thông số này được dùng để đánh giá hiệu quả của mô-đun kiểm chứng chức năng trong Kịch bản 1 dựa việc gán nhãn tương đồng (1) hoặc không tương đồng (0) chính xác cho các cặp vector đại diện cho các tập tin giống và khác nhau. Ngoài ra thông số này còn được sử dụng để đánh giá hiệu quả của các mô hình phát hiện mã độc trước các mẫu biến thể được tạo.

- **Tỉ lệ qua mặt – Evasion rate (ER)**

ER là tỷ lệ qua mặt thành công của biến thể khi chưa tích hợp bộ kiểm chứng chức năng, được tính với công thức (4.3), trong đó *e* là Số biến thể lẩn tránh thành công bộ phát hiện mã độc, *M* là tổng số tệp được kiểm tra.

$$ER = \frac{e}{M} \quad (4.3)$$

- **Tỉ lệ qua mặt còn chức năng - Evasive Rate with Functionality (ERFP)**

ERFP là tỷ lệ lẩn tránh thành công của biến thể khi sau tích hợp bộ kiểm chứng chức năng, được tính với công thức (4.4), trong đó $e_p$ là Số biến thể lẩn tránh thành công bộ phát hiện mã độc và bộ kiểm chứng chức năng (đảm bảo giữ được chức năng), *M* là Tổng số tệp được kiểm tra.

$$ERFP = \frac{e_p}{M} \tag{4.4}$$

## 4.3. Kết quả thực nghiệm

### 4.3.1. Phần huấn luyện

#### *4.3.1.1. Kịch bản 1: Đánh giá mô hình CNN*

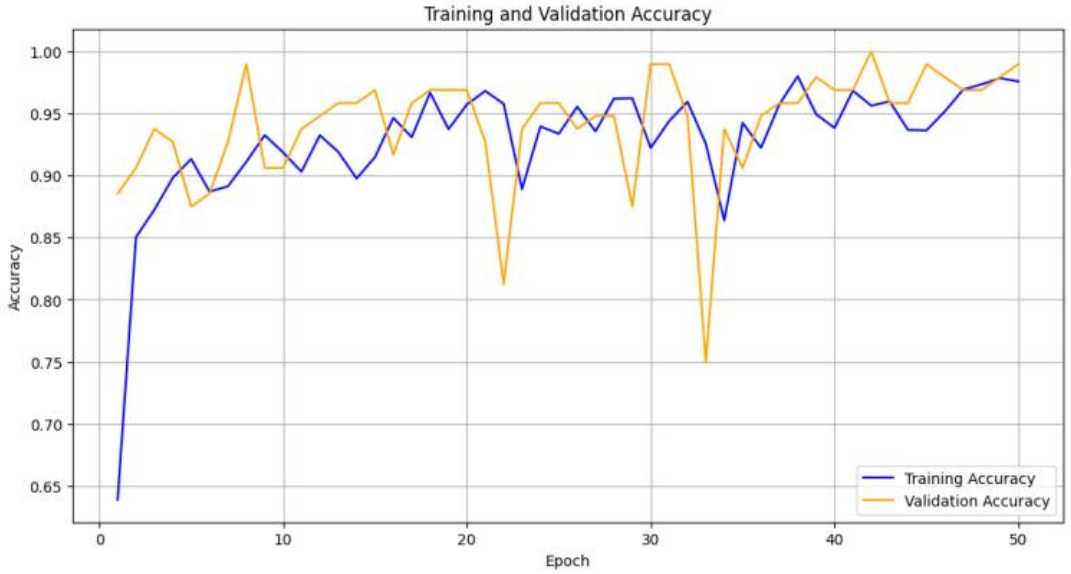
Sau khi huấn luyện mô hình CNN với bộ dữ liệu đã nêu ở 4.2.2a), chúng tôi thu về kết quả như **Bảng 4.8** và trực quan hóa kết quả huấn luyện ở **Hình 4.2**.

Chúng tôi sử dụng epoch cho phần huấn luyện là 50, thực hiện ba trường hợp cho ra ba kết quả mô hình huấn luyện. Kết quả tập Test cho thấy, accuracy của trường hợp thứ ba là kênh màu thứ 3 của ảnh trùng với kênh màu thứ hai cho kết quả cao nhất với trên 0,99, trường hợp thứ hai cho kết quả hơn 0,98 và thấp nhất là trường hợp thứ nhất với accuracy là hơn 0,84. Với ba kết quả này, chúng tôi lựa chọn kết quả của mô hình kênh màu thứ 3 trùng với kênh 2 để đưa vào mô hình RL.

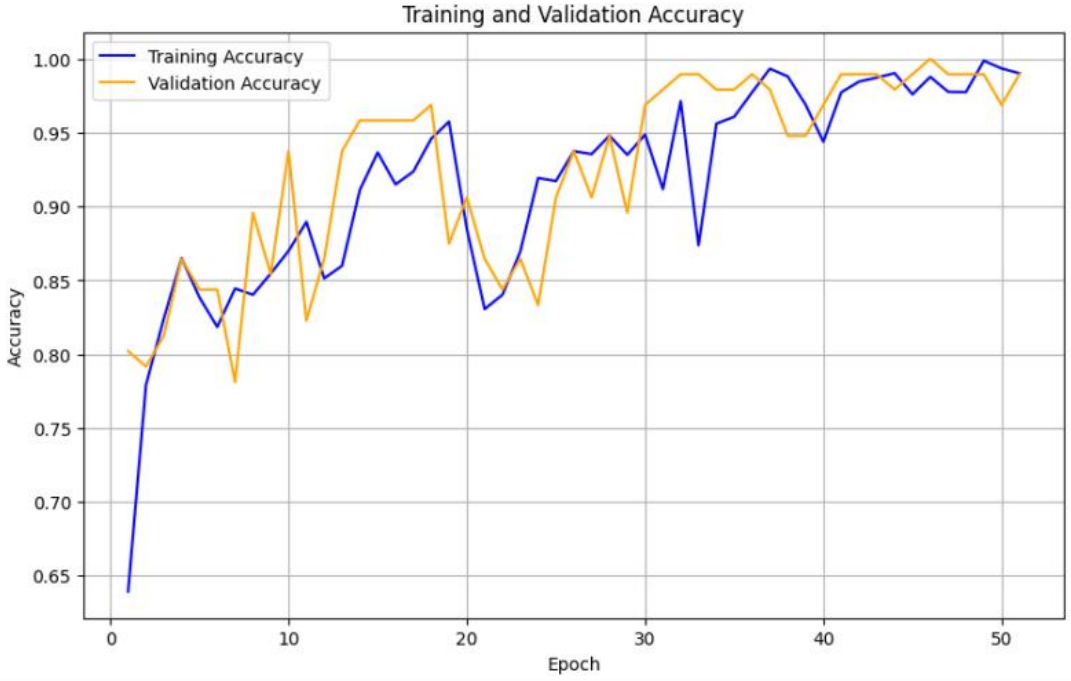
*Bảng 4.8. Kết quả thực nghiệm đánh giá mô hình CNN*

| **Trường hợp** | **Kênh màu thứ 3 của ảnh** | **Accuracy** |
|---|---|---|
| 1 | Giá trị 0 | 0,845312 |
| 2 | Giống với kênh màu đầu tiên | 0,985000 |
| 3 | Giống với kênh màu thứ hai | 0,993437 |

Để kiểm tra xem các mô hình có bị overfitting hay không, chúng tôi thông qua kết quả huấn luyện từ tập train và tập validation qua 50 epoch như sau.




a) Kênh thứ 3 là các giá trị 0




b) Kênh thứ 3 trùng với kênh thứ 1

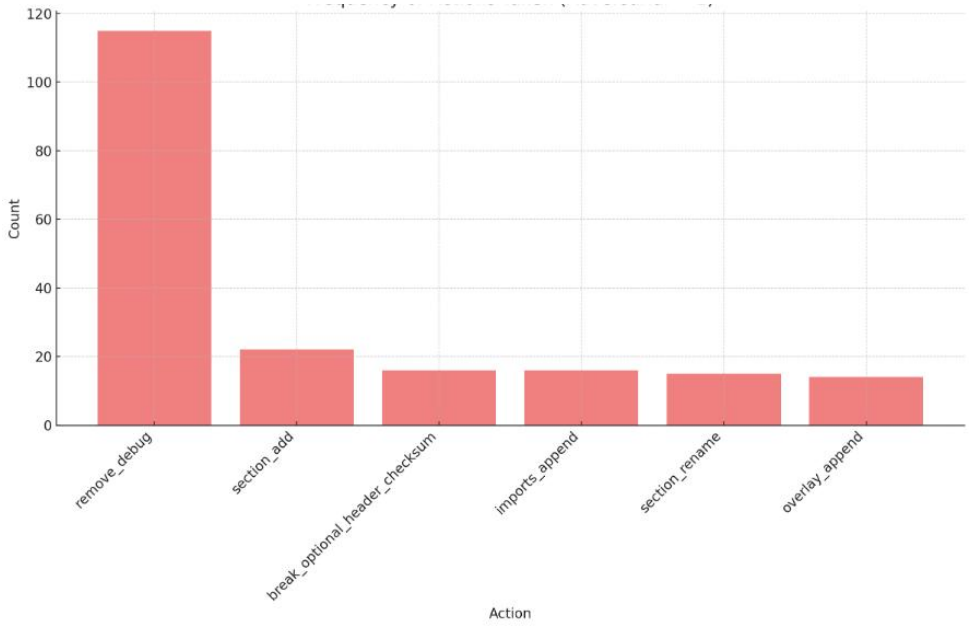

c) Kênh thứ 3 trùng với kênh thứ 2

*Hình 4.2. Kết quả huấn luyện và kiểm tra mô hình CNN với các cách gộp ảnh khác nhau.*

Kết quả cho thấy, cả ba mô hình về các cách gộp ảnh khác nhau đều không bị overfitting vì khoảng cách của giá trị Accuracy của tập huấn luyện và tập điều chỉnh được giữ ở mức không quá cao qua từng epoch và ở những epoch cuối khoảng cách này được rút lại ngắn nhất so với những epoch đầu. Trong ba trường hợp, thì trường hợp mô hình được huấn luyện bởi tập dữ liệu từ hình ảnh có kênh màu thứ ba trùng với kêu màu thứ hai là có giá trị Accuracy ổn định nhất so với hai trường hợp còn lại.

#### *4.3.1.2. Kịch bản 2: Đánh giá mô hình RL khi chưa tích hợp mô-đun kiểm chứng chức năng*

Việc huấn luyện mô hình RL trong trường hợp không sử dụng bộ kiểm chứng chức năng được hoàn thành với thời gian là hơn 1 giờ 7 phút. Kết thúc quá trình huấn luyện, agent thành công tạo ra được 123 biến thể vượt qua được bộ phát hiện malware, đạt tỉ lệ 123/1000 (12,3%). Các hành động được agent áp dụng và tần suất của chúng được thể hiện ở biểu đồ trong **Hình 4.3**.




*Hình 4.3. Thống kê số lượng các action và tần suất sử dụng trong các biến thể khi không có bộ kiểm chứng chức năng.*

Dựa vào kết quả trong biểu đồ trong biểu đồ ta nhận thấy rằng agent có xu hướng ưu tiên sử dụng action remove_debug với số lần sử dụng lên tới 115, trong khi đó các hành động còn lại được sử dụng là section_add, break_optional_header_checksum, imports_append, section_rename, overlay_append có số lần sử dụng tương đương nhau và khá ít, khoảng 14-22 lần. Lý giải cho việc remove_debug được sử dụng nhiều nhất có thể là do debug là nơi chứa các thông tin quan trọng của tập tin như tên hàm, biến hay cách mà tệp được xây dựng, các thông tin này thường được các chuyên gia bảo mật hoặc các phần mềm phát hiện mã độc sử dụng để phân tích và nhận diện mã độc nên việc xóa debug sẽ khiến cho các công cụ phân tích mã độc gặp khó khăn hoặc không thể nắm bắt cấu trúc cũng như các thông tin cần thiết để nhận diện mã độc. Một số hành động khác như section_append, signature_remove, và upx_pack/ upx_unpack sẽ không được agent sử dụng do gây ra lỗi trong quá trình tạo biến thể.

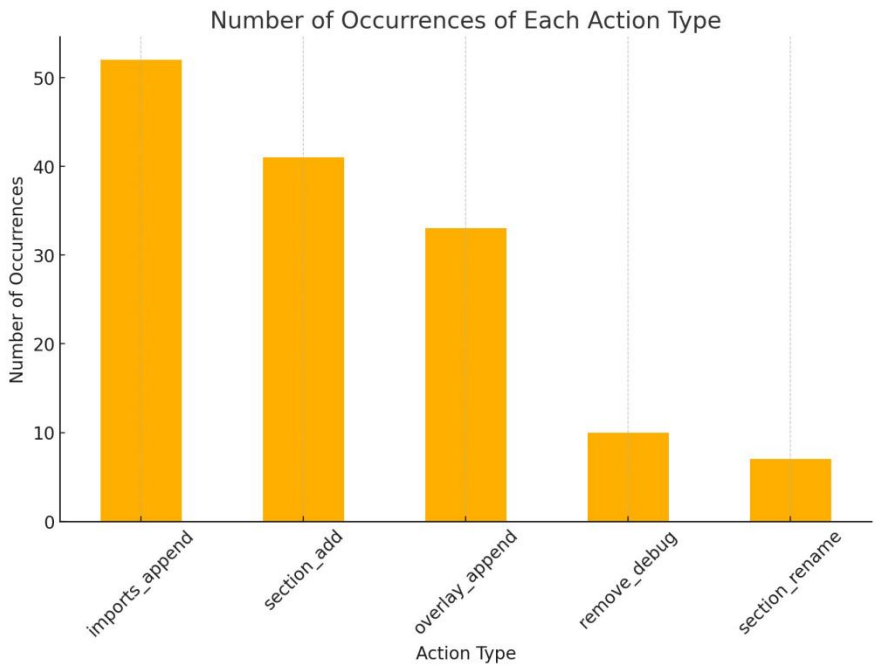
#### *4.3.1.3. Kịch bản 3: Đánh giá mô hình RL sau khi tích hợp mô-đun kiểm chứng chức năng*

Thời gian tiêu tốn cho việc hoàn thành huấn luyện mô mình RL trong trường hợp có sự tham gia của bộ kiểm chứng chức năng với cùng một tập dữ liệu là 8 giờ 15 phút, tức gấp 8 lần so với khi không có bộ kiểm chứng chức năng. Kết quả sau khi huấn luyện mô hình thì agent thành công trong việc tạo ra 90 biến thể vượt qua được cả hai bộ kiểm chứng chức năng và phát hiện mã độc, đạt tỉ lệ 90/1.000 (9%) thấp hơn kịch bản khi không có bộ kiểm chứng chức năng (12,3%). Điều này cho thấy rằng bộ kiểm chứng chức năng đóng vai trò quan trọng trong việc đảm bảo tính chính xác và độ tin cậy của mô hình, nhưng cũng đồng thời làm tăng đáng kể thời gian huấn luyện.

Việc tăng thời gian huấn luyện có thể xuất phát từ nhiều nguyên nhân, bao gồm việc tạo các vector đặc trưng biểu diễn thông tin cho các biến thể vừa tạo hay quá trình so sánh với biến thể gốc gặp vấn đề liên quan tới xử lý. Những bước này tuy làm chậm quá trình huấn luyện nhưng lại cần thiết để đảm bảo rằng mô hình học được không chỉ tối ưu về mặt hiệu suất mà còn tuân thủ các yêu cầu về chức năng và an toàn.

Theo kết quả thu được dựa trên **Hình 4.4** về các hành động và tần suất sử dụng của chúng trong quá trình huấn luyện, ta dễ dàng nhận thấy agent lần này có xu hướng ưu tiên lựa chọn 2 hành động là imports_append với 52 lần và section_add với 41 lần. Trong khi đó các hành động còn lại thì thấp hơn đáng kể, đặc biết là hành động remove_debug (10 lần) không còn được ưu tiên sử dụng sau khi mô hình RL tích hợp thêm bộ kiểm chứng chức năng. Lý giải việc agent thay đổi hành động ưu tiên từ remove_debug sang imports_append và section_add, nguyên nhân chính có thể là do việc gỡ bỏ thông tin debug có thể làm cho có thể làm thay đổi cách mã độc hoạt động hoặc loại bỏ các thành phần cần thiết, dẫn đến mất đi hoặc thay đổi các chức năng quan trọng. Trong khi đó thì ngược lại, hành động imports_append giúp thêm vào các hàm cần thiết, giúp đảm bảo các chức năng của mã độc vẫn hoạt động như mong muốn. Về hành động section_add thì tập

tin sẽ được thêm section, đặc biệt là các section không liên quan hoặc vô hại, các section được thêm này không được sử dụng trong quá trình thực thi file, do đó không làm ảnh hưởng tới chức năng gốc của file, đồng thời còn có thể làm tăng thêm độ phức tạp trong cấu trúc file, từ đó các bộ nhận diện mã độc sẽ khó khăn hơn trong việc phân tích.




*Hình 4.4. Thống kê số lượng các action và tần suất sử dụng trong các biến thể khi có bộ kiểm chứng chức năng.*

### 4.3.2. Phần đánh giá

#### *4.3.2.1. Đánh giá các biến thể được tạo ra trong quá trình huấn luyện*

Sau khi kết thúc quá trình huấn luyện mô hình RL với sự tham gia đầy đủ của bộ phát hiện mã độc và bộ kiểm chứng chức năng, 90 biến thể thành công được tạo ra được sử dụng cùng với các tập mã độc gốc tương ứng của chúng lần lượt được kiểm tra với các bộ phát hiện mã độc khác bên cạnh MalConv là Gradient Boosting, kết quả cuối cùng được thể hiện ở **Bảng 4.9**.

Dựa vào các kết quả, ta có thể thấy sự khác biệt rõ rệt giữa tỉ lệ vượt qua của các biến thể mã độc so với các mã độc gốc khi được kiểm tra bởi các bộ phát hiện mã độc khác nhau. Cụ thể:

- **MalConv**: Các biến thể mã độc có tỉ lệ vượt qua thành công lên đến 94%, trong khi các mã độc gốc không thể vượt qua được. Điều này chứng tỏ rằng các biến thể mã độc được tạo ra bởi quá trình huấn luyện mô hình RL với bộ phát hiện MalConv có khả năng né tránh phát hiện rất hiệu quả. Tỉ lệ cao này cũng phần nào thể hiện độ tin cậy của mô hình RL trong việc tạo ra các biến thể mã độc phức tạp, khó bị phát hiện hơn.
- **Gradient Boosting**: Tỉ lệ các biến thể mã độc vượt qua bộ kiểm chứng chức năng là 72%, cao hơn so với tỉ lệ 67% của các mã độc gốc. Dù mức độ chênh lệch không quá lớn, nhưng vẫn cho thấy rằng các biến thể mã độc đã được cải thiện khả năng né tránh phát hiện. Sự khác biệt này cho thấy các biến thể mã độc không chỉ hiệu quả khi đối phó với MalConv mà còn có khả năng vượt qua các bộ phát hiện khác, mặc dù độ hiệu quả không cao bằng.

*Bảng 4.9. So sánh tỉ lệ vượt qua các bộ phát hiện mã độc giữa các mã độc gốc và biến thể của chúng trong quá trình huấn lyện*

| | **Tỉ lệ vượt qua bộ phát hiện mã độc (%)** | |
|---|---|---|
| | **Biến thể mã độc** | **Mã độc gốc tương ứng** |
| MalConv | 94% | 0% |
| Gradient Boosting | 72% | 67% |

#### *4.3.2.2. Đánh giá khả năng tạo biến thể mã độc của mô hình sau khi được huấn luyện*

**Kết quả:**

- Với agent từ mô hình RL khi không tích hợp bộ kiểm chứng chức năng: số biến thể thành công vượt qua được bộ phát hiện mã độc là 84, đạt tỉ lệ né tránh bị phát hiện là 84/300 (28%).

- Với agent từ mô hình RL có tích hợp kiểm chứng chức năng: agent thành công tạo ra 62 trên tổng số 300 biến thể vượt qua được bộ kiểm chứng chức năng và bộ phát hiện mã độc, đạt tỉ lệ né tránh bị phát hiện là 62/300 (20,3%)

**Nhận xét:**

- Đối với MalConv: Sau khi tích hợp thêm bộ kiểm chứng chức năng, số lượng biến thể thành công vượt qua cả 2 bộ phát hiện mã độc và kiểm chứng chức năng tạo ra giảm đáng kể (62 so với 84). Điều này một lần nữa cho thấy rằng việc bổ sung bộ kiểm chứng chức năng làm cho quá trình tạo biến thể trở nên khó khăn hơn, vì mỗi biến thể cần phải vượt qua cả hai bộ để được xem là hợp lệ.
- Đối với Gradient Boosting:
    - Với mô hình sau khi chưa tích hợp kiểm chứng chức năng, 148/300 biến thể được tạo ra có khả năng vượt qua được bộ phát hiện mã độc này, tương ứng với tỉ lệ 49.3%, trong khi ở trường hợp có sự tham gia của bộ kiểm chứng chức năng, 99/300 biến thể vượt qua được cả 2 bộ phát hiện, đạt tỉ lệ 33%.
    - Từ kết quả trên và dựa vào **Bảng 4.10** để so sánh, chúng tôi nhận thấy dù là bộ phát hiện nào thì hiệu suất của mô hình có tích hợp bộ kiểm chứng chức năng vẫn luôn thấp hơn so với trường hợp còn lại. Tuy nhiên điều đó cũng chứng minh rằng bộ kiểm chứng chức năng không chỉ đóng vai trò quan trọng trong việc đảm bảo chức năng thực thi của biến thể mà còn góp phần giúp mô hình RL tạo ra các biến thể chất lượng hơn.

*Bảng 4.10. Khả năng vượt qua các bộ phát hiện mã độc của các biến thể được tạo bởi mô hình sau khi được huấn luyện*

| **Bộ phát hiện mã độc** | **ER** | **ERFP** |
|---|---|---|
| MalConv | 28% | 20,3% |
| Gradient Boosting | 49.3% | 33% |

# CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 5.1. Kết luận

Trước sự phát triển và thay đổi không ngừng của mã độc cũng như các biến thể của chúng thì việc phải liên tục thay đổi và cải tiến các phương pháp phòng chống, nhận diện mã độc luôn là một vấn đề quan trọng và dành được nhiều sự quan tâm. Nhận thức được vấn đề đó, nhóm chúng tôi đề xuất phương pháp tạo biến thể mã độc nhằm vượt qua được các bộ phát hiện mã độc, điều mà mã độc ban đầu không thể. Đề tài của nhóm chúng tôi tập trung nghiên cứu và áp dụng mô hình học sâu (RL) trong việc tạo ra các biến thể mới có thể đánh lừa được các bộ phát hiện mã độc rằng đó chỉ là các tập lành tính nhưng vẫn đảm bảo giữ nguyên được các chức năng thực thi của mã độc gốc. Từ các kết quả sau khi tiến hành thực nghiệm với bộ dữ liệu được thu thập từ VirusShare và malicia-project, chúng tôi rút ra được một số kết luận về các ưu nhược điểm của mô hình và phương pháp như sau:

**Ưu điểm:**

- Đối với mô hình RL: việc áp dụng các hành động phù hợp lên mã độc kết hợp với kiểm chứng chức năng thực thi thông qua so sánh tương đồng nhị phân thực sự có hiệu quả, hiệu suất của mô hình kể cả trong lúc huấn luyện cũng như lúc đánh giá đều cho ra kết quả khá cao đã chứng minh điều này. Đặc biệt hơn, các biến thể thành công tạo ra không chỉ thành công trong việc vượt qua được bộ phát hiện MalConv một cách dường như tuyệt đối, mà còn hoạt động tốt đối với một bộ phát hiện mã độc khác là Gradient Boosting.
- Riêng đối với phương pháp so sánh tương đồng nhị phân: việc bỏ đi một bước trong phương pháp gốc đã giúp cho việc tạo vector nhanh hơn đáng kể , giúp tiết kiệm một phần thời gian cho mô hình RL. Việc áp dụng mô hình CNN huấn luyện các ảnh tạo từ các vector đã cho hiệu suất rất cao, chứng

minh đây là mô hình phù hợp để dánh giá độ tương đồng nhị phân từ các tệp thực thi khi được kết hợp với phương pháp prov2vex đã tối ưu.

**Nhược điểm:**

- Đối với mô hình RL: trong quá thực nghiệm thì việc một số hành động trong không gian hành động gây ra lỗi cho biến thể với tần suất cao gây tổn thất tới số lượng biến thể đáng lẽ có thể tạo được, hay như số số lượng bộ phát hiện mã độc dùng cho việc kiểm tra các biến thể mã độc chưa được nhiều, dẫn tới việc các kết quả khi đánh giá mô hình chưa thực sự đa dạng.
- Riêng đối với phương pháp so sánh tương đồng nhị phân: để thực hiện phương pháp này, đòi hỏi cần phải qua nhiều công đoạn trong phần tiền xử lí dữ liệu. Từ việc lấy vector từ các tệp nhị phân, sau đó phải gộp thành các cặp giống và khác nhau đến việc biến đổi các vector này thành ảnh đã chiếm thời gian nhiều nhất trong việc huấn luyện mô hình CNN. Bên cạnh đó, dữ liệu để huấn luyện mô hình CNN này là các hình ảnh 3 kênh màu xám là loại dữ liệu mà chưa được thử nghiệm trong bất kì nghiên cứu nào trước đây, vì vậy, ngoài CNN, chúng tôi không đảm bảo loại dữ liệu này sẽ phù hợp với những mô hình phân loại khác.

## 5.2. Hướng phát triển

Từ các công việc, kết quả, đánh giá thu được, chúng tôi nhận thấy đề tài của mình vẫn còn khuyết thiếu nhưng cũng mang nhiều tiềm năng khai thác và phát triển hơn thế nữa. Sau đây là một số định hướng phát triển chúng tôi đề xuất:

- Thứ nhất, phương pháp prov2vex sau khi tối ưu hoá thời gian chạy đã giảm đáng kể thời gian trích xuất vector so với ban đầu. Tuy nhiên, việc bỏ đi bước chuẩn hoá thành dạng chính tắc phần nào làm lệch tính cốt lõi của phương pháp là phải thực hiện pháp băm trên dạng chính tắc này. Nên thế, chúng tôi muốn giữ nguyên các bước trong phương pháp này, đồng thời cải tiến kết hợp tối ưu thời gian chạy. Đây là hướng phát triển khá tốt và cũng đầy thử thách.

- Thứ hai, thay vì sử dụng CNN, chúng tôi có thể sử dụng SNN (Siamese Neural Network). Vì SNN có thể huấn luyện hiệu quả với tập dữ liệu nhỏ và nó không yêu cầu mỗi mẫu phải được gán nhãn rõ ràng, mà chỉ cần biết cặp dữ liệu có giống nhau hay không, điều này giúp giảm bớt công việc gán nhãn trong một số trường hợp. Còn CNN thường yêu cầu lượng dữ liệu lớn để đạt hiệu suất cao và mỗi mỗi mẫu dữ liệu phải được gán nhãn chính xác cho từng lớp.

- Thứ ba, nếu hướng phát triển đầu tiên đã đề ra có kết quả tốt, chúng tôi không cần dùng đến học máy trong mô-đun kiểm chứng chức năng. Vì lúc này, khi các tệp nhị phân đã trải qua đầy đủ các bước trong phương pháp prov2vex, các vector tạo thành có thể được đánh giá tương đồng bằng các thuật toán tính toán độ tương đồng như Jaccard hoặc Cosin trong công trình [5]. Điều này sẽ tiết kiệm thời gian hơn dùng mô hình học máy để tính toán vì các thuật toán này khá đơn giản.

- Thứ tư, phương pháp prov2vex đã tối ưu có thể được giữ nguyên, nhưng theo chúng tôi đã kiểm chứng thì thuật toán Jaccard và Cosin không còn khả năng đánh giá độ tương đồng giữa các vector này, nên tìm ra một thuật toán, một kỹ thuật khác để thay thế vai trò của hai thuật toán trên có thể là một hướng phát triển tiềm năng khác.

- Thứ năm, mở rộng không gian các hành động có thể áp dụng lên các mã độc để tạo biến thể, từ đó làm đa dạng hơn các biến thể và tăng khả năng lẩn tránh khỏi các bộ phát hiện.

- Thứ sáu, tăng số bộ phát hiện mã độc trong mô hình RL, thay vì agent phải tối ưu quyết định để vượt qua một thì giờ đây phải vượt qua được nhiều bộ phát hiện hơn, việc này tuy đòi hỏi nhiều thời gian hơn nhưng đồng thời cũng giúp cho hiệu suất mô hình trở nên hiệu quả hơn.

- Thứ bảy, mở rộng số lượng các loại tệp mà mô hình có thể tạo biến thể thay vì chỉ giới hạn ở loại tệp thực thi trên Windows (.exe), từ đó mô hình RL có thể tạo ra các biến thể có thể qua mặt được nhiều bộ phát hiện mã độc và hoạt động được trên đa dạng môi trường hơn.

- Thứ tám, nên tích hợp thêm phương pháp kiểm chứng chức năng bằng thông qua môi trường Cuckoo trong trường hợp phương pháp kiểm chứng chức năng bằng so sánh tương đồng nhị phân cho ra dự đoán không quá cao (dự đoán tương đồng chênh lệch không quá lớn so với mức 50%), lúc này môi trường Cuckoo sẽ được sử dụng để kiểm tra thêm một lần nữa xem liệu biến thể tạo ra có hoạt động chính xác hay không.

# TÀI LIỆU THAM KHẢO


[1] J. Boutsikas, M. E. Ere, C. Varga, E. Raff, C. Matuszek và C. Nicholas, "Evading Malware Classifiers via Monte Carlo Mutant Feature Discovery," trong *arXiv:2106.07860*, 2021.

[2] H. S. Anderson, A. Kharkar, B. Filar, D. Evans và P. Roth, "Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning," trong *arXiv:1801.08917*, 2018.

[3] H. D. T. Thu, T. D. Phan, H. L. Anh, L. N. Duy, K. N. Hoang và V.-H. Pham, "A Method of Mutating Windows Malwares using Reinforcement Learning with Functionality Preservation," trong *Proceedings of the 11th International Symposium on Information and Communication Technology*, 2022.

[4] N. Shalev và N. Partush, "Binary Similarity Detection Using Machine Learning," trong *Proceedings of the 13th Workshop on Programming Languages and Analysis for Security*, 2018.

[5] H. Guo, S. Huang, C. Huang, M. Zhang, Z. Pan, F. Shi, H. Huang, D. Hu và X. Wang, "A Lightweight Cross-Version Binary Code Similarity Detection Based on Similarity and Correlation Coefficient Features," *IEEE Access,* tập 8, 2020.

[6] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou và H. Huang, "Evading Anti-Malware Engines With Deep Reinforcement Learning," *IEEE Access,* tập 7, 2019.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage và A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine ,* tập 34, 2017.

[8] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro và C. Nicholas, "Malware Detection by Eating a Whole EXE," trong *The Workshops of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[9] A. P. Tuan, A. T. H. Phuong, N. V. Thanh và T. N. Van, "Malware Detection PE-Based Analysis Using Deep Learning Algorithm Dataset," 2018. Available: https://figshare.com/articles/dataset/Malware_Detection_PE-Based_Analysis_Using_Deep_Learning_Algorithm_Dataset/6635642.

[10] Y. Fujita, P. Nagarajan và T. Kataoka, "ChainerRL: A Deep Reinforcement Learning Library," *Journal of Machine Learning Research,* tập 22, 2021.

[11] Quarkslab, “LIEF - Library to instrument executable formats,” 2017. [Trực tuyến]. Available: https://github.com/lief-project.

# PHỤ LỤC

## A. Phương pháp gộp ảnh từ 2 vector đặc trưng

Dưới đây là đoạn mã giúp thêm vào kênh thứ ba theo kênh thứ nhất và kênh thứ 2 của ảnh:

```python
def vector_to_image(vector, output_image_path, image_shape):

    # Chia vector thành 2 ma trận 32x32
    image_channels = np.split(vector, 2)

    # Chuyển đổi mỗi ma trận thành ảnh 32x32,
    image = np.stack([channel.reshape((32, 32)) for channel in image_channels], axis=-1)
    image = np.concatenate((image, image[:, :, 0:1]), axis=2)  #với kênh thứ 3 là bản copy của kênh 1
    #image = np.concatenate((image, image[:, :, 1:2]), axis=2)  #với kênh thứ 3 là bản copy của kênh 2

    # Chuyển đổi kiểu dữ liệu sang uint8
    image = image.astype(np.uint8)

    # Lưu màng thành ảnh
    cv2.imwrite(output_image_path, image)
```

Hàm vector_to_image giúp chuyển đổi một vector thành một ảnh và lưu ảnh này vào một đường dẫn được chỉ định. Dưới đây là giải thích chi tiết về từng bước trong hàm:

### *a) Khai báo hàm và tham số*

```
def vector_to_image(vector, output_image_path, image_shape)
```

Chi tiết:

- vector: Một vector đầu vào cần chuyển đổi thành ảnh.
- output_image_path: Đường dẫn nơi lưu ảnh đầu ra.
- image_shape: Hình dạng của ảnh.

### *b) Chia vector thành các ma trận 32x32*

```
image_channels = np.split(vector, 2)
```

- Mục đích: Chia vector đầu vào thành hai phần bằng nhau.

- Chi tiết: Giả sử vector có độ dài 1024, nó sẽ được chia thành hai phần, mỗi phần có độ dài 512.

### *c) Chuyển đổi mỗi phần thành ảnh 32x32*

```
    image = np.stack([channel.reshape((32, 32)) for channel in image_channels], axis= -1)
    image = np.concatenate((image, image[:, :, 0:1]), axis=2)  # với kênh thứ 3 là bản copy của kênh 1
    #image = np.concatenate((image, image[:, :, 1:2]), axis=2)  #với kênh thứ 3 là bản copy của kênh 2
```

- Mục đích: Chuyển đổi mỗi phần của vector thành ma trận 32x32 và ghép chúng lại thành một ảnh có hai kênh.

- Chi tiết:

- channel.reshape((32, 32)): Chuyển đổi mỗi phần của vector (512 phần tử) thành ma trận 32x32.
- np.stack([...], axis=-1): Ghép hai ma trận này lại với nhau để tạo thành một ảnh với hai kênh (32x32x2).
- np.concatenate((image, image[:, :, 0:1]), axis=2): Thêm kênh thứ ba bằng cách sao chép kênh đầu tiên, tạo ra một ảnh RGB (32x32x3).
- np.concatenate((image, image[:, :, 1:2]), axis=2) :Thêm kênh thứ ba bằng cách sao chép kênh thứ hai, tạo ra một ảnh RGB (32x32x3).

***d) Chuyển đổi kiểu dữ liệu sang uint8***

```
    image = image.astype(np.uint8)
```

- Mục đích: Chuyển đổi kiểu dữ liệu của các giá trị ảnh sang uint8 (unsigned 8-bit integer).

- Chi tiết: Các giá trị trong ảnh sẽ nằm trong khoảng từ 0 đến 255, phù hợp với định dạng ảnh 8-bit (1 byte cho mỗi giá trị pixel), với 0 đại diện cho màu đen hoàn toàn và 255 đại diện cho màu trắng hoàn toàn (hoặc màu sáng nhất trong kênh màu cụ thể). Mặt khác, thư viện OpenCV (cv2) và nhiều thư viện xử lý ảnh khác yêu cầu ảnh đầu vào phải ở định dạng uint8 để hoạt động chính xác.

***e) Lưu mảng thành ảnh***

```
    cv2.imwrite(output_image_path, image)
```

- Mục đích: Lưu mảng dữ liệu dưới dạng một tệp ảnh.

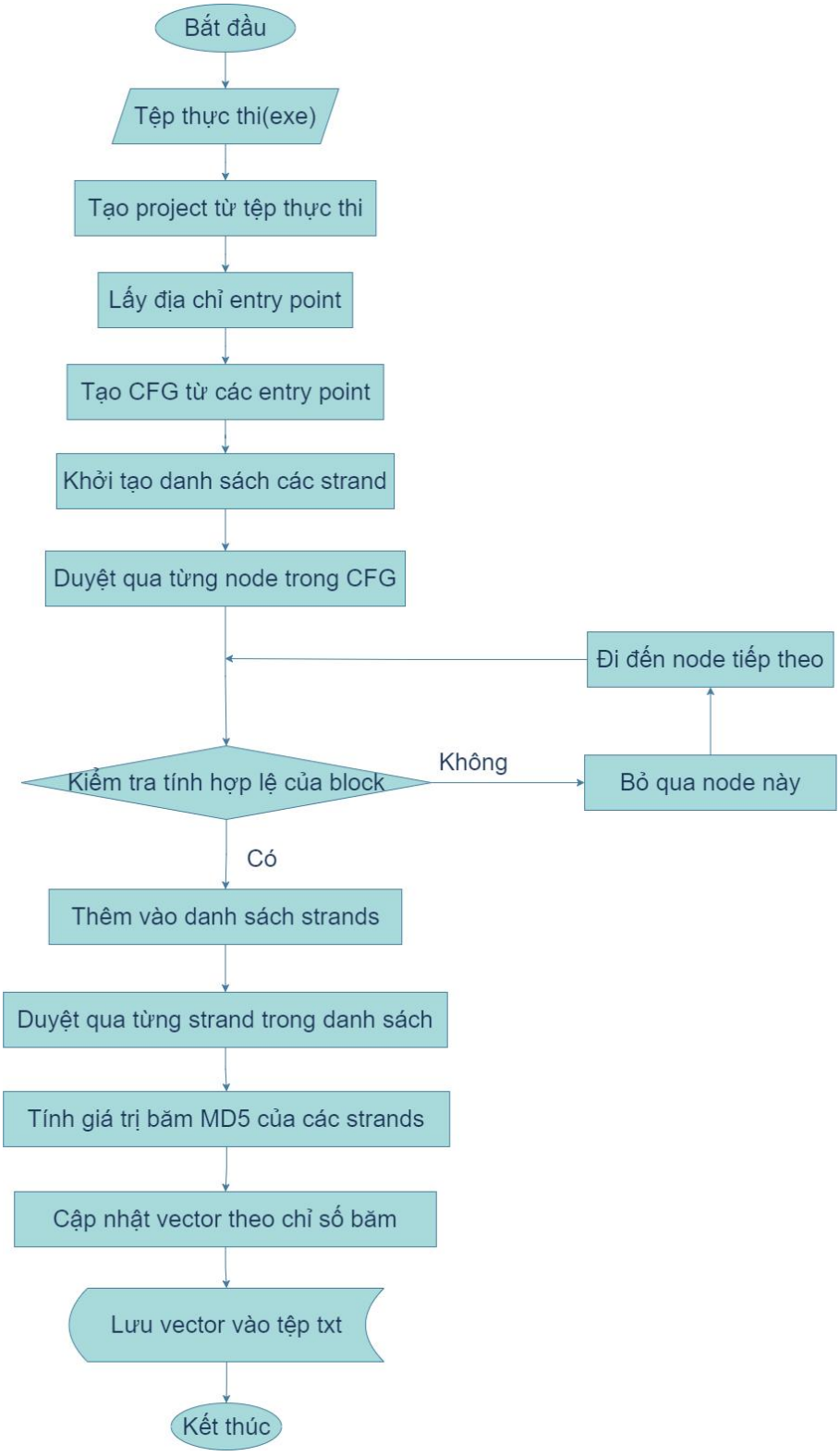
- Chi tiết: Dùng thư viện OpenCV (cv2) để lưu ảnh vào đường dẫn được chỉ định.

**Tóm lại** hàm vector_to_image thực hiện các bước sau:

- Chia vector đầu vào thành hai phần bằng nhau.
- Chuyển đổi mỗi phần thành ma trận 32x32.
- Ghép hai ma trận này lại với nhau để tạo thành một ảnh với hai kênh.
- Thêm kênh thứ ba bằng cách sao chép kênh đầu tiên, tạo ra một ảnh RGB.
- Chuyển đổi kiểu dữ liệu của các giá trị ảnh sang uint8.
- Lưu ảnh vào đường dẫn được chỉ định bằng thư viện OpenCV.

**B. Hàm chuyển đổi tập tin PE sang vector**

Chúng tôi sử dụng một hàm duy nhất để thực hiện việc chuyển đổi file PE thành vector, đó là hàm convert_exe_to_vector. Sau đây là luồng thực thi của hàm này:


Bắt đầu
Tệp thực thi(exe)
Tạo project từ tệp thực thi
Lấy địa chỉ entry point
Tạo CFG từ các entry point
Khởi tạo danh sách các strand
Duyệt qua từng node trong CFG
Đi đến node tiếp theo
Kiểm tra tính hợp lệ của block
Không
Bỏ qua node này
Có
Thêm vào danh sách strands
Duyệt qua từng strand trong danh sách
Tính giá trị băm MD5 của các strands
Cập nhật vector theo chỉ số băm
Lưu vector vào tệp txt
Kết thúc

Hàm convert_exe_to_vector sử dụng thư viện angr để phân tích một tệp thực thi và chuyển đổi nó thành một vector đại diện. Dưới đây là giải thích chi tiết hơn về từng phần của hàm:

***a) Khởi tạo dự án với angr***

```
project = angr.Project(exe_file_path, auto_load_libs=False,
load_options={})
```

- Mục đích: Khởi tạo một đối tượng angr.Project để phân tích tệp thực thi.
- Chi tiết:

- exe_file_path: Đường dẫn đến tệp thực thi cần phân tích.
- auto_load_libs=False: Chỉ tải tệp chính, không tải các thư viện liên quan để giảm độ phức tạp.
- load_options={}: Các tùy chọn tải khác, ở đây để trống.

***b) Lấy địa chỉ điểm bắt đầu thực thi***

```
entry_address = project.entry
```

- Mục đích: Lấy địa chỉ nơi chương trình bắt đầu thực thi.

***c) Tạo đồ thị luồng điều khiển (CFG)***

```
cfg = project.analyses.CFGFast()
```

- Mục đích: Tạo một đồ thị luồng điều khiển nhanh (CFG) cho tệp thực thi.
- Chi tiết: CFG đại diện cho các đường đi có thể có trong quá trình thực thi chương trình, giúp hiểu rõ cấu trúc và luồng điều khiển của chương trình.

***d) Lấy các chuỗi lệnh từ CFG***

```
strands = []
for node in cfg.graph.nodes():
    block = cfg.model.get_any_node(node.addr)
    if block is not None and hasattr(block.block, 'capstone') and
block.block.capstone is not None:
        strand = []
        for stmt in block.block.capstone.insns:
```

```
            expression = convert_instruction_to_expression(stmt)
            if expression:
                strand.append(expression)
        strands.append(strand)
```

- Mục đích: Lấy các chuỗi lệnh từ các khối trong CFG.
- Chi tiết:

- cfg.graph.nodes(): Lấy tất cả các nút trong CFG.
- cfg.model.get_any_node(node.addr): Lấy khối lệnh tại địa chỉ của nút hiện tại.
- Kiểm tra xem khối lệnh có chứa các lệnh đã giải mã (capstone) không.
- Chuyển đổi từng lệnh thành biểu thức bằng hàm convert_instruction_to_expression.
- Thu thập các biểu thức này thành các chuỗi lệnh (strands).

***e) Tạo biểu thức từ các chuỗi lệnh***

```
expressions = []
for strand in strands:
    expression = " ".join(strand)
    expressions.append(expression)
```

- Mục đích: Kết hợp các lệnh trong mỗi chuỗi thành một chuỗi biểu thức đơn lẻ.
- Chi tiết:

- Dùng phương thức join để kết hợp các biểu thức trong mỗi chuỗi thành một chuỗi duy nhất.
- Thu thập các chuỗi này vào danh sách expressions.

***f) Băm biểu thức và tạo vector***

```
vector = [0] * 2**10
for expression in expressions:
    md5_hash = hashlib.md5(expression.encode()).hexdigest()
    index = int(md5_hash, 16) % 2**10
```

```
    vector[index] += 1
```

- Mục đích: Chuyển đổi các biểu thức thành các giá trị băm và tạo một vector 1024 chiều.
- Chi tiết:

- vector = [0] * 2**10: Khởi tạo một vector 1024 chiều với tất cả các giá trị là 0.
- hashlib.md5(expression.encode()).hexdigest(): Tạo băm MD5 cho biểu thức.
- int(md5_hash, 16) % 2**10: Chuyển đổi băm thành số nguyên và lấy phần dư khi chia cho 1024 để xác định chỉ số trong vector.
- vector[index] += 1: Tăng giá trị tại vị trí chỉ số trong vector.
- return entry_address, vector: Trả về địa chỉ điểm bắt đầu và vector

Tóm lại, hàm convert_exe_to_vector giúp phân tích một tệp thực thi bằng cách sử dụng thư viện angr, tạo ra đồ thị luồng điều khiển, trích xuất các lệnh từ các khối trong đồ thị, chuyển đổi chúng thành các biểu thức, băm các biểu thức và ánh xạ chúng vào một vector 1024 chiều. Kết quả cuối cùng là địa chỉ điểm bắt đầu thực thi và vector đại diện của tệp thực thi, giúp cho việc phân tích và so sánh các tệp thực thi trở nên dễ dàng hơn trong các bài toán học máy.