BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ

NGUYỄN NGỌC TOÀN

NGHIÊN CỨU NÂNG CAO HIỆU QUẢ PHÁT HIỆN MÃ ĐỘC IOT DỰA TRÊN HỌC MÁY SỬ DỤNG CÁC ĐẶC TRƯNG CỦA TẬP TIN THỰC THI

LUẬN ÁN TIẾN SĨ NGÀNH AN TOÀN THÔNG TIN

HÀ NỘI - 2024

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ

Nguyễn Ngọc Toàn

NGHIÊN CỨU NÂNG CAO HIỆU QUẢ PHÁT HIỆN MÃ ĐỘC IOT DỰA TRÊN HỌC MÁY SỬ DỤNG CÁC ĐẶC TRƯNG CỦA TẬP TIN THỰC THI

Ngành: An toàn thông tin

Mã số: 9480202

LUẬN ÁN TIẾN SĨ CHUYÊN NGÀNH AN TOÀN THÔNG TIN

CÁN BỘ HƯỚNG DẪN KHOA HỌC **NGHIÊN CỨU SINH**

PGS.TS Lương Thế Dũng **Nguyễn Ngọc Toàn**

XÁC NHẬN CỦA ĐƠN VỊ ĐÀO TẠO

HÀ NỘI - 2024

LỜI CAM ĐOAN

Tôi xin cam đoan luận án "Nghiên cứu nâng cao hiệu quả phát hiện mã độc IoT dựa trên học máy sử dụng các đặc trưng của tập tin thực thi" là công trình nghiên cứu của riêng tôi, dưới sự hướng dẫn khoa học của PGS.TS Lương Thế Dũng, trừ những kiến thức tham khảo từ các tài liệu có liên quan đã được trích dẫn trong luận án.

Các kết quả, số liệu được trình bày trong luận án là hoàn toàn trung thực, một phần kết quả đã được công bố trên các Tạp chí và Kỷ yếu Hội thảo khoa học chuyên ngành (tại Danh mục công trình của tác giả), phần còn lại chưa từng được công bố trong bất kỳ công trình nào khác.

Hà Nội, ngày tháng năm 2024

TÁC GIẢ LUẬN ÁN

Nguyễn Ngọc Toàn

LỜI CẢM ƠN

Luận án này được nghiên cứu sinh (NCS) thực hiện trong quá trình học tập Tiến sĩ tại Học viện Kỹ thuật mật mã thuộc Ban Cơ yếu chính phủ. Trong quá trình học tập, NCS đã được các thầy, cô trong Học viện Kỹ thuật mật mã hướng dẫn và trang bị những kiến thức nền tảng cần thiết, đồng thời NCS có cơ hội tiếp xúc chuyên sâu về lĩnh vực mới và cấp thiết trong an toàn thông tin liên quan đến phát hiện mã độc IoT.

Trước hết, NCS xin bày tỏ lòng biết ơn sâu sắc tới thầy PGS.TS Lương Thế Dũng đã tận tình hướng dẫn, định hướng nghiên cứu khoa học, giúp NCS có thể vượt qua những khó khăn, thách thức trong suốt quá trình thực hiện nghiên cứu luận án. Tiếp đó, NCS xin gửi lời cảm ơn tới các cộng sự nhóm nghiên cứu MFC500 đã gợi mở các ý tưởng và hỗ trợ cho phương hướng phát triển nghiên cứu khoa học của NCS. NCS xin gửi lời cảm ơn tới Ban Giám đốc, Lãnh đạo Khoa An ninh mạng và phòng, chống tội phạm sử dụng công nghệ cao, các Phòng ban liên quan của Học viện An ninh nhân dân đã tạo điều kiện để NCS có thể tập trung nghiên cứu và thực hiện luận án này. Cảm ơn Tập đoàn Vingroup – Công ty CP và Chương trình học bổng thạc sĩ, tiến sĩ trong nước của Quỹ Đổi mới sáng tạo Vingroup (VINIF), Viện Nghiên cứu Dữ liệu lớn đã hỗ trợ một phần kinh phí trong quá trình NCS thực hiện các nghiên cứu của luận án này.

Cuối cùng, tôi xin bày tỏ lòng cảm ơn vô hạn đến gia đình đã luôn luôn là nguồn động lực để tôi nghiên cứu.

Xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC KÝ HIỆU, TỪ VIẾT TẮT

Ký hiệu, từ viết tắt	Viết đầy đủ	Nghĩa tiếng Việt đầy đủ
AI	Artificial Intelligence	Trí tuệ nhân tạo
ANN	Artificial Neural Networks	Mạng nơ-ron nhân tạo
ARM	Advanced RISC Machine	Kiến trúc vi xử lý ARM
CFG	Control Flow Graph	Đồ thị luồng điều khiển
CNN	Convolutional Neural Network	Mạng nơ-ron tích chập
CPU	Central Processing Unit	Bộ xử lý trung tâm
DFG	Data Flow Graph	Đồ thị luồng dữ liệu
DT	Decision Tree	Thuật toán cây quyết định
ELF	Linux Executable and Linkable Format	Định dạng tập tin thực thi và liên kết động trên Linux
FN	False Negative	Âm tính giả
FP	False Positive	Dương tính giả
Intel	Integrated Electronics	Kiến trúc vi xử lý do Intel sản xuất
IoT	Internet of things	Vạn vật kết nối Internet
IP	Internet Protocol	Giao thức Internet
kNN	k Nearest Neighbors	K-láng giềng gần nhất
LSTM	Long Short Term Memory	Bộ nhớ ngắn dài hạn
LR	Linear Regression	Hồi quy tuyến tính
MIPS	Microprocessor without Interlocked Pipeline Stages	Kiến trúc vi xử lý phát triển bởi MIPS Technologies
MLP	Multilayer Perceptron	Mạng nơ-ron nhân tạo truyền thẳng nhiều lớp
NCS	Nghiên cứu sinh	Nghiên cứu sinh
NB	Naive Bayes	Thuật toán Naive Bayes
NN	Neural Network	Mạng nơ-ron
Opcode	Operation Code	Mã thực thi
PowerPC	Performance Optimization With Enhanced RISC – Performance Computing	Kiến trúc vi xử lý do liên minh Apple - IBM – Motorola phát triển
PSI	Printable String Information	Chuỗi mang thông tin in được
RF	Random Forest	Thuật toán rừng ngẫu nhiên
RNN	Recurrent Neural Network	Mạng nơ-ron hồi quy
SPARC	Scalable Processor Architecture	Kiến trúc vi xử lý do Sun Microsystems và Fujitsu phát triển
SVM	Support vectơ machines	Thuật toán học máy vectơ hỗ trợ
TF- IDF	Term Frequency - Inverse Document Frequency	Trọng số của một từ trong văn bản
TN	True Negative	Tiêu cực chính xác
TP	True Positive	Tích cực chính xác

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH VẼ

MỞ ĐẦU

1. Tính cấp thiết

Cuộc cách mạng công nghiệp lần thứ 4 đã và đang tác động đến mọi mặt của đời sống xã hội. Với những yếu tố cốt lõi như trí tuệ nhân tạo, vạn vật kết nối Internet, dữ liệu lớn,… đã hình thành nhiều lĩnh vực, xuất hiện nhiều loại thiết bị mới và có nhiều thiết bị kết nối vào Internet như IP Camera, Smart TV, Router, Smartphone,… Theo dự báo của Statista[1] sẽ có hơn 30,9 tỉ thiết bị IoT kết nối vào năm 2025 và chiếm 75% tổng số thiết bị trên toàn thế giới. Trong an toàn thông tin, chủ đề mã độc là chủ đề được nhiều nhà nghiên cứu quan tâm hiện nay [64], [81], [85], [114]. Theo báo cáo của Sonicwall [107] số lượng mã độc tăng đột biến vào năm 2022 với 5,5 tỷ cuộc tấn công bằng mã độc, trong đó có hơn 112 triệu ghi nhận các mã độc nhằm vào thiết bị IoT, tăng 87% so với năm 2021. Các mã độc luôn phát triển với nhiều kỹ thuật phòng vệ tiên tiến, sử dụng nhiều kỹ thuật lẩn tránh các hệ thống phát hiện như các kỹ thuật làm rối, kỹ thuật đa hình, siêu đa hình, hoạt động đa nền tảng,… Việc nghiên cứu các giải pháp phát hiện mã độc nói chung và mã độc IoT nói riêng là cấp thiết trong đảm bảo an ninh, an toàn cho các hệ thống thông tin.

Việc xây dựng các giải pháp phát hiện mã độc IoT trong các tập tin thực thi được thực hiện dựa trên hai kỹ thuật chính bao gồm dựa trên dấu hiệu (signature-based) và dựa trên hành vi (behavior-based). Các giải pháp phát hiện mã độc dựa trên dấu hiệu có thể phát hiện chính xác được các mã độc đã biết và có trong cơ sở dữ liệu, tuy nhiên giải pháp không hiệu quả đối với các biến thể mã độc mới và mã độc chưa có dấu hiệu nhận biết. Vì vậy, với sự phát triển của mã độc trên các thiết bị IoT và sự phát triển của trí tuệ nhân tạo đã góp phần quan trọng trong thúc đẩy việc xây dựng các giải pháp phát hiện mã độc dựa trên hành vi hiệu quả hơn. Đặc biệt, các mô hình phát hiện mã độc dựa trên học máy có thể tạo ra các giải pháp có khả năng tự động phát hiện mã độc với độ chính xác cao và phát hiện được các biến thể và dòng mã độc mới trên thiết bị IoT.

Các đặc trưng của tập tin thực thi thường được sử dụng để xây dựng mô hình phát hiện mã độc dựa trên học máy. Các đặc trưng này đang được các nghiên cứu xây dựng dựa trên kỹ thuật phân tích tĩnh và động tập tin thực thi.

Đối với phân tích tĩnh, các đặc trưng của tập tin thực thi được trích xuất để sử dụng trong xây dựng các mô hình phát hiện mã độc dựa trên học máy bao gồm: Thông tin cấu trúc tiêu đề tập tin (header), mã thực thi (opcode), chuỗi mang thông tin in được (Printable String Information- PSI), đồ thị luồng điều khiển (Control Flow Graph - CFG), đồ thị luồng dữ liệu (Data Flow Graph - DFG),… Các đặc trưng tĩnh này đã được các nhóm nước ngoài đề xuất phương pháp trích xuất, lựa chọn phục vụ xây dựng mô hình phát hiện

[1] https://explodingtopics.com/blog/iot-stats

mã độc như Costin [2], Chin-wei Tien [17], Ding [125], Yan Shoshitaishvili [127],… Các phương pháp phân tích tĩnh đã đề xuất đã mạng lại hiệu quả trong phát hiện mã độc IoT, tuy nhiên các mô hình học máy sử dụng đặc trưng tĩnh đã trích xuất có độ chính xác cao lại cần số lượng đặc trưng cần thu thập nhiều, kích thước và thời gian phát hiện của mô hình lớn. Do đó, các phương pháp này chỉ có thể áp dụng với các tập tin thực thi đơn giản, không phù hợp với các tập tin phức tạp, có kích thước lớn. Các mô hình khó triển khai trong môi trường IoT hạn chế tài nguyên.

Tại Việt nam, luận án của tác giả Nguyễn Huy Trung [85] đã trình bày các phương pháp phát hiện một dòng mã độc cụ thể trên thiết bị IoT là IoT Botnet dựa trên đồ thị PSI được thu thập qua quá trình dịch ngược tập tin ELF. Tác giả đã áp dụng các mạng học sâu để phát hiện mã độc IoT với độ chính xác cao trên tập thử nghiệm với F1-score 98,6%. Tuy nhiên, phương pháp tác giả đề xuất phụ thuộc vào khả năng trích xuất các chuỗi PSI từ tập tin thực thi, đặc biệt độ chính xác trong phát hiện bị giảm đáng kể khi các chuỗi PSI bị mã hoá. Luận án cũng chỉ ra việc hạn chế và cần phát triển nghiên cứu đối với các dòng mã độc IoT đa dạng khác.

Đối với phân tích động, các đặc trưng của tập tin thực thi được thu thập phục vụ xây dựng các mô hình phát hiện mã độc bao gồm: Thông tin được thu thập hành vi mức hệ thống như lời gọi API, lời gọi hệ thống, giá trị thanh ghi, dữ liệu bộ nhớ,… và thông tin hành vi mức mạng như dữ liệu luồng mạng, kết nối mạng,... Môi trường thực hiện thu thập các đặc trưng trong phân tích động thường là môi trường mô phỏng hoặc ảo hoá được giám sát như sandbox, các thiết bị cài đặt tác tử. Trong các nghiên cứu của nhóm tác giả ngoài nước như Alberto [5], Yin Minn Pa Pa [90], Sriram [102], Jonas [107], Zhao [136],… đã sử dụng các đặc trưng động nêu trên. Tuy nhiên, các nghiên cứu cần thu thập nhiều thông tin hành vi, quá trình thực thi tập tin trong môi trường phân tích cần nhiều thời gian hơn quá trình thực thi trên thiết bị thực do việc đồng bộ tín hiệu. Các mô hình học sâu đã được sử dụng để nâng cao độ chính xác nhưng các nghiên cứu đối mặt với độ phức tạp tính toán lớn khi sử dụng các mạng học sâu phức tạp và khó triển khai trong thực tế cho các ứng dụng phát hiện mã độc IoT theo thời gian thực do cần thu thập nhiều đặc trưng hành vi mạng và ảnh hưởng đến băng thông, hoạt động của hệ thống mạng.

Trong nước, nhóm tác giải Lê Hải Việt đã đề xuất trong các nghiên cứu [65], [86], [114] các mô hình phát hiện mã độc IoT Botnet dựa trên đồ thị lời gọi hệ thống có hướng DSCG và kết hợp nhiều nguồn dữ liệu trích xuất từ kết quả phân tích động thu thập qua V-Sandbox với độ chính xác cao đạt 99,37%. Tuy nhiên, các nghiên cứu mới chỉ thử nghiệm với bộ dữ liệu chứa duy nhất một loại mã độc là IoT Botnet, thời gian khởi tạo, thực thi, giám sát các tập tin trong môi trường V-Sandbox còn dài dẫn đến hạn chế về mặt thời gian trong giải pháp phát hiện sớm mã độc IoT Botnet. Điều này cũng đã được đề

cập đến trong luận án tiến sĩ của tác giả [64]. Luận án của tác giả Tống Anh Tuấn [113] đã đề xuất 3 giải pháp phát hiện và phân loại một loại mã độc DGA Botnet sử dụng cách tiếp cận học máy và học sâu với độ chính xác cao khi so sánh với các giải pháp trước đó. Tuy nhiên, các giải pháp đề xuất cũng mới chỉ tập trung vào một loại mã độc DGA Botnet, chưa thử nghiệm trên các loại mã độc IoT khác, chưa có cơ chế phát hiện riêng cho các họ DGA Botnet có sự tương đồng cao hoặc các biến thể mã độc DGA Botnet.

Các nghiên cứu cũng chỉ ra rằng phương pháp phân tích động có thể thu thập được hành vi của các mã độc sử dụng các kỹ thuật làm rối, mã hoá mã, quan sát được các hành vi tương tác của mã độc trong môi trường thực thi. Tuy nhiên, các phương pháp này cần thu thập nhiều loại đặc trưng hành vi khác nhau như hành vi hệ thống, hành vi mạng, tương tác môi trường,… để có thể phát hiện mã độc. Thời gian xây dựng môi trường, thực thi tập tin mã độc trong môi trường giám sát nhiều sẽ ảnh hưởng đến hiệu quả khi phân tích số lượng mẫu lớn và khó triển khai trong các hệ thống phát hiện mã độc IoT đáp ứng thời gian thực. Bên cạnh đó, một số mô hình phát hiện mã độc IoT dựa trên đặc trưng động chưa đem lại độ chính xác cao trong phát hiện mã độc trên các tập dữ liệu IoT có sự mất cân bằng giữa các tập dữ liệu huấn luyện thử nghiệm.

Mặt khác, với sự phát triển đa dạng của các loại thiết bị IoT, bên cạnh dòng mã độc IoT hoạt động trên một kiến trúc vi xử lý cố định thì các mã độc IoT đã và đang tìm cách đa dạng hoá mục tiêu tấn công để có thể lây nhiễm vào nhiều loại thiết bị IoT khác nhau. Xuất hiện ngày càng nhiều mã độc và biến thể mã độc IoT mới có khả năng hoạt động trên nhiều loại thiết bị sử dụng kiến trúc vi xử lý khác nhau. Việc phát hiện mã độc hoạt động đa nền tảng kiến trúc dựa trên đặc trưng của tập tin thực thi là cần thiết. Việc xây dựng một mô hình có khả năng phát hiện mã độc hoạt động đa kiến trúc thay vì phải huấn luyện và tích hợp nhiều mô hình phát hiện mã độc IoT đơn kiến trúc trong một giải pháp sẽ có ý nghĩa to lớn trong thực tế môi trường IoT đa dạng thiết bị hiện nay.

Cách tiếp cận xây dựng mô hình phát hiện mã độc IoT đa kiến trúc dựa trên phương pháp trích xuất đặc trưng tĩnh ít phụ thuộc vào kiến trúc vi xử lý hoặc hệ điều hành đã chứng minh được khả năng phát hiện mã độc IoT hoạt động được trên nhiều kiến trúc vi xử lý. Tuy nhiên, hiệu quả của các mô hình phát hiện mã độc IoT dựa trên đặc trưng tĩnh phụ thuộc vào đặc trưng thu thập và biểu diễn của đặc trưng thu thập. Các thực nghiệm đánh giá chi tiết khả năng của các mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng tĩnh độc lập nền tảng kiến trúc vi xử lý chưa được triển khai trên các mẫu mã độc IoT thực tế và phổ biến.

Một cách tiếp khác là sử dụng đặc trưng động chung phổ biến trên các nền tảng kiến trúc. Tuy nhiên, việc thu thập đặc trưng động gặp khó khăn khi áp dụng đối với các mã độc hoạt động trên thiết bị IoT đa dạng về hệ điều hành, nền tảng kiến trúc, điều kiện kích hoạt hành vi độc hại, phát hiện môi trường phân tích và che giấu hành vi. Xây dựng mô

hình phát hiện mã độc có khả năng hoạt động đa nền tảng sử dụng đặc trưng động sẽ gặp nhiều khó khăn do sự khác biệt về môi trường thực thi tập tin và đặc điểm của đặc trưng động thu thập của từng nền tảng hệ điều hành hoặc nền tảng kiến trúc vi xử lý.

Hơn nữa, các tập dữ liệu mã độc IoT hoạt động trên các kiến trúc vi xử lý được các nghiên cứu thu thập từ các kho dữ liệu như Virus Total, Virus Share, Detux, IoTPOT,… Với đặc điểm riêng biệt của các thiết bị IoT, việc thu thập dữ liệu đang gặp phải nhiều khó khăn, thách thức, các tập dữ liệu IoT phục vụ huấn luyện mô hình có sự mất cân bằng giữa các lớp hoặc các loại mã độc, đặc biệt là mã độc hoạt động trên các thiết bị IoT mới, đa dạng kiến trúc vi xử lý. Các nghiên cứu xây dựng mô hình phát hiện dựa trên khai thác các tri thức mã độc các kiến trúc vi xử lý đã phổ biến hoặc mã độc truyền thống để xây dựng các mô hình dự đoán mã độc zero-day trên các kiến trúc vi xử lý mới, các thiết bị IoT phi chuẩn chưa đạt được hiệu quả tốt về độ chính xác. Vì vậy, việc xây dựng các mô hình phát hiện hiệu quả với độ chính xác phù hợp với thời gian và tài nguyên sử dụng, có khả năng phát hiện các biến thể mã độc trên các kiến trúc vi xử lý mới cần được quan tâm nghiên cứu hiện nay.

Từ những hiệu quả của ứng dụng học máy trong phát hiện mã độc IoT, luận án được thúc đẩy bởi một số vấn đề nghiên cứu mở sau đây:

Thứ nhất, các mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên đặc trưng của tập tin thực thi sử dụng nhiều loại đặc trưng thu thập từ phân tích tĩnh hoặc phân tích động, cần nhiều thời gian thu thập và xử lý lý đặc trưng. Việc sử dụng mạng học sâu, kết hợp nhiều đặc trưng hoặc thuật toán học máy trong xây dựng mô hình phát hiện IoT có thể nâng cao được độ chính xác cho mô hình nhưng chưa hiệu quả cao về số lượng đặc trưng sử dụng, thời gian thực thi tập tin có chứa mã độc, tốc độ tính toán, thời gian phát hiện và khả năng ứng dụng mô hình trong thực tế cho các ứng dụng phát hiện mã độc IoT theo thời gian thực hoặc các thiết bị IoT hạn chế tài nguyên.

Thứ hai, các mô hình phát hiện mã độc IoT đa kiến trúc dựa trên đặc trưng của tập tin thực thi dựa trên thu thập đặc trưng động hoặc tĩnh không phụ thuộc vào nền tảng kiến trúc, sử dụng các mạng học sâu phức tạp hoặc kết hợp nhiều loại mạng học sâu đã đem lại hiệu quả đối với một số dòng mã độc IoT nhưng chưa hiệu quả đối với các mã độc mới, mã độc tấn công có chủ đích hoặc mã độc zero-day trên thiết bị IoT đa dạng nền tảng. Các mô hình huấn luyện được khó triển khai trên các thiết bị IoT hạn chế tài nguyên.

Thứ ba, các nghiên cứu về mã độc trên thiết bị IoT thời gian qua tập trung nhiều vào mã độc trên thiết bị sử dụng hệ điều hành Android (chiếm hơn 40% so với các nghiên cứu liên quan) [114] và tập trung nhiều vào mã độc IoT Botnet, chưa nghiên cứu các họ mã độc IoT khác [64], [85], [113]. Tập dữ liệu phục vụ huấn luyện và đánh giá mô hình phát hiện trên các kiến trúc vi xử lý còn hạn chế, đặc biệt số lượng tập mẫu trên

các kiến trúc vi xử lý của thiết bị IoT mới hoặc chưa được thu thập trước đây. Nhiều tập dữ liệu đa kiến trúc đã công bố và được thử nghiệm trong các nghiên cứu đối mặt với vấn đề mất cân bằng dữ liệu các lớp mã độc, giữa các nền tảng kiến trúc vi xử lý.

Vì vậy, việc nghiên cứu nâng cao hiệu quả cho các mô hình phát hiện mã độc IoT hoạt động đơn nền tảng kiến trúc và đa nền tảng kiến trúc sử dụng đặc trưng của tập thi thực thi là cấp thiết, có ý nghĩa về mặt khoa học và thực tiễn hiện nay.

2. Mục tiêu nghiên cứu

Từ việc phân tích tính cấp thiết trong nội dung trên, luận án xác định mục tiêu nghiên cứu chính là nghiên cứu phát triển các mô hình phát hiện mã độc IoT dựa trên học máy nhằm nâng cao hiệu quả về mặt số lượng đặc trưng sử dụng, thời gian thu thập, kích thước, độ chính xác của mô hình trong phát hiện mã độc IoT hoạt động đơn và đa nền tảng kiến trúc vi xử lý, tăng khả năng tích hợp mô hình đã huấn luyện trong các giải pháp phát hiện mã độc tự động trong hệ thống IoT tài nguyên hạn chế. Các mục tiêu cụ thể gồm:

Một là, nghiên cứu phát triển giải pháp xây dựng mô hình phát hiện mã độc IoT hoạt động đơn kiến trúc phù hợp dựa trên học máy sử dụng đặc trưng lời gọi hệ thống của tập tin thực thi được trích xuất thông qua phương pháp phân tích động. Mô hình đề xuất giúp giảm số lượng và thời gian thu thập đặc trưng chuỗi lời gọi hệ thống thu thập từ quá trình phân tích động đảm bảo độ chính xác cao trong phát hiện mã độc IoT hoạt động đơn kiến trúc vi xử lý.

Hai là, nghiên cứu phát triển giải pháp xây dựng mô hình phát hiện mã độc IoT hoạt động đơn kiến trúc phù hợp dựa trên học máy sử dụng đặc trưng mã thực thi của tập tin thực thi được trích xuất thông qua phương pháp phân tích tĩnh. Mô hình đề xuất giúp giảm số lượng đặc trưng thu thập, thời gian huấn luyện và phát hiện, kích thước của mô hình nhưng vẫn đảm bảo độ chính xác trong phát hiện mã độc IoT hoạt động đơn kiến trúc vi xử lý.

Ba là, nghiên cứu đề xuất giải pháp xây dựng mô hình phát hiện mã độc IoT hoạt động đa kiến trúc phù hợp dựa trên học máy sử dụng phương pháp chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc vi xử lý khác nhau. Giải pháp đề xuất giúp mô hình phòng chống mất cân bằng dữ liệu huấn luyện, nâng cao độ chính xác, khả năng phát hiện mã độc hoạt động đa kiến trúc và mã độc zero-day trên kiến trúc mới.

3. Đối tượng và phạm vi nghiên cứu

3.1. Đối tượng nghiên cứu

Để đạt được mục tiêu nghiên cứu, đối tượng nghiên cứu của luận án là các tập tin ELF trên các thiết bị IoT sử dụng hệ điều hành Linux nhúng. Các tập tin ELF sẽ được thu thập từ các nguồn uy tín như các kho dữ liệu được các nhà khoa học chia sẻ, dữ liệu trích xuất từ phần sụn của thiết bị IoT để làm đối tượng nghiên cứu, thử nghiệm và đánh

giá kết quả của các mô hình phát hiện mã độc IoT.

3.2. Phạm vi nghiên cứu

Bài toán phát hiện mã độc IoT được xác định là khả năng phân biệt giữa các tập tin mã độc và tập tin lành tính. Luận án phát triển hướng tiếp cận mới trong xây dựng các mô hình phát hiện mã độc IoT hoạt động đơn kiến trúc vi xử lý và đa kiến trúc vi xử lý hiệu quả dựa trên học máy sử dụng đặc trưng biểu diễn dạng chuỗi thu thập từ phân tích động và phân tích tĩnh tập tin. Trong phạm vi luận án, việc sử dụng hai thuật ngữ mô hình phát hiện mã độc IoT và mô hình phát hiện mã độc IoT dựa trên học máy là đồng nhất. Phạm vi cụ thể như sau:

Thứ nhất, mô hình phát hiện mã độc là mô hình học máy có khả năng phân lớp tập tin thành các nhãn khác nhau. Trong phạm vi bài toán nghiên cứu của luận án, mô hình luận án tập trung là mô hình học máy có khả năng phân biệt tập tin thực thi thành hai nhãn là lành tính hoặc mã độc.

Thứ hai, có nhiều cách phân loại thiết bị IoT như dựa trên hãng sản xuất, dựa trên mục đích sử dụng, dựa trên tài nguyên gồm thiết bị IoT hiệu năng cao và thiết bị IoT tài nguyên hạn chế. Các thiết bị IoT tài nguyên hạn chế theo Bencheton [15] có thể kể đến như wifi router, IP camera, switch, smart Hub,… Mặt khác, theo nội dung báo cáo của Wendell E. Carter [124] các thiết bị IoT đang sử dụng phổ biến hệ điều hành Linux. Vì vậy, luận án chỉ tập trung nghiên cứu, phát hiện mã độc trên thiết bị IoT tài nguyên hạn chế sử dụng hệ điều hành Linux nhúng Kernel 2.6 hoặc 3.2 với các kiến trúc bộ vi xử lý trung tâm (Central Processing Unit - CPU) phổ biến như Integrated Electronics (Intel), Microprocessor without Interlocked Pipeline Stages (MIPS), Advanced RISC Machine (ARM), Scalable Processor Architecture (SPARC), Performance Optimization With Enhanced RISC – Performance Computing (PowerPC).

Thứ ba, có 3 phương pháp chính trong phân tích mã độc IoT là phân tích tĩnh, phân tích động và phân tích lai. Tuy nhiên, để đạt được mục tiêu nghiên cứu, luận án tiếp cận theo phương pháp phân tích động và phân tích tĩnh nhằm thu thập các đặc trưng biểu diễn dưới dạng chuỗi phục vụ xây dựng các mô hình phát hiện mã độc IoT.

Thứ tư, có hai cách tiếp cận mã độc đa nền tảng trên thiết bị IoT là theo hệ điều hành và theo kiến trúc vi xử lý trung tâm. Để đạt được mục tiêu nghiên cứu, luận án tiếp cận theo kiến trúc vi xử lý trung tâm. Khi đó, dựa vào khả năng hoạt động trên nền tảng kiến trúc vi xử lý, mã độc trên thiết bị IoT được phân loại gồm mã độc IoT đơn kiến trúc vi xử lý và mã độc IoT đa kiến trúc vi xử lý.

4. Nội dung và phương pháp nghiên cứu

4.1. Nội dung nghiên cứu

Với các mục tiêu nghiên cứu nêu trên, luận án tập trung vào 4 nội dung sau:

- Nghiên cứu, phân tích đặc điểm, xu hướng phát triển của mã độc IoT. Từ đó khảo

sát, phân tích, đánh giá hiệu quả của các mô hình phát hiện mã độc IoT dựa trên học máy để làm cơ sở đề xuất, giải quyết các bài toán xây dựng mô hình phát hiện mã độc IoT hiệu quả dựa trên học máy sử dụng đặc trưng của tập tin thực thi.

- Nghiên cứu xây dựng mô hình phát hiện mã độc IoT hoạt động đơn kiến trúc phù hợp giúp giảm số lượng và thời gian thu thập đặc trưng chuỗi lời gọi hệ thống thu thập từ quá trình phân tích động đảm bảo độ chính xác cao. Đánh giá hiệu quả của mô hình đề xuất và so sánh kết quả thực nghiệm với các nghiên cứu khác có liên quan.

- Nghiên cứu xây dựng mô hình phát hiện mã độc IoT hoạt động đơn kiến trúc phù hợp giúp giảm số lượng đặc trưng thu thập đặc trưng mã thực thi từ quá trình phân tích tĩnh tập tin thực thi, thời gian huấn luyện và phát hiện, kích thước của mô hình nhưng vẫn đảm bảo độ chính xác tốt. Đánh giá hiệu quả của mô hình đề xuất và so sánh kết quả thực nghiệm với các nghiên cứu khác có liên quan.

- Nghiên cứu đề xuất giải pháp xây dựng mô hình phát hiện mã độc IoT hoạt động đa kiến trúc phù hợp dựa trên học máy sử dụng phương pháp chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc vi xử lý khác nhau giúp mô hình phòng chống mất cân bằng dữ liệu huấn luyện, nâng cao độ chính xác, khả năng phát hiện mã độc hoạt động đa kiến trúc và mã độc zero-day trên kiến trúc mới. Đánh giá hiệu quả của mô hình đề xuất và so sánh với các mô hình hiệu quả khác đã công bố cùng hướng tiếp cận.

4.2. Phương pháp nghiên cứu

- *Phương pháp nghiên cứu lý thuyết:*

Khảo sát, phân tích, tổng hợp, đánh giá các công trình nghiên cứu khoa học có liên quan đến luận án trong và ngoài nước để xác định những vấn đề cần tiếp tục nghiên cứu theo hướng của luận án. Việc tìm kiếm các công trình nghiên cứu khoa học có liên quan được thực hiện thông qua nguồn mở trên Internet và các bài báo cáo tại hội thảo uy tín trong lĩnh vực an toàn thông tin.

Từ đó lựa chọn các nội dung, vấn đề còn tồn tại trong xây dựng các mô hình phát hiện mã độc IoT dựa trên học máy để nghiên cứu và đề xuất các mô hình phát hiện mã độc IoT hiệu quả hơn. Hệ thống các vấn đề nghiên cứu để đề xuất bài toán và so sánh, đánh giá kết quả.

- *Phương pháp nghiên cứu thực nghiệm:*

Thực nghiệm các mô hình đề xuất trên các tập dữ liệu tập tin thực thi đã công bố. Đánh giá hiệu quả của các mô hình sử dụng các tiêu chí đánh giá phù hợp và so sánh với kết quả thực nghiệm các nghiên cứu có liên quan.

5. Các đóng góp chính của luận án

Luận án có 3 đóng góp chính bao gồm:

- ***Đóng góp 1*:** Luận án phát triển mô hình phát hiện mã độc IoT đơn kiến trúc vi xử lý sử dụng đặc trưng chuỗi lời gọi hệ thống ngắn thu thập từ phân tích động tập tin

thực thi nhưng vẫn đảm bảo tiêu chí độ chính xác tương đồng các nghiên cứu liên quan dựa trên đề xuất huấn luyện các mô hình dựa đoán chuỗi phục vụ phân lớp tập tin thực thi sử dụng mạng học sâu LSTM. Mô hình phát hiện mã độc IoT sử dụng một loại đặc trưng dạng chuỗi duy nhất là chuỗi lời gọi hệ thống với độ dài chuỗi cần thu thập là 150 góp phần làm giảm thời gian thu thập đặc trưng trong phát hiện mã độc IoT và là cơ sở để xây dựng các giải pháp phát hiện mã độc IoT tích hợp trong các hệ thống bảo mật đáp ứng thời gian thực.

- ***Đóng góp 2****:* Luận án phát triển mô hình phát hiện mã độc IoT đơn kiến trúc vi xử lý sử dụng tối thiểu số lượng mã thực thi để thu thập chuỗi mã thực thi từ phân tích tĩnh tập tin thực thi nhưng vẫn đảm bảo tiêu chí độ chính xác tương đồng các nghiên cứu liên quan dựa trên đề xuất sử dụng phương pháp trích rút đặc trưng hiệu quả. Mô hình phát hiện mã độc IoT chỉ sử dụng số lượng 20 mã thực thi hiệu quả nhất trên nền tảng MIPS phục vụ xây dựng giải pháp trích chọn chuỗi mã thực thi đại diện cho tập tin thực thi góp phần làm giảm thời gian thu thập đặc trưng, giảm tài nguyên sử dụng của mô hình phát hiện, có thể phù hợp với các thuật toán học máy truyền thống đơn giản và là cơ sở để xây dựng các giải pháp phát hiện mã độc IoT tích hợp trong các hệ thống bảo mật IoT tài nguyên hạn chế.

- ***Đóng góp 3***: Luận án phát triển mô hình phát hiện mã độc IoT đa kiến trúc hiệu quả về độ chính xác, có khả năng phát hiện mã độc trên kiến trúc vi xử lý mới hoặc ít dữ liệu và tri thức về mã độc dựa trên đề xuất phương pháp chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc vi xử lý khác nhau. Mô hình phát hiện mã độc IoT đa kiến trúc vi xử lý đã đạt hiệu quả về độ chính xác tốt khi tăng cường được tập dữ liệu huấn luyện và chỉ sử dụng các thuật toán học máy truyền thống đơn giản, kích thước, thời gian phát hiện phù hợp khi triển khai trong môi trường IoT tài nguyên hạn chế, có khả năng dự đoán mã độc zero-day đa kiến trúc trên các thiết bị IoT đa dạng.

Các nghiên cứu của luận án nhằm hướng tới giải pháp hiệu quả, hoàn chỉnh dựa trên quy trình phát hiện mã độc trong thực tế để phát hiện mã độc IoT hoạt động đơn kiến trúc và đa kiến trúc vi xử lý sử dụng hệ điều hành Linux nhúng. Các thực nghiệm đã đưa ra kết quả tốt và mở ra nhiều hướng phát triển trong tương lai. Toàn bộ mã nguồn nghiên cứu của luận án được công bố mở tại địa chỉ: *https://gitlab.com/ngoctoan*.

6. Cấu trúc của luận án

Với các kết quả nghiên cứu đã thực hiện, ngoài phần mở đầu, kết luận thì luận án được trình bày gồm 4 chương:

Chương 1: Trình bày tổng quan về mã độc IoT, phát hiện mã độc IoT dựa trên học máy và đánh giá hiệu quả của một số mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng của tập tin thực thi. Từ đó đề xuất bài toán nghiên cứu nâng cao hiệu quả cho mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng của tập tin

thực thi.

Chương 2: Trình bày đề xuất mô hình phát hiện mã độc IoT đơn kiến trúc vi xử lý hiệu quả sử dụng đặc trưng động của tập tin thực thi.

Chương 3: Trình bày đề xuất mô hình phát hiện mã độc IoT đơn kiến trúc vi xử lý hiệu quả sử dụng đặc trưng tĩnh của tập tin thực thi.

Chương 4: Trình bày đề xuất mô hình phát hiện mã độc đa kiến trúc hiệu quả dựa trên phương pháp chuyển đổi đặc trưng của tập tin chéo kiến trúc vi xử lý.

CHƯƠNG 1. TỔNG QUAN VỀ MÃ ĐỘC IOT VÀ PHÁT HIỆN MÃ ĐỘC IOT DỰA TRÊN HỌC MÁY

1.1. Tổng quan về mã độc IoT

1.1.1. Khái niệm, đặc điểm của thiết bị IoT

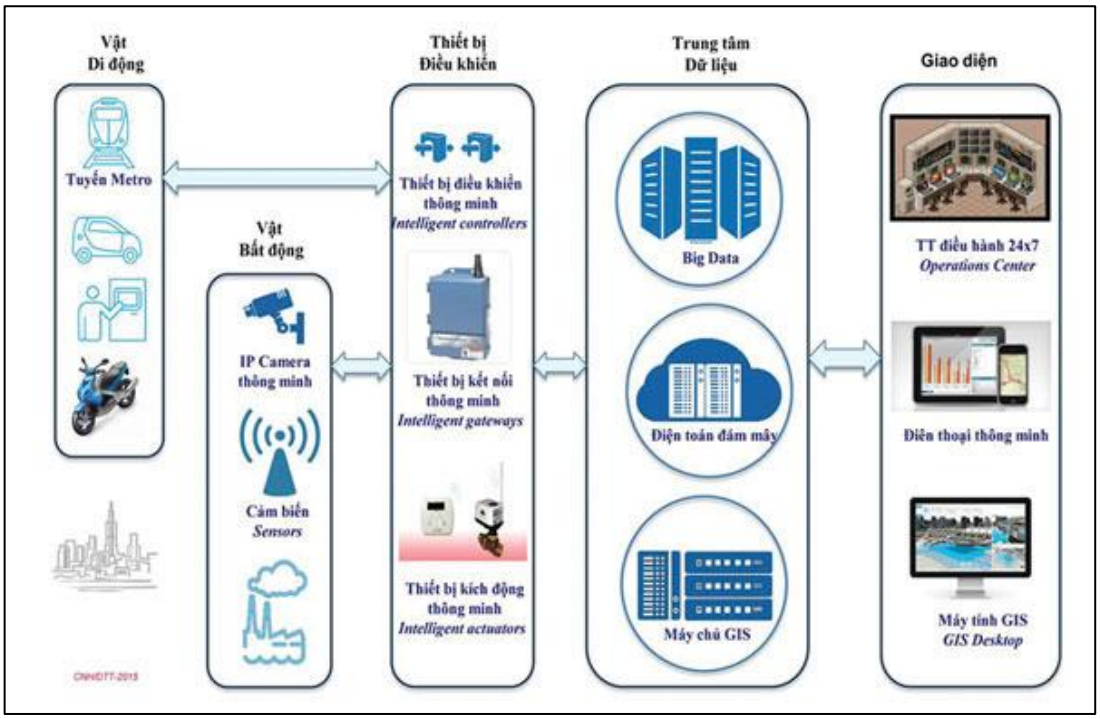
Hiện nay có nhiều cách hiểu về IoT và thiết bị IoT đã được đưa ra. Khái niệm IoT được Liên minh Viễn thông thế giới (ITU) chỉ ra rằng "IoT là một cơ sở hạ tầng mang tính toàn cầu cho xã hội thông tin, mang đến những dịch vụ tiên tiến bằng cách kết nối các vật thể (cả vật lý và ảo) dựa trên sự tồn tại của thông tin, dựa trên khả năng tương tác của các thông tin đó và dựa trên các công nghệ truyền thông" [53]. Với việc tương tác, thu thập, xử lý dữ liệu, các hệ thống IoT đã phát huy được mọi thứ để cung cấp dịch vụ cần thiết đến tất cả các ứng dụng khác nhau, đồng thời bảo đảm quyền riêng tư và tính bảo mật. Tương tự, Congressional Research Service [24] đã định nghĩa "IoT là một hệ thống các thiết bị có mối quan hệ với nhau được kết nối tạo thành một mạng có thể trao đổi dữ liệu mà không cần sự tương tác giữa người và máy". IoT được hiểu là một tập hợp các thiết bị điện tử có thể chia sẻ thông tin với nhau. Từ điển Oxford đã đưa ra khái niệm "IoT là việc kết nối các thiết bị trong các vật dụng hàng ngày thông qua internet để cho phép chúng chia sẻ dữ liệu với nhau"[2]. Theo nghiên cứu [51] IoT được định nghĩa "là một mạng lưới các đối tượng vật lý".

Về mặt tổng quát, chưa có một khái niệm thống nhất về thiết bị IoT, tuy nhiên với các khái niệm đã đưa ra cơ bản đã chỉ ra đặc điểm nổi bật nhất của loại thiết bị IoT là khả năng kết nối, chia sẻ và tương tác. Vì vậy, luận án sử dụng định nghĩa thiết bị IoT tổng quát như sau:

Định nghĩa 1.1. *Thiết bị IoT là các "vật thể" (vật lý hoặc ảo hóa) đa dạng về hệ điều hành và kiến trúc vi xử lý, có khả năng kết nối, chia sẻ dữ liệu, tương tác dựa trên các công nghệ truyền thông phục vụ các mục đích khác nhau của người sử dụng.*

Hiện nay, có nhiều thiết bị IoT phổ biến mới như điện thoại thông minh, đồng hồ thông minh, ti vi thông minh, IP Camera, thiết bị truy cập mạng không dây, ... Một mô hình ứng dụng các thiết bị IoT được minh hoạ qua hình 1.1. Trong đó, phần lớn các thiết bị IoT sử dụng hệ điều hành Linux nhúng [114]. Do đó, luận án chỉ tập trung thử nghiệm, đánh giá các mô hình đề xuất với tập tin định dạng ELF trên hệ điều hành Linux nhúng nhưng không mất đi tính tổng quát khi áp dụng trên các hệ điều hành khác.

[2] https://www.oxfordlearnersdictionaries.com/definition/english/internet-of-things



Hình 1.1. Mô hình ứng dụng thiết bị IoT[3].

Với khả năng kết nối và hoạt động, thiết bị IoT có những đặc trưng khác biệt với những công nghệ khác gồm:

- *Tính kết nối:* Các thiết bị IoT có thể kết nối và tương tác thường xuyên, liên tục với nhau. Vì vậy, có sự tác động, ảnh hưởng an ninh, an toàn thông tin giữa các thiết bị IoT trong cùng một hệ thống IoT.

- *Tính di động và khó kiểm soát*: Các thiết bị IoT thường có khả năng tự hành cao [135]. Các thiết bị IoT có thể truy cập mà không phụ thuộc vào không gian và thời gian thông qua môi trường mạng. Vì vậy, có thể dẫn đến các vấn đề an ninh, an toàn thông tin khi thu thập và trao đổi dữ liệu giữa các thiết bị IoT.

- *Tính biến động trạng thái*: Các thiết bị IoT thay đổi trạng thái linh hoạt phụ thuộc vào vị trí, chức năng và tốc độ di chuyển. Vì vậy, các giải pháp bảo mật cần được điều chỉnh phù hợp với trạng thái hoạt động của từng loại thiết bị IoT.

- *Tính không đồng nhất:* Các thiết bị IoT được sản xuất từ nhiều hãng, nhiều quốc gia trên thế giới, vì vậy nó có sự đa dạng về nền tảng hệ điều hành, kiến trúc vi xử lý. Bên cạnh đó, với việc ra đời của các thiết bị IoT mới, công nghệ mới và các mục đích kinh doanh khác, các hãng đã thực hiện nhiều thay đổi riêng trong kiến trúc vi xử lý của các thiết bị. Vì vậy, việc nghiên cứu tổng quát các thiết bị IoT trên thị trường là một công việc còn gặp nhiều thách thức.

- *Tính hạn chế tài nguyên thiết bị:* Theo C. Lévy-Bencheton [15] các thiết bị IoT được chia thành 2 loại chính là thiết bị hạn chế tài nguyên và hiệu năng cao. Theo Việt

[3] https://iot.dtt.vn/GiaiphapIoT.html

[64], thiết bị IoT hạn chế tài nguyên là các thiết bị IoT có cấu tạo hạn chế về dung lượng bộ nhớ, năng lực xử lý dữ liệu, băng thông truyền tải dữ liệu,… Nhằm mục đích kinh doanh, giảm thiểu chi phí sản xuất, các thiết bị IoT thường được thiết kế nhỏ gọn, bộ nhớ, khả năng lưu trữ, tính toán được trang bị ở mức hạn chế và ít có các cơ chế đảm bảo an ninh, an toàn thông tin. Thông tin chi tiết tài nguyên một số thiết bị IoT dòng SOHO đang được sử dụng thể hiện trong Bảng 1.1.

Bảng 1.1. Thông tin tài nguyên một số thiết bị IoT dòng SOHO.

Loại thiết bị	Hãng sản xuất	Phiên bản	Flash Memory	RAM	Số lượng thiết bị trực tuyến trên Zoomeye[4]
IP Camera	D-link	DCS-932L	4MB	32MB	477.291
	D-link	DCS-930L	4MB	32MB	466.058
	D-link	DCS-950G A4	4MB	32MB	261
Thiết bị định tuyến	D-Link	DIR-615 F1, F2	2MB	16MB	5
	Linksys	WRT120N	2MB	32MB	892
	Linksys	WRT110	4MB	16MB	423.776
	Netgear	DG834G v1, v2, v3, v4	4MB	16MB	98.082
	Netgear	WNR1000 v3	4MB	16MB	30.887
	Netgear	DG834GT	4MB	16MB	15.720
	Netgear	WNR612 v2	4MB	16MB	5.481
	TP-Link	TL-WR841N	4MB	32MB	930.064
	TP-Link	TL-WR840N	4MB	32MB	604.011
	TP-Link	TL-WR940N	4MB	32MB	567.443
	D-Link	DIR-615 C1, C2, D, E	4MB	32MB	462.004
Thiết bị chuyển mạch	D-link	Dlink DGS-1100-05	2MB	128MB	4
	D-link	Dlink DGS-1100-05PD	2MB	128MB	2
	D-link	Dlink DGS-1100-16	8MB	128MB	7
	FS	FS S3150-8T2FP	16MB	128MB	45
	D-link	Dlink DES-1210-28	16MB	128MB	42
	Netgear	Netgear GS310TP	32MB	128MB	171.489
	Netgear	Netgear GS110TP v3	32MB	128MB	100.297

Các thiết bị điển hình khảo sát đang hoạt động trên Internet trong Bảng 1.1 cho thấy sự hạn chế về bộ nhớ. Vì vậy, việc triển khai các giải pháp bảo mật sẵn có phổ biến hiện nay trên các thiết bị IoT gặp nhiều khó khăn và chưa phù hợp với các thiết bị hạn chế tài nguyên.

1.1.2. Khái niệm, đặc điểm của mã độc IoT

Một cách tổng quan, mã độc là các chương trình máy tính được tạo ra với mục đích làm hại đến tính mật, tính toàn vẹn hoặc tính sẵn sàng của dữ liệu, ứng dụng và hệ điều hành của của hệ thống [68]. Mã độc trên các thiết bị IoT là các mã độc nhằm vào các thiết bị IoT, chủ yếu là các thiết bị nhúng đa kiến trúc vi xử lý. Hầu hết mã độc trên các

[4] *https://www.zoomeye.hk/ (Truy cập tìm kiếm ngày 26/5/2024)*

thiết bị IoT được xây dựng dựa trên một loại ngôn ngữ lập trình và được biên dịch chéo (cross-compiler) thành tập tin thực thi trên nhiều kiến trúc vi xử lý khác nhau. Tập tin thực thi nhị phân thường chứa tất cả các thư viện liên kết để giảm sự phụ thuộc vào môi trường bên ngoài và tăng cơ hội thực thi trên các thiết bị IoT khác nhau.

Như vậy, tương tự mã độc trên thiết bị di động [7] và mã độc Linux [36] thì mã độc IoT được xem là mã độc lây nhiễm trên các thiết bị IoT hoặc hệ thống IoT. Với cách tiếp cận khái niệm như vậy và kết hợp với phạm vi nghiên cứu của đề tài, khái niệm mã độc IoT sử dụng trong luận án được xác định như sau:

Định nghĩa 1.2. *Mã độc IoT là mã độc có khả năng xâm nhập, lây nhiễm và hoạt động trên các thiết bị IoT.*

Theo nghiên cứu của John Aycock [55], các loại mã độc thường có 3 đặc trưng cơ bản gồm: Tính tự nhân bản, tính tăng trưởng và tính ký sinh.

- *Tính tự nhân bản (Self-replicating):* Mã độc lây nhiễm bằng cách tự tạo ra các bản sao chép mới hoặc các thể hiện của chính nó. Tuy nhiên, tính tự nhân bản không được xem xét nếu các bản sao hoặc các biến thể của mã độc được tạo dưới các tác động có chủ đích của người dùng.

- *Tính tăng trưởng (Population growth*): Là sự thay đổi lớn về số lượng mã độc, đặc biệt là mã độc có khả năng tự nhân bản. Mã độc không có tính tự nhân bản sẽ không có tính tăng trưởng, nhưng mã độc không có tính tăng trưởng thì vẫn có thể có tính tự nhân bản.

- *Tính ký sinh (Parasitic):* Nếu mã độc muốn tồn tại và thực hiện lây nhiễm thì cần có các tập tin thực thi khác để ký sinh (làm vật chủ). Mã độc chỉ lây nhiễm khi các tập tin vật chủ được thực thi.

Ngoài những đặc trưng chung của mã độc thì mã độc IoT có một số đặc trưng riêng biệt được tạo ra bởi các đặc điểm của thiết bị IoT [14], [25], [61] như sau:

- *Mục tiêu tấn công đa dạng*: Mã độc IoT tập trung vào các thiết bị IoT và mạng lưới thiết bị IoT đa dạng kiến trúc vi xử lý và hệ điều hành.

- *Lây lan nhanh chóng*: Mã độc IoT có thể lây lan nhanh, ảnh hưởng đến một số lượng lớn các thiết bị trong thời gian ngắn do số lượng các thiết bị IoT và sự kết nối của các thiết bị IoT.

- *Khó phát hiện:* Mã độc IoT có thể khó để phát hiện và loại bỏ khỏi các thiết bị do các thiết bị IoT thường ít triển khai các giải pháp bảo mật.

- *Tấn công mạng lưới:* Mã độc IoT có thể tấn công vào mạng lưới của các thiết bị IoT để lấy thông tin, tạo mạng botnet hoặc tấn công vào các hệ thống khác.

- *Khai thác các lỗ hổng bảo mật*: Mã độc IoT thường sử dụng các lỗ hổng bảo mật của các thiết bị IoT để tấn công, lây nhiễm.

Với sự đa dạng và tính không đồng nhất của các thiết bị IoT, ngày càng xuất hiện

nhiều mẫu mã độc có khả năng hoạt động trên nhiều dòng thiết bị IoT có sự khác biệt về hệ điều hành và kiến trúc vi xử lý. Trong luận án, NCS tập trung nghiên cứu phát hiện mã độc IoT đơn kiến trúc vi xử lý và mã độc IoT đa kiến trúc vi xử lý được tổng quát như sau:

Định nghĩa 1.3. *Mã độc IoT đơn kiến trúc là loại mã độc IoT được thiết kế để thực thi trên một kiến trúc bộ vi xử lý trung tâm cố định và định sẵn.*

Các kiến trúc bộ vi xử lý trung tâm của thiết bị IoT phổ biến như Intel, MIPS, ARM, SPARC, PowerPC. Mỗi kiến trúc của thiết bị IoT có nguyên lý và cơ chế hoạt động khác nhau. Vì vậy, các dòng mã độc IoT trên từng kiến trúc sẽ có các đặc trưng khác nhau. Việc phát hiện mã độc IoT trên mỗi loại kiến trúc cần có sự khác biệt để đem lại hiệu quả tối ưu nhất.

Định nghĩa 1.4. *Mã độc IoT đa kiến trúc là loại mã độc IoT có thể thực thi và hoạt động trên nhiều kiến trúc bộ vi xử lý trung tâm khác nhau.*

Để tăng khả năng lây lan trong môi trường IoT, một mã độc IoT có thể được thiết kế để hoạt động trên nhiều kiến trúc vi xử lý khác nhau. Việc phát hiện mã độc IoT đa kiến trúc cần áp dụng các phương pháp để xác định các biến thể của một loại mã độc và mã độc zero-day hoạt động trên các kiến trúc vi xử lý khác nhau. Trong phạm vi của luận án, mã độc IoT zero-day đa kiến trúc được định nghĩa như sau:

Định nghĩa 1.5. *Mã độc IoT zero-day đa kiến trúc là loại mã độc mới chưa được biết đến trước đây hoạt động trên các thiết bị IoT đa dạng nền tảng kiến trúc vi xử lý.*

1.1.3. Phân loại mã độc IoT

Trong các nghiên cứu thực hiện phân loại mã độc IoT [87], nhiều cách thức phân loại đã được các nghiên cứu đưa ra như dựa trên mục đích tấn công, phương thức lây nhiễm, cách thức hoạt động, mục tiêu thiết bị tấn công,….Tương tự đối với mã độc nói chung, mã độc IoT thường được các nghiên cứu phân loại và gán nhãn dựa trên dựa trên hoạt động của mã độc bao gồm các loại chính như sau:

- *Botnet:* Là loại mã độc có khả năng tạo ra các mạng lưới thiết bị IoT bị lây nhiễm để thực hiện các cuộc tấn công từ chối dịch vụ phân tán. Điển hình có thể kể đến như mã độc IoT botnet “Mirrai” với khả năng tự lây nhiễm vào các thiết bị IoT bằng cách khai thác các mật khẩu mặc định yếu của thiết bị IoT.

- *Worm*: Là loại mã độc có khả năng tự sao chép và lây lan từ thiết bị IoT này sang thiết bị IoT khác không cần thông qua sự tác động của người dùng. Ví dụ IoT worm “Linux.Darlloz” với có khả năng tự lây nhiễm diện rộng đối nhiều thiết bị định tuyến.

- *Trojan*: Là loại mã độc ẩn dưới dạng chương trình ứng dụng hợp pháp để lây nhiễm vào thiết bị. Ví dụ trojan “SoundMiner” có khả năng trích xuất dữ liệu từ các thiết bị sử dụng hệ điều hành Android.

- *Spyware*: Là loại mã độc gián điệp với khả năng lây nhiễm và thu thập thông tin

nhạy cảm như dữ liệu cá nhân, hình ảnh, âm thanh từ các thiết bị IoT. Ví dụ spyware "sKyWIper" với khả năng tấn công diện rộng trên các thiết bị IoT và đánh cắp thông tin, lắng nghe các tín hiệu Microphone, bật Bluetooth trên thiết bị nếu có và gửi thông tin dò quét được tới thiết bị điều khiển gần nhất của hacker.

- *Rootkit*: Là loại mã độc có khả năng ẩn giấu trên thiết bị nhằm truy cập từ xa tới thiết bị đã lây nhiễm thông qua chỉnh sửa nhân của hệ điều hành hoặc phần sụn của thiết bị IoT. Rootkit "Cloaker" đã được phát hiện với khả năng ẩn giấu thông qua khai thác đặc trưng của kiến trúc vi xử lý ARM.

- *Virus*: Là loại mã độc cần sự tương tác của người dùng để kích hoạt và lây nhiễm trên thiết bị. Với các đặc điểm này, virus chưa được các hacker quan tâm phát triển cho các thiết bị IoT tài nguyên hạn chế và thiết bị IoT ít tương tác với người dùng.

Ngoài ra, theo Aohui Wang [10], mã độc IoT có thể phân loại theo hình thức tấn công của mã độc gồm mã độc tấn công vét cạn và mã độc tấn công thực thi mã từ xa. Theo dự án IoT OWASP TOP 10 [52], trong năm 2019, lỗ hổng mật khẩu thiết bị yếu, phổ biến và dễ bẻ khoá được xếp hạng thứ nhất. Điều này cũng minh chứng cho việc ngày càng hình thành nhiều mã độc thực hiện tấn công dò quét mật khẩu trên thiết bị để tự động lây nhiễm diện rộng trong hệ thống IoT. Tiếp đến trong Bảng xếp hạng là các lỗ hổng về các dịch vụ truy cập thiết bị không an toàn, lỗ hổng các kết nối trong hệ sinh thái IoT không an toàn, lỗ hổng trong cập nhật hệ điều hành, thư viện,.. Vì vậy, các thiết bị IoT thiếu cơ chế an toàn này đang là mục tiêu phổ biến để hacker thực hiện tấn công tự động từ xa thông quan các mã độc.

1.1.4. Xu hướng phát triển của mã độc IoT

Với các đặc điểm nêu trên của thiết bị IoT, mã độc IoT ngày càng phát triển với nhiều kỹ thuật tấn công, lây nhiễm, hành vi độc hại và biến thể mới. Trong 10 năm trở lại đây, một số dòng mã độc IoT và các biến thể điển hình theo các năm được thể hiện chi tiết trong Bảng 1.2.

Bảng 1.2. Xu hướng phát triển của mã độc IoT thời gian qua.

Tên mã độc/Các biến thể	**Năm phát hiện**	**Đối tượng lây nhiễm**	**Đặc điểm chính**
Linux. Darlloz	2013	Các thiết bị IoT sử dụng nhiều kiến trúc vi xử lý	- Khai thác lỗ hổng mã CVE-2012-1823. - Có khả năng ngăn chặn người dùng truy cập tới thiết bị đã lây nhiễm. - Có khả năng xoá tập tin do các dòng mã độc khác đã sinh ra trên thiết bị.
Dofloo/ Spike/ MrBlack và các biến thể	2014	Các thiết bị IoT sử dụng kiến trúc ARM, MIPS	- Sử dụng Agent-handler để liên lạc và điều khiển mạng Botnet. - Giả mạo tập tin "rc.local" để duy trì tính thường trực trên thiết bị IoT. - Điều phối mật độ tấn công DDoS thông qua tính toán

			hiệu năng của thiết bị IoT.
Bashlite/ Gafgyt và các biến thể	2014	Các thiết bị IoT sử dụng nhiều kiến trúc vi xử lý	- Sử dụng Agent-handler để liên lạc và điều khiển mạng Botnet. - Có khả năng lây nhiễm đa kiến trúc vi xử lý. - Hình thành mạng Botnet để thực hiện tấn công DDoS qua các kỹ thuật Flood.
XOR. DDoS	2015	Nhiều dòng thiết bị IoT khác nhau	- Sử dụng mã hoá XOR để mã hoá các kết nối với máy chủ và mã nguồn của mã độc. - Hình thành mạng Botnet để thực hiện nhiều kỹ thuật tấn công DDoS.
Mirai	2016	Các thiết bị IoT sử dụng nhiều kiến trúc vi xử lý	- Khai thác lỗ hổng bảo mật của thiết bị IoT để tự lây lan. - Có thể biên dịch để thực thi trên nhiều loại kiến trúc vi xử lý. - Sử dụng các kỹ thuật tấn công DDoS đơn giản.
Bricker Bot	2017	Nhiều loại thiết bị IoT khác nhau	- Là một mã độc IoT botnet dựa trên busybox. - Tự lây nhiễm thông qua khai thác lỗ hổng bảo mật của thiết bị IoT. - Thực hiện nhiều hành vi độc hại như cấu hình lại thiết bị, thay đổi phần sụn thiết bị,… - Có khả năng xoá dữ liệu, phá hủy các thiết bị sau khi đã lây nhiễm.
Triton	2017	Hệ thống thiết bị điều khiển công nghiệp	- Có khả năng làm hỏng thiết bị trong hệ thống điều khiển điện công nghiệp. - Có khả năng thực thi trên nhiều kiến trúc vi xử lý khác nhau.
Satori	2017	Các thiết bị IoT như router wifi	- Tấn công các thiết bị thông qua khai thác lỗ hổng CVE-2014-8361 - Sử dụng một số kỹ thuật tấn công tương tự mã độc Mirai. - Tạo ra mạng IoT botnet lớn thông qua việc tập trung lây nhiễm vào các thiết bị router wifi. - Có khả năng thay đổi địa chỉ ví tiền "ảo" như Bitcoin, ETH để đánh cắp tài khoản.
VPN Filter	2018	Các thiết bị định tuyến và thiết bị lưu trữ trên mạng	- Có khả năng đánh cắp dữ liệu lưu trữ trên thiết bị. - Có khả năng tấn công nhiều loại thiết bị định tuyến khác nhau và lây nhiễm thông qua khai thác lỗ hổng bảo mật. - Có thể tồn tại lâu dài trên thiết bị ngay cả khi thiết bị được khởi động lại.
Hakai	2018	Các thiết bị định tuyến	- Là một loại mã độc IoT botnet. - Tấn công các thiết bị định tuyến thông qua khai thác lỗ hổng zero-day hoặc tấn công brute-force. - Dựa trên kỹ thuật mã độc Mirai và Gafgyt đã sử dụng. - Sử dụng nhiều kỹ thuật tấn công DDoS.
Chalubo	2019	Các thiết bị IoT sử dụng nhiều loại kiến trúc vi xử lý	- Là loại mã độc IoT có khả năng thực hiện tấn công DDoS. - Tập trung tấn công các máy chủ sử dụng dịch vụ SSH qua môi trường Internet. -Sử dụng mật mã dòng ChaCha.
HEH	2020	Các thiết bị	- Lây nhiễm vào các thiết bị IoT có sử dụng dịch vụ Telnet

(Hoax calls)		IoT sử dụng các kiến trúc vi xử lý khác nhau	hoạt động trên cổng dịch vụ 23 hoặc 2323 TCP/IP. - Cho phép thực hiện shellcode tấn công từ xa thiết bị đã lây nhiễm. - Lây nhiễm được trên nhiều loại thiết bị IoT sử dụng kiến trúc vi xử lý khác nhau.
Mozi	2021	Thiết bị IoT như network gateways và DVR	- Là một loại IoT botnet. - Có khả năng tạo mạng lưới IoT botnet và lấy cắp thông tin trên thiết bị. - Khai thác các lỗ hổng bảo mật đã biết sau khi lây nhiễm. - Sử dụng kỹ thuật lẩn tránh UPX anti-unpack.
Rapper Bot	2022	Các thiết bị IoT	-Sử dụng một phần chức năng của Mirai. -Tập trung tấn công các thiết bị sử dụng hệ điều hành Linux có sử dụng dịch vụ SSH. -Sử dụng mã hoá XOR 2 lớp để phòng tránh việc phân tích và lẩn tránh sự phát hiện.

Qua phân tích các dòng mã độc IoT nêu trên, xu hướng phát triển của mã độc IoT hiện nay thường tập trung vào các nội dung sau đây:

- *Tấn công, lây nhiễm vào các thiết bị IoT có ít cơ chế bảo mật*: Một số dòng mã độc IoT được thiết kế để tấn công, lây nhiễm vào các thiết bị IoT thông qua các lỗ hổng bảo mật, đặc biệt khai thác các lỗ hổng đã biết trên các thiết bị IoT để lây lan trong một hệ thống IoT như Mirrai (2016), Bricker Bot, Satori (2017), VPN Filter, Hakai (2018), HEH (2020), Mozi (2021), ...

- *Sử dụng các kỹ thuật khai thác mới*: Mã độc IoT có thể sử dụng các chức năng khai thác tự động hóa để tìm ra các lỗ hổng và lây lan nhanh chóng qua môi trường mạng như Mirrai (2016), Hakai (2018), Rapper Bot (2022), ...

- *Sử dụng trí tuệ nhân tạo*: Mã độc IoT sử dụng trí tuệ nhân tạo để phát triển các kỹ thuật tấn công mới, tìm kiếm, khai thác các lỗ hổng trong các thiết bị IoT và lẩn tránh các biện pháp bảo mật. Bên cạnh đó, các tri thức của các mã độc cũ cũng được sử dụng để tạo ra các biến thể và các dòng mã độc mới có các hành vi phức tạp và tấn công nguy hiểm hơn trên các thiết bị IoT như mã độc Mirrai (2016) và các biến thể về sau.

- *Đa dạng hoá mục tiêu*: Các mã độc IoT đã đa dạng hoá mục tiêu tấn công để có thể lây nhiễm vào nhiều loại thiết bị IoT khác nhau. Các dòng mã độc hoạt động đa nền tảng xuất hiện ngày càng nhiều và nguy hiểm hơn.

- *Tăng cường tính năng phân tán:* Nhiều dòng mã độc IoT được thiết kế phức tạp hơn và hoạt động phân tán trên nhiều thiết bị IoT. Các thành phần độc hại được ẩn giấu trên nhiều thiết bị với các kỹ thuật khác nhau.

- *Tăng cường tính tương tác*: Mã độc IoT có khả năng tương tác với các thiết bị khác trong mạng để phát tán và lây lan. Chúng có thể tìm cách tấn công và chiếm quyền kiểm soát các thiết bị IoT khác, tạo thành các mạng botnet lớn.

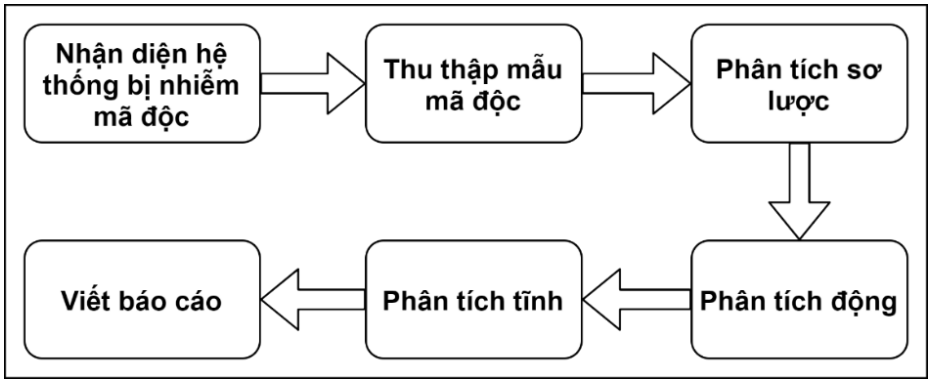
Các mã độc trên thiết bị IoT thường được phát triển và tạo ra nhiều biến thể để

hoạt động trên một kiến trúc vi xử lý định sẵn hoặc có khả năng hoạt động trên nhiều kiến trúc vi xử lý khác nhau. Ngày càng xuất hiện nhiều mã độc, biến thể mã độc mới có hành vi phức tạp, lây lan trên diện rộng các thiết bị IoT. Vì vậy, để phát hiện và ngăn chặn các mã độc IoT hiện nay, cần nghiên cứu áp dụng học máy trong phát hiện cả mã độc IoT đơn kiến trúc và mã độc IoT đa kiến trúc.

1.2. Tổng quan về phát hiện mã độc IoT dựa trên học máy

1.2.1. Quy trình phân tích phục vụ phát hiện mã độc IoT

Bài toán phân tích mã độc được chia thành 3 nhóm chính gồm [132]: (1) Phát hiện mã độc nhằm cung cấp khả năng phân biệt các tập tin mã độc giữa các tập tin lành tính; (2) Phân loại mã độc nhằm cung cấp khả năng xác định mẫu tập tin thuộc về loại mã độc nào; (3) Xác định sự tiến hóa của mã độc nhằm phát hiện ra mối liên hệ kế thừa, di truyền giữa các mã độc. Với bài toán phát hiện mã độc thường được các chuyên gia phân tích thực hiện thông qua quá trình phân tích tập tin thực thi trong hình 1.2 [68].



Hình 1.2. Quy trình phân tích mã độc.

- *Bước 1*: Nhận diện hệ thống bị nhiễm mã độc thông qua việc phát hiện sự cố và xác minh nguồn lây nhiễm mã độc.

- *Bước 2*: Thu thập mẫu mã độc và phân loại mẫu mã độc

- *Bước 3*: Phân tích sơ lược nhằm phân tích đặc trưng tập tin đã thu thập, kiểm tra các kỹ thuật nén tập tin sử dụng, kiểm tra các kết quả phân tích và cơ sở dữ liệu mã độc đã có trước đó.

- *Bước 4:* Phân tích động tập tin thực thi. Phân tích động là kỹ thuật giám sát các hành vi trong quá trình thực thi các tập tin đó, từ đó phát hiện các hành vi độc hại, hành vi bất thường. Một số môi trường được sử dụng để phân tích động mã độc IoT như các sandbox, các bộ mô phỏng dựa trên QEMU, thiết bị thực tế cài đặt tác tử.

- *Bước 5:* Phân tích tĩnh tập tin thực thi. Phân tích tĩnh là phương pháp phân tích mà không cần thực thi tập tin để phát hiện mã độc hại. Quá trình phân tích tĩnh đòi hỏi người phân tích cần xem xét mã nguồn của tập tin hoặc một định dạng có thể đọc hiểu sau quá trình dịch ngược tập tin như mã assembly và hiểu được luồng thực thi của chương trình. Quá trình dịch ngược tập tin được thực hiện với sự hỗ trợ một số công cụ dịch ngược.

- *Bước 6:* Viết báo cáo về hoạt động của tập tin nhằm xác định hành vi độc hại và phân loại mã độc.

Trong quy trình trên, hai bước phân tích động và phân tích tĩnh là rất cần thiết có thể tìm hiểu, đánh giá đầy đủ hành vi, hoạt động của một tập tin trong thực tế nhưng lại đòi hỏi nhiều kiến thức chuyên sâu của người phân tích. Khi số lượng mẫu tập thực thi thu thập ngày càng nhiều và phức tạp, việc thực hiện các bước phân tích sơ lược, phân tích động và phân tích tĩnh để xác định hành vi độc hại của tập tin là không đáp ứng được yêu cầu thực tiễn. Vì vậy, ứng dụng học máy trong xây dựng các mô hình phát hiện mã độc IoT là cần thiết trong việc xác định hành vi độc hại của một tập tin thực thi. Kết quả so sánh phân tích mã độc bởi chuyên gia phân tích và sử dụng học máy được thể hiện trong Bảng 1.3.

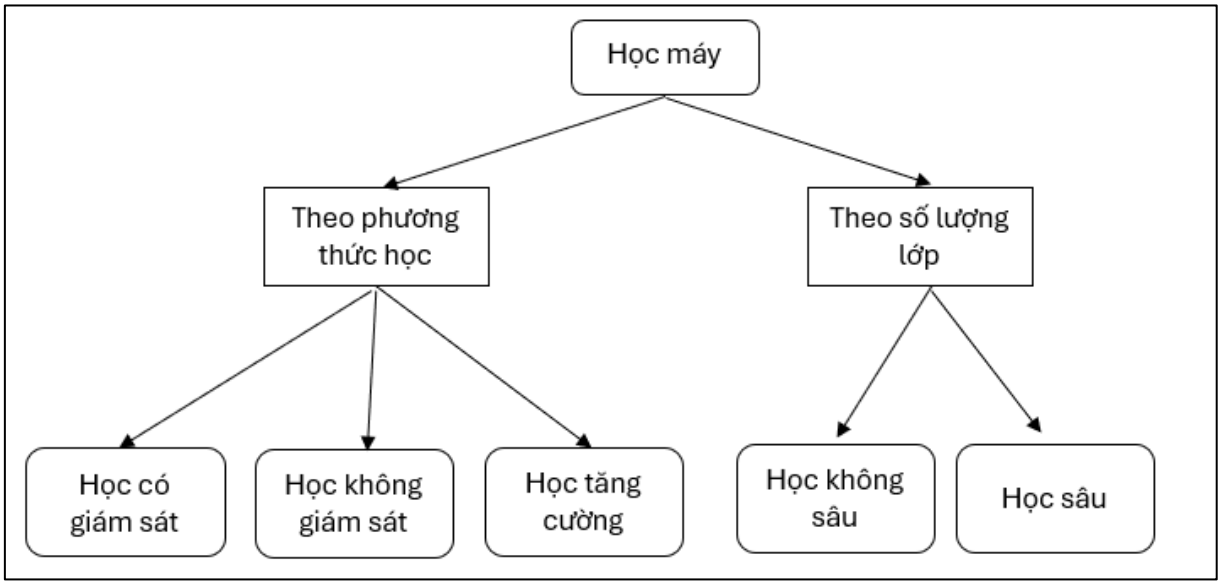
Bảng 1.3. So sánh phân tích mã độc bởi chuyên gia phân tích và sử dụng học máy.

	Phân tích mã độc bởi chuyên gia phân tích	**Phân tích mã độc bởi các mô hình học máy**
Khả năng tự động hoá	Cần sự can thiệp của con người.	Có thể tự động hoá cao.
Khả năng mở rộng phân tích	Không hiệu quả khi số lượng tập tin lớn.	Xử lý được số lượng tập tin, quy mô lớn.
Tốc độc phân tích	Cần nhiều thời gian phân tích từng mẫu tập tin, không đáp ứng kịp thời việc phát hiện và phản ứng trước các cuộc tấn công mã độc.	Phân tích được nhiều tập tin trong thời gian ngắn, đáp ứng kịp thời việc phát hiện và phản ứng trước các cuộc tấn công mã độc.
Khả năng phát hiện các biến thể mới	Khó nhận biết các mã độc chưa từng phân tích hoặc mã độc sử dụng các kỹ thuật tấn công mới	Có khả năng nhận diện các mẫu mã độc mới và các biến thể mã độc.
Phát hiện dựa trên các đặc trưng phức tạp	Khó nhận diện các mẫu phức tạp, các mối quan hệ ẩn giữa các thành phần mã độc.	Có khả năng nhận diện, phân loại dựa trên nhiều đặc điểm, mỗi quan hệ phức tạp.
Sai sót phân tích	Có thể sai sót do lỗi chủ quan của con người hoặc thiếu kinh nghệm phân tích.	Hoạt động nhất quán, không bị ảnh hưởng bởi yếu tố chủ quan.
Cập nhật, điều chỉnh phương pháp	Khó cập nhật kiến thức liên tục và điều chỉnh phương pháp khi xuất hiện các mã độc mới.	Có thể cập nhập mô hình liên tục thông qua huấn luyện và cải thiện dữ liệu mới.
Kết hợp nhiều nguồn dữ liệu	Khó tích hợp và phân tích nhiều nguồn dữ liệu khác nhau.	Có khả năng kết hợp và phân tích nhiều nguồn dữ liệu khác nhau.

Các mô hình phát hiện mã độc IoT dựa trên học máy có thể thay thế các chuyên gia phân tích mã độc để phát hiện và phân loại mã độc trong tập tin thực thi. Việc áp dụng học máy trong xây dựng mô hình phát hiện mã độc IoT không chỉ giúp tự động hóa và tăng tốc quá trình phát hiện mà còn nâng cao hiệu quả và khả năng phát hiện các mẫu mã độc mới và phức tạp. Điều này giúp hệ thống đảm bảo an ninh, an toàn mạng luôn ở trạng thái sẵn sàng và hiệu quả trong việc bảo vệ hệ thống IoT trước các mối đe

dọa ngày càng tinh vi của kẻ tấn công.

Học máy (Machine Learning) có liên hệ chặt chẽ với tính toán thống kê, tập trung vào dự đoán nhãn bằng cách sử dụng các hệ thống máy tính. Hiện nay có nhiều cách phân loại các thuật toán học, nhóm tác giả Janiesch [48] đưa ra 2 cách phân loại các thuật toán học máy như hình 1.3.



Hình 1.3. Phân loại thuật toán học máy.

- *Theo phương thức học*, các thuật toán học máy được chia làm 3 loại gồm: Học có giám sát (Supervise learning) là thuật toán dự đoán đầu ra của một dữ liệu mới dựa trên các cặp đầu vào, đầu ra đã biết từ trước. Các cặp dữ liệu này còn được gọi là "dữ liệu, nhãn". Học không giám sát (Unsupervise learning) là việc học chỉ có dữ liệu đầu vào mà không biết được nhãn hay đầu ra. Thuật toán học không giám sát dựa vào cấu trúc của dữ liệu để thực hiện một công việc nào đó như phân nhóm. Học tăng cường (Reinforcement learning) là việc học sử dụng mô tả trạng thái hiện tại của hệ thống, chỉ định một mục tiêu, cung cấp một danh sách các hành động cho phép và các ràng buộc môi trường để cho ra các kết quả thay vì cung cấp các cặp đầu vào và đầu ra. Việc học được thực hiện thông qua quá trình hướng đến mục tiêu dựa trên nguyên tắc thử sai để tối đa hóa một phần thưởng hoặc một mục tiêu.

- *Theo số lượng lớp*, các thuật toán học máy được phân chia thành 2 loại gồm: Học không sâu (Shallow learning) là các thuật toán học máy mà chỉ có một hoặc ít lớp ẩn trong kiến trúc của chúng. Học không sâu thường đề cập đến các mô hình học máy mà không có số lượng lớp ẩn lớn hoặc không có khả năng tự học và hiểu biết sâu về dữ liệu. Các thuật toán học không sâu thường được sử dụng trong các nhiệm vụ đơn giản, có cấu trúc, nơi dữ liệu đầu vào không quá phức tạp và không yêu cầu mức độ biểu diễn sâu về thông tin. Các thuật toán học không sâu bao gồm hồi quy tuyến tính, cây quyết định, Naive Bayes và mạng nơ-ron tiêu chuẩn với một hoặc hai lớp ẩn. Các mô hình này

thường được sử dụng trong các nhiệm vụ phân loại và dự đoán đơn giản, nơi các mô hình không cần phải hiểu biết sâu về dữ liệu hoặc tạo ra biểu diễn phức tạp của thông tin. Học sâu (Deep learning) là việc xây dựng và huấn luyện các mô hình máy học sử dụng các kiến trúc mạng nơ-ron có nhiều lớp, đặc biệt là các mạng nơ-ron nhân tạo với số lớp ẩn lớn. Mục đích của học sâu là để mô hình hóa và tìm hiểu các biểu diễn phức tạp từ dữ liệu không cấu trúc hoặc dữ liệu có cấu trúc phức tạp.

Các kết quả nghiên cứu mô hình phát hiện mã độc sử dụng phương pháp học máy như [40], [64], [114], [126] cho thấy các thuật toán học máy không sâu có giám sát sử dụng trong bài toán phân lớp tập tin thực thi như: Decision Tree, k Nearest Neighbors, Support Vectơ Machine, Random Forest,… thường được sử dụng và đã đem lại hiệu quả trong các kịch bản thử nghiệm với tập dữ liệu phù hợp do sự đơn giản, dễ triển khai trong thực tế. Tuy nhiên, hiệu quả của các thuật toán học máy này phụ thuộc nhiều vào độ chính xác của các đặc trưng đã trích xuất và lựa chọn. Bên cạnh đó, việc nghiên cứu các phương pháp trích xuất phù hợp với các đặc trưng hành vi giúp cải thiện hiệu quả phát hiện mã độc. Việc trích chọn, tiền xử lý các đặc trưng yêu cầu nhiều kiến thức chuyên gia đối với mỗi đặc trưng trích xuất từ tập tin thực thi. Do đó, các mô hình phát hiện mã độc IoT sử dụng các thuật toán học máy không sâu sử dụng các đặc trưng của tập tin thực thi vẫn còn các hạn chế, đặc biệt trong khả năng phát hiện mã độc mới.

Để khắc phục hạn chế trên của các mô hình dựa trên học không sâu, một số mô hình phát hiện mã độc dựa trên học sâu đã được đề xuất để khai thác các biểu diễn phức tạp từ tập dữ liệu huấn luyện. Tuy nhiên, các mô hình học sâu thường đòi hỏi một lượng lớn dữ liệu huấn luyện và môi trường có khả năng tính toán lớn, dữ liệu đầu vào của các mô hình học sâu phải được xử lý và chuyển đổi sang dạng số, tức là dựa vào một không gian mới thường gọi là “embedding”. Trong phạm vi luận án, thuật ngữ “embedding” được hiểu như sau:

Định nghĩa 1.6. Embedding là cách biểu biễn một đặc trưng dạng chuỗi (mã thực thi, lời gọi hệ thống,…) dưới dạng vectơ trong không gian nhiều chiều. Trong luận án, embedding được hiểu là biểu diễn vectơ.

Trong ngôn ngữ tự nhiên, embedding giúp mã hoá (biểu diễn) ý nghĩa của từ hoặc cụm từ vào các vectơ số học. Biểu diễn từ thường tạo ra các vectơ có kích thước cố định trong không gian vectơ. Các từ có ý nghĩa tương tự sẽ có biểu diễn gần nhau trong không gian vectơ này. Hiện nay, có nhiều các giải pháp để biểu diễn vectơ như Word2vec, CBOW, GloVe, Doc2Vec[66] hoặc các mô hình học sâu như FastText, Transformer, BERT, … Lựa chọn được phương pháp embedding phù hợp sẽ giúp xây dựng các mô hình có khả năng hiểu được ngữ nghĩa và mối quan hệ giữa các từ được biểu diễn.

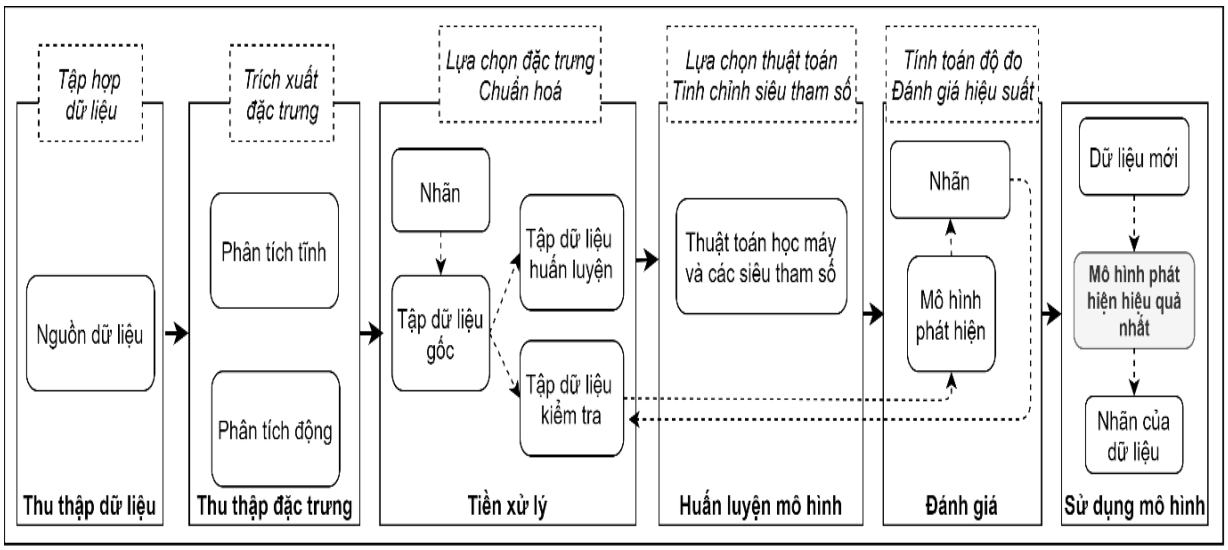
Vì vậy, trong phát hiện mã độc IoT dựa trên học máy, cần lựa chọn các thuật toán

học máy phù hợp với đặc điểm của từng loại đặc trưng trích xuất từ tập tin và yêu cầu cụ thể của bài toán triển khai mô hình phát hiện trong môi trường IoT hạn chế tài nguyên.

1.2.2. Mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng của tập tin thực thi

Mô hình phát hiện mã độc IoT dựa trên học máy là một mô hình được xây dựng dựa trên sử dụng các kỹ thuật và phương pháp học máy để phát hiện mã độc từ tập dữ liệu đầu vào. Mô hình là sự kết hợp giữa việc sử dụng các kỹ thuật học máy với các đặc trưng đã thu thập để xây dựng các bộ phân lớp tập tin hoặc dự đoán mã độc. Mục tiêu của mô hình phát hiện mã độc IoT dựa trên học máy là xác định những mẫu, đặc trưng hoặc hành vi có thể liên quan đến mã độc dựa trên dữ liệu huấn luyện trước đó.

Quá trình xây dựng một mô hình phát hiện mã độc IoT dựa trên học máy thường bao gồm 6 bước được mô tả trong hình 1.4.



Hình 1.4. Quá trình xây dựng mô hình phát hiện mã độc IoT dựa trên học máy.

❖ *Bước 1: Thu thập dữ liệu*

Thu thập dữ liệu là việc tập hợp các mẫu tập tin mã độc hoặc lành tính từ các nguồn tin cậy. Quá trình thu thập dữ liệu phụ thuộc vào mục tiêu bài toán xây dựng mô hình phát hiện mã độc IoT. Dữ liệu có thể được thu thập từ các nguồn khác nhau như các tập tin thực thi, cơ sở dữ liệu đã công bố, các cảm biến, hệ thống honeypot, honeynet trong hệ thống mạng, tập tin trích xuất từ thiết bị thực tế,...

Với đặc điểm hoạt động của thiết bị IoT và mã độc IoT, việc thu thập các tập tin trên thiết bị IoT hạn chế tài nguyên đặt ra nhiều khó khăn, thách thức. Các tập dữ liệu mã độc IoT đã công bố còn hạn chế so với các tập dữ liệu mã độc truyền thống. Đặc biệt, tập dữ liệu mã độc hoạt động trên nhiều hệ điều hành, nhiều kiến trúc vi xử lý chưa phổ biến còn hạn chế về số lượng và loại mã độc. Số lượng các mẫu trong các tập dữ liệu chưa có sự cân bằng giữa các nhãn. Vì vậy, cần có các nghiên cứu nhằm tăng cường tập dữ liệu phục vụ xây dựng các mô hình phát hiện mã độc IoT hiệu quả hơn.

❖ *Bước 2: Thu thập đặc trưng*

Trong phân tích và phát hiện mã độc, dựa trên phương pháp thu thập đặc trưng từ tập tin thực thi có thể chia đặc trưng thành hai dạng chính bao gồm: Đặc trưng động và đặc trưng tĩnh. Đặc trưng động là đặc trưng thu thập thông qua kỹ thuật phân tích động tập tin. Các đặc trưng động thường được sử dụng trong xây dựng mô hình phát hiện mã độc IoT dựa trên học máy bao gồm các tác vụ mà mã độc thực hiện, các lời gọi hệ thống, các sự kiện được sinh ra trong quá trình thực thi, thông tin chiếm dụng tài nguyên của thiết bị mục tiêu, các gói tin được trao đổi qua mạng. Đặc trưng tĩnh là đặc trưng được thu thập thông qua kỹ thuật phân tích tĩnh tập tin. Các đặc trưng tĩnh thường được sử dụng trong xây dựng mô hình phát hiện mã độc IoT dựa trên học máy gồm các thuộc tính của tập tin như chuỗi string, mã hash, chuỗi opcode, đồ thị luồng điều khiển, thông tin header của tập tin.

Phân tích tĩnh và phân tích động tập tin đều có những ưu, nhược điểm được trình bày trong Bảng 1.4. Các đặc trưng động và đặc trưng tĩnh đã thu thập sẽ được lựa chọn phù hợp với từng yêu cầu của bài toán để tiến hành tiền xử lý và huấn luyện các mô hình phát hiện mã độc IoT hiệu quả.

Bảng 1.4. Ưu và nhược điểm của phân tích tĩnh và phân tích động tập tin.

Phương pháp	**Phân tích tĩnh**	**Phân tích động**
Ưu điểm	- Tài nguyên thực hiện phân tích và thời gian thực thi thấp. - Chi tiết hoá được toàn bộ luồng điều khiển chương trình. - Xác định đầy đủ các khả năng kích hoạt của chương trình thực thi. - Không cần xây dựng môi trường thực thi tập tin.	- Loại bỏ được ảnh hưởng của các kỹ thuật gây rối. - Không phục thuộc vào công cụ dịch ngược và ngôn ngữ lập trình. - Có khả năng phát hiện được mã độc đa hình, mã hoá. - Giám sát được quá trình thực thi cụ thể của một chương trình để quyết định một tập tin là mã độc hay không.
Nhược điểm	- Không hiệu quả đối với mã độc sử dụng các kỹ thuật gây rối. - Không hiệu quả đối với mã độc sử dụng kỹ thuật đa hình và mã hóa. - Phụ thuộc nhiều vào khả năng dịch ngược từ tập tin thực thi.	- Phụ thuộc vào khả năng mô phỏng môi trường thực thi và giám sát hành vi tập tin khi thực thi. - Có nguy cơ lây nhiễm mã độc với môi trường bên ngoài hệ thống phân tích. - Cần xác định điều kiện kích hoạt để chương trình thực thi.

Bên cạnh đó, với cách tiếp cận dựa trên biểu diễn có thể có các đặc trưng gồm: Đặc trưng dạng chuỗi, đặc trưng dạng đồ thị, đặc trưng dạng hình ảnh, đặc trưng dạng số,… Các đặc trưng dạng chuỗi thường được sử dụng trong phát hiện mã độc IoT gồm: Chuỗi strings, chuỗi API calls, chuỗi system calls, chuỗi opcode, … Mỗi loại đặc trưng dạng chuỗi sẽ được thu thập, trích xuất thông qua các phương pháp khác nhau để phục vụ xây dựng mô hình phát hiện mã độc IoT hiệu quả.

Với mỗi đặc trưng thu thập được và dạng biểu diễn sẽ các phương pháp xử lý và trích chọn đặc trưng khác nhau phục vụ xây dựng các mô hình phát hiện mã độc IoT dựa trên học máy phù hợp. Để đạt được các mục tiêu đã đặt ra trong luận án, NCS sẽ tiếp cận các đặc trưng biểu diễn dạng chuỗi được trích xuất từ phân tích động và phân tích tĩnh tập tin thực thi để xây dựng các mô hình phát hiện mã độc IoT đơn kiến trúc và mã độc IoT đa kiến trúc.

❖ *Bước 3: Tiền xử lý*

Đây là một giai đoạn quan trọng trong xây dựng mô hình, nó giúp xây dựng mô hình học máy chính xác hơn. Dữ liệu thô ban đầu có một số đặc điểm như dữ liệu bị thiếu sót, không nhất quán, nhiễu, vì vậy dữ liệu cần được xử lý trước khi đưa vào huấn luyện mô hình học máy. Sau khi chuẩn hóa, dữ liệu được tiền xử lý trong trường hợp số chiều của vectơ đặc trưng quá lớn. Điều này sẽ giúp tối ưu hóa kết quả dự đoán và giảm thời gian thực thi mô hình học máy. Vì vậy, một số kỹ thuật trích chọn đặc trưng đã được phát triển để giải quyết vấn đề giảm thiểu các biến không liên quan và dư thừa trong tập đặc trưng. Lựa chọn đặc trưng phù hợp sẽ giúp hiểu rõ dữ liệu, giảm yêu cầu tính toán, giảm kích thước của vectơ đặc trưng và cải thiện hiệu suất của mô hình phát hiện.

Sau đó, dữ liệu được chia thành tập huấn luyện, tập kiểm tra để phục vụ huấn luyện và đánh giá mô hình. Trong xây dựng mô hình phát hiện mã độc IoT dựa trên học máy, có nhiều phương pháp chia dữ liệu huấn luyện và đánh giá như phương pháp holdout, cross-validation, stratified sampling, leave-one-out, bootstrapping,… Mỗi phương pháp chia dữ liệu huấn luyện, đánh giá có những ưu và nhược điểm riêng biệt. Lựa chọn phương pháp cần phù hợp với tính chất của dữ liệu, số lượng mẫu và mục tiêu của bài toán phát hiện mã độc IoT cụ thể. Ưu và nhược điểm của các phương pháp được thể hiện trong Bảng 1.5.

Bảng 1.5. So sánh một số phương pháp phân chia tập dữ liệu huấn luyện và đánh giá mô hình phát hiện mã độc IoT.

Phương pháp	Phân chia	Ưu điểm	Nhược điểm	Phù hợp bài toán
Holdout validation [104]	Phân chia dữ liệu thành hai phần: Một phần dùng để huấn luyện mô hình và một phần để đánh giá hiệu suất của mô hình.	Đơn giản, dễ thực hiện.	Kết quả đánh giá có thể không ổn định vì sự biến đổi của dữ liệu.	Dữ liệu lớn và ít nhiễu.
Cross-validation [102]	Chia dữ liệu thành k-fold. Mỗi lượt, một phần của dữ liệu được sử dụng để đánh giá và các phần còn lại được sử dụng để huấn luyện mô hình. Kết quả được lấy trung bình từ các lượt.	Đánh giá mô hình tốt hơn với dữ liệu nhỏ.	Tăng đáng kể thời gian tính toán, đặc biệt với dữ liệu lớn.	Dữ liệu nhỏ và cần đánh giá chính xác.

Stratified sampling [67]	Phân chia dữ liệu đảm bảo các nhóm con (tầng) khác nhau trong một tập có thể đại diện cho tập mẫu đó. Việc lấy các mẫu từ một tập sẽ dựa trên việc xác định kích thước cho mỗi tầng (tỷ lệ có thể cố định hoặc không cố định cho mỗi tầng).	Đảm bảo tính đại diện và độ chính xác của mẫu trong nghiên cứu thống kê, đặc biệt khi tập dữ liệu không đồng nhất.	Cần xác định kích thước cho mỗi tầng phù hợp với từng bài toán và đặc điểm của tập dữ liệu.	Dữ liệu không cân đối về số lượng mẫu giữa các lớp.
Leave-one-out [118]	Huấn luyện mô hình với tất cả các mẫu ngoại trừ một mẫu, sau đó đánh giá trên mẫu đã được bỏ ra. Lặp lại cho tất cả các mẫu.	Cung cấp thông tin chính xác nhất về hiệu suất mô hình.	Thời gian tính toán lớn khi dữ liệu lớn, có thể dẫn đến "overfitting".	Dữ liệu nhỏ và cần đánh giá chính xác.
Boot-Strapping [118]	Tạo ra nhiều bộ dữ liệu mới từ dữ liệu gốc.	Tạo ra dữ liệu mẫu có thể phong phú hơn.	Có thể dẫn đến overfitting nếu không sử dụng đúng cách.	Dữ liệu ít và cần phong phú hơn.

❖ *Bước 4: Huấn luyện mô hình*

Huấn luyện là việc sử dụng tập dữ liệu đã tiền xử lý để huấn luyện mô hình học máy và tinh chỉnh các tham số của các thuật toán học máy, mạng học sâu nhằm tối ưu hóa hiệu suất của mô hình. Việc lựa chọn các thuật toán học máy, mạng học sâu phù hợp là cơ sở quan trọng trong xây dựng một mô hình phát hiện mã độc IoT có khả năng ứng dụng trong thực tế.

Để huấn luyện mô hình, tập hợp dữ liệu huấn luyện được sử dụng để tinh chỉnh các tham số và để kiểm tra hiệu suất của bộ phân lớp thì tập dữ liệu kiểm tra được sử dụng. Trong quá trình huấn luyện mô hình, dữ liệu kiểm tra không được sử dụng để huấn luyện bộ phân lớp.

❖ *Bước 5: Đánh giá*

Đánh giá là một phần quan trọng trong quy trình phát triển mô hình, nó giúp tìm ra mô hình tốt nhất để đại diện cho dữ liệu và mô hình được chọn sẽ hoạt động tốt trong tương lai. Sau khi huấn luyện mô hình, đánh giá mô hình bằng cách sử dụng tập dữ liệu kiểm tra và tính toán các độ đo để đánh giá hiệu quả của mô hình. Nếu kết quả đánh giá chưa tốt, cần quay trở lại bước tiền xử lý và tiến hành huấn luyện lại. Nhiều cách thức đã được đưa ra trong các nghiên cứu để so sánh và đánh giá hiệu quả của mô hình phát hiện mã độc IoT dựa trên học máy gồm:

- *Kiểm tra độ tin cậy mô hình dựa trên các dữ liệu đã được gán nhãn*: Kiểm tra độ tin cậy của mô hình bằng cách đưa vào các mẫu đã được phân loại hoặc xác định nhãn để xem xét mô hình có thể phân loại (gán nhãn) đúng hay không. Nếu mô hình không thể gán nhãn đúng theo nhãn đã được xác định thì mô hình có thể không đáng tin cậy và

cần được điều chỉnh.

- *Đánh giá độ tin cậy của mô hình trên các tập dữ liệu mới:* Để đánh giá độ tin cậy của mô hình trên các tập dữ liệu mới, có thể sử dụng phương pháp cross-validation hoặc hold-out validation.

- *Đánh giá dựa trên tiêu chí các độ đo*: Một số độ đo thường được các nghiên cứu sử dụng để đánh giá độ chính xác của mô hình phát hiện mã độc bao gồm:

+ Accuracy: Accuracy được tính bằng tỉ lệ số lượng mẫu dự đoán đúng trên tổng số lượng mẫu. Tuy nhiên, độ chính xác có thể không phản ánh chính xác hiệu quả của mô hình đối với các bài toán bất định hoặc mất cân bằng dữ liệu. Vì vậy, để giải quyết các vấn đề mất cân bằng dữ liệu có thể kết hợp với các độ đo khác.

+ Precision và Recall: Precision và Recall được sử dụng chủ yếu trong trường hợp số lượng tập mẫu thuộc các lớp khác nhau có tỉ lệ không đồng đều. Các độ đo được tính toán theo các công thức sau đây:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{NP} + \text{TN} + \text{FN}} \tag{1.1}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{1.2}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1.3}$$

Trong đó: True positive (TP) cho biết rằng một mẫu mã độc được phát hiện chính xác và gán nhãn là mã độc. True negative (TN) cho biết rằng một mẫu lành tính được phát hiện chính xác và gán nhãn là lành tính. False positive (FP) cho biết rằng một mẫu lành tính bị phát hiện sai và gán nhãn là mã độc. False negative (FN) cho biết một mẫu mã độc không được phát hiện và được gán nhãn sai là lành tính.

+ F1-score: F1-score là trung bình điều hòa giữa Precision và Recall. F1-score được sử dụng để đánh giá chất lượng của mô hình khi số lượng các mẫu thuộc các lớp khác nhau là không đồng đều. F1-score cho mỗi lớp *i* được tính theo công thức sau:

$$F1_i = \frac{2 \text{ x } Precision_i \text{ x } Recall_i}{Precision_i + Recall_i} \tag{1.4}$$

+ F1-weight: F1-weight là một phép đo tổng hợp của Precision và Recall của mô hình phân loại, tỷ lệ cân nhắc đối với mỗi lớp dữ liệu dựa trên trọng số của lớp đó. Công thức tính F1-weight sử dụng trọng số của từng lớp nhằm đánh giá hiệu suất của mô hình trên dữ liệu không cân bằng. F1-weight được tính theo công thức sau:

$$\text{F1-weight} = \frac{\sum_{i=1}^{n} w_i \text{ x } F1_i}{\sum_{i=1}^{n} w_i} \tag{1.5}$$

trong đó, *n* là số lượng các lớp, w_i là trọng số của lớp *i.* Nếu không có trọng số nào được cung cấp thì mỗi lớp được coi là có trọng số bằng nhau.

+ F1-Macro: F1-macro là một phép đo trung bình của F1-score của từng lớp mà không cân nhắc trọng số của từng lớp. F1-macro không quan tâm đến kích thước của

từng lớp và xem xét mức độ đóng góp của mỗi lớp vào kết quả tổng thể. Nó có thể phù hợp khi các lớp có sự cân bằng hoặc không cân bằng về kích thước và không yêu cầu xác định trọng số đặc biệt cho từng lớp. F1-macro được tính theo công thức sau:

$$\text{F1-macro} = \frac{\sum_{i=1}^{n} F1_i}{n} \quad (1.6)$$

trong đó, *n* là số lượng các lớp trong bài toán phân loại.

- *Đánh giá dựa trên tài nguyên sử dụng*: Đánh giá hiệu quả của mô hình phát hiện mã độc IoT dựa trên tài nguyên sử dụng là một phương pháp đánh giá tính thực tế của mô hình. Tài nguyên sử dụng bao gồm thời gian tính toán và bộ nhớ. Một mô hình học máy tốt là mô hình có khả năng phân loại chính xác các dữ liệu mới một cách nhanh chóng và hiệu quả với tài nguyên tính toán và bộ nhớ hạn chế. Để đánh giá hiệu quả về mặt tài nguyên sử dụng của mô hình phát hiện mã độc IoT dựa trên học máy có thể sử dụng các yếu tố sau:

+ Thời gian huấn luyện: Là thời gian cần thiết để mô hình học máy được đào tạo trên một tập dữ liệu huấn luyện cụ thể. Nếu thời gian đào tạo quá lâu, thì mô hình khó có thể sử dụng được trong môi trường thực tế.

+ Thời gian phát hiện: Là thời gian mà mô hình học máy cần để phân loại một mẫu mới. Nếu thời gian dự đoán quá lâu, thì mô hình không phù hợp để sử dụng trong môi trường thực tế.

+ Kích thước mô hình: Là dung lượng bộ nhớ yêu cầu để lưu trữ mô hình. Nếu kích thước mô hình quá lớn, thì mô hình có thể không sử dụng được trong môi trường có tài nguyên hạn chế.

- *Đánh giá dựa trên độ phức tạp của mô hình học máy*: Đánh giá hiệu quả của một mô hình phát hiện mã độc IoT dựa trên độ phức tạp của mô hình học máy là một trong những phương pháp phổ biến được sử dụng để đánh giá và so sánh khả năng học và khả năng xử lý của các mô hình khác nhau. Tuy nhiên, đánh giá hiệu quả của mô hình phát hiện chỉ dựa trên độ phức tạp mô hình học máy là không đủ để đưa ra một quyết định chính xác về tính hiệu quả của mô hình. Một mô hình học máy có độ phức tạp cao có thể cung cấp kết quả tốt hơn một mô hình đơn giản, tuy nhiên, nó cũng có thể dẫn đến các vấn đề như overfitting [131] khi mô hình quá tập trung vào dữ liệu huấn luyện và không thể áp dụng được cho các tập dữ liệu mới. Ngược lại, một mô hình đơn giản có thể dẫn đến underfitting [131] khi mô hình không đủ phức tạp để học được các mối quan hệ phức tạp trong dữ liệu. Vì vậy, trong luận án tiêu chí đánh giá dựa trên độ phức tạp của mô hình học máy sẽ được tham chiếu thêm với các tiêu chí đánh giá quan trọng khác nêu trên để quyết định lựa chọn thuật toán học máy và mạng học sâu phù hợp.

Trong môi trường IoT, độ chính xác của mô hình cần được cân nhắc với thời gian huấn luyện, thời gian phát hiện và kích thước mô hình để đánh giá khả năng ứng dụng

thực tế của mô hình. Tùy thuộc vào tính chất của bài toán và dữ liệu, có thể sử dụng một hoặc nhiều cách thức nêu trên để đánh giá hiệu quả của một mô hình phát hiện mã độc IoT. Vì vậy, các tiêu chí đánh giá sẽ được luận án lựa chọn để đánh giá cho từng mô hình trong các đề xuất.

❖ *Bước 6: Sử dụng mô hình*

Đây là việc dùng mô hình phát hiện đã huấn luyện được để dự đoán các kết quả cho dữ liệu mới và đưa ra quyết định dựa trên kết quả dự đoán. Sau giai đoạn đánh giá, một mô hình có hiệu quả tốt nhất được tạo ra và sử dụng trong xây dựng các giải pháp phát hiện mã độc IoT trong môi trường thực tế.

Các bước quan trọng trong quá trình xây dựng mô hình phát hiện mã độc dựa trên học máy cần tập trung giải quyết bao gồm: Trích xuất, lựa chọn được các đặc trưng hiệu quả và lựa chọn mô hình huấn luyện phù hợp để giải quyết các vấn đề khác nhau trong phát hiện mã độc IoT [82].

1.3. Đánh giá hiệu quả của một số mô hình phát hiện mã độc bị IoT dựa trên học máy sử dụng đặc trưng của tập tin thực thi

Hiệu quả của mô hình phát hiện mã độc IoT dựa trên học máy thường được đánh giá dựa trên nhiều tiêu chí khác nhau đã được phân tích trong nội dung 1.2.2 của luận án và tuỳ thuộc vào từng bài toán cụ thể. Trong môi trường IoT, yêu cầu phát hiện chính xác, nhanh chóng, hạn chế tài nguyên sử dụng của các mô hình phát hiện mã độc IoT là rất cần thiết khi ứng dụng các kết quả nghiên cứu vào trong hệ thống mạng của các cơ quan, tổ chức, doanh nghiệp. Đặc biệt, với các hệ thống mạng trọng yếu, hệ thống mạng lớn với nhiều loại thiết bị IoT, mức độ yêu cầu đảm bảo an toàn thông tin cao, đòi hỏi cần phát hiện nhanh chóng, chính xác, có khả năng tự động phát hiện các cuộc tấn công có chủ đích, các biến thể mã độc mới, các mã độc hoạt động đa kiến trúc nhằm vào các hệ thống mạng quan trọng này. Với các hệ thống IoT hiện nay, các mô hình phát hiện mã độc ngoài hiệu quả về độ chính xác thì các yêu cầu về phòng chống mất cân bằng dữ liệu huấn luyện, hạn chế thời gian trích xuất đặc trưng, thời gian phát hiện, kích thước mô hình và tăng khả năng phát hiện mã độc có thể hoạt động đa nền tảng cần được quan tâm nghiên cứu.

1.3.1. Đánh giá hiệu quả của mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên học máy sử dụng đặc trưng của tập tin thực thi

Với sự phát triển của các thuật toán học máy, mạng học sâu, việc xây dựng các mô hình phát hiện mã độc IoT đơn kiến trúc đã đạt được nhiều thành tựu đáng kể trong thời gian qua. Các đặc trưng của tập tin thực thi trích xuất từ phân tích động và phân tích tĩnh đã mạng lại nhiều kết quả khả quan trong phát hiện mã độc trên các thiết bị IoT. Các đặc trưng của tập tin được thu thập, lựa chọn phù hợp với từng tập dữ liệu, mô hình học máy phục vụ các bài toán khác nhau trong phát hiện mã độc. Cụ thể như sau:

- Hiệu quả của mô hình sử dụng đặc trưng của tập tin trích xuất từ phân tích động, các mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên học máy sử dụng đặc trưng này có thể phát hiện được các mã độc IoT sử dụng kỹ thuật che giấu và các mã độc có tính chất động. Các đặc trưng động đã được sử dụng và đem lại hiệu quả trong xây dựng các mô hình phát hiện mã độc IoT dựa trên học máy gồm: Memory usage [59], [77], Instruction traces [27], Network traffic [6], [38], [42], [44], API call [16], [76], [78], [99], [126], [135], System call [59], [115],[120], … Một số mô hình phát hiện mã độc IoT hiệu quả dựa trên học máy sử dụng đặc trưng động thể hiện như Bảng 1.6.

Bảng 1.6. Một số mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên học máy sử dụng đặc trưng động của tập tin thực thi.

Tác giả	**Năm công bố**	**Đặc trưng động**	**Phương pháp học máy**	**Tập dữ liệu thử nghiệm**	**Đánh giá độ chính xác của mô hình**
Esraa Saleh Alomari [38]	2023	Network traffic	Sử dụng mạng nơ-ron Long Short Term Memory (LSTM)	Tập dữ liệu 35 đặc trưng của 100.000 bản ghi dữ liệu luồng mạng gồm 50.000 của tập tin mã độc và 50.000 của tập tin lành tính.	Tỷ lệ giảm số lượng đặc trưng nằm trong khoảng từ 18,18% đến 42,42%, với mức giảm hiệu suất tương ứng là 0,07% đến 5,84%.
Lê Hải Việt [64]	2022	Đồ thị lời gọi hệ thống có hướng	Sử dụng các thuật toán học máy SVM, DT, RF, kNN	Tập dữ liệu gồm 5.023 mẫu IoT botnet và 3.888 mẫu lành tính.	Độ chính xác trung bình Accuracy là 96,89% và AUC là 98,9%.
Shanxi Li [99]	2022	Chuỗi lời gọi API	Sử dụng mạng nơ-ron Graph Neural Network	Tập dữ liệu gồm 6.686 mẫu mã độc và 6.938 mẫu lành tính.	Độ chính xác accuracy cao nhất trong phát hiện mã độc IoT là 98,32%.
Ce Li [16]	2022	Chuỗi lời gọi API	Sử dụng mạng nơ-ron Bi-LSTM	Tập dữ liệu thu thập từ thực tế.	Kết quả tốt nhất với các độ đo Accuracy là 97,31% và F1-score là 97,24%.
Vikas Sihag [120]	2021	Lời gọi hệ thống, phân tích Binder, lưu lượng mạng,…	Sử dụng các mạng học sâu	Tập dữ liệu gồm 13.533 ứng dụng thu thập từ các kho phần mềm, tiện ích trên nền tảng Android.	Độ chính xác accuracy trong phát hiện mã độc là 98,08%.
Jeon [59]	2020	Bộ nhớ, hành vi mạng, lời gọi hệ thống, tiến trình, hệ thống tập tin ảo hoá	Sử dụng mạng học sâu Convolution -al Neural Network (CNN)	Tập dữ liệu nhóm tác giả thu thập từ các tập tin thực thi trên hệ thống nhúng gồm 1.000 mẫu mã độc và 401 mẫu lành tính.	Độ chính xác accuracy trong phát hiện mã độc là 99,28%.

G. Bendiab [42]	2020	Lưu lượng mạng	Sử dụng các mạng học sâu	Tập dữ liệu tác giả thu thập qua giám sát các tập tin mã độc và tập tin lành tính gồm 1000 tập tin pcap.	Mô hình phát hiện mã độc cho kết quả tốt nhất đạt độ chính xác accuracy là 94,5%.
W.Jung [122]	2020	Tài nguyên máy chủ sử dụng	Sử dụng mạng CNN kết hợp với đặc trưng tiêu thụ điện năng	Tập dữ liệu gồm 12.000 bản ghi thông tin tiêu thụ điện năng trên các thiết bị IoT.	Mô hình phát hiện mã độc IoT Botnet có độ chính xác tốt nhất đạt accuracy là 96,5% và thời gian huấn luyện là 170 giây.
M. K. Alzaylaee [78]	2020	Lời gọi API, mục đích, phân quyền	Sử dụng mạng nơ-ron sâu Deep neural Network	Tập dữ liệu trên nền tảng thiết bị di động gồm 31.125 mẫu mã độc và lành tính.	Sử dụng 420 đặc trưng động và tĩnh để xây dựng mô hình phát hiện mã độc Android với accuracy là 97,8%.
M.Ficco [76]	2019	Lời gọi API	Sử dụng chuỗi Markov	Tập dữ liệu gồm 24.000 tập tin apk mã độc và 22.000 tập tin apk lành tính.	Mô hình phát hiện mã độc IoT Botnet đạt độ chính xác accuracy tốt nhất là 89%.
Trần Nghi Phú[115]	2019	Lời gọi hệ thống	Sử dụng các thuật toán học máy SVM, RF, NB	Tập dữ liệu trên nền tảng MIPS gồm 3.223 tập tin mã độc và 228 tập tin lành tính.	Mô hình phát hiện mã độc đạt kết quả tốt nhất với độ đo F1-Weight là 97,44%.
A. O. Prokofiev [6]	2018	Lưu lượng mạng	Sử dụng thuật toán Logistic Regression	Tập dữ liệu gồm dữ liệu mạng thu thập từ 100 botnet hướng đến các thiết bị IoT.	Mô hình phát hiện mã độc cho độ chính xác cao nhất đạt 97,3%.

Bên cạnh đó, với kết quả so sánh hiệu quả về thời gian thu thập đặc trưng, tiền xử lý của một số mô hình phát hiện mã độc dựa trên học máy sử dụng đặc trưng động của tập tin thực thi trong Bảng 1.7.

Bảng 1.7. So sánh hiệu quả về mặt thời gian một số mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên học máy sử dụng đặc trưng động của tập tin thực thi.

Tác giả	**Đặc trưng động**	**Thuật toán học máy/ mạng học sâu**	**Tập dữ liệu thử nghiệm**		**Độ chính xác (%)**	**Thời gian thu thập đặc trưng động/mẫu (phút)**
			Số lượng mã độc	**Số lượng lành tính**		
Tobiyama [103]	Thông tin tiến trình hệ thống	Recurrent Neural Network (RNN)	81	69	96	5
Damodaran	Lời gọi hệ	Mô hình	745	40	98	10

[3]	thống, mã thực thi	Markov ẩn				
Hansen [101]	Lời gọi hệ thống	RF	5.000	837	98	3,33
Nguyễn Long Giang [86]	Lời gọi hệ thống, luồng mạng, yêu cầu tài nguyên	SVM, KNN, DT, RF, hàm hợp nhất Voting, Linear Regression (LR)	5.023	3.888	99	3
Pascanu [95]	Lời gọi hệ thống	RNN và Echo State Networks	25.000	25.000	95	Ít nhất 15 bước, không báo cáo cụ thể thời gian

Như vậy, các nghiên cứu [3], [16], [59], [76], [78], [85], [86], [99], [101], [115], [120] đã sử dụng đặc trưng lời gọi hệ thống thu thập từ phân tích động để xây dựng các mô hình phát hiện mã độc IoT đơn kiến trúc. Các mô hình này đã đem lại độ chính xác tốt hơn sử dụng các đặc trưng động khác như lưu lượng mạng và tài nguyên máy chủ. Tuy nhiên, một số nghiên cứu cần thu thập nhiều thông tin hành vi từ phân tích động, quá trình thu thập đặc trưng lời gọi hệ thống cho từng mẫu lớn (trên 3 phút/mẫu). Các mô hình học máy, mạng học sâu đã được kết hợp để nâng cao độ chính xác nhưng các nghiên cứu đối mặt với độ phức tạp tính toán lớn khi sử dụng các mạng học sâu phức tạp và cần huấn luyện nhiều mô hình phân lớp khác nhau để phát hiện mã độc. Vì vậy, khi triển khai mô hình phát hiện đã được huấn luyện trong các giải pháp bảo mật trên môi trường IoT tài nguyên hạn chế hoặc cần đảm bảo yêu cầu phát hiện, cảnh báo sớm mã độc cho nhiều tập tin thực thi trong thời gian ngắn sẽ gặp nhiều khó khăn. Mặt khác, một số tập dữ liệu được các nghiên cứu thử nghiệm có sự chênh lệch và hạn chế về số mẫu tập tin lành tính thu thập được. Hạn chế trong việc thu thập các mẫu tập tin lành tính có khả năng thực thi trong môi trường phân tích động cũng đã được chỉ ra trong các luận án [64][114].

- *Hiệu quả đối với đặc trưng của tập tin trích xuất từ phân tích tĩnh,* các đặc trưng đã được sử dụng hiệu quả trong xây dựng các mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên học máy gồm: Strings [128], Bytes n-gram [133], Opcode [32], [37], [46], [47], [115], Function call graph [19], [69], Control flow-based opcode [117], [125], entropy-based [28], Grayscale image [57], [74], ... Một số mô hình phát hiện mã độc IoT hiệu quả dựa trên học máy sử dụng đặc trưng tĩnh của tập tin thực thi thể hiện như Bảng 1.8. Phương pháp phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng tĩnh cho độ chính xác cao và độ tin cậy tốt (các nghiên cứu đã khảo sát cho kết quả chính xác trong phát hiện mã độc đạt khoảng 93% đến 99%) khi phát hiện các mã độc IoT có đặc trưng tĩnh tương tự nhau. Tuy nhiên, các nghiên cứu sử dụng phương pháp này đã được chỉ ra các hạn chế trong phát hiện mã độc có sử dụng các kỹ thuật che giấu hoặc

các mã độc IoT có tính chất động.

Bảng 1.8. Một số mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên học máy sử dụng đặc trưng tĩnh của tập tin thực thi.

Tác giả	Năm công bố	Đặc trưng tĩnh	Phương pháp/Mô hình phát hiện	Tập dữ liệu thử nghiệm	Đánh giá độ chính xác của mô hình
Chia-Yi Wu [19]	2023	Function call grahp	Sử dụng phương pháp state-of-the-art graph embedding kết hợp với SVM	Tập dữ liệu IoT gồm 108.000 tập tin mã độc	Tỷ lệ phát hiện chính xác accuracy là 98,88%
Muham-mad Asam [74]	2022	Greyscale images	Sử dụng mạng CNN	Tập dữ liệu IoT nền tảng hệ điều hành Linux gồm 14.733 ảnh của tập tin mã độc và 2.486 ảnh của tập tin lành tính	Các độ đo Accuracy là 97,93%, F1-Score là 93,94%, Precision là 98,64%
Safa Ben Atitallah [9]	2022	Images của PE file	Sử dụng 3 mô hình mạng học sâu CNN và sử dụng phương pháp Random Forest Voting	Tập dữ liệu gồm 14.226 RGB converted images của 26 họ mã độc	Độ chính xác của mô hình phát hiện mã độc nằm trong khoảng 98% đến 99%
Nguyễn Huy Trung [85]	2021	Đồ thị PSI	Sử dụng mạng CNN	Tập dữ liệu gồm 6.165 mã độc IoT botnet và 3.845 mẫu lành tính	Độ chính xác của mô hình phát hiện đạt accuracy là 97,8%
Zahra Moti [134]	2021	Raw byte code	Sử dụng mạng nơ-ron CNN và mạng LSTM cùng với kỹ thuật GAN để tạo các mẫu mã độc mới	Tập dữ liệu IoT thử nghiệm gồm 271 mẫu mã độc và 281 mẫu lành tính	Độ chính xác 97,56% và thời gian huấn luyện là 2,5 phút, thời gian phát hiện là 0,004 giây
Hamid Darabia-n [46]	2020	Opcode	Sử dụng các thuật toán học máy KNN, SVM, Multilayer Perceptron (MLP), RF, DT và AdaBoost	Tập dữ liệu thử nghiệm gồm 247 tập tin mã độc và 269 tập tin lành tính trên nền tảng ARM	Độ đo F-measure là 99% khi sử dụng 36 đặc trưng opcode
Hisham Alasmar-y [45]	2019	CFG	Sử dụng phương pháp tính toán 23 thuộc tính lý thuyết đồ thị của CFG	Tập dữ liệu thử nghiệm gồm 5.853 mã độc trên hệ điều hành Android	Độ chính xác cao nhất accuracy đạt 99,66%
Ensieh Modiri Dovom [37]	2019	Opcode	Chuyển đổi opcode vào không gian vectơ và áp dụng kỹ thuật fuzzy pattern tree	Các tập dữ liệu gồm 22.000 mẫu trên hệ điều hành Windows và tập dữ liệu trên kiến trúc ARM gồm 128 mẫu mã độc và 1.078 mẫu lành tính	Độ chính xác cao nhất của mô hình trong phát hiện mã độc IoT botnet đạt 99,83%

Trần Nghi Phú [117]	2019	CFG	Sử dụng thuật toán học máy SVM kết hợp với phương pháp trích chọn đặc trưng n-gram	Tập dữ liệu trên kiến trúc MIPS với 7.000 mẫu	Độ chính xác cao nhất accuracy đạt 99,34%
A Dehgha-ntanha [4]	2019	Opcode graph	Sử dụng mạng học sâu và đồ thị opcode	Tập dữ liệu trên kiến trúc ARM gồm 128 mã độc và 1.078 mẫu lành tính	Độ chính xác cao nhất accuracy đạt 98,37%
Jiawei Su [57]	2018	Grayscale image	Sử dụng mạng nơ-ron và biểu diễn các mẫu nhị phân như ảnh đa mức xám	Tập dữ liệu thu thập từ IOTPOT gồm 500 mã độc IoT DDoS	Độ chính xác cao nhất accuracy đạt 94%
Hamed Haddad Pajouh [47]	2018	Opcode	Sử dụng mạng nơ-ron và chuỗi opcode	Tập dữ liệu IoT trên kiến trúc ARM gồm 281 mã độc và 279 mẫu lành tính	Độ chính xác của mô hình phát hiện mã độc là 98,18%

Kết quả trong Bảng 1.8 cũng cho thấy rằng đặc trưng mã thực thi được trích xuất dựa trên phương pháp phân tích tĩnh tập tin thực thi trong các nghiên cứu [4], [37], [46], [47] đã tạo ra các mô hình phát hiện mã độc IoT hiệu quả cao về độ chính xác (trên 98%), tuy nhiên các phương pháp đã đề xuất có độ phức tạp cao trong trích xuất đặc trưng dạng đồ thị và sử dụng mạng học sâu kiến trúc phức tạp, chưa có các đánh giá về mặt tài nguyên sử dụng trong huấn luyện mô hình học máy và mô hình phát hiện sau khi huấn luyện. Một số nghiên cứu thử nghiệm với tập dữ liệu IoT chưa đủ lớn và chỉ tập trung vào nền tảng kiến trúc vi xử lý của thiết bị di động như ARM.

Mặt khác, để đánh giá hiệu suất của một số mô hình phát hiện mã độc IoT dựa trên đặc trưng tĩnh, nhóm tác giả Ngô Quốc Dũng [92] đã thực nghiệm và so sánh hiệu quả dựa trên một tập dữ liệu tác giả thu thập với 6.165 mẫu mã độc và 3.845 mẫu lành tính. Kết quả độ chính xác và thời gian trích xuất đặc trưng, tiền xử lý và thời gian phân lớp được chi tiết trong Bảng 1.9.

Bảng 1.9. So sánh hiệu suất một số mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên học máy sử dụng đặc trưng tĩnh của tập tin thực thi.

Đặc trưng tĩnh	**Thuật toán học máy/mạng học sâu**	**Độ chính xác (%)**	**Thời gian trích xuất đặc trưng và tiền xử lý (phút)**	**Thời gian phân lớp (giây)**
ELF-header	RIPPER	99,8	110	0,75
	PART	99,8		1,27
	DT	99,6		1
String-based	SVM	98	5	12,4
	kNN	99,8		1
	DT	99,4		8,75
	RF	99,7		9,71
Image-based	Neural Network	89,1	14	139

CFG-based	SVM	89	7200	1,45
	LR	85		0,5
	RF	95		1,75
PSI -graph	Neural Network	98,7	88	107

Kết quả các thực nghiệm của nhóm tác giả đã chứng minh được hiệu quả về độ chính xác cao của một số mô hình phát hiện mã độc IoT dựa trên học máy, tuy nhiên độ phức tạp chi phí thời gian trong trích xuất và thời gian phân lớp còn lớn. Thời gian phân lớp nhỏ nhất là 0,5 giây khi áp dụng với mô hình sử dụng thuật toán LR nhưng độ chính xác còn hạn chế chỉ đạt 85% và thời gian trích xuất, tiền xử lý vượt quá yêu cầu đáp ứng trong môi trường thực tế.

Với sự phát triển của mã độc IoT, dựa trên ưu và nhược điểm của phương pháp trích xuất đặc trưng của tập tin thực thi, việc xây dựng các mô hình phát hiện mã độc IoT dựa trên đặc trưng động hoặc đặc trưng tĩnh sẽ góp phần nâng cao khả năng phát hiện các mã độc mới, mã độc có nhiều hành vi độc hại phức tạp hiện nay. Bên cạnh đó, một số mô hình phát hiện mã độc IoT được xây dựng dựa trên sự kết hợp của cả hai loại đặc trưng động và đặc trưng tĩnh. Việc kết hợp đặc trưng động và đặc trưng tĩnh sẽ cung cấp thông tin phong phú và đầy đủ hơn về hành vi của mã độc, giúp cải thiện độ chính xác của phương pháp phát hiện và giảm số lượng các kết quả dương tính giả. Tuy nhiên, việc sử dụng cả hai loại đặc trưng đòi hỏi nhiều kỹ thuật kết hợp phức tạp và nhiều tài nguyên tính toán hơn so với việc sử dụng một loại đặc trưng duy nhất [129]. Các thuật toán học sâu như mạng nơ-ron có thể giúp giải quyết các bài toán phức tạp hơn so với các mô hình học máy không sâu truyền thống. Đối với bài toán phát hiện mã độc IoT, một số mạng học sâu có thể được sử dụng để phân loại các mẫu độc hại với độ chính xác cao. Tuy nhiên, các mô hình này có phức tạp tính toán lớn, cần số lượng dữ liệu huấn luyện lớn, thời gian huấn luyện và phát hiện nhiều, khó triển khai trong thực tế cho các ứng dụng phát hiện mã độc IoT theo thời gian thực và các thiết bị IoT hạn chế tài nguyên. Mặt khác, các mô hình phát hiện mã độc IoT đơn kiến trúc mặc dù đã đạt độ chính xác cao dựa trên đặc trưng tĩnh và động trích xuất từ tập tin thực thi nhưng nhiều nghiên cứu đã chỉ ra hạn chế của mô hình phát hiện chỉ thử nghiệm với tập dữ liệu có số lượng nhỏ tập tin thu thập được, số lượng các mẫu có sự chênh lệch lớn giữa các tập đã được gán nhãn. Các tập dữ liệu thử nghiệm được thu thập riêng bởi các nhà nghiên cứu và chưa có đánh giá toàn diện về tính tin cậy và hiệu quả của các tập dữ liệu này, nhiều tập dữ liệu, cách trích chọn đặc trưng là không công khai [29], vấn đề mất cân bằng dữ liệu có thể ảnh hưởng lớn đến kết quả dự đoán của mô hình. Mô hình dự đoán không chính xác trên nhóm dữ liệu có ít mẫu vì đa phần kết quả dự đoán thường sẽ thiên về một nhóm dữ liệu có nhiều mẫu. Với một bài toán phân lớp mã độc cho dữ liệu là 10000 mẫu có nhãn là A và 500 mẫu có nhãn là B, nếu dữ liệu được chia thành tập dữ liệu huấn luyện và kiểm tra theo tỉ lệ 80/20 thì tập đánh giá trên tập dữ liệu đó gồm 2000

mẫu nhãn A và 100 mẫu nhãn B. Giả sử mô hình dự đoán tất cả nhãn đều là A, thì giá trị độ chính xác Accuracy là 2000/2100 (khoảng trên 95%). Vì vậy, mô hình phát hiện dù không phân lớp chính xác tất cả các tập tin nhãn B nhưng lại cho kết quả giá trị accuracy rất cao.

Vì vậy, trong xây dựng mô hình phát hiện mã độc IoT đơn kiến trúc hiệu quả dựa trên đặc trưng của tập tin thu thập từ phân tích động, phân tích tĩnh bên cạnh việc đảm bảo yêu cầu về độ chính xác thì cần xem xét các yêu cầu về thời gian trích xuất đặc trưng, tiền xử lý, kích thước và thời gian phát hiện của mô hình. Bên cạnh đó, để giải quyết các bài toán phát hiện mã độc có khả năng hoạt động trên nhiều loại thiết bị khác nhau về hệ điều hành hoặc kiến trúc vi xử lý, các mô hình phát hiện mã độc IoT đa nền tảng cần được nghiên cứu, xây dựng.

1.3.2. Đánh giá hiệu quả của mô hình phát hiện mã độc IoT đa kiến trúc dựa trên học máy sử dụng đặc trưng của tập tin thực thi

Trong hệ thống IoT có sự đa dạng các loại thiết bị IoT và các mã độc có thể được thiết kế dành riêng cho một loại nền tảng IoT hoặc nhiều loại nền tảng IoT khác nhau. Việc phát hiện mã độc IoT đa kiến trúc có thể được thực hiện thông qua xây dựng nhiều mô hình phát hiện mã độc IoT đơn kiến trúc. Tuy nhiên, việc xây dựng nhiều mô hình sẽ đòi hỏi nhiều tập dữ liệu trên các kiến trúc để xây dựng mô hình, tăng thời gian phát hiện, tài nguyên sử dụng của giải pháp khi triển khai. Vì vậy, việc xây dựng một mô hình phát hiện mã độc IoT đa nền tảng kiến trúc dựa trên học máy có thể giải quyết các hạn chế trên, tăng tính toàn diện, linh hoạt, khả năng phát hiện đa dạng các loại mã độc IoT và các biến thể. Trong xây dựng mô hình phát hiện mã độc IoT đa kiến trúc dựa trên học máy sử dụng đặc trưng của tập tin thực thi, một số phương pháp đã được các nghiên cứu sử dụng bao gồm:

- *Dựa trên dữ liệu đa nguồn*: Các mô hình được xây dựng dựa trên dữ liệu kết hợp từ nhiều nguồn thu thập, trích xuất khác nhau như thông tin hệ thống, thông tin hành vi, thông tin thiết bị,… như các nghiên cứu [36], [16],...

- *Dựa trên kỹ thuật trích chọn đặc trưng đa kiến trúc*: Các đặc trưng được trích chọn phù hợp với nhiều loại thiết bị IoT khác nhau để phục vụ huấn luyện mô hình phát hiện mã độc IoT như các nghiên cứu [21], [112], [125], [130],...

- *Dựa trên kết hợp nhiều mô hình đơn kiến trúc*: Các mô hình phát hiện được xây dựng dựa trên kết hợp nhiều mô hình đơn kiến trúc để có khả năng phát hiện mã độc trên nhiều kiến trúc khác nhau của thiết bị IoT. Việc kết hợp nhiều mô hình phát hiện đơn kiến trúc cần đảm bảo các yêu cầu về tốc độ và tài nguyên sử dụng để đảm bảo khả năng phát hiện mã độc trên nhiều loại thiết bị IoT đa dạng kiến trúc vi xử lý.

- *Dựa trên mô hình phát hiện IoT chéo kiến trúc*: Các mô hình học máy có khả năng huấn luyện trên một tập dữ liệu kiến trúc nguồn để phát hiện mã độc IoT trên kiến

trúc đích như các nghiên cứu [31], [116],... Từ đó, phương pháp có thể tạo ra các mô hình có khả năng phát hiện các mã độc trên nhiều nền tảng kiến trúc khác nhau hoặc mô hình phát hiện các biến thể của một loại mã độc trên nhiều nền tảng kiến trúc thiết bị IoT đa dạng.

Việc sử dụng các phương pháp trên trong phát hiện mã độc IoT đa kiến trúc dựa trên học máy đã chứng minh hiệu quả bước đầu. Tuy nhiên, các phương pháp, mô hình luận án đã khảo sát trong Bảng 1.10 chưa cho hiệu quả cao về độ chính xác phù hợp với tài nguyên tính toán và chưa đề cập đến khả năng phát hiện, dự báo mã độc trên kiến trúc vi xử lý mới.

Bảng 1.10. Một số mô hình phát hiện mã độc IoT đa kiến trúc dựa trên học máy.

Tác giả	**Năm công bố**	**Phương pháp**	**Kết quả đạt được**	**Hạn chế chính**
Ding và các cộng sự [125]	2014	Trích xuất đặc trưng dòng điều khiển mã thực thi để xây dựng mô hình phát hiện	- Mô hình có khả năng phát hiện mã độc IoT đơn kiến trúc và mã độc IoT đa kiến trúc. - Các kết quả thực nghiệm cho thấy mô hình phát hiện đạt được độ chính xác cao hơn các phương pháp dựa trên văn bản.	- Nghiên cứu đối mặt với bài toán có độ phức tạp NP-hard trong duyệt đồ thị. - Phương pháp chỉ có thể áp dụng với các tập tin thực thi đơn giản, không phù hợp với các tập tin phức tạp, có kích thước lớn.
Mohannad Alhanahnah và các cộng sự [70]	2018	Sinh dấu hiệu (signature) dựa trên mô hình phân cụm mã độc nhiều giai đoạn	- Mô hình có khả năng phát hiện mã độc và các biến thể mã độc mới trên nhiều kiến trúc vi xử lý. - Các thử nghiệm được tiến hành với 2 tập dữ liệu thu thập từ hệ thống thực tế với tỉ lệ phát hiện cao nhất accuracy là 95,5%.	- Tập dữ liệu nhóm tác giả tự thu thập từ thực tế và chỉ gồm 5.150 mẫu mã độc IoT. - Khả năng phát hiện mã độc IoT mới trong thực nghiệm chỉ mới đạt là 85,2%.
Cozzi [36]	2018	Mô hình phát hiện mã độc trên Linux dựa trên phân tích dữ liệu metadata, các đặc trưng động và đặc trưng tĩnh	Mô hình đã chứng minh khả năng phát hiện mã độc với tập dữ liệu tập tin mã độc trên Linux hoạt động trên 10 kiến trúc vi xử lý khác nhau với tập tin có kích thước từ 134 bytes đến 14,8 MB qua các thực nghiệm.	- Tập dữ liệu thử nghiệm có sự chênh lệch lớn giữa các nền tảng kiến trúc, một số kiến trúc có số lượng mẫu nhỏ (dưới 1% so với tổng số mẫu). - Chưa có các đánh giá chi tiết về độ chính xác trong phát hiện mã độc IoT hoạt động trên các nền tảng kiến trúc trong tập dữ liệu thử nghiệm.

Phú và các cộng sự [116]	2019	Xây dựng mô hình phát hiện mã độc đa kiến trúc dựa trên phát hiện chéo nền tảng kiến trúc	Các mô hình đề xuất đã có khả năng phát hiện mã độc hoạt động đa kiến trúc vi xử lý thông qua các thực nghiệm đánh giá trên tập dữ liệu các nền tảng kiến trúc vi xử lý khác nhau.	- Độ chính xác trong phát hiện chưa cao với khả năng phát hiện mã độc hoạt động chéo nền tảng có trường hợp chỉ đạt accuracy là 58,2%. - Việc phát hiện mã độc đa kiến trúc còn phụ thuộc vào độ chính xác và hiệu quả của việc xây dựng ngôn ngữ trung gian Vex.
Lee và cộng sự [130]	2020	Mô hình phát hiện mã độc đa nền tảng dựa trên đặc trưng PSI	Mô hình phát hiện với tập dữ liệu 122.504 tập tin mã độc ELF thu thập từ VirusTotal trên các kiến trúc x86, x86-64, MIPS, ARM, SPARC, PowerPC và cho kết quả thử nghiệm phát hiện mã độc đạt độ chính xác accuracy trung bình là 97,2% .	- Hạn chế về số chiều không gian lớn do số lượng lớn các PSI độc lập được tìm thấy với 14 triệu đặc trưng PSI. - Tập dữ liệu thử nghiệm có sự chênh lệch lớn giữa các nhãn, đặc biệt một số kiến trúc chưa thu thập được mẫu mã độc.
Vasan và các cộng sự [31]	2020	Mô hình phát hiện mã độc IoT đa kiến trúc (MTHAEL) sử dụng mạng học sâu RNN và CNN	- Thử nghiệm phân loại mã độc trên kiến trúc ARM gồm 3.427 mã độc và 2.035 tập tin lành tính đạt độ chính xác cao nhất accuracy là 99,98%, thời gian phát hiện của mô hình là 0,32 giây. - Kết quả huấn luyện và phát hiện chéo kiến trúc vi xử lý đạt độ chính xác accuracy là 95,72%.	Độ chính xác accuracy chỉ là 58,2% khi huấn luyện trên kiến trúc vi xử lý MIPS để phát hiện trên kiến trúc vi xử lý Intel 80386.
Chin-wei Tien và các cộng sự [16]	2020	Phân tích mã độc IoT dựa trên việc lựa chọn các đặc trưng trong tập tin ELF và đặc trưng mã thực thi	Framework đã chứng minh qua thực nghiệm khả năng phát hiện các mã độc trên nhiều kiến trúc vi xử lý.	Việc lựa chọn các đặc trưng trong nghiên cứu chưa hiệu quả đối với các mã độc IoT sử dụng các kỹ thuật làm rối, mã độc dạng “droppers” do không chưa các mã khai thác trong chương trình, vì vậy nó ảnh hưởng lớn đến độ chính xác của mô hình phát hiện.
Jingmei Li [58]	2020	Sử dụng phương pháp học tăng cường để phân	Độ chính xác trong các thử nghiệm đạt giá trị accuracy cao nhất trên	- Mô hình được thử nghiệm trên tập dữ liệu nhỏ thu thập từ

		lớp mã độc đa kiến trúc	98%.	VirusShare và VX Heavens gồm 600 mẫu mã độc, chưa đảm bảo tính khách quan khi đánh giá mô hình. - Tác giả chỉ ra rằng mô hình chưa hiệu suất thấp khi số lượng mẫu của một lớp nhỏ hơn 1000.
Wan và cộng sự [112]	2020	Mô hình phát hiện mã độc dựa trên chuỗi byte thu thập từ tập tin thực thi	Sử dụng một tập dữ liệu hơn 111 nghìn mẫu cho mỗi tập dữ liệu tập tin lành tính và mã độc để đánh giá và chứng minh khả năng phát hiện mã độc đa kiến trúc.	Phương pháp hoạt động chưa hiệu quả khi có sự hiện diện của việc tái sử dụng mã chương trình giữa các họ mã độc.
Chaganti và các cộng sự [21]	2022	Mô hình phát hiện dựa trên mạng Bi-GRU-CNN sử dụng đặc trưng chuỗi byte trích xuất từ tập tin ELF	- Mô hình có khả năng phát hiện và phân lớp mã độc độc lập với kiến trúc vi xử lý. - Mô hình được thử nghiệm với kết quả độ chính xác đạt 100% đối với trường hợp phát hiện phần mềm độc hại IoT và 98% đối với phân loại các dòng mã độc IoT.	Nghiên cứu mới chỉ đưa ra rằng mô hình đề xuất có khả năng phát hiện và phân loại mã độc đa kiến trúc dựa vào sự độc lập của đặc trưng chuỗi byte và các loại kiến trúc vi xử lý, chưa có các thực nghiệm tin cậy để đánh giá hiệu suất phát hiện mã độc IoT đa kiến trúc.
Zhao và các cộng sự [136]	2023	Phát hiện mã độc cho các thiết bị IoT đa dạng kiến trúc vi xử lý dựa trên đám mây PaaS sử dụng mạng học sâu để xử lý dữ liệu và trích xuất đặc trưng của gói tin mạng	Các kết quả thực nghiệm cho độ chính xác trung bình accuracy của mô hình phát hiện là 97,18% và recall là 99,01% khi phát hiện các mẫu ELF.	- Phương pháp cần có sự kết nối mạng ổn định để triển khai, tốc độ phát hiện có thể bị ảnh hưởng bởi tốc độ xử lý của đám mây, không phát hiện được các loại phần mềm độc hại mới. - Tập dữ liệu thử nghiệm nhỏ chỉ với 1.719 mẫu trên kiến trúc ARM và x86-32.

Bên cạnh đó, các nghiên cứu phát hiện mã độc IoT đa kiến trúc vi xử lý đã thu thập và sử dụng tập dữ liệu thử nghiệm mô tả trong Bảng 1.11.

Bảng 1. 11. Một số tập mẫu phục vụ xây dựng mô hình phát hiện mã độc IoT đa kiến trúc.

Tác giả	**Nguồn thu thập mẫu**		**Số lượng mẫu**		**Kiến trúc vi xử lý**
	Lành tính	**Mã độc**	**Lành tính**	**Mã độc**	

Alhanahnah [70]	openwrt	IoTPOT	130	5.150	ARM, MIPS, Intel, PowerPC, x86-64, Renesas SH, Motorola, SPARC
Cozzi [36]	-	Virus Total và tự thu thập	-	10.548	ARM, MIPS, Intel, PowerPC, x86-64. Motorola, SPARC, Hitachi SH
Lee [130]	-	Virus Total	-	122.504	ARM, MIPS, x86, x86-64, PowerPC, SPARC
Nguyễn Huy Trung [85]	SOHO IoT devices	IoTPOT	6.031	4.002	ARM, MIPS, PowerPC, SPARC
Lê Hải Việt [64]	IoT device firmwares	IoTPOT, Virus Total, VirusShare	3.388	5.023	MIPS, ARM, x86, PowerPC
Trần Nghi Phú [114]	Ubuntu OS-x86-64, Intel, Frimware of router images	IoTPOT, Detux, VirusShare	4.107	19.710	Intel, MIPS, PowerPC, SPARC, Motorola, Rensenas SH
Tien [17]	Ubuntu 16.04 LTS	CZ.NIC, IoTPOT	2.157	6.251	x86, x64, ARM
Vasan [31]	Frimware router images	IoTPOT, Detux, VirusShare	5.655	15.482	ARM, MIP, Intel, x86-64, PowerPC
Wan [112]	D-link, Zyxel, Netgear, IDIS, Belkin, MikroTik	VirusTotal	111.353	111.610	ARM, MIPS, x86, x86-64, PowerPC, SPARC, Renesas SH

Các tập dữ liệu mã độc thử nghiệm chủ yếu được các nhóm tác giả thu thập từ các nguồn Internet tương tự nhau như: IoTPOT, VirusShare, VirusTotal ,… Số mẫu huấn luyện có các nhãn và trên các kiến trúc vi xử lý khác nhau có sự chênh lệch lớn. Với các đặc điểm của thiết bị IoT tài nguyên hạn chế, việc xây dựng các ứng dụng sẵn có được nhà sản xuất hạn chế đến mức tối đa để đảm bảo tài nguyên sử dụng trên thiết bị nên số lượng chương trình ứng dụng trên thiết bị này không nhiều và đa dạng như máy tính trước đây. Vì vậy, việc thu thập mẫu lành tính và trên các thiết bị IoT này gặp nhiều khó khăn và không đa dạng mẫu. Điều này có thể dẫn đến vấn đề mất cân bằng dữ liệu trong huấn luyện mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng của tập tin thực thi. Mặt khác, với sự phát triển đa dạng, nhanh chóng của các loại thiết bị IoT hiện nay về hệ điều hành và nền tảng kiến trúc vi xử lý sử dụng, việc thu thập các mẫu tập tin trên các dòng thiết bị IoT mới đặt ra nhiều yêu cầu về phương pháp, môi trường, công cụ thu thập, trích xuất các tập tin từ hệ thống IoT. Việc tận dụng tri thức từ các tập dữ liệu đã có trước đây hoặc tạo ra dữ liệu mới từ dữ liệu ban đầu sẽ có thể mang lại nhiều hiệu quả trong phát hiện mã độc IoT đa nền tảng kiến trúc trong tương lai.

1.4. Bài toán nâng cao hiệu quả cho mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng của tập tin thực thi

1.4.1. Bài toán nghiên cứu

Với các đánh giá hiệu quả của một số mô hình phát hiện mã độc IoT dựa trên học máy đã trình bày ở trên, mỗi mô hình phát hiện mã độc IoT dựa trên học máy đã khảo sát đều có những ưu và nhược điểm khác nhau. Hạn chế chính của các nghiên cứu xây dựng mô hình học máy trong phát hiện mã độc IoT dựa trên đặc trưng tập tin thực thi bao gồm:

Thứ nhất, các mô hình phát hiện mã độc IoT đơn kiến trúc dựa học máy sử dụng đặc trưng động hoặc đặc trưng tĩnh của tập tin thực thi có độ chính xác cao đang cần sử dụng số lượng đặc trưng nhiều, cách thức trích xuất đặc trưng còn phức tạp, cần nhiều thời gian giám sát và thu thập, mô hình phát hiện cần nhiều thời gian huấn luyện và phát hiện, kích thước mô hình phát hiện chưa được đánh giá hoặc đã đánh giá nhưng chưa phù hợp với thiết bị IoT tài nguyên hạn chế.

Thứ hai, các mô hình phát hiện mã độc IoT đa kiến trúc dựa trên học máy sử dụng đặc trưng của tập tin phần lớn tập trung xây dựng dựa trên các đặc trưng độc lập nền tảng và không phụ thuộc vào kiến trúc vi xử lý của các loại thiết bị IoT. Các tập dữ liệu mã độc IoT đa kiến trúc đã công bố còn hạn chế về số lượng mẫu và loại kiến trúc vi xử lý cho phép mã độc hoạt động. Một số mô hình phát hiện mã độc IoT đa kiến trúc dựa trên huấn luyện và phát hiện chéo kiến trúc vi xử lý đã được đề xuất nhưng hiệu quả chưa đồng đều giữa các kiến trúc vi xử lý. Các mô hình có khả năng dự báo mã độc zero-day đa kiến trúc và biến thể mã độc trên nền tảng kiến trúc mới chưa được tập trung giải quyết.

Thứ ba, vấn đề không cân bằng dữ liệu huấn luyện giữa các nhãn, giữa các nền tảng kiến trúc cũng ảnh hưởng đến các thử nghiệm và đánh giá hiệu quả của một số mô hình phát hiện mã độc IoT đơn kiến trúc và đa kiến trúc đã đề xuất. Đặc biệt là hạn chế về số mẫu tập tin lành tính có khả năng thực thi đầy đủ hành vi hệ thống trong môi trường Sandbox và số lượng các mẫu mã độc trên kiến trúc vi xử lý như SPARC, PowerPC.

Vì vậy, để giải quyết các hạn chế nêu trên trong xây dựng mô hình phát hiện mã độc IoT hiệu quả dựa trên học máy và đặc trưng của tập tin thực thi, luận án nghiên cứu bài toán như sau:

Nghiên cứu xây dựng các mô hình phát hiện mã độc trên thiết bị IoT đa dạng kiến trúc vi xử lý dựa trên học máy sử dụng ít thời gian và số lượng đặc trưng biểu diễn dạng chuỗi của tập tin thực thi cần thu thập từ phân tích động và phân tích tĩnh nhằm nâng cao khả năng tích hợp trong hệ thống IoT tài nguyên hạn chế, đảm bảo khả năng áp dụng cho các tập tin thực thi trong môi trường IoT với độ chính xác tốt và có thể dự báo mã độc IoT zero-day chéo kiến trúc vi xử lý.

1.4.2. Giải quyết bài toán

Để giải quyết bài toán, luận án xây dựng các mô hình phát hiện mã độc IoT đơn kiến trúc và đa kiến trúc hiệu quả dựa trên học máy sử dụng đặc trưng biểu diễn dạng chuỗi của tập tin thực thi thu thập thông qua các phương pháp phân tích động, phân tích tĩnh. Trong đó, tập trung nâng cao hiệu quả của các mô hình phát hiện dựa trên nghiên cứu các phương pháp thu thập, lựa chọn đặc trưng động và đặc trưng tĩnh, xử lý dữ liệu đa nền tảng kiến trúc vi xử lý và lựa chọn thuật toán học máy phù hợp với môi trường IoT.

❖ ***Thu thập và lựa chọn đặc trưng hiệu quả trong phát hiện mã độc IoT***

Hiện nay, trong các nghiên cứu xây dựng mô hình phát hiện mã độc IoT luận án đã khảo sát, các tập dữ liệu thử nghiệm được thu thập riêng bởi các nhà nghiên cứu và chưa có đánh giá toàn diện về tính tin cậy và hiệu quả của các tập dữ liệu này, nhiều tập dữ liệu, cách trích chọn đặc trưng là không công khai [29]. Các đặc trưng động và đặc trưng tĩnh được thu thập thông qua các công cụ, hệ thống hỗ trợ phân tích động và phân tích tĩnh. Với sự phát triển của mã độc trên các thiết bị IoT, cần lựa chọn phương pháp trích xuất, thu thập đặc trưng phù hợp phục vụ xây dựng các mô hình phát hiện mã độc IoT hiệu quả trong thực tế. Việc xây dựng các giải pháp sử dụng cả mô hình phát hiện mã độc IoT dựa trên đặc trưng động và mô hình phát hiện dựa trên đặc trưng tĩnh sẽ góp phần nâng cao khả năng phát hiện các mã độc mới, mã độc sử dụng các kỹ thuật lẩn tránh và mã độc có nhiều hành vi độc hại phức tạp hiện nay. Bên cạnh đó, các mô hình phát hiện sử dụng kết hợp cả đặc trưng động và đặc trưng tĩnh sẽ cung cấp thông tin phong phú và đầy đủ hơn về hành vi của mã độc, giúp cải thiện độ chính xác. Tuy nhiên, việc sử dụng cả hai loại đặc trưng để xây dựng một mô hình đòi hỏi nhiều kỹ thuật kết hợp phức tạp và nhiều tài nguyên tính toán hơn so với việc sử dụng một loại đặc trưng duy nhất [129].

Để đạt được mục tiêu nghiên cứu, luận án tiếp cận xây dựng các mô hình phát hiện mã độc IoT sử dụng đặc trưng biểu diễn dạng chuỗi thu thập dựa trên phương pháp phân tích động hoặc phân tích tĩnh tập tin thực thi. Các đặc trưng biểu diễn dạng chuỗi tồn tại 2 xu hướng tiếp cận chính gồm:

- *Coi các đặc trưng trong chuỗi là những dữ liệu có thuộc tính rời rạc, không liên quan đến nhau*: Các dữ liệu này được sinh ra từ những phân phối độc lập và không giống nhau. Các nhà nghiên cứu đã áp dụng các phương pháp xử lý dữ liệu có đặc trưng rời rạc để trích xuất đặc trưng phục vụ quá trình phân lớp. Việc phân lớp một dữ liệu mới phụ thuộc vào cách kiểm tra các giá trị thuộc tính rời rạc của mẫu này có thuộc tập giá trị của thuộc tính rời rạc lớp tương ứng hay không. Do tập hợp có tính hoán vị nên tính chất thứ tự của các giá trị thuộc tính sẽ bị loại bỏ trong quá trình xử lý dữ liệu. Sự loại bỏ này có thể dẫn đến giảm độ chính xác của mô hình phát hiện mặc dù có thể sử dụng ít tài nguyên tính toán.

- Coi các đặc trưng trong chuỗi là những dữ liệu có thuộc tính tuần tự, có thứ tự trước sau và mối liên hệ lẫn nhau: Việc này giống như các "từ" trong một đoạn văn được viết theo một trật tự nhất định. Nếu các từ trong đoạn văn đó bị hoán vị ngẫu nhiên mà không ảnh hưởng đến tần suất xuất hiện trong đoạn văn thì việc nắm bắt ý nghĩa của nó sẽ rất khó khăn. Cần có cách thức xử lý dữ liệu để không làm mất đi đặc trưng tuần tự. Cách xử lý phổ biến là dùng kỹ thuật word embedding để biến đổi các "từ" thành vectơ số đặc trưng cho ý nghĩa của "từ" trong ngữ cảnh. Kỹ thuật đảm bảo các "từ" tương tự nhau sẽ có giá trị vectơ gần giống nhau. Đối với cách tiếp cận này, kỹ thuật word2vec dựa trên một mạng nơ-ron 2 lớp đơn giản và được sử dụng phổ biến với hai cách gồm: Sử dụng ngữ cảnh để dự đoán mục tiêu (Continuos Bag of Words – CBOW) và sử dụng một từ để dự đoán ngữ cảnh mục tiêu (Skip-gram). Tuy nhiên, word2vec có một số nhược điểm như không có khả năng nhận diện được những "từ" không có trong tập dữ liệu huấn luyện, mạng nơ-ron có kích thước lớn, tăng thời gian tính toán nếu thực hiện các thuật toán Gradient Descent tìm cực trị,…

Vì vậy, luận án lựa chọn cách tiếp cận trích chọn và xử lý dữ liệu đặc trưng động và đặc trưng tĩnh biểu diễn dưới dạng chuỗi như những dữ liệu có thuộc tính tuần tự nhằm hạn chế mất mát đặc trưng quan trọng về tính tuần tự. Bên cạnh đó, sử dụng các kỹ thuật trích chọn đặc trưng và tiền xử lý dữ liệu là các bước quan trọng để loại bỏ nhiễu và chuẩn bị dữ liệu cho việc huấn luyện mô hình học máy. Các kỹ thuật trích chọn đặc trưng và tiền xử lý dữ liệu như xử lý đặc trưng, lựa chọn đặc trưng quan trọng, đặc trưng có mức độ đại diện cao, chuẩn hóa dữ liệu, loại bỏ các giá trị bị khuyết và loại bỏ nhiễu có thể giúp cải thiện hiệu quả của mô hình phát hiện mã độc IoT dựa trên học máy. Việc lựa chọn các kỹ thuật trích xuất đặc trưng và tiền xử lý phù hợp giúp giảm chi phí tính toán và tăng độ chính xác của mô hình phát hiện mã độc IoT.

❖ ***Lựa chọn thuật toán học máy hiệu quả trong phát hiện mã độc IoT***

Mỗi thuật toán học máy, mạng học sâu đều có khả năng áp dụng hiệu quả trong xây dựng mô hình phát hiện độc IoT. Các thuật toán học sâu như mạng nơ-ron có thể giúp giải quyết các bài toán phức tạp hơn so với các mô hình học máy không sâu truyền thống. Đối với bài toán phát hiện mã độc IoT, một số mạng học sâu có thể được sử dụng để phân loại các mẫu độc hại với độ chính xác cao. Tuy nhiên, các mô hình này có phức tạp tính toán lớn, thời gian huấn luyện và phát hiện nhiều, khó triển khai trong thực tế cho các ứng dụng phát hiện mã độc IoT theo thời gian thực và các thiết bị IoT hạn chế tài nguyên. Do đó, việc lựa chọn thuật toán học máy không sâu truyền thống hay mạng học sâu cần được nghiên cứu phù hợp với loại đặc trưng sử dụng và mục đích bài toán phát hiện mã độc IoT cụ thể.

Trong xây dựng mô hình phát hiện mã độc IoT dựa trên các thuật toán học máy không sâu sử dụng đặc trưng dạng chuỗi của tập tin thực thi, các thuật toán học máy

thường được sử dụng và chứng minh hiệu quả trong các nghiên cứu bao gồm: Support Vectơ Machines, Naive Bayes, Random Forest, Decision Tree,…

- Support Vectơ Machines (SVM) là là một thuật toán học máy sử dụng để phân loại dữ liệu bằng cách tìm siêu phẳng (hyperplane) tốt nhất phân chia các điểm dữ liệu của các lớp khác nhau. SVM sẽ tìm ra một khoảng cách (margin) sao cho nó tối đa hóa khoảng cách từ các điểm dữ liệu gần nhất thuộc mỗi lớp đến siêu phẳng.

- Decision Tree (DT) là một cây phân cấp có cấu trúc được sử dụng để phân loại các đối tượng dựa trên một loạt các quy tắc. Các thuộc tính của đối tượng (ngoại trừ các thuộc tính phân loại) có thể thuộc các kiểu dữ liệu khác nhau bao gồm các giá trị nhị phân, danh nghĩa, thứ tự, định lượng, trong khi thuộc tính phân loại phải có kiểu dữ liệu là nhị phân hoặc thứ tự. Cho trước dữ liệu về các đối tượng bao gồm các thuộc tính và các lớp của chúng, cây quyết định sẽ tạo ra các quy tắc để dự đoán lớp của dữ liệu chưa nhìn thấy.

- Random Forest (RF) là một loại trong nhóm thuật toán cây quyết định.Thuật toán dựa trên nguyên lý tạo ra nhiều cây quyết định và kết hợp kết quả của chúng để đưa ra dự đoán cuối cùng. RF coi mỗi cây quyết định là một cử tri độc lập (giống như một cuộc bầu cử thực sự). Kết thúc cuộc bầu chọn, câu trả lời có nhiều phiếu bầu nhất từ cây quyết định sẽ được chọn.

- Naive Bayes (NB) là một thuật toán dựa trên định lý Bayes của lý thuyết xác suất để đưa ra phán đoán cũng như phân loại dữ liệu dựa trên dữ liệu quan sát và thống kê.

Mỗi thuật toán đều có những ưu và nhược điểm khi áp dụng đối với tập dữ liệu khác nhau. Việc so sánh ưu và nhược điểm của các thuật toán dựa trên các tiêu chí hiệu suất, hiệu quả, độ phức tạp và khả năng áp dụng được thể hiện trong Bảng 1.13.

Bảng 1.12. Ưu và nhược điểm một số thuật toán học máy truyền thống.

	SVM	**NB**	**RF**	**DT**
Ưu điểm	- Hiệu suất tốt trong không gian có số chiều cao. - Hiệu quả khi có một ranh giới phân loại rõ ràng. - Sử dụng một phép đo margin để tối ưu hóa độ phức tạp của mô hình.	- Hoạt động tốt với các tập dữ liệu nhỏ và có nhiễu. - Độ phức tạp thấp, không yêu cầu nhiều tài nguyên tính toán. - Đơn giản và dễ triển khai.	- Hiệu suất tốt trong việc xử lý dữ liệu lớn, nhiều đặc trưng. - Giảm overfitting bằng cách sử dụng các cây quyết định ngẫu nhiên và bỏ phiếu và có thể xác định độ quan trọng của các đặc trưng.	- Có thể xử lý cả dữ liệu dạng số, dữ liệu rời rạc và có khả năng xử lý dữ liệu có nhiễu. - Dễ hiểu và diễn giải.
Nhược điểm	- Cần tinh chỉnh siêu tham số để đạt hiệu suất tốt. - Thời gian huấn luyện có thể tăng nhanh khi dữ liệu lớn. - Khó áp dụng trực	- Giả định về sự độc lập giữa các đặc trưng có thể không phù hợp với thực tế. - Kết quả dự đoán có thể	- Thời gian huấn luyện có thể tăng nhanh với dữ liệu lớn và số lượng cây lớn. - Khó hiểu và khó diễn giải.	- Dễ bị overfitting, đặc biệt khi cây quyết định quá sâu và phức tạp. - Khả năng tổng quát hóa

	tiếp với dữ liệu không tuyến tính và có nhiều nhiễu.	không chính xác khi áp dụng với dữ liệu phức tạp.		không tốt khi áp dụng cho dữ liệu mới.

Hiệu suất và ưu, nhược điểm của mỗi thuật toán học máy có thể thay đổi tuỳ thuộc vào loại dữ liệu cụ thể và việc cấu hình siêu tham số. Vì vậy, trong quá trình xây dựng các mô hình phát hiện mã độc IoT dựa trên các thuật toán học máy không sâu, luận án nghiên cứu đề xuất các thuật toán học máy truyền thống phù hợp với đặc trưng đã trích chọn và lựa chọn các siêu tham số phù hợp đối với mỗi thuật toán học máy để đạt được hiệu quả tốt nhất.

Trong xây dựng mô hình phát hiện mã độc IoT dựa trên các mạng học sâu để dự đoán chuỗi đặc trưng dạng của tập tin thực thi, các mạng nơ-ron như RNN có thể được xem xét và lựa chọn. Mạng nơ-ron RNN được thiết kế để xử lý dữ liệu có cấu trúc dạng chuỗi và cho phép ghi nhớ thông tin từ các bước trước đó trong chuỗi để dự đoán phần tử tiếp theo. Cơ chế chính của RNN là mạng nơ-ron đệ quy, trong đó thông tin từ mỗi bước được truyền tiếp về sau thông qua chuỗi các đơn vị nơ-ron. Trong dự đoán từ trong chuỗi, mạng nơ-ron RNN sẽ xử lý một chuỗi các từ hoặc ký tự để từ đó dự đoán từ tiếp theo trong chuỗi. Tuy nhiên, mạng nơ-ron RNN gặp hạn chế khi xử lý chuỗi dài hoặc gặp vấn đề biến mất đạo hàm (gradient vanishing). Mạng học sâu Long Short Term Memory là một mô hình học sâu RNN được giới thiệu vào năm 1997 bởi Hochreiter và Schmidhuber để xử lý vấn đề biến mất đạo hàm trong mạng RNN cổ điển. Mạng nơ-ron LSTM được sử dụng trong nhiều ứng dụng bao gồm xử lý ngôn ngữ tự nhiên, dịch máy, nhận dạng giọng nói, nhận dạng hình ảnh,... Các mạng nơ-ron LSTM được đánh giá là có khả năng học và ghi nhớ thông tin dài hạn tốt hơn so với mô hình RNN cổ điển.

Một mạng nơ-ron LSTM chứa một tập các cổng (gate). Mỗi ô nhớ (cell) trong LSTM có một cấu trúc bộ nhớ dài hạn trong cấu trúc của một trạng thái cell được cập nhật vào, ra. Một cổng quên được xem xét ở trạng thái ẩn và đầu vào mới. Nó sẽ quyết định thông tin nào có thể bị lãng quên một cách an toàn. Sau đó, cổng đầu vào xác định thông tin từ đầu vào mới được đưa vào trạng thái cell cần nhớ. Cuối cùng, cổng đầu ra lấy thông tin từ đầu vào, trạng thái cell và trạng thái ẩn để tạo ra đầu ra cho bước hiện tại. Chìa khóa của mạng nơ-ron LSTM là trạng thái ô nhớ. Trạng thái ô nhớ chạy thẳng xuống toàn bộ chuỗi và chỉ có một số trao đổi tuyến tính không đáng kể. Do đó, thông tin chỉ chạy dọc theo nó mà không thay đổi. Thông tin về trạng thái ô nhớ có thể được thêm vào hoặc loại bỏ bởi các cấu trúc được gọi là cổng. Các cổng được tạo thành từ một lớp mạng nơ-ron sigmoid và phép toán nhân theo số điểm, đây là các cách tùy chọn để thông tin đi qua. Lớp mạng nơ-ron sigmoid trả về các số trong khoảng [0,1], biểu thị số lượng mỗi phần tử sẽ được cho qua. Giá trị bằng 0 có nghĩa là "không để gì đi qua", giá trị bằng 1 có nghĩa là "để mọi thứ đi qua". Ba trong số các cổng này được tạo ra để bảo vệ và quản lý trạng thái ô [106].

Trong mô hình ngôn ngữ tự nhiên, mạng nơ-ron LSTM có thể được sử dụng để dự đoán từ tiếp theo trong một câu. Vì vậy, luận án lựa chọn phương pháp tiếp cận sử dụng mạng nơ-ron LSTM để xây dựng các mô hình dự đoán các chuỗi đặc trưng trích xuất từ các tập tin thực thi, từ đó xây dựng các bộ phân lớp tập tin với các nhãn tương ứng.

Mặt khác, để nâng cao độ chính xác từ các mạng học sâu, phương pháp Bagging (Bootstrap Aggregating) đã được đề xuất sử dụng trong xây dựng mô hình phát hiện mã độc IoT dựa trên dự đoán chuỗi đặc trưng trích xuất từ tập tin thực thi. Bagging là một kỹ thuật kết hợp các mô hình dự đoán yếu thành một mô hình dự đoán mạnh hơn để cải thiện hiệu suất của mô hình. Các bước chính của Bagging bao gồm:

- Tạo ra các tập con của dữ liệu: Bagging sử dụng kỹ thuật Bootstrap để tạo ra các tập con ngẫu nhiên.

- Huấn luyện nhiều mô hình: Một số lượng N mô hình dự đoán được huấn luyện trên các tập dữ liệu con, mỗi tập con được tạo ra từ quá trình Bootstrap.

- Kết hợp dự đoán: Kết quả dự đoán cuối cùng được tính toán bằng cách kết hợp dự đoán từ tất cả các mô hình con. Thông thường, kết quả cuối cùng có thể là giá trị trung bình đối với bài toán hồi quy hoặc là kết quả dự đoán được bình chọn nhiều nhất đối với bài toán phân loại.

Độ phức tạp có thể tăng lên nếu sử dụng nhiều mạng học sâu khác nhau khi huấn luyện N mô hình dự đoán. Vì vậy, cần cân nhắc, đánh giá hiệu quả về độ chính xác và lựa chọn các mô hình dự đoán phù hợp với các yêu cầu bài toán.

❖ ***Tăng cường dữ liệu trong huấn luyện mô hình phát hiện mã độc IoT***

Tập dữ liệu huấn luyện là một yếu tố rất quan trọng đối với việc huấn luyện mô hình học máy. Tập dữ liệu huấn luyện có thể giúp mô hình học máy hiểu được các mẫu và đặc trưng trong dữ liệu, từ đó cải thiện khả năng dự đoán trên các dữ liệu mới. Tuy nhiên, khi tập dữ liệu huấn luyện không đủ lớn hoặc đại diện cho đủ các trường hợp có thể xảy ra trong thực tế, mô hình học máy có thể không hội tụ đúng cách (không đạt được một trạng thái tối ưu hoặc mong muốn) hoặc bị "overfitting" (mô hình có hiệu suất rất cao trên dữ liệu huấn luyện nhưng lại kém hiệu quả khi áp dụng vào dữ liệu kiểm tra hoặc dữ liệu thực tế). Khi tập dữ liệu quá nhỏ, mô hình không có đủ thông tin để học một cách tổng quát và chính xác. Mô hình có thể không học được các đặc điểm quan trọng và phổ quát của dữ liệu, dẫn đến việc hội tụ tại một điểm không phải là tối ưu. Trong phát hiện mã độc IoT, sự hạn chế trong thu thập các mẫu tập tin thực thi, số lượng các mẫu có sự chênh lệch lớn giữa các tập đã được gán nhãn có thể ảnh hưởng lớn đến kết quả dự đoán của mô hình.

Ví dụ 1.1: Với một bài toán phân lớp mã độc cho tập dữ liệu gồm 10.000 mẫu có nhãn là A và 500 mẫu có nhãn là B, nếu dữ liệu được chia thành tập dữ liệu huấn luyện

và kiểm tra theo tỉ lệ 80/20 thì tập đánh giá trên tập dữ liệu đó gồm 2.000 mẫu nhãn A và 100 mẫu nhãn B. Giả sử mô hình dự đoán tất cả nhãn đều là A, thì giá trị độ chính xác Accuracy là 2.000/2.100 (khoảng trên 95%). Vì vậy, mô hình phát hiện không phân lớp chính xác tất cả các tập tin nhãn B nhưng lại cho kết quả giá trị accuracy rất cao. Nếu sử dụng mô hình này trong thực tế thì mô hình sẽ dự đoán không chính xác trên nhóm dữ liệu có ít mẫu vì đa phần kết quả dự đoán thường sẽ thiên về một nhóm dữ liệu có nhiều mẫu.

Để giảm thiểu mất cân bằng dữ liệu, việc tăng cường tập dữ liệu tập tin thực thi là rất cần thiết, tuy nhiên quá trình thu thập, lấy mẫu dữ liệu và gán nhãn các tập tin thực thi trên môi trường IoT là công việc đặt ra nhiều thách thức. Đặc biệt là thu thập các mẫu tập tin của các biến thể mã độc trên thiết bị IoT mới. Trong trường hợp không thể thu thập thêm dữ liệu từ thực tế, thì một số phương pháp tăng cường dữ liệu phục vụ xây dựng mô hình phát hiện mã độc IoT bao gồm:

- *Tạo ra các phiên bản khác nhau của dữ liệu*: Là phương pháp tạo ra các phiên bản khác nhau của dữ liệu sử dụng các kỹ thuật biến đổi dữ liệu. Đối với dữ liệu hình ảnh, các phép biến đổi như lật, xoay và co giãn có thể giúp tạo thêm dữ liệu từ dữ liệu gốc. Đối với dữ liệu văn bản, các phép chuyển đổi từ, chuỗi trong văn bản có thể được xem xét để mở rộng dữ liệu. Đối với dữ liệu dạng chuỗi có thể sử dụng các phép biến đổi, chèn nhiễu hoặc xáo trộn chuỗi để tạo ra dữ liệu mới từ chuỗi ban đầu.

- *Sử dụng các phương pháp sinh dữ liệu mới hoặc tạo dữ liệu giả*: Sử dụng các thuật toán tạo dữ liệu tổng hợp hoặc mô phỏng để sinh ra dữ liệu mới từ dữ liệu hiện có như GANs (Generative Adversarial Networks) và VAE (Variational Autoencoders) [24].

Các phương pháp nêu trên có thể tạo ra dữ liệu mới và giúp mô hình huấn luyện học được các khuôn mẫu khác nhau của dữ liệu. Tuy nhiên các phương pháp này có thể tạo ra các dữ liệu mới giống với dữ liệu gốc, vì vậy việc học các tri thức mới sẽ không đem lại hiệu quả cao, đặc biệt trong phát hiện mã độc zero-day trên nền tảng mới. Để khắc phục nhược điểm trên, luận án lựa chọn phương pháp xử lý dữ liệu phục vụ tăng cường dữ liệu huấn luyện thông qua chuyển đổi dữ liệu từ các nguồn khác nhau như dữ liệu truyền thống, các tập dữ liệu đã chuẩn hoá và có nhiều tri thức để tạo ra các dữ liệu trên thiết bị IoT mới. Việc kết hợp dữ liệu đã có và dữ liệu chuyển đổi sẽ có thể giúp mô hình học được nhiều khuôn mẫu trước đó và tri thức mới trong quá trình huấn luyện.

1.5. Kết luận chương 1

Trong chương 1, luận án đã trình bày những nội dung cơ bản về thiết bị IoT và mã độc IoT với các xu hướng phát triển của mã độc trên các thiết bị IoT. Luận án cũng đã đánh giá, phân tích quy trình phân tích và cách thức xây dựng mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng của tập tin thực thi. Bên cạnh đó, luận án phân

tích, đánh giá một số mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng của tập tin thực thi hiệu quả. Từ đó, cho thấy việc nghiên cứu các mô hình phát hiện mã độc trên thiết bị IoT hiệu quả dựa trên học máy và đặc trưng của tập tin thực thi còn nhiều hạn chế và cần được cải tiến. Vì vậy, đề tài luận án tập trung vào nghiên cứu nâng cao hiệu quả cho các mô hình phát hiện mã độc trên thiết bị IoT tài nguyên hạn chế dựa trên học máy sử dụng đặc trưng của tập tin thực thi thu thập thông qua phương pháp phân tích động và phân tích tĩnh.

Từ nội dung đã phân tích, luận án xác định các hướng tiếp cận nâng cao hiệu quả của mô hình phát hiện mã độc IoT tập trung vào phương pháp thu thập, lựa chọn đặc trưng, tăng cường tập dữ liệu và sử dụng phương pháp học máy phù hợp, hiệu quả với các đặc trưng của tập tin thực thi thu thập thông qua phân tích động và phân tích tĩnh. Từ đó đề xuất bài toán nghiên cứu của luận án và giải quyết bài toán thông qua các mô hình phát hiện mã độc IoT đơn kiến trúc và đa kiến trúc được trình bày trong các chương tiếp theo của luận án.

CHƯƠNG 2: XÂY DỰNG MÔ HÌNH PHÁT HIỆN MÃ ĐỘC IOT ĐƠN KIẾN TRÚC HIỆU QUẢ SỬ DỤNG ĐẶC TRƯNG ĐỘNG CỦA TẬP TIN THỰC THI

2.1. Mở đầu

Phương pháp phân tích động để thu thập đặc trưng của tập tin thực thi sẽ loại bỏ được các kỹ thuật gây rối, mã hoá mã nguồn như các kỹ thuật "packed" và "obfuscated" mã độc sử dụng. Tuy nhiên, việc xây dựng, lựa chọn môi trường cho phép mã độc bộc lộ hoàn toàn các hành vi và giám sát và phân tích đầy đủ các hành vi là yêu cầu quan trọng trong phân tích động. Trong xây dựng mô hình phát hiện mã độc IoT dựa trên phân tích động, việc thu thập, trích xuất đặc trưng là bước quan trọng, ảnh hưởng lớn tới khả năng ứng dụng trong môi trường IoT hạn chế tài nguyên và độ chính xác của mô hình. Tập đặc trưng phải được thu thập trong thời gian tối thiểu nhất và mô tả đầy đủ nhất về chức năng, mục đích của tập tin mã độc. Trên cơ sở đó, các phương pháp trích chọn đặc trưng, tiền xử lý phù hợp với môi trường IoT cần được áp dụng để xây dựng mô hình phát hiện mã độc IoT hiệu quả dựa trên học máy.

Hiện nay, các đặc trưng động của tập tin thực thi có thể được thu thập thông qua việc giám sát tập tin thực thi trong các môi trường khác nhau gồm các giải pháp Honeypot như hệ thống IoTPOT [90], các giải pháp xây dựng môi trường ảo hoá như Cuckoo Sandbox [13], CWSandbox [18], Linon [62], REMnux [97], Detux [34], Lisa [30], F-Sandbox [115], V-Sandbox [64]… Mỗi giải pháp thường có ưu, nhược điểm khác nhau, vì vậy cần lựa chọn được giải pháp phù hợp để thu thập đầy đủ hành vi phục vụ xây dựng các mô hình phát hiện mã độc IoT hiệu quả. Dữ liệu thu thập từ phân tích động phục vụ xây dựng các mô hình phát hiện mã độc IoT dựa trên học máy chủ yếu là luồng mạng, chuỗi lời gọi API, chuỗi lời gọi hệ thống và tương tác với tài nguyên của hệ thống như tương tác tệp tin, thư mục, thanh ghi CPU, RAM,…. Đối với mã độc IoT, đặc trưng lời gọi hệ thống thu thập từ giám sát tập tin thực thi đã đem lại nhiều hiệu quả trong xây dựng các mô hình phát hiện dựa trên học máy như trong các nghiên cứu [79], [80], [84], [94], [115], [123],... khi sử dụng kết hợp phương pháp trích chọn đặc trưng như n-gram. Tuy nhiên, các phương pháp chọn đặc trưng n-gram chỉ xem xét tần suất xuất hiện của các đặc trưng trong chuỗi lời gọi hệ thống nên hiệu quả về tài nguyên sử dụng và khả năng phát hiện biến thể mã độc mới còn hạn chế.

Mặt khác, phần lớn các nghiên cứu về phát hiện mã độc trước đây NCS đã khảo sát đang tập trung vào các thiết bị IoT như máy tính cá nhân (PC, laptop), điện thoại di động sử dụng kiến trúc Intel và ARM. Hiện nay, các nghiên cứu cần tập trung phát triển các mô hình phát hiện mã độc IoT trên các kiến trúc vi xử lý khác như MIPS [115]. Bên cạnh đó, các mô hình đã đề xuất chưa xem xét hiệu quả của mô hình phát hiện trong các trường hợp có sự mất cân bằng giữa các tập dữ liệu huấn luyện. Đặc biệt, đối với các trường hợp xây dựng mô hình phát hiện mã độc IoT cho các thiết bị sử dụng hệ điều

hành nhúng khó thu thập về mẫu tập tin lành và các thiết bị sử dụng nền tảng kiến trúc vi xử lý mới.

Với những phân tích nêu trên, chương 2 đề xuất mô hình phát hiện mã độc IoT đơn kiến trúc sử dụng đặc trưng chuỗi lời gọi hệ thống được trích xuất thông qua phương pháp phân tích động tập tin thực thi. Hiện nay, có một số định nghĩa về lời gọi hệ thống như sau: Theo Marko [75] "một lời gọi hệ thống là một cơ chế để ứng dụng yêu cầu một dịch vụ từ nhân của hệ điều hành bên dưới". Lawrence Williams[5] đưa ra định nghĩa "lời gọi hệ thống của một hệ điều hành là một cơ chế cung cấp giao diện giữa một tiến trình và hệ điều hành đó". Đây được coi là một phương pháp để một chương trình máy tính yêu cầu một dịch vụ từ nhân của hệ điều hành. Từ những cách định nghĩa trên, luận án thống nhất xây dựng khái niệm lời gọi hệ thống trong IoT như sau:

Định nghĩa 2.1. *Lời gọi hệ thống là một cơ chế trong hệ điều hành cho phép ứng dụng tương tác với nhân hệ điều hành để yêu cầu thực hiện các tác vụ hệ thống nào đó như đọc, ghi dữ liệu, tạo tiến trình, quản lý bộ nhớ, truy cập các tài nguyên phần cứng,...*

Khi một ứng dụng yêu cầu một tác vụ hệ thống, nó sẽ gọi các lời gọi hệ thống tương ứng với tác vụ đó. Hệ điều hành sau đó sẽ thực hiện các tác vụ và trả về kết quả cho ứng dụng. Lời gọi hệ thống có thể được xem là một cách để các ứng dụng tương tác với hệ thống nhằm thực hiện các chức năng cần thiết.

Một lời gọi hệ thống gồm có tên và các tham số như hình 2.1 dưới đây:

```
execve("./Sample/kyudib", ["./Sample/kyudib"],
0x7fadaacc /* 12 vars */) = 0
brk(NULL) = 0x672000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x779b4000
uname(sysname="Linux", nodename="debian-mips") = 0
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT(No such file or directory)
access("/etc/ld.so.preload", R_OK) = -1 ENOENT(No such file or directory)
open("/etc/ld.so.cache", O_RDONLY) = 3
fstat64(3, st_mode=S_IFREG|0644, st_size=15133, ...) = 0
mmap(NULL, 15133, PROT_READ, MAP_PRIVATE, 3, 0) = 0x779b0000
close(3)
```

Hình 2.1. Nhật ký chuỗi lời gọi hệ thống bắt đầu của một mã độc IoT.

Thông thường, để giảm độ phức tạp trong xây dựng các phương pháp và mô hình phát hiện mã độc, các nhà nghiên cứu chỉ xem xét mức tên của lời gọi hệ thống như execve, brk, nmap,… để xây dựng các đặc trưng phục vụ huấn luyện các mô hình học máy.

[5] https://www.guru99.com/system-call-operating-system.html

Chương trình độc hại và chương trình lành tính có các hành vi tương tác hệ thống khác nhau như chương trình độc hại yêu cầu nhiều kết nối mạng hoặc truy cập các tài nguyên "nhạy cảm" thường xuyên hơn. Mỗi lời gọi hệ thống riêng lẻ không thể mô tả hành vi của một chương trình. Do đó, một số lời gọi hệ thống theo trình tự cần được xem xét để xác định hành vi mức mã (code-level) của chương trình. Nắm bắt được sự phụ thuộc giữa các lời gọi hệ thống trong một chuỗi lời gọi hệ thống có thể phục vụ việc phân loại hành vi bình thường và hành vi độc hại của một chương trình.

Định nghĩa 2.2. *Chuỗi lời gọi hệ thống là một tập các lời gọi hệ thống liên tiếp và có thứ tự để thực hiện một tác vụ hệ thống nhất định.*

Trong một chuỗi lời gọi hệ thống, các lời gọi hệ thống được sắp xếp theo thứ tự để thực hiện một tác vụ. Một lời gọi hệ thống được sử dụng như là đầu vào cho lời gọi hệ thống tiếp theo trong một chuỗi lời gọi hệ thống. Ví dụ, trên hệ điều hành Windows, việc đọc dữ liệu từ một tập tin trên ổ đĩa cứng, một ứng dụng sẽ gọi lần lượt các lời gọi hệ thống gồm open(),lseek(), read(), close() theo một thứ tự nhất định để thực hiện tác vụ đó.

Hạn chế chính của các nghiên cứu phát hiện mã độc IoT sử dụng chuỗi lời gọi hệ thống đã được NCS khảo sát bao gồm:

Thứ nhất, các nghiên cứu sử dụng đặc trưng chuỗi lời gọi hệ thống phần lớn chỉ xem xét tần suất xuất hiện của các lời gọi hàm mà chưa khai thác hết đặc trưng tuần tự và mối liên hệ giữa các lời gọi trong chuỗi lời gọi hệ thống thu thập từ một tập tin. Các mô hình phát hiện mã độc IoT có độ chính xác cao đang sử dụng nhiều loại đặc trưng động cần thu thập hoặc độ dài chuỗi lời gọi hệ thống cần thu thập phải đủ lớn để huấn luyện. Các phương pháp trích chọn đặc trưng chuỗi lời gọi hệ thống nhằm hạn chế số lượng lời gọi hệ thống cần thu thập như nén (compressing) và giảm đặc trưng không thể loại bỏ hoàn toàn các đặc trưng gây nhiễu và gây mất thông tin quan trọng làm ảnh hưởng đến hiệu quả của mô hình phát hiện mã độc IoT.

Thứ hai, các nghiên cứu nâng cao độ chính xác cho mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đồ thị chuỗi lời gọi hệ mới chỉ xem xét mối liên hệ hạn chế giữa các đặc trưng lời gọi hệ thống và có nhược điểm liên qua đến số chiều, độ phức tạp của mô hình và sự phụ thuộc vào kiến trúc vi xử lý hoặc hệ điều hành.

Thứ ba, các phương pháp Ensemble [105], Bagging [83] đã được các nghiên cứu sử dụng để nâng cao độ chính xác cho mô hình phát hiện. Tuy nhiên, việc sử dụng kết hợp nhiều thuật toán học máy, mạng học sâu khác nhau có thể làm tăng độ phức tạp, thời gian tính toán, phát hiện. Việc sử dụng kết hợp cùng một thuật toán hoặc kiến trúc mạng học sâu cho các nhãn dữ liệu chưa được xem xét, đánh giá tính hiệu quả trong các bài toán phát hiện mã độc IoT. Chưa có nghiên cứu sử dụng các phương pháp này để cải thiện mạnh mẽ hiệu suất của mô hình dự đoán, đặc biệt là khi sử dụng trong các bài toán phát hiện nhanh mã độc với dữ liệu lớn và không cân bằng giữa các nhãn huấn luyện.

Thứ tư, theo khảo sát của luận án, các nghiên cứu phát hiện mã độc IoT đang tập trung nhiều trên các thiết bị sử dụng nền tảng hệ điều hành Android hoặc kiến trúc vi xử lý phổ biến trước đây như Intel, ARM. Chưa có nhiều nghiên cứu tập trung xây dựng mô hình phát hiện mã độc cho các kiến trúc vi xử lý của thiết bị IoT tài nguyên hạn chế phổ biến khác trong các hệ thống mạng hiện nay như bộ định tuyến, bộ chuyển mạch, điểm truy cập mạng không dây và camera IP. Khả năng ứng dụng các mô hình phát hiện mã độc sử dụng lời gọi hệ thống trong hệ thống IoT còn hạn chế còn do thời gian thu thập và độ dài chuỗi lời gọi hệ thống lớn.

Bên cạnh đó, với sự phát triển của các mạng học sâu, việc sử dụng mạng học sâu LSTM đã chứng minh được hiệu quả trong huấn luyện các mô hình dự đoán từ trong một chuỗi và việc sử dụng phương pháp Bagging đối với từng tập dữ liệu khác nhau để khai thác các đặc trưng từ các tập dữ liệu có thể đem lại nhiều hiệu quả mới trong xây dựng mô hình phát hiện mã độc. Do vậy, hướng tiếp cận sử dụng phương pháp Bagging với các mạng học sâu sử dụng đặc trưng chuỗi lời gọi hệ thống thu thập từ phân tích động tập tin có thể đem lại hiệu quả trong xây dựng các mô hình phát hiện mã độc nói chung và mã độc IoT nói riêng.

Luận án tập trung nghiên cứu đề xuất một mô hình phát hiện mã độc IoT đơn kiến trúc hiệu quả sử dụng phương pháp Bagging dựa trên một kiến trúc mạng nơ-ron LSTM đối với từng tập dữ liệu chuỗi lời gọi hệ thống trích xuất từ phân tích động tập tin thực thi. Việc sử dụng các mô hình huấn luyện từ một kiến trúc mạng LSTM trên các tập dữ liệu khác nhau đã gán nhãn có thể giúp tăng khả năng tìm hiểu và xử lý dữ liệu đối với từng tập chuỗi lời gọi hệ thống. Việc huấn luyện trên từng tập dữ liệu cũng góp phần nâng cao hiệu quả cho mô hình phát hiện trong trường hợp có sự chênh lệch lớn hoặc hạn chế về số lượng của các tập dữ liệu được gán nhãn. Đặc biệt, trong việc thu thập các tập dữ liệu trên thiết bị IoT, các nghiên cứu [64],[114] đã cho thấy việc thu thập các tập tin ELF lành tính có đầy đủ điều kiện để có thể thực thi trên môi trường phân tích động gặp nhiều khó khăn. Các nghiên cứu trên gặp hạn chế về số lượng tập tin ELF lành tính có đầy đủ thư viện của ngôn ngữ lập trình để thực thi độc lập trên thiết bị hoặc tương thích với các hệ điều hành nhúng. Vì vậy, việc chỉ áp dụng các mô hình học máy, mạng học sâu theo cách tiếp cận đơn nhất sẽ có thể không hiệu quả cao trong phát hiện mã độc IoT khi có sự mất cân bằng dữ liệu huấn luyện.

Thông thường, các mã độc hại thường lây nhiễm (chèn) vào tập tin lành tính như virus, trojan, backdoor,… Khi thực thi tập tin mã độc, các chức năng của chương trình bình thường gốc bị lây nhiễm mã độc vẫn hoạt động, do đó, nhiều lời gọi hệ thống của chương trình bình thường gốc vẫn nằm trong chuỗi lời gọi hệ thống. Do đó, cần thu thập được đầy đủ các lời gọi hệ thống của tập tin và sử dụng phương pháp xây dựng bộ phân lớp chuỗi mã thực thi phù hợp. Đã có các mô hình phát hiện mã độc dựa trên đặc trưng

lời gọi hệ thống sử dụng một mạng nơ-ron để phân lớp như nghiên cứu của Mathew và cộng sự [98], Maniath và cộng sự [100], Yingying Liu và cộng sự [121], tuy nhiên các mô hình này dựa trên việc huấn luyện mô hình phân lớp sử dụng mạng nơ-ron LSTM để học và phân biệt các biểu diễn lời gọi hệ thống của tập tin lành tính và tập tin độc. Vì vậy, để học và phân biệt được các chuỗi lời gọi hệ thống này, mạng nơ-ron LSTM thường cần nhiều dữ liệu để huấn luyện, các chuỗi lời gọi hệ thống cần đủ dài để mô hình có đủ thông tin để phân lớp.

Với đặc trưng lời gọi hàm hệ thống thu thập từ phân tích động đoạn mã độc hại và mã lành tính trên thiết bị IoT có sự khác nhau lớn và cần phương pháp biểu diễn mạnh mẽ cho từng tập dữ liệu, luận án hướng đến phương pháp huấn luyện các mô hình học trên từng nhãn dữ liệu mã độc và lành tính. Việc huấn luyện các mô hình học sâu trên từng tập dữ liệu đã gán nhãn có thể giúp mạng học sâu dự đoán chuỗi như LSTM sẽ học được nhanh và chính xác hơn các đặc trưng riêng biệt từ các tập dữ liệu này và có thể lựa chọn các siêu tham số cho mỗi mô hình phù hợp với đặc điểm và số lượng của từng tập dữ liệu. Điều này giúp giải quyết vấn đề chệnh lệch số lượng giữa các nhãn huấn luyện và xác định được độ dài chuỗi lời gọi hệ thống ngắn nhất đủ để có thể phân biệt được chuỗi lời gọi hệ thống tạo ra từ tập tin mã độc và lành tính. Do đó, chương 2 sẽ trình bày mô hình phát hiện mã độc IoT đơn kiến trúc hiệu quả với việc thu thập chuỗi lời gọi hệ thống ngắn thu thập từ phân tích động nhưng vẫn đảm bảo các tiêu chí về độ chính xác của mô hình dựa trên phương pháp Bagging nhưng sử dụng cùng một kiến trúc mạng nơ-ron LSTM để huấn luyện trên hai tập dữ liệu đã được gán nhãn mã độc và lành tính.

Mô hình đề xuất trong chương này giải quyết vấn đề sau đây:

- *Cho M = {M_1, M_2, ..., M_n} với M_i là mã độc (i = 1,2,...,m). M-Model là mô hình dự đoán chuỗi lời gọi hệ thống xây dựng từ tập M.*

- *Cho B = {B_1, B_2, ..., B_k} với B_j là lành tính (j = 1,2,...,n). B-Model là mô hình dự đoán chuỗi lời gọi hệ thống xây dựng từ tập B.*

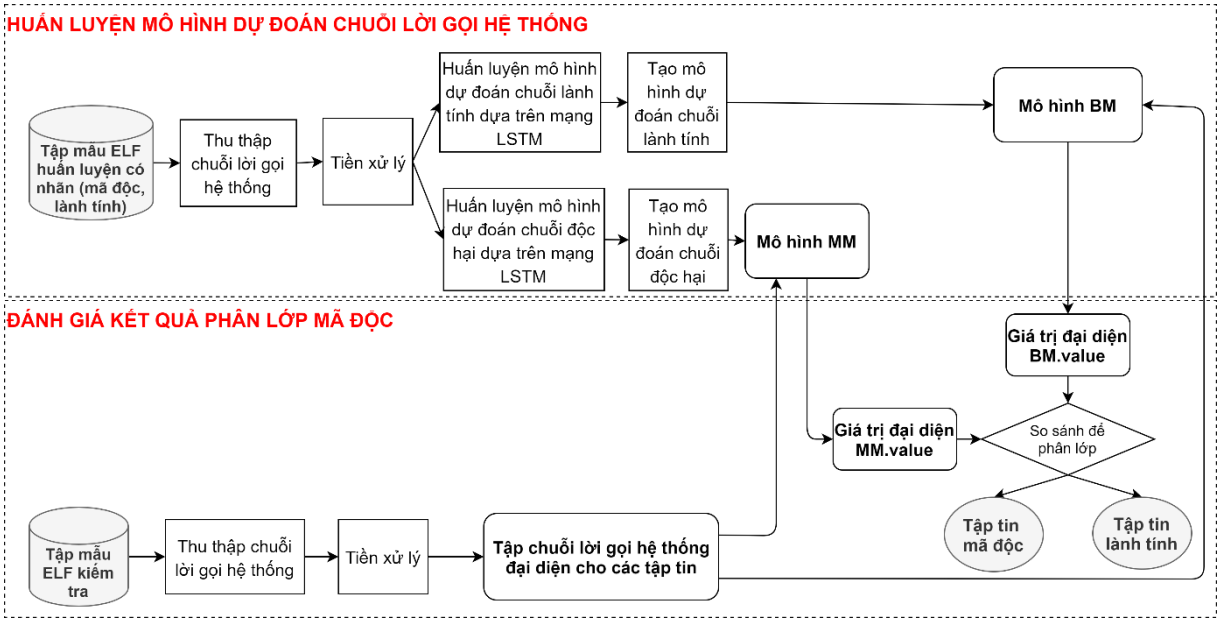
- *Cho S_1^L = {s_1, s_2, ..., s_L} là một chuỗi lời gọi hệ thống thu thập từ tập tin có độ dài L, mỗi lời gọi hệ thống s_t (t = 1,2,...,L) trong tập S_1^L sẽ xác định xác suất xuất hiện tương ứng của chuỗi lời gọi hệ thống S_1^t đối với M-Model và B-Model lần lượt là $P_{M-Model}(S_1^t)$ và $P_{B-Mode}(S_1^t)$.*

- *Cần tìm giá trị k = $arg\,min_k$ $|P_{M-Model}(S_1^t) - P_{B-Model}(S_1^t)|$ và độ chính xác được đảm bảo khi so sánh với các mô hình khác có liên quan.*

2.2. Xây dựng mô hình phát hiện mã độc IoT đề xuất dựa trên chuỗi lời gọi hệ thống

Quá trình xây dựng mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên đặc trưng chuỗi lời gọi hệ thống thu thập từ phương pháp phân tích động tập tin thực thi đề xuất

gồm 2 giai đoạn chính: Huấn luyện mô hình dự đoán chuỗi lời gọi hệ thống và đánh giá kết quả phân lớp mã độc dựa trên hai mô hình dự đoán chuỗi lời gọi hệ thống đã huấn luyện. Tổng thể quá trình xây dựng mô hình phát hiện mã độc IoT đơn kiến trúc sử dụng đặc trưng động của tập tin thực thi đề xuất được mô tả trong hình 2.2.



Hình 2.2. Kiến trúc tổng quan quá trình xây dựng mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên chuỗi lời gọi hệ thống thu thập từ phân tích động đề xuất.

2.2.1. Thu thập chuỗi lời gọi hệ thống

Chuỗi lời gọi hệ thống có thể được thu thập và phân tích bằng các công cụ khác nhau hoặc thông qua môi trường sandbox. Một sandbox có thể phù hợp đối với việc thu thập các chuỗi lời gọi hệ thống trên từng nền tảng hệ điều hành và kiến trúc vi xử lý khác nhau. Kết quả so sánh một số sandbox phục vụ thu thập chuỗi lời gọi hệ thống của tập tin thực thi được thể hiện trong Bảng 2.1.

Bảng 2.1. So sánh một số sandbox phục vụ thu thập chuỗi lời gọi hệ thống của tập tin thực thi.

Tên sandbox	**Hỗ trợ hệ điều hành**	**Đa kiến trúc vi xử lý**	**Tương tác với tệp hệ thống**	**Chương trình mô phỏng/ ảo hoá**	**Mô phỏng NVRAM**	**Mô phỏng kết nối C&C**
Cuckoo [13]	Windows, Android, Linux	Có	Có	Vmware, KVM, Virtual Box	Không	Không
REMnux [97]	Ubuntu	Không	Không	Vmware	Không	Không
Limon [62]	Linux	Không	Có	Vmware	Không	Không
Padawan [91]	Linux	Có	Có	QEMU	Không	Không
LiSa [33]	Linux	Có	Có	QEMU	Không	Không
V-Sandbox [64]	Windows, Linux	Có	Có	QEMU	Không	Có
F-Sandbox	Linux	Có	Có	QEMU	Có	Có

[115]						

Từ kết quả so sánh, hai giải pháp V-Sandbox và F-Sandbox có khả năng phù hợp với công việc thu thập chuỗi lời gọi hệ thống từ một tập tin thực thi trong môi trường IoT. Tuy nhiên, với khả năng mô phỏng NVRAM như thiết bị thật để kích hoạt các hành vi của tập tin và hạn chế số lượng dữ liệu thu thập khi chỉ thu thập một loại hành vi lời gọi hệ thống nhằm tiết kiệm thời gian xử lý báo cáo kết quả thu thập thì luận án lựa chọn giải pháp F-Sandbox để thu thập chuỗi lời gọi hệ thống của các tập tin thực thi trên môi trường IoT. F-Sandbox [115] sử dụng giải pháp Kprobe để thu thập các lời gọi hệ thống được tạo ra từ chương trình đang chạy và các tiến trình con của nó. INetSim được sử dụng để mô phỏng các dịch vụ mạng và Internet giúp các mã độc bộc lộ các hành vi yêu cầu có kết nối mạng hoặc máy chủ C&C.

Cấu trúc của F-Sandbox gồm 4 thành phần chính được thể hiện trong hình 2.3 gồm: Máy ảo cài đặt QEMU, bộ điều khiển (Sandbox Controller), bộ giám sát máy ảo (QEMU Monitor), máy chủ mô phỏng dịch vụ Internet (INetSim Server).

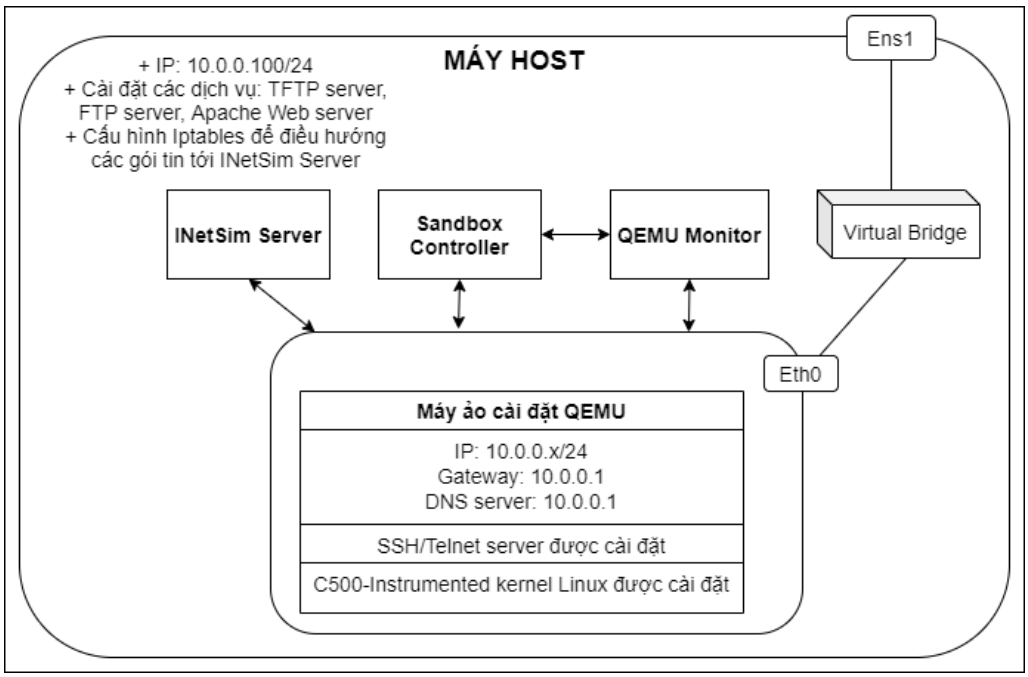
- Máy ảo cài đặt QEMU là một máy ảo trên nền tảng phần mềm QEMU[6]. Máy ảo có khả năng mô phỏng và ảo hoá thiết bị mạng hỗ trợ mô phỏng 26 kiến trúc CPU khác nhau, đặc biệt các kiến trúc vi xử lý trên các thiết bị IoT như MIPS, ARM,... và hỗ trợ trên các hệ điều hành Windows, Linux. Bên cạnh đó, máy ảo cài đặt QEMU được NCS tích hợp cài đặt thêm dịch vụ SSH Server để tương tác với bên ngoài thông qua Sandbox Controller.

- Sandbox Controller là thành phần tương tác với QEMU Monitor thông qua việc gọi các lệnh hiển thị cấu hình mạng, khôi phục các bản ảnh (snapshot) đã lưu trữ khi khởi tạo. Sandbox Controller tương tác với máy ảo cài đặt thông qua thực thi các câu lệnh SSH đến máy ảo và truyền tải, cấp quyền, yêu cầu thực thi các tập tin trong máy ảo QEMU.

- QEMU Monitor là thành phần thực hiện giám sát, quản lý tương tác từ Sandbox Controller và Máy ảo QEMU.

- INetSim Server là công cụ giả lập các dịch vụ của mạng Internet, chứa các kịch bản Perl được sử dụng để mô phỏng các dịch vụ mạng phổ biến như DNS, HTTP, HTTPS, FTP, TELNET, SSH,...

[6] http://wiki.qemu.org



Hình 2.3. Cấu trúc của F-Sandbox.

Để tiến hành thu thập chuỗi lời gọi hệ thống từ các tập tin thực thi, mỗi tập tin thực thi ELF được thực thi trong môi trường F-Sandbox với một cấu hình đồng nhất để thu thập các tập tin nhật ký các lời gọi hệ thống của tập tin. Sau đó, một chuỗi lời gọi hệ thống được trích xuất dựa trên tên của các lời gọi hệ thống thu thập được. Để có được chuỗi lời gọi hệ thống hoàn chỉnh nhất, các chuỗi lời gọi hệ thống trên mỗi kiến trúc vi xử lý được xây dựng bởi tất cả các lời gọi hệ thống trên kiến trúc tương ứng có thể xuất hiện được luận án tập hợp từ các nguồn khác nhau.

Việc thu thập các chuỗi lời gọi hệ thống của các tập tin ELF được thực hiện với giải pháp F-Sandbox trên cùng một cấu hình phần cứng, có cùng một môi trường mạng mô phỏng và thực thi trong cùng một khoảng thời gian giống nhau. Đối với thu thập chuỗi lời gọi hệ thống, thời gian thực thi tập tin càng nhiều thì số lượng các lời gọi hệ thống thu được càng lớn. Theo nghiên cứu của Raymond Canzanese [96] việc thu thập lời gọi hệ thống phải thực hiện trong khoảng thời gian tối thiểu 5 giây hoặc độ dài tối thiểu là 1.500 đối với phát hiện mã độc trên kiến trúc i386 sử dụng hệ điều hành Windows. Hansen [101] đã tiến hành thu thập lời gọi hệ thống trên tập dữ liệu sử dụng hệ điều hành Windows trong thời gian 200 giây. Tuy nhiên, đối với tập dữ liệu trên kiến trúc IoT như MIPS kết quả thu được lại có sự khác biệt. Vì vậy cần xác định khoảng thời gian thu thập phù hợp và tương đồng với các nghiên cứu khác có liên quan để tiến hành so sánh, đánh giá hiệu quả của mô hình phát hiện đề xuất.

Để thu thập lời gọi hệ thống của một tập tin thực thi, các tập tin được đưa vào một thư mục đã được cấu hình trên máy chủ HOST, sau đó khởi chạy F-sandbox để kết nối SSH đến máy ảo QEMU thông qua giao thức SFTP trên Sandbox Controller phục vụ truyền tải tập tin vào máy ảo QEMU, cấp quyền và thực thi tập tin trong

khoảng thời gian đã cấu hình. Các công cụ được tích hợp trong máy ảo QEMU như Strace sẽ tiến hành thu thập các lời gọi hệ thống và lưu trữ dưới dạng tập tin nhật ký theo chuỗi thời gian. Một tập tin thực thi có thể tạo nhiều tiến trình và một tiến trình sẽ tạo ra một bản ghi nhật ký các lời gọi hệ thống. Bên cạnh đó, nếu trong quá trình thực thi tập tin có xuất hiện các yêu cầu kết nối mạng thì F-Sandbox sẽ được chuyển hướng tới INetSim Server thông qua việc cấu hình IPtable và DNS-poisoning nhằm phản hồi yêu cầu để tập tin có thể tiếp tục thực thi và bộc lộ các hành vi khác.

Kết quả của quá trình thu thập đối với một tập tin thực thi là các tập tin nhật ký lời gọi hệ thống tương ứng với các tiến trình được tạo ra khi thực thi tập tin đó trong F-Sandbox. Các tập tin nhật ký này sẽ được tiền xử lý phục vụ quá trình huấn luyện trong các bước tiếp theo.

2.2.2. Tiền xử lý dữ liệu

Đầu tiên, các tập tin nhật ký lời gọi hệ thống được tạo từ một tập tin được ghép nối lại với nhau theo thứ tự các tiến trình con tạo ra để phục vụ xây dựng một chuỗi lời gọi hệ thống đại diện cho tập tin thực thi. Nếu chuỗi lời gọi hệ thống có độ dài quá ngắn so với một chương trình thực thi bình thường sẽ bị xóa bỏ khỏi tập dữ liệu chuỗi lời gọi hệ thống. Những tập tin này thường là các tập tin bị lỗi thực thi hoặc tập tin không thu thập được hành vi trong môi trường F-Sandbox do sử dụng các kỹ thuật lẩn tránh.

Tiếp theo, để phục vụ huấn luyện các mô hình dự đoán chuỗi lời gọi hệ thống dựa trên các mạng nơ-ron LSTM, hai bộ dữ liệu gồm Mal-SysCallLog là các chuỗi lời gọi hệ thống thu thập từ tất cả các tập tin ELF có nhãn mã độc và Beg-SysCallLog là các chuỗi lời gọi hệ thống thu thập từ tất cả các tập tin ELF có nhãn lành tính được xây dựng. Đối với mỗi tập Mal-SysCallLog và Beg-SysCallLog, mỗi lời gọi hệ thống trong chuỗi được xem xét như là một từ trong ngôn ngữ tự nhiên. Mỗi chuỗi lời gọi hệ thống thu thập từ một tập tin được coi như một câu trong một văn bản. Văn bản được tạo ra bằng việc ghép nối tất cả các chuỗi lời gọi hệ thống thu thập từ tập tin có nhãn mã độc hoặc lành tính tương ứng.

Cuối cùng, quá trình chuyển đổi các chuỗi lời gọi hệ thống thành các vectơ được thực hiện để phục vụ xây dựng các mô hình dự đoán chuỗi trong bước tiếp theo. Quá trình chuyển đổi các chuỗi lời gọi hệ thống thành các vectơ phục vụ quá trình huấn luyện các mô hình dự đoán chuỗi lời gọi hệ thống được mô tả trong thuật toán 2.1. Kích thước từ điển lời gọi hệ thống được xác định thông qua thu thập tất cả các lời gọi hệ thống có thể xuất hiện của nền tảng kiến trúc tương ứng. Luận án tiến hành xác định từ điển lời gọi hệ thống từ các nguồn đã công bố và thực nghiệm thu thập lời gọi hệ thống trên lượng lớn các tập tin thực thi của nền tảng kiến trúc tương ứng. Ví dụ, trên nền tảng kiến trúc MIPS, luận án xác định được từ điển lời gọi hệ thống gồm 345 syscall. Việc chuyển đổi các chuỗi lời gọi hệ thống được tiến hành trên từng tập dữ liệu Mal-SysCallLog và

Beg-SysCallLog.

Thuật toán 2.1. Tiền xử lý chuỗi lời gọi hệ thống.

Đầu vào*: Một chuỗi lời gọi hệ thống (syscall) với ngưỡng độ dài L*
Đầu ra: *Hai vectơ sau khi biến đổi chuỗi lời gọi hệ thống*

 # khởi tạo danh sách x và y
1: x ← []
2: y ← []
3: **For** i **in** [0, min (L, length(syscall))]
 # Tách chuỗi syscall thành các chuỗi con có độ dài từ 0 tới i
4: Slice_syscall ← substring(systemcall,0,i)
 # Thêm đệm là giá trị "0" thông qua padding vào đầu chuỗi con vừa tách để đảm bảo độ dài các chuỗi là L
5: x_padded ← padding (slice_syscall, max_length=L)
 # Mã hoá x_padded và y_padded thành ma trận one-hot encoding với độ dài bằng kích thước từ điển lời gọi hệ thống trên kiến trúc tương ứng (vocab_size)
6: x_onehot ← onehot (x_padded, num_class = vocab_size)
7: y_onehot ← onehot (syscall[i], num_class = vocab_size)
 # Thêm dữ liệu x_onehot, y_onehot vào danh sách x và y
8: x.append (x_onehot)
9: y.append (y_onehot)
10: **Endfor**
11: **Return** x, y

2.2.3. Huấn luyện mô hình dự đoán chuỗi lời gọi hệ thống dựa trên mạng nơ-ron LSTM

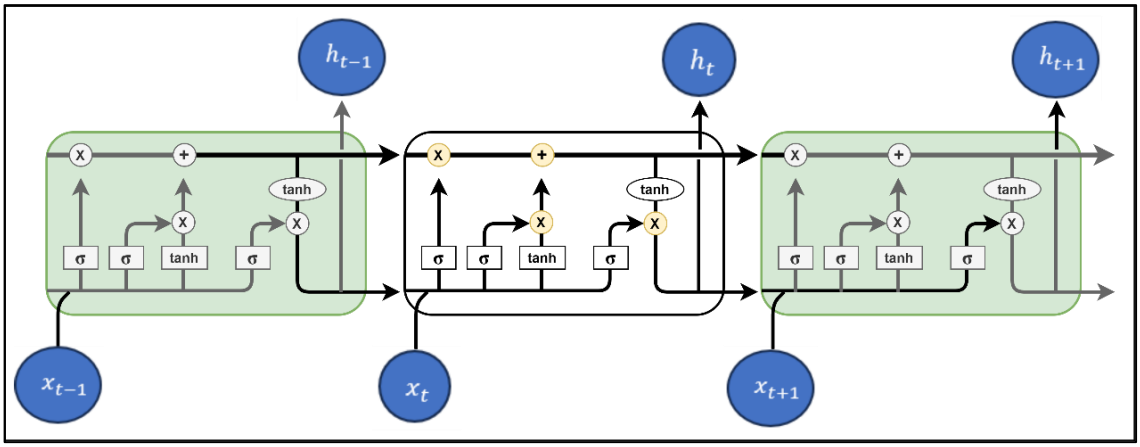
Mô hình ngôn ngữ thống kê đã giải quyết được vấn đề dự đoán dữ liệu chuỗi thời gian trong dữ liệu tuần tự. Mô hình phổ biến trong phát hiện mã độc dựa trên n-gram đã có thể giải quyết một số bài toán xây dựng mô hình phát hiện mã độc dựa trên đặc trưng biểu diễn dạng chuỗi. Tuy nhiên nó không thể dự đoán được đặc trưng nếu chưa tồn tại trong tập đặc trưng huấn luyện. Khác với phương pháp n-gram, mạng nơ-ron LSTM sử dụng tất cả các lời gọi hệ thống trước đó trong một chuỗi lời gọi hệ thống để dự đoán lời gọi hệ thống tiếp theo. Nhiều thông tin ngữ cảnh được chọn từ các chuỗi lời gọi hệ thống hơn so với một số mô hình học máy khác như mô hình học máy sử dụng phương pháp n-gram. Do đó, một mô hình học máy có thể được xây dựng dựa trên mạng nơ-ron LSTM để dự đoán phân lớp của chuỗi lời gọi hệ thống. Tuy nhiên, với các đặc trưng riêng biệt trong hành vi khi thực thi các tập tin mã độc IoT và lành tính, việc sử dụng phương pháp bagging đối với từng tập dữ liệu đã gán nhãn sẽ có khả năng tìm hiểu và xử lý dữ liệu đặc trưng dạng chuỗi trích xuất từ phân tích động tập tin IoT đa dạng hơn. Bên cạnh đó, việc khai thác tốt đặc trưng của từng tập dữ liệu lành tính và mã độc thông qua từng mô hình cho mỗi tập dữ liệu có thể hạn chế ảnh hưởng khi các tập dữ liệu có

sự mất cân bằng.

Vì vậy, các mô hình dự đoán chuỗi được tiến hành xây dựng dựa trên phương pháp Bagging sử dụng mạng học sâu LSTM đối với từng tập dữ liệu đã được gán nhãn. Mỗi mô hình dự đoán chuỗi được huấn luyện cùng kiến trúc mạng nơ-ron LSTM nhưng sử dụng các tham số khác nhau để khai thác tối đa các đặc trưng chuỗi lời gọi hệ thống của từng tập dữ liệu. Trong phạm vi bài toán phát hiện mã độc của luận án, hai mô hình dự đoán chuỗi lời gọi hệ thống là mô hình lành tính B-Model (Benign Model) và mô hình độc hại M-Model (Malware Model) từ hai tập dữ liệu đã tiền xử lý tương ứng là Beg-SysCallLog và Mal-SysCallLog được huấn luyện phục vụ phân hai lớp. Việc sử dụng hai mạng LSTM giúp xây dựng mô hình phát hiện có khả năng tìm hiểu và xử lý dữ liệu đa dạng, có nhiều thông tin riêng biệt do đặc trưng thứ tự các lời gọi hệ thống trong chuỗi trích xuất từ tập tin mã độc IoT và tập tin lành tính có nhiều sự khác biệt.

Cuối cùng, việc phân loại chuỗi lời gọi hệ thống được dự đoán dựa trên kết quả dự đoán được bình chọn giá trị đại diện tính toán dựa trên từng mô hình đã huấn luyện. Trong luận án, hai mô hình dự đoán chuỗi là cơ sở tiền đề quan trọng để tính toán giá trị đại diện khi tính toán xác suất xảy ra chuỗi trên từng mô hình phục vụ phân lớp chuỗi lời gọi hệ thống chưa xác định nhãn với độ dài của chuỗi lời gọi hệ thống cần thu thập tối thiểu nhất.

Kiến trúc mạng nơ-ron LSTM được luận án sử dụng để xây dựng các mô hình dự đoán chuỗi bao gồm ba lớp: Lớp đầu vào, lớp đầu ra và lớp ẩn. Lớp đầu vào là một vectơ được biểu diễn bằng mã hóa one-hot. Lớp đầu ra là một vectơ của phân phối xác suất. Một lời gọi hệ thống được coi là một từ trong mô hình ngôn ngữ tự nhiên. Chuỗi lời gọi hệ thống được coi là một câu trong mô hình ngôn ngữ tự nhiên. Trong xây dựng mô hình huấn luyện dựa trên mạng nơ-ron LSTM, mỗi lời gọi hệ thống tiếp theo trong chuỗi được dự đoán theo tất cả các lời gọi hệ thống trước đó trong chuỗi. Cách thức xây dựng kiến trúc khối nhớ trong mạng LSTM được thể hiện như hình 2.4.



Hình 2.4. Cách thức xây dựng kiến trúc khối nhớ trong mạng LSTM [7].

[7] https://www.researchgate.net/figure/Basic-representation-of-the-LSTM_fig1_353163587

Để huấn luyện hai mô hình dự đoán chuỗi lời gọi hệ thống cho từng tập dữ liệu lần lượt là B-Model và M-Model, các lời gọi hệ thống trong từng tập được mã hoá dưới dạng vectơ để đưa vào mạng nơ-ron LSTM. Các mô hình được huấn luyện dựa trên kiến trúc mạng LSTM cơ bản với các bước sau đây:

Bước 1: LSTM sẽ quyết định xem thông tin nào sẽ cho phép đi qua ô trạng thái và nó được kiểm soát bởi hàm *sigmoid* trong lớp cổng quên. Đầu tiên, nó nhận đầu vào là hai giá trị và trả về một giá trị trong khoảng [0,1]. Nếu giá trị bằng 0 thể hiện rằng "bỏ qua toàn bộ thông tin" và bằng 1 thể hiện là "giữ lại toàn bộ thông tin". Điều này giống như trong ngôn ngữ tự nhiên, chúng ta đang cố gắng dự báo từ tiếp theo dựa vào toàn bộ những từ trước đó, ô trạng thái có thể bao gồm loại của chủ ngữ hiện tại để sử dụng cho đại từ ở các câu tiếp theo. Tuy nhiên, chủ ngữ không cố định và khi có chủ ngữ mới xuất hiện thì chúng ta cần quên đi chủ ngữ cũ. Vì vậy, lớp cổng quên cho phép cập nhật thông tin mới và lưu trữ giá trị của nó khi có thay đổi theo thời gian.

Bước 2: LSTM sẽ quyết định loại thông tin nào sẽ được lưu trữ trong ô trạng thái. Đầu tiên, một lớp ẩn của hàm *sigmoid* được gọi là lớp cổng vào quyết định giá trị bao nhiêu sẽ được cập nhật. Tiếp theo, lớp ẩn của hàm *tanh* sẽ tạo ra một vectơ của một giá trị trạng thái mới và có thể được thêm vào ô trạng thái. Sau đó, một cập nhật cho trạng thái được tạo ra thông qua kết quả kết hợp của lớp cổng vào và lớp ẩn của hành tanh.

Sau đó, trạng thái cũ sẽ được nhân với giá trị lớp cổng quên tương ứng với việc quên đi những thông tin quyết định được phép quên. Phần tử đề cử là một giá trị mới được sử dụng để tính toán tương ứng với việc bao nhiêu thông tin được cập nhật vào mỗi giá trị trạng thái.

Bước 3: LSTM quyết định trả về kết quả đầu ra. Kết quả đầu ra sẽ dựa trên ô trạng thái. Đầu tiên, chúng chạy qua một lớp *sigmoid* nơi quyết định phần nào của ô trạng thái sẽ ở đầu ra. Sau đó, ô trạng thái được đưa qua hàm *tanh* để chuyển giá trị về khoảng [-1,1] và nhân với đầu ra của một cổng *sigmoid*, do đó nó chỉ trả về phần mà chúng ta quyết định.

Vì vậy, thông qua xây dựng kiến trúc khối nhớ, mô hình có khả năng dự đoán và tính toán xác suất của lời gọi hệ thống tiếp theo trong một chuỗi thông qua tất cả lời gọi hệ thống trong chuỗi lời gọi hệ thống trước đó. Mô hình thu được một vectơ đầu ra có phân phối xác suất P_{h_i} (xác suất của sự kiện h_i là đại diện cho lời gọi hệ thống thứ *i* trong một chuỗi lời gọi hệ thống s_i (i = *2,3,..,L*) tính theo công thức 2.8 dưới đây:

$$P_{h_i} = \mathrm{p}(s_i|S_1^{i-1}) \quad (2.8)$$

Trong đó, $p(s_i|S_1^{i-1})$ là xác suất có điều kiện của s_i (sự kiện *i*) trước toàn bộ chuỗi các sự kiện trước đó S_1^{i-1} (từ sự kiện thứ 1 đến sự kiện thứ *i*-1). Xác suất xuất hiện lời gọi hệ thống hiện tại h_i phụ thuộc vào toàn bộ các lời gọi hệ thống trước đó trong chuỗi.

Để tính xác suất của cả chuỗi các sự kiện, chúng ta cần tính xác suất đồng thời của

tất cả các sự kiện xảy ra theo thứ tự. Theo lý thuyết xác suất, xác suất đồng thời của các sự kiện liên tiếp là tích của các xác suất có điều kiện của các sự kiện đó. Vì vậy, với một chuỗi lời gọi hệ thống có độ dài *L* thì xác suất để tạo ra chuỗi lời gọi hệ thống S_1^L (chuỗi các lời gọi hệ thống từ s_1 đến s_L) được tính theo công thức tích các xác suất có điều kiện và p(s_1) là xác suất của lời gọi hệ thống đầu tiên trong chuỗi, có thể được coi là đã biết hoặc không điều kiện (vì không có từ trước đó), nên công thức trên thường được đơn giản hóa như sau:

$$p(S_1^L) = \prod_{i=2}^{L} p(s_i \mid S_1^{i-1}) \quad (2.9)$$

Quá trình dự đoán chuỗi lời gọi hệ thống tiếp theo trong một chuỗi các lời gọi hệ thống dựa trên xác suất của từng lời gọi hệ thống $s_i, s_2, \ldots, s_n$, mô hình sẽ tính xác suất P_{h_i} cho mỗi lời gọi hệ thống sau đó lời gọi hệ thống có xác suất lớn nhất sẽ được chọn làm lời gọi hệ thống dự đoán. Quá trình có thể được mô tả qua công thức 2.10 dưới đây:

$$\text{Predicted\_syscall} = (s_i | P_i, 1 \le i \le n) \quad (2.10)$$

với $P_i = \max(P_{h_i})$ là xác suất lớn nhất của lời gọi hệ thống thứ *i* (h_i) trong chuỗi s_i, *n* là số lời gọi hệ thống khác nhau hoặc độ dài của chuỗi s_i.

Hàm mất mát (loss function) mô tả sự khác biệt giữa lời gọi hệ thống được dự đoán tiếp theo và lời gọi hệ thống đích trong chuỗi. Mục tiêu của huấn luyện mạng nơ-ron LSTM là tối thiểu hoá hàm mất mát này. Hàm mất mát thường được lựa chọn dựa trên bài toán cụ thể mà mạng nơ-ron LSTM giải quyết. Đối với bài toán phân lớp đầu ra, hàm mất mát thường được sử dụng là Cross-Entropy Loss (hay còn gọi là Log Loss). Cross-Entropy Loss đo lường sự khác biệt giữa phân phối xác suất dự đoán của mô hình và phân phối xác suất thực tế của các lớp. Vì vậy, trong xây dựng các mô hình dự đoán chuỗi MM và BM, luận án lựa chọn Cross-Entropy Loss để tính toán hàm mất mát. Hàm mất mát này càng giảm khi xác suất dự đoán của mô hình tiến gần hơn về phân phối xác suất thực tế. Công thức của Cross-Entropy Loss được luận án sử dụng biểu diễn như sau:

$$\text{Cross-Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{2} y_{ij}.log\ (p_{ij}) \quad (2.11)$$

Trong đó: *N* là số lượng chuỗi lời gọi hệ thống, y_{ij} là chỉ số hàm xác định đúng nhãn của mẫu *i* trong lớp *j* (nhận giá trị là 1 nếu đúng lớp, 0 nếu sai lớp), p_{ij} là xác suất dự đoán của mô hình cho mẫu *i* thuộc vào lớp *j*.

Bên cạnh đó, thuật toán lan truyền ngược backpropagation [23] được sử dụng để tính toán lỗi và cập nhật các trọng số của mạng. Sau nhiều lần lặp lại, trọng số hội tụ và giai đoạn huấn luyện được dừng lại. Thuật toán tối ưu hóa Adam Optimizer [22] được sử dụng tối ưu hoá mô hình học sâu LSTM. Adam kết hợp các ưu điểm của hai phương pháp tối ưu hóa phổ biến là AdaGrad và RMSProp. Adam sử dụng cả giá trị trung bình động của gradient (mức độ thay đổi của hàm mất mát) và giá trị trung bình động của

bình phương gradient để điều chỉnh tốc độ học. Vì vậy, Adam Optimizer có khả năng tự động điều chỉnh tốc độ học, giảm yêu cầu tinh chỉnh các siêu tham số khi huấn luyện mô hình học dựa trên mạng LSTM.

Quá trình xây dựng hai mô hình dự đoán chuỗi lời gọi hệ thống MM và BM dựa trên một kiến trúc mạng học sâu LSTM được thể hiện trong đoạn giả mã của thuật toán 2.2.

Thuật toán 2.2. Xây dựng mô hình dự đoán chuỗi lời gọi hệ thống dựa trên mạng nơ-ron LSTM.

Đầu vào: *Chuỗi tất cả các lời gọi của tập dữ liệu (X)*
Đầu ra*: Mô hình dự đoán chuỗi lời gọi hệ thống (model)*

1: Model ← Khởi tạo mô hình dự đoán dựa trên mạng học sâu LSTM
 # Khởi tạo hàm mất mát sử dụng cross entropy
2: Criterion ← Cross-Entropy Loss
 # Khởi tạo hàm tối ưu hoá đối với các tham số trong mô hình
3: Optimizer () ← Adam (Parameter of Model)
4: **For** $\boldsymbol{s_i}$ **in** X:
 # Đặt lại giá trị gradient của các tham số trong thuật toán tối ưu hoá
5: Optimizer zero gradient ()
6: P_{h_i} ← model ($\boldsymbol{s_i}$) = p($s_i|S_1^{i-1}$)
 # Giá trị dự đoán $P_i = max\ (P_{h_i})$
7: Predicted_syscall=($s_i|\boldsymbol{P_i}$,1≤i≤n)
8: Loss ← criterion (predicted_syscall, $\boldsymbol{s_i}$)
 # Lan truyền ngược thông qua hàm Loss Backward
9: Loss Backward ()
 # Cập nhật trọng số cho mô hình thông qua hàm Optimizer
10: Optimizer ()
11: **Endfor**
12: **Return** Model

2.2.4. Xây dựng bộ phân lớp chuỗi lời gọi hệ thống dựa trên các mô hình dự đoán chuỗi đã huấn luyện

Trong thu thập hành vi lời gọi hệ thống của các tập tin phụ thuộc phần lớn vào môi trường phân tích động tập tin. Với sự phát triển của trí tuệ nhân tạo và các công nghệ mới, đã có các nghiên cứu chỉ ra rằng mã độc có khả năng phát hiện môi trường phân tích động hoặc thay đổi hành vi khi lây nhiễm trên nhiều thiết bị [56][110]. Vì vậy, để có thể giúp đánh giá đầy đủ hơn về hành vi hoạt động của mỗi mẫu mã độc và kết hợp giá trị dự đoán từ các mô hình dự đoán chuỗi đã xây dựng từ các tập dữ liệu. Luận án lựa chọn phương pháp xây dựng các giá trị đại diện cho một tập tin cần thu thập chuỗi lời gọi hệ thống. Trong bài toán phát hiện mã độc IoT của luận án, sau giai đoạn huấn luyện hai mô hình dự đoán chuỗi lời gọi hệ thống (B-Model và M-Model), với một chuỗi lời gọi hệ thống cần xác định nhãn sinh ra từ tập tin (S_1^m), hai giá trị đại diện được xác định dựa trên xác suất trung bình tính toán từ mô hình dự đoán chuỗi với *N* lần thu thập chuỗi lời gọi hệ thống của một tập tin. Hai giá trị đại diện cho tập

tin lần lượt là representative _value_B-Model (S_1^m) và representative _value_M-Model (S_1^m) .

Đầu tiên, với mỗi mô hình dự đoán chuỗi đã được huấn luyện, thuật toán lan truyền ngược có thể tính toán xác suất p($s_i \mid S_1^{i-1}$) với i=2,3,…,L. Sau khi tính toán đối với toàn bộ một chuỗi lời gọi hệ thống, một xác suất xảy ra của toàn bộ chuỗi lời gọi hệ thống có độ dài *L* được tính theo công thức P(S_1^L). Cả hai mô hình M-Model và B-Model đều được sử dụng để tính toán xác suất cho một chuỗi lời gọi hệ thống cần xác định nhãn.

Sau đó, một giá trị đại diện được xác định để mô tả mức độ tương quan giữa các lần thu thập chuỗi lời gọi hệ thống từ một tập tin đối với từng mô hình M-Model và B-Model. Các tập tin thực thi được thực hiện thu thập chuỗi lời gọi hệ thống *N* lần. Giá trị đại diện được định nghĩa dựa trên giá trị trung bình của xác suất. Việc tính giá trị đại diện theo trung bình cộng xác suất sẽ gặp hạn chế khi áp dụng cho các xác suất rất nhỏ hoặc lớn vì nó có thể bị ảnh hưởng bởi các giá trị biên. Vì vậy, luận án lựa chọn tính giá trị đại diện theo trung bình hàm mũ xác suất để giúp tăng cường độ nhạy của phép đo trung bình, đặc biệt là đối với các xác suất nhỏ và giúp cho các giá trị này có ảnh hưởng ít hơn đến kết quả cuối cùng. Giá trị đại diện theo trung bình hàm mũ được tính theo công thức sau đây:

$$\text{Representative \_value\_(model)} = \exp\left(\frac{\sum_{i=1}^{N} log\ P_i(S_1^m)}{N}\right) \qquad (2.12)$$

Việc lặp lại *N* lần thu thập chuỗi lời gọi hệ thống sẽ giúp giải quyết các bài toán hiếm gặp trong thu thập chuỗi lời gọi hệ thống của mã độc như mã độc che giấu hành vi, mã độc phát hiện môi trường ảo hoá. Với các đặc điểm của mã độc trên các thiết bị IoT hạn chế về tài nguyên, thông thường ít sử dụng các kỹ thuật che giấu hành vi độc hại. Để giảm độ phức tạp trong thu thập chuỗi lời gọi hệ thống và xử lý tính toán, chúng ta có cũng thể xem xét giá trị *N* phù hợp. Trường trường hợp thử nghiệm trong luận án thấy rằng với đặc điểm mã độc trên thiết bị IoT sử dụng nền tảng kiến trúc MIPS thu thập được ít thay đổi hành vi hệ thống nên luận án lựa chọn N =1 để thực hiện các đánh giá. Trong một số trường hợp mã độc có sử dụng các kỹ thuật đa hình, siêu đa hình, thay đổi hành vi thì giá trị N cần được xem xét lựa chọn tăng số lần thu thập chuỗi để đảm bảo phát hiện tất cả các hành vi độc hại của tập tin mã độc IoT.

Cuối cùng, nhãn của tập tin được phân loại dựa trên so sánh hai giá trị *representative_value_M-Model* và *representative_value_B-Model*. Nếu giá trị *representative _value_M-Model* lớn hơn giá trị *representative _value_B-Model* thì tập tin sẽ được xác định là mã độc. Ngược lại, tập tin sẽ được xác định là lành tính.

Quá trình phân lớp chuỗi lời gọi hệ thống thu thập từ một tập tin thực thi dựa trên 2 mô hình dự đoán chuỗi sử dụng mạng nơ-ron LSTM được trình bày bằng giả mã như Thuật toán 2.3.

Thuật toán 2.3. Phát hiện mã độc IoT dựa trên hai mô hình dự đoán chuỗi lời gọi hệ thống sử dụng mạng nơ-ron LSTM.

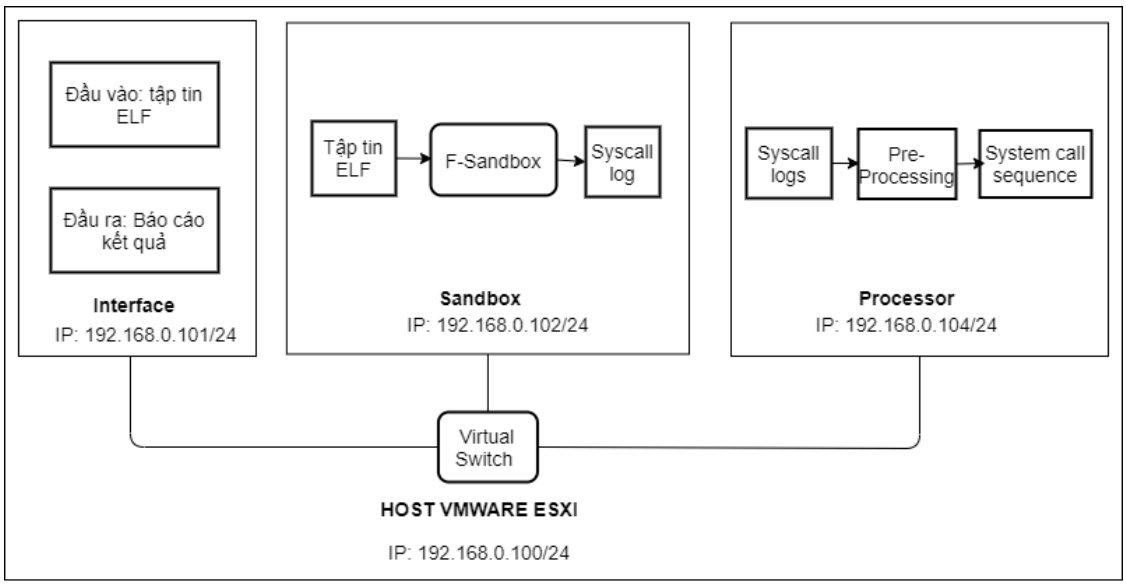
Đầu vào: *Chuỗi lời gọi hệ thống cần xác định nhãn S_1^L; 2 mô hình dự đoán chuỗi lời gọi hệ thống M-Model và B-Model.*
Đầu ra*: Nhãn của tập tin sinh ra chuỗi S_1^L.*

 # Tiền xử lý chuỗi lời gọi hệ thống S_1^L
1: Preprocess_syscall (S_1^L)
 # Xác định giá trị đại diện của tập tin dựa trên mô hình M-Model
2: p(S_1^L) ← M − Model (S_1^L)
3: Representative _value_mal ← exp($\frac{\sum_{i=1}^{N} log\ p_i}{N}$)
 # Xác định giá trị đại diện của tập tin dựa trên mô hình B-Model
4: p(S_1^L) ← B − Model (S_1^L)
5: Representative _value_beg ← exp($\frac{\sum_{i=1}^{N} log\ p_i}{N}$)
 # So sánh hai giá trị đại diện của tập tin để quyết định nhãn
6: **If** Representative _value_mal > Representative _value_beg **then**
7: $Malware \leftarrow Nhãn\ (S_1^L)$
8: **Else**
9: $Benign \leftarrow Nhãn\ (S_1^L)$
10: **Endif**
11: **Return** $Nhãn\ (S_1^L)$

2.3. Thực nghiệm và đánh giá mô hình phát hiện mã độc IoT đề xuất

2.3.1. Môi trường thực nghiệm

❖ *Xây dựng môi trường thu thập các tập dữ liệu chuỗi lời gọi hệ thống*: NCS tiến hành thử nghiệm thu thập chuỗi lời gọi hệ thống từ các tập tin ELF dựa trên F-Sandbox được triển khai trên môi trường ảo hoá sử dụng giải pháp Vmware ESXi với 3 máy tính ảo được cài đặt và kết nối theo mô hình dưới đây:



Hình 2.5. Môi trường thử nghiệm thu thập chuỗi lời gọi hệ thống.

Một máy chủ HP ProLiant DL 380p Gen8 với cấu hình Intel Xeon CPU E5-2620 v2, 12 CPUs x 2.095 GHz, RAM 2x32GB được ảo hoá sử dụng giải pháp Vmware ESXi v6.5.0. Sau đó, 03 máy ảo với tên gọi là Interface, Sandbox, Processor được tạo dựng trên môi trường này.

+ Máy Interface: Là máy tính thực hiện nhiệm vụ cung cấp giao diện giữa người dùng với hệ thống thu thập các chuỗi lời gọi hệ thống. Trên đó cài đặt hệ điều hành Ubuntu 18.04 LTS với các dịch vụ Apache Web server và dịch vụ SSH. Tập tin thực thi ELF sẽ được truyền tải lên máy chủ thông qua giao diện web được cấu hình trên máy tính này.

+ Máy Sandbox: Là máy tính cài đặt F-Sandbox để thu thập nhật ký các lời gọi hệ thống của một tập tin. Trên đó cài đặt hệ điều hành Ubuntu 18.04 LTS và các dịch vụ SSH Server với các kết nối mạng đến máy Interface và Processor.

+ Máy Processor: Là máy tính được cài đặt các công cụ xử lý các nhật ký lời gọi hệ thống để xây dựng các tập dữ liệu chuỗi lời gọi hệ thống. Máy Processor được cài đặt hệ điều hành Ubuntu 18.04 LTS và dịch vụ SSH Server để có thể truyền tải dữ liệu với các máy tính khác trong hệ thống.

❖ *Môi trường thử nghiệm huấn luyện và đánh giá mô hình phát hiện mã độc IoT dựa trên mạng nơ-ron LSTM và chuỗi lời gọi hệ thống*: Luận án tiến hành thử nghiệm dựa trên thư viện Tensorflow và sử dụng GPU trên Google Colaboratory để tăng tốc huấn luyện. Luận án thực nghiệm nhiều lần để lấy kết quả trung bình khi đánh giá các độ đo độ chính xác trên các ngưỡng độ dài chuỗi lời gọi hệ thống.

2.3.2. Xây dựng tập dữ liệu thử nghiệm

Thực nghiệm sử dụng tập dữ liệu tập tin thực thi trên môi trường IoT được xây dựng từ một bộ dữ liệu C500-IoT dataset công bố bởi Phú và các cộng sự [115]. Các mẫu mã độc là các tập tin được thu thập từ các nguồn khác nhau trên Internet gồm IoTPOT, VirusShare, VirusTotal, Detux và các chương trình có sẵn trên Embedded Linux. Đây cũng là các nguồn thu thập của các tập dữ liệu thử nghiệm về phát hiện mã độc IoT như [17], [31], [70], [85], [130]. Tập dữ liệu C500-IoT dataset đã được một số nhà nghiên cứu trong và ngoài nước sử dụng để đánh giá các phương pháp phát hiện mã độc IoT tại các nghiên cứu [31], [64], [85], [114]. Chi tiết tập dữ liệu C500-IoT dataset được mô tả trong Bảng 2.2.

Bảng 2.2. Thông tin thống kê bộ dữ liệu C500-IoT dataset

Nền tảng kiến trúc	VirusShare	IoTPot	Detux	Shared	Tổng số tập tin mã độc	Tổng số tập tin lành tính
MIPS	1.603	935	3.282	798	5.022	1.899
Intel	5.492	570	29	5	6.086	1.438
ARM	2.117	912	35	26	3.038	530
X86-64	586	320	11	3	914	180

PowerPC	699	353	12	4	1.060	60
Motorola	1.455	294	4	5	1.748	0
RenesasSH	646	310	2	3	955	0
SPARC	584	299	8	4	887	0

Để đánh giá hiệu quả của mô hình phát hiện mã độc IoT luận án đề xuất, NCS sử dụng một tập dữ liệu các tập tin thực thi trên kiến trúc MIPS từ bộ dữ liệu C500-IoT dataset. Tuy nhiên, trong tập dữ liệu này chỉ gồm 2.628 mẫu tập tin thực thi ELF đầy đủ thư viện đóng gói sẵn với 2.370 tập tin mã độc và 258 tập tin lành tính. Do đặc thù của các thiết bị sử dụng nền tảng kiến trúc MIPS, giới hạn về tài nguyên lưu trữ và xử lý nên số lượng tập tin lành tính khả năng thực thi độc lập với các hệ điều hành nhúng phục vụ còn hạn chế. Đây cũng là các hạn chế mà các nghiên cứu phát hiện mã độc IoT khác như [64],[114] gặp phải khi thu thập tập dữ liệu lành tính trong thực tế.

Vì vậy, luận án đã tiến hành thu thập bổ sung thêm vào tập dữ liệu thử nghiệm các chương trình lành tính từ các chương trình tiện ích của các nhà cung cấp phần mềm tin cậy và đã được đánh giá bởi công cụ quét mã độc trực tuyến Virus Total[8]. Tập dữ liệu tập tin thực trên môi trường IoT sử dụng nền tảng kiến trúc MIPS thử nghiệm trong luận án bao gồm 300 tập tin lành tính và 930 tập tin mã độc. Trong đó có 37 họ mã độc khác khau với nhiều họ mã độc IoT phổ biến như: LightAidra/ Aidra/ Zendran, Dofloo/ Spike/ MrBlack/ Wrkatk/ Sotdas/ AES.DDoS/ DnsAmp, Gafgyt/ BASHLITE/ Lizkebab/ Torlus, Tsunami/ Kaiten, SecurityRisk, Moose, Hajime, Trojan.Gen, vv… Việc lựa chọn tập dữ liệu cũng để đánh giá thêm khả năng phát hiện mã độc IoT của mô hình đã đề xuất khi có sự không cân bằng dữ liệu và hạn chế số lượng mẫu tập tin lành tính (số lượng mẫu lành tính chỉ chiếm gần 1/4 tổng số lượng tập mẫu thử nghiệm).

Việc trích xuất, thu thập chuỗi lời gọi hệ thống và xây dựng mô hình phát hiện mã độc IoT dựa trên mạng nơ-ron LSTM được áp dụng tương tự trên các kiến trúc vi xử lý khác trong bộ dữ liệu IoT. Vì vậy, việc thử nghiệm trên tập dữ liệu các tập tin nền tảng kiến trúc MIPS vẫn đảm bảo tính tổng quát trong đánh giá mô hình phát hiện mã độc IoT.

2.3.3. Xây dựng các tập chuỗi lời gọi hệ thống

Bước 1: *Thu thập các tập tin nhật ký lời gọi hệ thống của tập tin thực thi trong môi trường F-Sandbox.*

Các tập tin thực thi ELF được thực thi lần lượt trong môi trường F-Sandbox để thu thập các tập tin nhất ký chứa thông tin về các lời gọi hệ thống tương ứng với từng tập tin thực thi và nhãn ban đầu. Qua quá trình quan sát việc thu thập lời gọi hệ thống các tập tin thử nghiệm trên kiến trúc MIPS, thời gian thực thi tập tin càng nhiều thì số lượng

[8] https://www.virustotal.com/gui/

các chuỗi lời gọi hệ thống thu thập được từ một tập tin càng lớn. Bên cạnh đó, với các tập tin thử nghiệm, sau khoảng thời gian 30 giây, các lời gọi hệ thống thu được tăng lên chậm lại. Vì vậy, trong thử nghiệm NCS tiến hành cấu hình F-Sandbox thu thập nhật ký lời gọi hệ thống từ các tập tin trong thời gian 30 giây, đây cũng là khoảng thời gian tương tự các nghiên cứu khác đã thu thập đối với thực thi các tập tin ELF kiến trúc MIPS [114]. Kết quả thu thập một nhật ký lời gọi hệ thống từ một tập tin ELF trên nền tảng MIPS thể hiện trong hình 2.12.

```
execve("./nmap", ["./nmap", "-sS", "192.168.2.158"], 0x7fb3fc14 /* 13 vars */) = 0
brk(NULL)                               = 0xaf2000
brk(0xaf2480)                           = 0xaf2480
set_thread_area(0xaf9460)               = 0
brk(0xaf2910)                           = 0xaf2910
set_thread_area(0xaf98e0)               = 0
open("/dev/urandom", O_RDONLY)          = 3
read(3, "\350]p\341", 4)                = 4
close(3)                                = 0
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
brk(0xaf3910)                           = 0xaf3910
brk(0xaf4000)                           = 0xaf4000
geteuid()                               = 0
gettimeofday({tv_sec=1537336184, tv_usec=649662}, NULL) = 0
getpid()                                = 2457
open("/dev/urandom", O_RDONLY|O_LARGEFILE) = 3
read(3, "\277\217\3168^6\207\300\3431B\356Z\1\366\276\365\376\333\36\257B\17\22}\33L\0054\16|\204"..., 244) = 2
close(3)                                = 0
gettimeofday({tv_sec=1537336184, tv_usec=653290}, NULL) = 0
time(NULL)                              = 1537336184 (2018-09-19T05:49:44+0000)
open("/etc/TZ", O_RDONLY)               = -1 ENOENT (No such file or directory)
open("/etc/localtime", O_RDONLY)        = 3
read(3, "TZif2\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\1\0\0\0\1\0\0\0\0"..., 68) = 68
lseek(3, -68, SEEK_END)                 = 50
read(3, "UTC\0\0\0TZif2\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\1\0\0"..., 68) = 68
close(3)                                = 0
rt_sigaction(SIGINT, {sa_handler=0x428c04, sa_mask=~[HUP RT_67 RT_68 RT_69 RT_70 RT_71 RT_72 RT_73 RT_74 RT_75
rt_sigaction(SIGTERM, {sa_handler=0x428c04, sa_mask=~[HUP RT_67 RT_68 RT_69 RT_70 RT_71 RT_72 RT_73 RT_74 RT_75
rt_sigaction(SIGQUIT, {sa_handler=0x428c04, sa_mask=~[HUP RT_67 RT_68 RT_69 RT_70 RT_71 RT_72 RT_73 RT_74 RT_75
open("/dev/tty", O_RDONLY|O_NONBLOCK|O_LARGEFILE) = 3
ioctl(3, TIOCGPGRP, 0x7ffd7764)         = 0
```

Hình 2.6. Một tập tin nhật ký lời gọi hệ thống ghi nhận được từ F-Sandbox.

Bước 2: Thu thập tên của các lời gọi hệ thống trong các tập tin nhật ký sinh ra bởi một tập tin thực thi.

Một tập tin thực thi trong môi trường F-Sandbox có thể tạo ra nhiều tiến trình con, mỗi tiến trình con tạo ra một tập tin nhật ký lời gọi hệ thống. Do vậy, việc thu thập chuỗi lời gọi hệ thống để mô tả hành vi của tập tin thực thi cần đảm bảo thứ tự của các tiến trình con được tạo ra. Bên cạnh đó, cần xây dựng chương trình để trích xuất ra chuỗi tên lời gọi hệ thống từ các tập tin nhật ký. Luận án tiếp cận theo phương pháp xây dựng "từ điển lời gọi hệ thống" để trích xuất, thu thập chuỗi lời gọi hệ thống. Đối với kiến trúc MIPS sử dụng hệ điều hành Linux nhúng, luận án khảo sát, thu thập được tổng số 345 lời gọi hệ thống khác nhau có thể xuất hiện trong chuỗi lời gọi hệ thống. Từ đó, luận án xây dựng chương trình thu thập chuỗi lời gọi hệ thống từ tên của các lời gọi hệ thống trong các tập tin nhật ký sinh ra theo thứ tự tiến trình con tạo ra.

Kết quả thu thập chuỗi lời gọi hệ thống từ các tập tin ELF đã được gán nhãn thể hiện như thống kê trong Bảng 2.3.

Bảng 2.3. Kết quả thu thập chuỗi lời gọi hệ thống từ các tập tin.

Nhãn	Mã độc	Lành tính
Số lượng tập tin ELF	930	300
Số lượng tập tin thu thập chuỗi lời gọi hệ thống thành công	928	300
Độ dài trung bình của chuỗi	327	305

Đối với tập dữ liệu thử nghiệm, độ dài của các chuỗi lời gọi hệ thống trên cả hai tập mã độc và lành tính phổ biến trong khoảng trên 300 lời gọi hệ thống. Các chuỗi lời gọi hệ thống sau khi thu thập được phân chia thành các tập khác nhau theo độ dài tối thiểu của các lời gọi hệ thống để đưa vào huấn luyện nhằm xác định ngưỡng thu thập phù hợp nhất cho phát hiện mã độc IoT trên nền tảng kiến trúc MIPS. Dựa trên các ngưỡng trong các nghiên cứu khác như [64], [114],… đã công bố trước đây, để có cơ sở so sánh, đánh giá hiệu quả của mô hình đối với các ngưỡng độ dài chuỗi lời gọi hệ thống mà không mất đi tính tổng quát, luận án đề xuất lựa chọn phân chia tập dữ liệu thành các ngưỡng tương đồng với các nghiên cứu này phục vụ thử nghiệm. Số lượng các chuỗi lời gọi hệ thống phân chia theo độ dài tối thiểu được thể hiện trong Bảng 2.4.

Bảng 2.4. Kết quả các tập chuỗi lời gọi hệ thống theo các ngưỡng độ dài chuỗi.

Độ dài chuỗi lời gọi hệ thống tối thiểu	**50**	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**	**1000**
Số lượng mẫu mã độc	928	927	905	866	861	860	621	594	108	67	45
Số lượng mẫu lành tính	300	300	282	264	193	164	157	149	136	127	88

Qua phân tích dữ liệu thu được, NCS có một số đánh giá như sau:

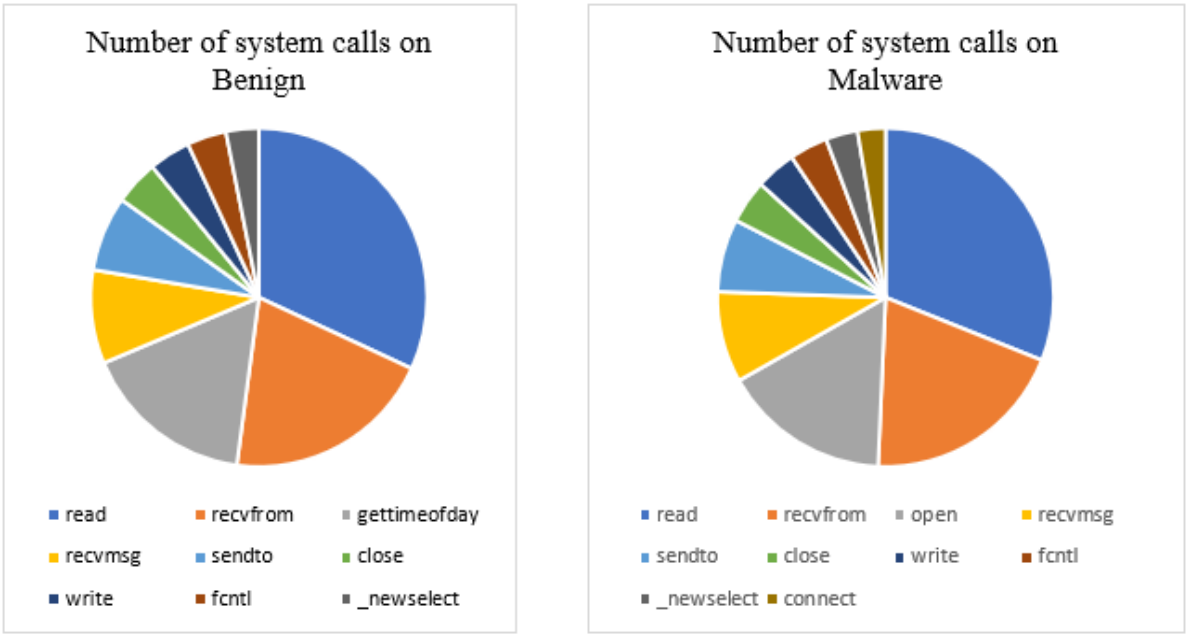
- Có một số tập tin mã độc (2/930 chiếm tỉ lệ 2% tổng số lượng mẫu thử nghiệm) không thu được chuỗi lời gọi hệ thống do F-Sandbox chưa đảm bảo đủ điều kiện kích hoạt của các tập tin mã độc này. Độ dài trung bình của tập chuỗi lời gọi hệ thống thu được từ tập tin mã độc không có sự khác biệt nhiều đối với độ dài trung bình của tập chuỗi lời gọi hệ thống thu được từ tập tin lành tính. Vì vậy, việc phân lớp tập tin chỉ dựa trên độ dài chuỗi lời gọi hệ thống thu thập được là không khả thi.

- Đa số các tập tin có chuỗi lời gọi hệ thống nằm trong khoảng 300 đến 500, miền giá trị độ dài chuỗi lời gọi hệ thống ngắn có thể được lý giải do các chương trình viết trên IoT thường đơn giản hơn trên máy tính truyền thống do hạn chế về tài nguyên và các chức năng. Độ dài tối đa của một câu trong ngôn ngữ tự nhiên thường ngắn hơn độ dài của chuỗi lời gọi hệ thống. Vì vậy, việc sử dụng mạng nơ-ron LSTM cho trường hợp

chuỗi lời gọi hệ thống cần có sự khác biệt khi áp dụng với ngôn ngữ tự nhiên để có thể đem lại hiệu quả tốt.

- Số lượng các lời gọi hệ thống khác nhau trên nền tảng MIPS chỉ có 345 lời gọi hệ thống, điều này là khác biệt với số lượng từ trong ngôn ngữ tự nhiên. Phân tích nhật ký lời gọi hệ thống thu thập được, tập tin mã độc chỉ sử dụng 136 lời gọi hệ thống khác nhau, tập tin lành tính sử dụng 127 lời gọi hệ thống khác nhau và đa số các lời gọi hệ thống sử dụng trong cả hai tập trùng nhau. Trong bộ dữ liệu thực nghiệm, chỉ có 160 lời gọi hệ thống xuất hiện trong cả hai tập phần mềm độc hại và lành tính, trong khi số lượng từ khác nhau trong ngôn ngữ tự nhiên là hơn 10.000 từ. Các lời gọi hệ thống xuất hiện nhiều nhất trên cả hai bộ dữ liệu mã độc và mã sạch là tương đối giống nhau thể hiện trong hình 2.13, đặc biệt là hai lời gọi hệ thống là "read" và "recvfrom" xuất hiện nhiều nhất trong cả hai bộ dữ liệu. Vì vậy, việc sử dụng từng lời gọi hệ thống riêng lẻ để phân loại chuỗi lời gọi hệ thống trong trường hợp này là không khả thi. Cần nghiên cứu các phương pháp phân tích, đánh giá các đặc trưng khác của chuỗi lời gọi hệ thống.

Tập dữ liệu các chuỗi lời gọi hệ thống theo ngưỡng độ dài phục vụ quá trình huấn luyện và kiểm tra được phân chia ngẫu nhiên theo tỉ lệ 80/20 để tiến hành các thử nghiệm đánh giá hiệu quả của mô hình phát hiện.



Hình 2.7. Tỷ lệ các lời gọi hệ thống xuất hiện nhiều nhất.

2.3.4. Kết quả xây dựng mô hình phát hiện mã độc IoT

Luận án tiến hành thử nghiệm mô hình phát hiện mã độc IoT dựa trên cùng một trên kiến trúc mạng nơ-ron LSTM cho hai tập dữ liệu đã được gán nhãn nhưng khác nhau về các tham số. Kiến trúc mạng nơ-ron LSTM được sử dụng để huấn luyện hai mô hình dự đoán chuỗi MM và BM có các giá trị thông số và hàm như Bảng 2.5.

Bảng 2.5. Các thông số chính sử dụng trong kiến trúc mạng nơ-ron LSTM để huấn luyện các mô hình dự đoán chuỗi lời gọi hệ thống.

Các thông số/ hàm trong mạng nơ-ron LSTM	**Giá trị sử dụng**	**Ý nghĩa tham số**
Mô hình (model)	Sequential()	Mô hình được xây dựng theo cách tuần tự, các lớp được thêm vào theo thứ tự từ đầu đến cuối
Các lớp LSTM	model.add()	Các lớp của mạng LSTM sử dụng: các lớp đầu tiên có return_sequences=True và 1 lớp cuối cùng return_sequences=False
Đầu vào cho lớp đầu tiên	SYSCALL_LEN, len(vocab)+1	SYSCALL_LEN là hiểu dài của chuỗi đầu vào và len(vocab)+1 là kích thước của mỗi vector đầu vào
Số đơn vị ẩn (units)	number	Số chiều của trạng thái ẩn
Các lớp Dense	units=len(vocab)+1	Số lượng lớp đầu ra
Hàm kích hoạt (activation)	softmax	Hàm softmax được sử dụng để chuyển đầu ra thành xác suất cho từng lớp
Hàm mất mát (loss)	categorical_crossentropy	Hàm mất mát categorical_crossentropy được sử dụng
Bộ tối ưu hoá (optimizer)	adam	Sử dụng thuật toán tối ưu hóa adam

Các thử nghiệm được thực hiện nhiều lần, mỗi lần thực hiện 30 epoch để điều chỉnh các tham số chính của các mạng nơ-ron LSTM nhằm lựa chọn các giá trị tham số phù hợp như trong Bảng 2.6.

Bảng 2.6. Các tham số chính của các mạng nơ-ron LSTM sử dụng để xây dựng các mô hình dự đoán chuỗi lời gọi hệ thống.

Các tham số	**Ý nghĩa tham số**	**Giá trị sử dụng trong BM**	**Giá trị sử dụng trong MM**
test_size	Tỷ lệ tập mẫu sử dụng cho tập kiểm tra	0.3	0.3
random_state	Khởi tạo quá trình lấy mẫu ngẫu nhiên	2020	2020
batch_size	Số lượng mẫu dữ liệu được sử dụng trong mỗi lần cập nhật trọng số của mô hình	64	128
epochs	Số chu kỳ huấn luyện đầy đủ trên toàn bộ tập dữ liệu	500	200

Các thực nghiệm cũng được thực hiện nhiều lần để đánh giá hiệu suất của mô hình dựa trên độ dài của chuỗi lời gọi hệ thống, số lượng lớp ẩn và số lượng đơn vị ẩn trong

mỗi lớp. Từ đó, xác định độ dài chuỗi lời gọi hệ thống, số lớp ẩn và số đơn vị ẩn trong mỗi lớp cho hiệu quả đối với tập dữ liệu thử nghiệm trên kiến trúc MIPS.

Để đánh giá tính hiệu quả của mô hình đề xuất, các phương pháp đánh giá dựa trên các độ đo như Accuracy, F1-Micro và F1- Weight được sử dụng. Dựa trên những giá trị này, việc đánh giá độ chính xác của mô hình phát hiện sẽ khắc phục được hạn chế về độ tin cậy khi đánh giá mô hình có sự chênh lệch số lượng tập mẫu mã độc và lành tính. Những số liệu đánh giá này được áp dụng thường xuyên trong cộng đồng nghiên cứu để cung cấp các đánh giá toàn diện về các vấn đề mất cân bằng số lượng các mẫu trong quá trình huấn luyện mô hình.

- Kết quả đánh giá hiệu suất của mô hình dựa trên độ dài của chuỗi lời gọi hệ thống:

Trong mô hình luận án đề xuất, mạng nơ-ron LSTM được khởi tạo ban đầu với 4 lớp ẩn và 1000 đơn vị (unit) trong mỗi lớp. Có nhiều lớp ẩn và số đơn vị trong mỗi lớp có thể giúp mô hình học được các biểu diễn và mô hình hoá các mối quan hệ phức tạp trong dữ liệu, từ đó tăng khả năng dự đoán, có khả năng biểu diễn dữ liệu chi tiết hơn, phức tạp hơn. Tuy nhiên, tăng số lớp ẩn có thể làm tăng nguy cơ overfiting, tăng độ phức tạp tính toán, thời gian huấn luyện. Vì vậy, dựa trên các khảo sát và thực nghiệm của NCS, 4 lớp ẩn (3 lớp *sigmoid* và 1 lớp *tanh*) và 1000 unit thường được sử dụng trong mạng LSTM để huấn luyện các mô hình dự đoán chuỗi. Để xác định ngưỡng độ dài chuỗi lời gọi hệ thống cần thu thập của một tập tin, mô hình phát hiện mã độc IoT được đánh giá với các tập dữ liệu chuỗi lời gọi hệ thống có ngưỡng độ dài khác nhau trong các kịch bản thử nghiệm lần lượt là 50, 100, 150, 200, 250, 300, 350, 400, 450, 500. Kết quả thực nghiệm được thể hiện trong Bảng 2.7 là các độ đo trung bình của 10 lần thử nghiệm trên cùng môi trường và tại các thời gian khác nhau.

Bảng 2.7. Kết quả đánh giá mô hình phát hiện mã độc IoT theo ngưỡng độ dài của chuỗi lời gọi hệ thống.

Độ dài của chuỗi lời gọi hệ thống	**50**	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
Accuracy	95,83	96,47	98,37	89,67	88,86	88,59	87,5	85,87	85,67	85,59
F1-Macro	94,94	95,37	97,81	86,23	82,23	81,06	79,02	75,10	75,09	75,07
F1-Weight	95,79	96,53	98,38	89,79	87,74	87,22	85,9	83,56	83,36	83,31

Bảng 2.7 chỉ ra rằng khi ngưỡng độ dài chuỗi lời gọi hệ thống là 150, kết quả phát hiện mã độc có thể đạt độ chính xác cao nhất với accuracy là 98,37%, F1-Macro là 97,81% và F1-Weight là 98,38%. Với kết quả đánh giá các độ đo độ chính xác với các kịch bản thử nghiệm đối với các tập dữ liệu sử dụng ngưỡng độ dài chuỗi lời gọi hệ thống trên cũng có thể được lý giải qua việc mạng nơ-ron LSTM không thể ghi nhớ thêm thông tin và bị nhiễu khi chuỗi lời gọi hệ thống sử dụng quá dài, như trong thử

nghiệm ngưỡng độ dài của chuỗi lời gọi hệ thống lớn hơn 200. Khi sử dụng chuỗi lời gọi hệ thống quá ngắn, mạng nơ-ron sẽ chưa học đủ thông tin cần thiết để phân loại mã độc như khi ngưỡng độ dài chuỗi lời gọi hệ thống nhỏ hơn 100. Vì vậy, với kịch bản thử nghiệm ngưỡng độ dài của chuỗi lời gọi hệ thống đang cho kết quả các độ đo độ chính xác tốt nhất trong các kịch bản thử nghiệm.

❖ *Kết quả đánh giá hiệu suất của mô hình dựa trên số lớp ẩn và số đơn vị ẩn trong mỗi lớp:*

Hiệu quả của mô hình phát hiện được thử nghiệm lần lượt theo số lớp ẩn trong mạng nơ-ron từ 1 đến 6 để đánh giá các độ đo độ chính xác. Mỗi lớp của tất cả các mạng có 1000 đơn vị ẩn và độ dài của chuỗi lời gọi hệ thống là 150. Kết quả đánh giá được thể hiện trong Bảng 2.8.

Bảng 2.8. Đánh giá theo số lớp ẩn.

Số lớp ẩn	**1**	**2**	**3**	**4**	**5**	**6**
Accuracy	90,49	93,75	96,2	98,37	96,47	95,92
F1-Macro	88,44	91,81	94,73	97,81	95,20	94,74
F1-Weight	90,95	93,87	96,17	98,38	96,47	96,03

Bảng 2.8 cho thấy mô hình phát hiện cho các độ đo hiệu quả nhất với Accuracy là 98,37%, F1-Macro là 97,81 và F1-Weight là 98,38% đạt được khi sử dụng 4 lớp ẩn. Khi số lượng các lớp ẩn là một, mạng nơ-ron LSTM không thể nắm bắt đủ thông tin hữu ích từ chuỗi lời gọi hệ thống để dự đoán chính xác chuỗi. Trong khi số lớp ẩn lớn hơn 4, mạng nơ-ron LSTM bị overfiting do mô hình quá phức tạp (nhiều lớp). Như vậy mạng nơ-ron LSTM sử dụng 4 lớp ẩn là phù hợp nhất trong mô hình phát hiện mã độc trên tập dữ liệu thử nghiệm.

Bên cạnh đó, để xác định số đơn vị trong mỗi lớp hiệu quả, mô hình phát hiện được thử nhiệm với số lớp ẩn là 4, độ dài chuỗi lời gọi hệ thống là 150 và thay đổi số lượng nơ-ron (units) trong lớp ẩn. Kết quả đánh giá thể hiện như Bảng 2.9.

Bảng 2.9. Đánh giá theo số lượng nơ-ron trong lớp ẩn.

Số lượng đơn vị	**500**	**1000**	**1500**	**2000**
Accuracy	95,88	98,37	96,19	95,01
F1-Macro	94,69	97,81	95,02	94,02
F1-Weight	95,99	98,38	96,26	95,12

Bảng 2.9 cho thấy khi số lượng nơ-ron trong lớp ẩn là 1000, mô hình có các độ đo hiệu quả nhất với Accuracy là 98,37%, F1-Macro là 97,81% và F1-Weight là 98,38%. Điều này có thể lý giải bởi khi số lượng nơ-ron trong lớp ẩn nhỏ hơn 1000, mô hình không hiểu đầy đủ thông tin của các chuỗi để phân loại hiệu quả. Khi số lớn hơn 1000, mô hình bị overfiting và ảnh hưởng đến kết quả phân loại tập dữ liệu kiểm tra.

Như vậy, từ các kết quả thử nghiệm, luận án thấy rằng mô hình phát hiện sử dụng mạng LSTM với 4 lớp ẩn, 1000 unit và chuỗi lời gọi hệ thống có ngưỡng độ dài là 150 khi xây dựng mô hình MM và BM sẽ cho kết quả phát hiện mã độc IoT trên kiến trúc MIPS hiệu quả về các độ đo độ chính xác.

Bên cạnh đó, trong môi trường thử nghiệm của luận án, tốc độ phát hiện trung bình cho các chuỗi lời gọi hệ thống trong tập kiểm tra là 2,38 chuỗi lời gọi hệ thống/s (tương ứng thời gian phát hiện mã độc trung bình cho một tập tin chuỗi lời gọi hệ thống là 0,42s). Thời gian phát hiện trên đối với mô hình phát hiện dựa trên mạng học sâu là phù hợp khi đánh giá với các nghiên cứu khác trong nội dung 1.3.1 của luận án.

2.3.5. So sánh hiệu quả của mô hình đề xuất với các mô hình khác có liên quan

❖ *So sánh hiệu quả mô hình đề xuất với mô hình phát hiện mã độc IoT sử dụng một mạng nơ-ron LSTM duy nhất:*

Việc sử dụng một mạng nơ-ron LSTM để thực hiện phân lớp tập tin đã đem lại hiệu quả trong các nghiên cứu

Để thực hiện so sánh, mô hình phát hiện mã độc IoT trên kiến trúc MIPS sử dụng đặc trưng chuỗi lời gọi hệ thống được huấn luyện dựa trên một mạng nơ-ron LSTM như trong các nghiên cứu [98], [121] đã phân tích trong phần mở đầu được thử nghiệm trên cùng một bộ dữ liệu, cùng thiết bị và môi trường thực nghiệm như mô hình NCS đã sử dụng. Kết quả so sánh được thể hiện trong Bảng 2.10.

Bảng 2.10. So sánh mô hình đề xuất với mô hình phân lớp nhị phân sử dụng 1 mạng nơ-ron LSTM duy nhất.

Mô hình phát hiện / **Ngưỡng độ dài chuỗi lời gọi hệ thống**	**Mô hình phân lớp nhị phân trên một mạng LSTM duy nhất**			**Mô hình luận án đề xuất**		
	Accuracy	**F1-Macro**	**F1-Weight**	**Accuracy**	**F1-Macro**	**F1-Weight**
50	73,87	60,13	60,29	95,83	94,94	95,79
100	73,89	60,16	60,31	96,47	95,37	96,53
150	73,92	60,23	61,02	**98,37**	**97,81**	**98,38**
200	73,94	60,31	61,09	89,67	86,23	89,79
250	73,98	60,69	61,45	88,86	82,02	87,74
300	74,35	62,16	62,86	88,59	81,06	87,22
350	74,88	62,79	63,01	87,5	79,02	85,9
400	76,36	66,22	67,48	85,87	75,10	83,56
450	78,54	69,37	70,68	85,67	75,09	83,36
500	**80,16**	**72,01**	**71,42**	85,59	75,07	83,31

Kết quả đánh giá các độ đo trong Bảng 2.10 cho thấy rằng mô hình phát hiện mã độc IoT sử dụng một mạng nơ-ron LSTM duy nhất có xu hướng phát hiện chính xác khi chuỗi lời gọi hệ thống sử dụng có độ dài tăng dần. Điều này phù hợp với đặc điểm các

tập tin mã độc trong tập dữ liệu thường là các chương trình lành tính bị mã độc hại lây nhiễm thêm vào. Do đó, mô hình cần chuỗi lời gọi hệ thống đủ lớn để mô hình học được sự khác biệt giữa chuỗi lời gọi hệ thống thu thập từ tập mã độc và tập lành tính. Điều này đã được minh chứng trong các thử nghiệm của các nghiên cứu đã khảo sát trong nội dung mở đầu với độ dài chuỗi lời gọi hệ thống rất lớn và số lượng dữ liệu các tập cần có sự cân bằng cao để đem lại hiệu quả. Các mô hình phát hiện đề xuất dựa trên việc huấn luyện mạng nơ-ron LTSM cho từng tập dữ liệu đã chứng minh được khả năng học hiệu quả đối từng đặc trưng khác biệt của dữ liệu chuỗi mã thực thi thu thập từ tập tin lành tính và tập tin mã độc.

❖ *So sánh với các mô hình phát hiện mã độc IoT sử dụng học máy và phương trích chọn đặc trưng n-gram*

Để thực hiện so sánh, ba mô hình phát hiện sử dụng học máy và phương pháp trích chọn đặc trưng n-gram (thử nghiệm với 2-gram) sử dụng trong các nghiên cứu phát hiện mã độc IoT như [39], [71], [109], [115] được tiến hành thực nghiệm để đánh giá. Ba mô hình so sánh là 3 mô hình phát hiện mã độc IoT trên kiến trúc MIPS điển hình sử dụng chuỗi lời gọi hệ thống được sử dụng trong luận án của Phú [114]. Các đánh giá, so sánh hiệu quả các mô hình được tiến hành thử nghiệm trên cùng một bộ dữ liệu, cùng thiết bị và môi trường thực nghiệm. Kết quả so sánh được thể hiện trong Bảng 2.11.

Bảng 2.11. Kết quả so sánh mô hình đề xuất với 3 mô hình học máy khác.

Ngưỡng độ dài chuỗi lời gọi hệ thống	**F1-Macro**				**F1-Weight**			
	RF [114]	**SVM [114]**	**NB [114]**	**Mô hình đề xuất**	**RF [114]**	**SVM [114]**	**NB [114]**	**Mô hình đề xuất**
50	85,03	79,81	67,21	95,34	91,48	89,13	66,56	95,49
100	85,60	84,65	69,01	95,37	91,73	90,82	67,35	96,53
150	84,95	82,67	70,19	**97,81**	92,29	90,83	68,76	**98,38**
200	85,48	82,54	77,79	86,23	92,45	91,02	83,38	89,79
250	85,18	82,43	77,89	82,02	92,48	91,04	83,38	87,74
300	85,13	80,88	77,81	81,06	92,61	90,73	83,51	87,22
400	91,82	88,09	78,84	75,10	97,04	95,61	93,00	83,56
500	**92,11**	**90,64**	**79,49**	75,07	**97,09**	**96,34**	**93,46**	83,31

Từ kết quả trên chúng ta thấy rằng khi độ dài của chuỗi lời gọi hệ thống là 500, ba mô hình phát hiện mã độc IoT trên kiến trúc MIPS sử dụng phương pháp trích chọn n-gram cho kết quả độ đo F1-Macro và F1-Weight cao nhất. Điều này cũng thể hiện được tính chất của các mô hình học máy truyền thống là việc hạn chế trong khả năng tự động học các biểu diễn phức tạp từ dữ liệu. Do đó, các mô hình học máy truyền thống cần sử dụng chuỗi lời gọi hệ thống dài để có đủ thông tin và thể hiện được toàn bộ hành vi của chương trình giúp mô hình phân biệt chính xác các đặc trưng chuỗi đã trích xuất. Mô hình phát hiện mã độc IoT trên kiến trúc MIPS mà luận án đề xuất hiệu quả hơn về các độ đo

độ chính xác do mạng nơ-ron đã khắc phục được hạn chế trong việc học các biểu biểu và mô hình đã có đủ thông tin quan trọng để biểu diễn đặc trưng khi chỉ cần sử độ dài chuỗi lời gọi hệ thống ngắn. Các thực nghiệm cũng cho thấy rằng mã độc IoT trên tài nguyên hạn chế có nhiều điểm khác biệt so với mã độc trên thiết bị máy tính truyền thống khi đặc trưng của hành vi độc hại được F-sandbox ghi nhận từ các lời gọi hệ thống đầu tiên của chuỗi và mô hình đề xuất cho độ chính xác tốt khi chuỗi ngắn, giảm dần khi chuỗi dài do chứa các đặc trưng lời gọi hệ thống gây nhiễu hoặc tương đồng giữa hai tập.

❖ *Đánh giá hiệu quả của mô hình đề xuất với các mô hình phát hiện sử dụng chuỗi lời gọi hệ thống ngắn khác có liên quan:*

Nhằm đánh giá hiệu quả của mô hình sử dụng độ dài chuỗi lời gọi hệ thống ngắn và vẫn đảm bảo độ chính xác tương đồng các mô hình phát hiện có liên quan khác. Luận án phân tích, so sánh mô hình đề xuất với các mô hình khác có liên quan sử dụng đặc trưng lời gọi hệ thống thu thập từ phân tích động tập tin. Mô hình đề xuất chỉ cần sử dụng duy nhất một loại đặc trưng thu thập từ quá trình phân tích động tập tin là chuỗi lời gọi hệ thống với độ dài chuỗi cần thu thập ngắn là 150 đã có thể cho độ chính xác tốt tương đồng với một số nghiên cứu khác luận án đã khảo sát. Đánh giá, so sánh hiệu quả với các nghiên cứu khác có liên quan được thể hiện trong Bảng 2.12. Kết quả cho thấy mô hình đã khai thác tốt đặc trưng tuần tự của từng tập dữ liệu chuỗi lời gọi hệ thống thu thập từ tập tin mã độc và tập tin lành tính. Với chuỗi lời gọi hệ thống ngắn cần thu thập, mô hình có khả năng phân lớp tập tin với ít thời gian thu thập chuỗi lời gọi hệ thống trong môi trường phân tích động hơn trong thực tế. Do đó, mô hình đề xuất có thể được sử dụng để tích hợp mô hình phát hiện vào các giải pháp phát hiện mã độc cần ít thông tin và thời gian thu thập từ thực thi tập tin nhưng vẫn đảm bảo độ chính xác cao trong các hệ thống bảo mật thực tế.

Bảng 2.12. So sánh mô hình đề xuất với các nghiên cứu liên quan.

Tác giả	Đặc trưng sử dụng	Mô hình/phương pháp	Độ dài chuỗi hiệu quả	Accuracy (%)
Maniath [100]	Lời gọi hệ thống	Mạng LSTM	779.980	96,67
Pascanu [95]	Lời gọi hệ thống	Bagging (RNN, ESN)	3.000	95
Hansen [101]	Lời gọi hệ thống	RF	596	98,13
Phú [114]	Lời gọi hệ thống	RF, 2-gram, PCA	300	95,3
Việt [64]	Lời gọi hệ thống	SVM	300	98,26
Mô hình đề xuất	Lời gọi hệ thống	Bagging (two LSTM)	150	98,37

2.4. Kết luận chương 2

Trong chương này đã đề xuất mô hình phát hiện mã độc IoT dựa trên phương pháp Bagging sử dụng cùng một kiến trúc mạng nơ-ron LSTM cho từng tập dữ liệu chuỗi lời gọi hệ thống trích xuất thông qua phân tích động đã được gán nhãn. Mô hình đề xuất sử

dụng chuỗi lời gọi hệ thống ngắn hơn so với các mô hình phát hiện mã độc IoT khác đã khảo sát nhưng vẫn đạt hiệu quả cao về tiêu chí độ chính xác, ít bị tác động bởi vấn đề mất cân bằng dữ liệu huấn luyện giữa các nhãn trong phát hiện mã độc IoT trên kiến trúc MIPS. Do đó, mô hình có thể được huấn luyện, tạo mô hình phát hiện để sử dụng tích hợp trong các hệ thống phát hiện và cảnh báo mã độc IoT trong thực tế.

Tuy nhiên, trong các thử nghiệm của luận án đã chứng minh phương pháp phân tích động chưa thu thập được toàn bộ đặc trưng lời gọi hệ thống của tất cả các tập tin khi thực thi, một số tập tin độc hại chưa thực thi trong môi trường F-sandbox. Việc sử dụng chuỗi lời gọi hệ thống ngắn đã đem lại hiệu quả trong các thử nghiệm nhưng có thể gặp khó khăn khi phát hiện các mã độc sử dụng các kỹ thuật che giấu hành vi, chưa bộc lộ hành vi ngay sau khi thực thi tập tin. Vì vậy, để xây dựng giải pháp phát hiện mã độc IoT tổng thể theo quy trình phân tích mã độc, luận án sẽ nghiên cứu đề xuất mô hình phát hiện mã độc IoT hiệu quả dựa trên học máy sử dụng đặc trưng thu thập từ phân tích tĩnh tập tin. Chi tiết về mô hình phát hiện mã độc IoT đề xuất sẽ được trình bày và thảo luận trong nội dung tiếp theo của luận án.

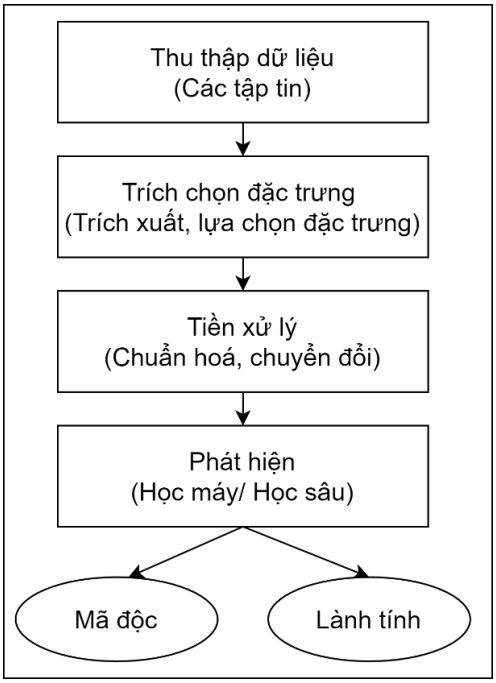
Ý tưởng và kết quả thực nghiệm của mô hình đề xuất trong chương này đã được trình bày, công bố trên tạp chí quốc tế: "*A novel approach to detect IoT malware by system calls and Long Short-Term Memory model*", Journal of Theoretical and Applied Information Technology 31st August 2021, Vol. 99. No. 16, 2021 (SCOPUS, Q4), ISSN: 1992-8645.

CHƯƠNG 3: XÂY DỰNG MÔ HÌNH PHÁT HIỆN MÃ ĐỘC IOT ĐƠN KIẾN TRÚC HIỆU QUẢ SỬ DỤNG ĐẶC TRƯNG TĨNH CỦA TẬP TIN THỰC THI

3.1. Mở đầu

Trong phương pháp xây dựng mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên đặc trưng dạng chuỗi của tập tin thực thi, phân tích động đã đem lại hiệu quả trong xây dựng mô hình phát hiện mã độc IoT dựa trên mạng nơ-ron LSTM, tuy nhiên phương pháp cũng bộc lộ một số hạn chế khi thực hiện thu thập hành vi lời gọi hệ thống với các mã độc sử dụng các kỹ thuật che giấu hành vi. Để giải quyết các trường hợp nêu trên và phân tích lượng lớn các tập tin IoT đòi hỏi xây dựng môi trường thực thi phức tạp thì các mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng tĩnh của tập tin cần được xem xét, lựa chọn. Phân tích tĩnh sẽ giúp giảm thời gian xây dựng môi trường và tài nguyên thực hiện phân tích, phân tích được nhiều thông tin liên quan đến hoạt động của tập tin, không cần xây dựng môi trường giám sát tập tin thực thi và xác định điều kiện kích hoạt để tập tin bộc lộ hết hành vi. Để giải quyết bài toán phân tích tĩnh một lượng lớn tập tin thực thi, các phương pháp học máy kết hợp với đặc trưng tĩnh được sử dụng để xây dựng mô hình phát hiện mã độc IoT.

Theo khảo sát của luận án, trong 10 năm trở lại đây, quá trình xây dựng mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng thu thập từ phân tích tĩnh tập tin gồm 4 bước: Thu thập dữ liệu, trích chọn đặc trưng, tiền xử lý, phát hiện mã độc được thể hiện trong hình 3.1.



Hình 3.1. Tổng quan quá trình xây dựng mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng tĩnh.

Hiệu quả của các mô hình phát hiện và dự đoán mã độc IoT đều phụ thuộc chặt chẽ vào các đặc trưng sử dụng để xây dựng mô hình. Quá trình trích chọn đặc trưng phù

hợp trong xây dựng mô hình không chỉ góp phần nâng cao độ chính xác của mô hình phát hiện mà còn giúp giảm độ phức tạp và tiết kiệm tài nguyên sử dụng của mô hình phục vụ tích hợp trong môi trường IoT tài nguyên hạn chế. Ngày nay, các phương pháp trích chọn đặc trưng bao gồm phương pháp lọc (filters), trình bao bọc (wrapper), phương pháp nhúng (embedded) hoặc kết hợp các phương pháp này thường được sử dụng trong xây dựng các mô hình phát hiện mã độc IoT hạn chế tài nguyên. Các phương pháp lọc sẽ chọn một tập hợp con các đặc trưng mà không làm thay đổi biểu diễn ban đầu của chúng [54]. Trích chọn đặc trưng dựa trên trình bao bọc bao gồm một mô hình phân loại để đánh giá sự phù hợp của các đặc trưng.

Canedo và cộng sự [119] đã phân tích 7 phương pháp trích chọn đặc trưng dựa trên bộ lọc, hai phương pháp trích chọn đặc trưng bao bọc và hai phương pháp nhúng để tạo tập dữ liệu "microarray" theo bốn phân loại học máy. Xu và cộng sự [12] đã so sánh các phương pháp trích chọn đặc trưng bao bọc và lựa chọn tập hợp đặc trưng con dựa trên bộ lọc liên quan đến hiệu suất phân loại và thời gian thực hiện. Các nghiên cứu này chỉ ra rằng các phương pháp lọc nhanh hơn các phương pháp trích chọn đặc trưng bao bọc nhưng hiệu suất phân loại thấp hơn nhiều so với các phương pháp bao bọc. Mặt khác, việc nghiên cứu một mô hình phát hiện có tốc độ nhanh là cần thiết khi đối mặt với các bài toán số lượng phần mềm độc hại ngày càng tăng theo cấp số nhân. Các mô hình phát hiện mã độc sử dụng ít về mặt thời gian có thể được huấn luyện dựa trên các đặc trưng tĩnh trích xuất từ các chương trình thực thi.

Các đặc trưng tĩnh được thu thập sẽ kết hợp với các thuật toán học máy, mạng học sâu phù hợp để xây dựng các mô hình phát hiện và phân loại mã độc hiệu quả. Việc sử dụng đặc trưng mã thực thi trích xuất từ phân tích tĩnh đã mang lại nhiều hiệu quả trong việc phát hiện mã độc nói chung và mã độc IoT nói riêng. Mã thực thi sử dụng trong luận án được xây dựng theo các khái niệm dưới đây:

Định nghĩa 3.1. *Mã thực thi (Operation code – opcode) là chỉ thị của một lệnh trong ngôn ngữ máy được thực hiện trên bộ vi xử lý nhằm đưa ra hướng dẫn các thao tác cho phần cứng thực hiện lệnh đó.*

Một mã thực thi là một chỉ thị lệnh đơn có thể được thực thi bởi bộ vi xử lý trung tâm. Mỗi loại kiến trúc vi xử lý khác nhau sẽ sử dụng một số loại mã thực thi khác nhau. Theo thống kê của NCS, các tập tin ELF trên kiến trúc MIPS có hơn 400 tên mã thực thi khác nhau, trên kiến trúc ARM có hơn 700 tên mã thực thi khác nhau, kiến trúc PowerPC có khoảng 400 mã thực thi khác nhau. Mỗi mã thực thi có thể chứa các thông tin về các thanh ghi và vị trí của dữ liệu, cũng như các tham số liên quan đến hoạt động được thực hiện, vì vậy nó có thể được sử dụng hiệu quả trong phát hiện mã độc.

Trong ngôn ngữ assembly, mỗi mã thực thi có hai phần là tên mã thực thi ở phần đầu và các tham số. Các phương pháp phổ biến sẽ lấy tên mã thực thi để đại diện, bỏ

qua phần tham số do sự phức tạp và sự phụ thuộc vào ngữ cảnh của các tham số này. Qua các khảo sát, đánh giá ở chương 1 và mục tiêu của luận án tối thiểu độ dài chuỗi mã thực thi, luận án đề xuất sử dụng thông tin tên mã thực thi để xây dựng chuỗi mã thực thi phục vụ xây dựng mô hình phát hiện mã độc. Do đó, với cách lấy tên mã thực thi, các mã thực thi sẽ được biểu diễn ở tên chỉ thị ngắn gọn như lw, sll, addu, sw, nop,… Chuỗi mã thực thi sử dụng trong luận án được định nghĩa như sau:

Định nghĩa 3.2. *Chuỗi mã thực thi là chuỗi các tên mã thực thi liên tiếp được trích xuất từ tập tin thực thi và có khả năng mô tả hành vi của một chương trình hoặc đoạn chương trình.*

Trong minh hoạ hình 3.2, chuỗi mã thực thi thu thập được từ chương trình sẽ là *lw lw sll addu sw lw nop addiu sw*.

```
loc_403300:                             # CODE XREF: connectTimeout+1C8↓j
                lw      $v0, 0xE0+var_BC($fp)
                lw      $v1, 0xE0+var_C0($fp)
                sll     $v0, 2
                addu    $v0, $v1
                sw      $zero, 0($v0)
                lw      $v0, 0xE0+var_BC($fp)
                nop
                addiu   $v0, 1
                sw      $v0, 0xE0+var_BC($fp)
```

Hình 3.2. Minh hoạ thu thập chuỗi mã thực thi từ các tập tin ELF trên kiến trúc MIPS.

Hạn chế chính của các nghiên cứu phát hiện mã độc IoT đơn kiến trúc sử dụng chuỗi mã thực thi luận án đã được khảo sát bao gồm:

Thứ nhất, trong trích chọn đặc trưng mã thực thi, phương pháp trích chọn đặc trưng lọc nhanh hơn các phương pháp trích chọn đặc trưng bao bọc nhưng do số lượng đặc trưng nhiều, nhiều đặc trưng gây nhiễu nên độ chính xác trong phát hiện và phân loại lại thấp hơn so với các phương pháp bao bọc. Trong môi trường IoT cần đảm bảo sự phù hợp giữa tốc độ xử lý và độ chính xác trong xây dựng các mô hình phát hiện mã độc IoT.

Thứ hai, các mô hình phát hiện cho độ chính xác cao đang sử dụng các phương pháp trích xuất đặc trưng mã thực thi dạng đồ thị phức tạp, huấn luyện mô hình dựa trên các thuật toán học máy, mạng học sâu phức tạp. Điều này dẫn đến thời gian trích xuất chuỗi mã thực thi, kích thước mô hình phát hiện sau khi huấn luyện lớn và khó để ứng dụng các mô hình phát hiện này trong hệ thống IoT hạn chế tài nguyên hoặc đáp ứng yêu cầu tốc độ xử lý.

Thứ ba, các mô hình phát hiện mã độc IoT dựa trên đặc trưng mã thực thi cho độ chính xác cao đã khảo sát chỉ mới được thử nghiệm với các nền tảng của máy tính cá nhân và thiết bị di động, chưa tập trung trên các kiến trúc vi xử lý của thiết bị IoT phổ biến khác như MIPS. Mỗi nền tảng kiến trúc vi xử lý có các đặc trưng mã thực thi khác nhau, vì vậy

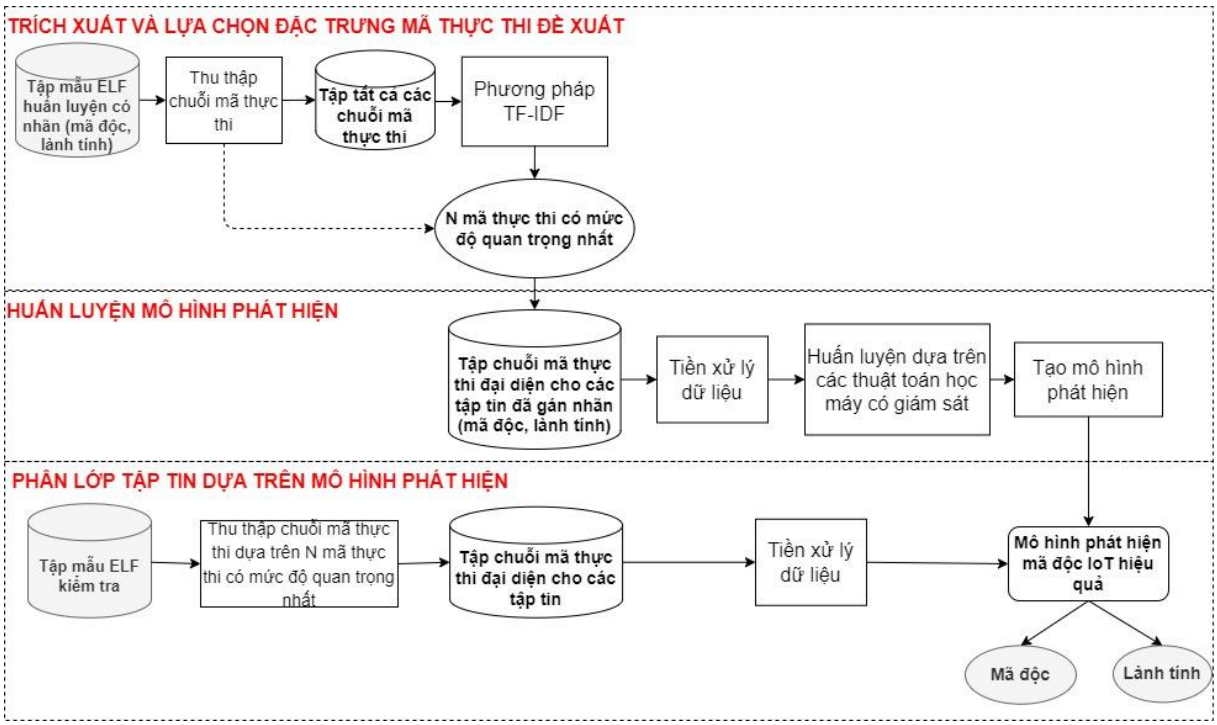
cần có các nghiên cứu đánh giá thử nghiệm trên các nền tảng khác nhau để đưa ra các phương pháp xây dựng mô hình phát hiện phù hợp với loại đặc trưng tĩnh này.

Vì vậy, *để nâng cao hiệu quả cho mô hình phát hiện mã độc IoT đơn kiến trúc sử dụng đặc trưng tĩnh của tập tin thực thi, chương 3 đề xuất kết hợp phương pháp trích xuất, thu thập đặc trưng chuỗi mã thực thi hiệu quả, phù hợp với các mô hình học máy truyền thống trong xây dựng mô hình phát hiện mã độc IoT trên nền tảng kiến trúc MIPS. Mô hình phát hiện mã độc IoT đơn kiến trúc sử dụng tối thiểu số lượng mã thực thi cần thu thập từ phân tích tĩnh tập tin, sử dụng các thuật toán học máy truyền thống để giảm độ phức tạp tính toán, thời gian tính toán và kích thước mô hình phát hiện sau khi huấn luyện nhưng vẫn đảm bảo độ chính xác cao tương đồng các nghiên cứu khác luận án đã khảo sát.*

Có nhiều phương pháp trích xuất và lựa chọn đặc trưng tĩnh của tập tin thực thi sử dụng trong phát hiện mã độc đã được nghiên cứu. Theo cách tiếp cận đặc trưng biểu diễn dạng chuỗi của luận án, một mã thực thi được coi là một "từ" trong mô hình ngôn ngữ. Một chuỗi mã thực thi được coi là một "câu" trong mô hình ngôn ngữ. Vì vậy, chúng ta có thể dự đoán ý nghĩa của một câu dựa vào một số "từ khóa" trong câu. Do đó, trong một chuỗi mã thực thi, mỗi mã thực thi có một mức độ ý nghĩa khác, một số mã thực thi có thể mô tả ý nghĩa và đại diện cho chuỗi mã thực thi đó. Việc lựa chọn được các mã thực thi có mức độ quan trọng cao trong chuỗi sẽ hạn chế được số lượng đặc trưng mã thực thi cần trích xuất, thu thập trong phân tích tĩnh. Trong xử lý ngôn ngữ tự nhiên việc xác định mức độ quan trọng của một từ trong câu đã có thể được giải quyết thông qua nhiều phương pháp như Information Gain, Term Frequency – Inverse Document Frequency,… Đối với phân tích chuỗi mã thực thi, luận án cũng tiếp cận dựa trên sự tương đồng này để lựa chọn các mã thực thi quan trọng cho các kiến trúc vi xử lý trong xây dựng mô hình phát hiện mã độc IoT hiệu quả. Luận án sẽ tiến hành phân tích, đánh giá các phương pháp trích xuất, lựa chọn, thuật toán học máy truyền thống hiện có có để lựa chọn phương pháp trích xuất đặc trưng chuỗi mã thực thi phù hợp đối với các tập tin thực thi trên nền tảng kiến trúc MIPS phục vụ xây dựng mô hình phát hiện hiệu quả về thời gian xử lý và kích thước mô hình, độ chính xác phù hợp với môi trường thiết bị IoT tài nguyên hạn chế.

3.2. Xây dựng mô hình phát hiện mã độc IoT dựa trên chuỗi mã thực thi đề xuất

Quá trình xây dựng mô hình phát hiện mã độc IoT dựa trên đặc trưng chuỗi mã thực thi đề xuất gồm 3 giai đoạn chính: Trích xuất và lựa chọn đặc trưng; Huấn luyện mô hình phát hiện; Phân lớp tập tin dựa trên mô hình phát hiện. Tổng thể quá trình xây dựng mô hình đề xuất được mô tả trong hình 3.3.



Hình 3. 3. Quá trình xây dựng mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên học máy sử dụng đặc trưng chuỗi mã thực thi.

3.2.1. Đề xuất phương pháp trích xuất, lựa chọn đặc trưng mã thực thi

- Đối với quá trình trích xuất chuỗi mã thực thi dựa trên phân tích tĩnh tập tin thực thi: Có nhiều phương pháp trích xuất mã thực thi từ tập tin thực thi đã được đề xuất như khảo sát trong nghiên cứu của nghiên cứu sinh [TC4]. Các phương pháp trích xuất chuỗi mã thực thi dựa trên đồ thị sẽ đối mặt với việc sử dụng tài nguyên tính toán lớn, việc lưu trữ phức tạp và cần nhiều thời gian để trích xuất cho một tập tin thực thi. Các phương pháp trích xuất mã thực thi dựa trên đồ thị như [17],[116],[117],… có thể đem nhiều thông tin về luồng thực thi của một chương trình khi trích xuất chuỗi mã thực thi nhưng độ phức tạp tính toán và tài nguyên sử dụng để thu thập trích xuất có thể tăng cao khi tập tin có kích thước lớn hoặc phức tạp.

Để khắc phục một số hạn chế trên, phương pháp trích xuất đặc trưng chuỗi mã thực thi được thực hiện thông qua các công cụ dịch ngược tập tin thực thi. Các phương pháp dịch ngược có thể khắc phục được vấn đề tài nguyên và tốc độ trích xuất đặc trưng mã thực thi nhưng lại đối mặt với việc độ dài chuỗi mã thực thi thu thập được lớn, việc xử lý và huấn luyện mô hình phát hiện mã độc IoT trực tiếp trên các chuỗi mã thực thi này không đem lại hiệu quả cao. Các công cụ dịch ngược sẽ thu thập tất cả các thông tin mã thực thi của chương trình thực thi gốc và đoạn mã độc hại lây nhiễm trong tập tin, vì vậy độ dài chuỗi mã thực thi thu thập được lớn và nhiều thông tin không có giá trị phân lớp hoặc gây nhiễu từ mã thực thi của chương trình thực thi gốc.

Để phù hợp và nâng cao khả năng phát triển giải pháp bảo mật trong môi trường IoT tài nguyên hạn chế thì việc xây dựng một giải pháp trích xuất chuỗi mã thực thi đơn

giản, hiệu quả phù hợp với thuật toán học máy là yêu cầu cần thiết.

Để đạt được mục tiêu nghiên cứu của luận án, cách tiếp cận trích xuất chuỗi mã thực thi dựa trên phương pháp dịch ngược tập tin thực thi được lựa chọn sử dụng trong mô hình đề xuất. Tuy nhiên, luận án tiếp cận trích xuất chuỗi mã thực thi dựa trên "từ điển opcode" đề xuất là danh sách mã thực thi quan trọng trên một nền tảng kiến trúc vi xử lý tương ứng được xây dựng dựa trên phương pháp xác định mức độ quan trọng.

Phương pháp trích xuất chuỗi mã thực thi từ tập tin thực thi ELF trên hệ điều hành Linux luận án đề xuất được thực hiện thông qua các trong sơ đồ hình 3.4.

Hình 3.4. Quá trình trích xuất chuỗi mã thực thi từ tập tin ELF.

Bước 1: Tập tin thực thi ELF được dịch ngược thành tập tin dạng ASM thông qua công cụ dịch ngược tập tin thực thi hỗ trợ kiến trúc vi xử lý và hệ điều hành tương ứng. Việc thu thập được chính xác, đầy đủ các đặc trưng mã thực thi của tập tin phụ thuộc vào hiệu quả của các công cụ dịch ngược. Vì vậy, luận án tiến hành các đánh giá và thử nghiệm dịch ngược các tập tin để phân tích, lựa chọn, đề xuất công cụ phù hợp đối với nền tảng kiến trúc vi xử lý luận án tập trung nghiên cứu.

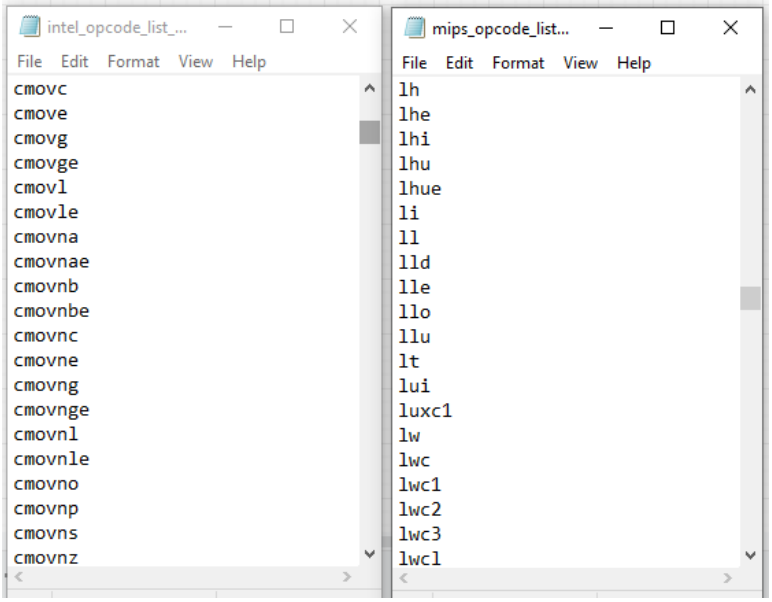
Hiện nay, một số công cụ dịch ngược được các nghiên cứu sử dụng đối với các tập tin ELF bao gồm: IDA Pro [50], Capstone [20], OllyDBG [88],… Mỗi công cụ đều có những ưu và nhược điểm trong dịch ngược các tập tin thực thi. Kết quả so sánh các công cụ dịch ngược được thể hiện trong Bảng 3.1. Từ kết quả phân tích và quá trình thử nghiệm dịch ngược các tập tin ELF trên nền tảng MIPS, luận án thấy rằng công cụ IDA Pro đem lại nhiều ưu điểm vượt trội so với các công cụ dịch ngược khác. Vì vậy, trong mô hình đề xuất, công cụ IDA Pro được luận án lựa chọn để thực hiện chức năng dịch ngược tập tin thực thi ELF.

Bảng 3.1. So sánh các công cụ hỗ trợ dịch ngược tập tin ELF.

	IDA Pro	**OllyDBG**	**Capstone**
Ưu điểm	- Có khả năng phân tích các định dạng tập tin khác nhau. - Cung cấp cho người dùng nhiều tính năng nâng cao như tìm kiếm chuỗi, phân tích hệ thống	- Miễn phí và có tính năng gỡ rối tốt, cho phép người phân tích theo dõi các biến và hàm trong quá trình thực thi. - Cung cấp cho người	- Miễn phí và mã nguồn mở. - Hỗ trợ nhiều kiến trúc vi xử lý khác nhau. - Được viết bằng ngôn ngữ C, dễ dàng tích hợp vào các ứng dụng

	cục bộ, phân tích mã, … - Có khả năng tùy chỉnh và mở rộng để thêm các tính năng mới, phù hợp với nhu cầu của người dùng.	dùng các tính năng tìm kiếm, phân tích, xử lý và phân tích phần mềm độc hại. - Giao diện người dùng thân thiện và dễ sử dụng.	khác.
Nhược điểm	-Là phần mềm thương mại, có phí. - Giao diện người dùng không thân thiện và khó sử dụng cho người dùng mới.	- Không có tính năng phân tích các định dạng tập tin khác nhau như IDA Pro. - Không cung cấp các tính năng phân tích phức tạp như IDA Pro.	- Không có các tính năng phức tạp như IDA Pro. - Không cung cấp giao diện người dùng đồ họa, chỉ có API để tích hợp vào các ứng dụng.

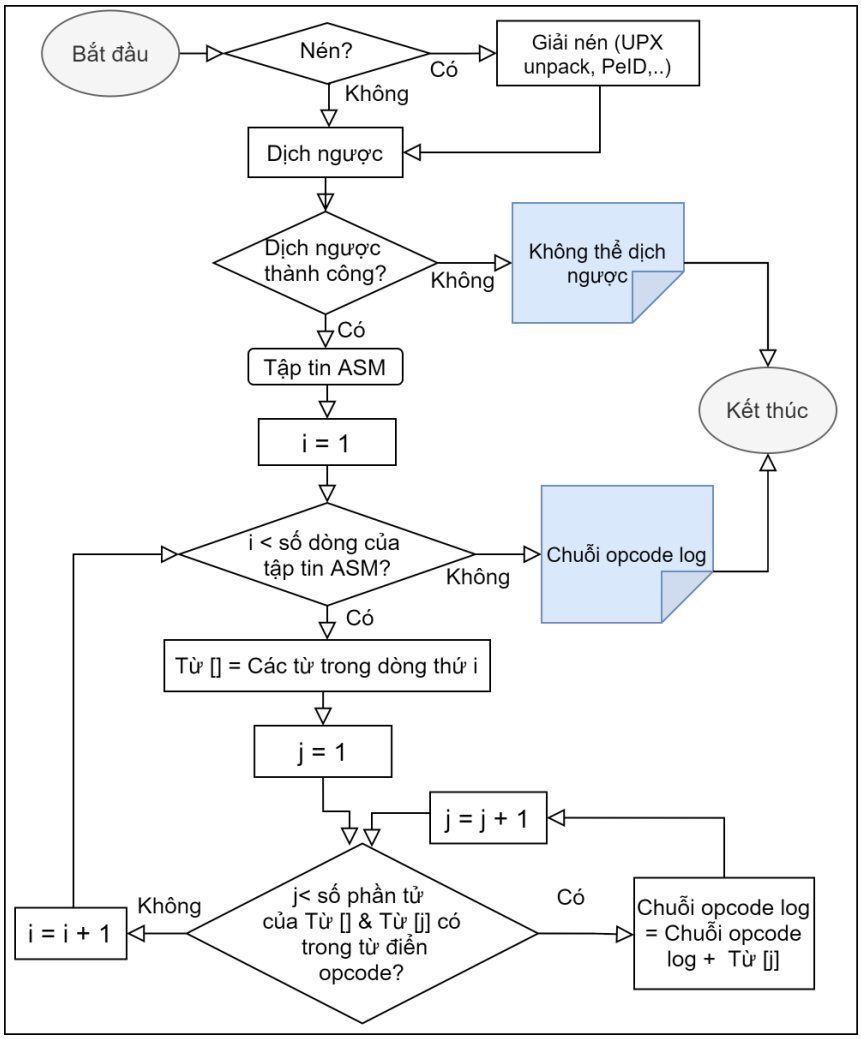
Bước 2: Các tập tin dạng ASM chứa mã dịch ngược (dạng hợp ngữ) sẽ được sử dụng để trích xuất chuỗi mã thực thi thông qua "từ điển opcode" xây dựng trên nền tảng kiến trúc vi xử lý tương ứng với thiết bị IoT. "Từ điển opcode" đầy đủ ban đầu được luận án xây dựng thông qua việc thu thập tên các mã thực thi từ các nhà sản xuất CPU và các công trình khoa học đã công bố. Bên cạnh đó, việc xây dựng từ điển mã thực thi được thực hiện qua các thực nghiệm dịch ngược và phân tích tập tin thực thi thủ công để xác định, bổ sung các mã thực thi có trong tập tin đã dịch ngược nhưng chưa tìm thấy từ các nguồn Internet. "Từ điển opcode" đầy đủ ban đầu của một số nền tảng kiến trúc của thiết bị IoT được luận án thu thập công bố mở tại địa chỉ: "*https://gitlab.com/ngoctoan/caimp/-/tree/main/Opcode%20Extraction?ref_type=heads*". Số lượng, tên và chức năng của các mã thực thi cũng có sự khác biệt giữa các nền tảng kiến trúc như trong "từ điển opcode" hai nền tảng Intel và MIPS như hình 3.5.



Hình 3.5. Từ điển mã thực thi trên kiến trúc vi xử lý Intel và MIPS.

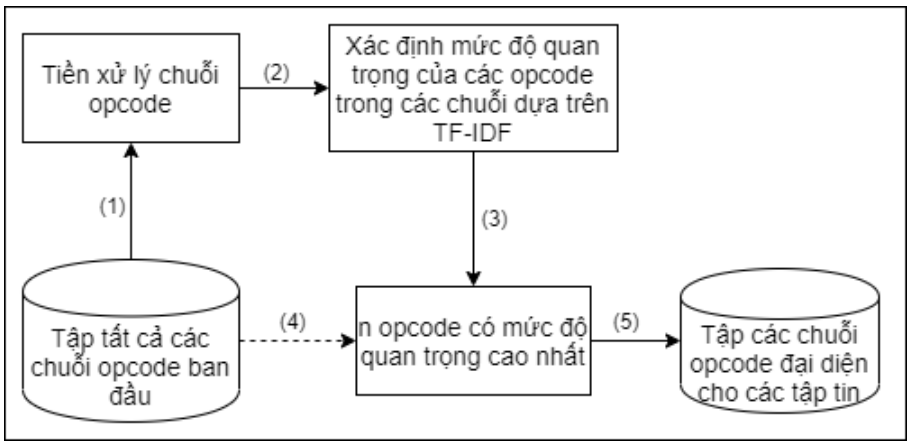
Quá trình thu thập chuỗi mã thực thi của một tập tin dựa trên "từ điển opcode"

được luận án đề xuất như mô tả trong thuật toán hình 3.6.



Hình 3.6. Sơ đồ hoá thuật toán thu thập chuỗi mã thực thi đề xuất

- *Đề xuất "từ điển opcode" hiệu quả phục vụ xây dựng tập chuỗi mã thực thi đại diện cho các tập tin*: Số lượng và chất lượng của các đặc trưng mã thực thi trong chuỗi cần được xem xét trong huấn luyện mô hình học máy phục vụ phân loại tập tin ELF để đảm bảo độ chính xác cao và sử dụng ít tài nguyên nhất. Do đó, luận án đề xuất sử dụng phương pháp TF-IDF trong xử lý ngôn ngữ tự nhiên để xác định các mã thực thi quan trọng nhất trong xây dựng đặc trưng chuỗi mã thực thi, từ đó xây dựng lại "từ điển opcode" hiệu quả cho từng nền tảng kiến trúc phục vụ xây dựng mô hình phát hiện mã độc IoT. Quá trình xác định "từ điển opcode" hiểu quả là n opcode có mức độ quan trọng cao nhất phục vụ xây dựng đặc trưng chuỗi mã thực thi cho xây dựng mô hình phát hiện mã độc được thể hiện trong hình 3.7.



Hình 3.7. Sơ đồ phương pháp lựa chọn mã thực thi quan trọng và xây dựng chuỗi mã thực thi đại diện cho các tập tin.

Với hướng tiếp cận chuỗi mã thực thi trích xuất từ một tập tin là một đặc trưng tuần tự dạng chuỗi tương đồng với các "câu" trong ngôn ngữ tự nhiên, luận án xác định mức độ quan trọng của một mã thực thi trong chuỗi mã thực thi dựa trên việc tính toán trọng số của mã thực thi trong một chuỗi. Trong các nghiên cứu trong mô hình ngôn ngữ tự nhiên, mô hình Term Frequency – Inverse Document Frequency (TF-IDF) đã đem lại nhiều hiệu quả trong tổng hợp văn bản và xác định mức độ quan trọng của các từ trong đoạn văn [HT1]. Giá trị TF-IDF là trọng số của một từ trong văn bản thu được thông qua số liệu thống kê và cho thấy tầm quan trọng của từ này trong một văn bản chứa nó. Vì vậy, TF-IDF có thể được áp dụng để xác định mức độ quan trọng của các mã thực thi trong tập các chuỗi mã thực thi thu thập được. Giá trị TF được sử dụng để tính toán tần suất xuất hiện của các mã thực thi trong chuỗi mã thực thi. Độ dài mỗi chuỗi mã thực thi khác nhau, số lần xuất hiện của các mã thực thi có thể khác nhau rất lớn. Vì vậy, số lần xuất hiện của các mã thực thi sẽ được chia cho độ dài của chuỗi (điều này tương tự tổng số từ trong một văn bản). Trong trường hợp luận án sử dụng, giá trị TF của mã thực thi *x* được tính theo công thức như sau đây:

$$TF(x, s) = \frac{fr(x)}{sum(s)} \quad (3.1)$$

trong đó *fr(x)* là số lần mã thực thi *x* xuất hiện trong chuỗi mã thực thi *s*, *sum(s)* là tổng số mã thực thi của chuỗi mã thực thi *s*.

Giá trị IDF là sự đo lường mức độ quan trọng của một mã thực thi trong tập các mã thực thi. Khi chỉ tính tần suất xuất hiện của mã thực thi, các mã thực thi được coi là quan trọng như nhau. Tuy nhiên, có một số mã thực thi thường được sử dụng nhiều nhưng không quan trọng để thể hiện ý nghĩa của chuỗi mã thực thi. Một số mã thực thi trong tập dữ liệu lại mô tả được đặc trưng của hành vi độc hại trên hệ thống. Do đó, giá trị IDF có khả năng xác định lại các trọng số tương ứng cho các mã thực thi quan trọng luôn xuất hiện. Giá trị IDF luận án sử dụng được tính theo công thức sau đây:

$$IDF(x, D) = log\ (\frac{N}{D(x)}) \quad (3.2)$$

trong đó *N* là tổng số chuỗi mã thực thi được tạo ra, *D(x)* là số chuỗi mã thực thi chứa mã thực thi *x*.

Việc xác định tầm quan trọng của các mã thực thi trong chuỗi mã thực thi có nhiều điểm chung với mức độ liên quan của từ trong một văn bản. Việc kết hợp hai giá trị TF và IDF sẽ giúp nâng cao hiệu quả trong việc xác định mức độ quan trọng của các mã thực thi, một số mã thực thi xuất hiện nhiều nhưng thực tế có thể không quan trọng. Các nhà nghiên cứu đã có nhiều kết quả khả quan về TF-IDF của từ trong văn bản như các nghiên cứu [93], [108]. Có hai công thức chính để tính giá trị TF-IDF thường được sử dụng trong ngôn ngữ tự nhiên gồm: TF-IDF cơ bản và TF-IDF nâng cao như TF-IDF Normalization, Smooth IDF, Max IDF,…Dựa trên các đặc điểm của mã thực thi và chuỗi mã thực thi thu thập được, luận án đề xuất sử dụng cách tính TF-IDF của mã thực thi *x* theo công thức cơ bản và được biểu diễn như sau:

$$TF\text{-}IDF\ (x) = TF(x).IDF(x,D) \quad (3.3)$$

Thông qua việc xác định giá trị TF-IDF của mã thực thi trong tập dữ liệu mã thực thi, NCS sử dụng *n* mã thực thi có trọng số cao nhất để xây dựng "từ điển opcode" hiệu quả cho kiến trúc vi xử lý tương ứng. Khi lựa chọn *n* quá lớn, số lượng mã thực thi trong từ điển và độ chuỗi mã thực thi thu thập được từ một tập tin có thể tăng lên, độ phức tạp khi xây dựng mô hình có thể nâng cao. Tuy nhiên, khi lựa chọn *n* quá nhỏ, số lượng mã thực thi cần thu thập ít, chuỗi mã thực thi thu thập được có thể không đủ thông tin để đại diện được cho tập tin thực thi và giảm độ chính xác khi xây dựng mô hình phát hiện mã độc. Vì vậy, giá trị *n* mã thực thi có mức độ quan trọng cao nhất sẽ phụ thuộc vào từng nền tảng kiến trúc vi xử lý được luận án xác định dựa trên các thực nghiệm và các nghiên cứu có liên quan khác.

Cuối cùng, "từ điển opcode" hiệu quả được sử dụng để xây dựng chương trình trích xuất chuỗi mã thực thi đại diện cho các tập tin thực thi phục vụ xây dựng mô hình phát hiện mã độc IoT như hình 3.8.

```
# Tải lên tập tin chứa "từ điển opcode hiệu quả"
input = open ('từ điển opcode', 'r')
opcodeList = input.readlines()
opcodeDict = dict()
for i in opcodeList:
    for x in i.split():
        opcodeDict(x)=i
input.close()
# Thu thập chuỗi mã thực thi đại diện cho từng tập tin thực thi dựa trên "từ điển opcode" đề xuất
path = "Đường dẫn đến thư mục lưu các tập tin .asm chứa kết quả dịch ngược"
os.chdir(path)
for file in os.listdir():
    if file.endwith(".asm"):
        file_path = f"{path}/{file}"
        with open(file_path, encoding="utf8", 'r') as f:
            print (file_path)
            output = open (file_path + '.txt', 'w')
            all = f.readlines()
            for line in all:
                for word in line.split():
                    if word in opcodeDict:
                        output.write(word + ' ')
            f.close()
            output.close()
# Kết quả trả về là các tập tin .txt chứa chuỗi mã thực thi đại diện cho tập tin
```

Hình 3.8. Thu thập chuỗi mã thực thi dựa trên "từ điển opcode" đề xuất.

Kết quả của quá trình trích chọn, thu thập là một tập dữ liệu các chuỗi mã thực thi đại diện cho các tập tin ELF phục vụ huấn luyện và phân lớp tập tin.

3.2.2. Huấn luyện mô hình phát hiện mã độc IoT

Tập dữ liệu các chuỗi mã thực thi đại diện cho các tập tin được thu thập sẽ được tiền xử lý thông qua việc chuẩn hoá, chuyển đổi phù hợp với các thuật toán sử dụng để xây dựng mô hình phát hiện mã độc IoT. Tập dữ liệu chuỗi mã thực thi được phân chia thành hai tập dữ liệu huấn luyện và tập đánh giá.

Ngoài ra, phương pháp n-gram được sử dụng để chuyển đổi đặc trưng các chuỗi mã thực thi đại diện cho tập tin phục vụ huấn luyện các mô hình máy học. N-gram là tuần xuất hiện hiện của *n* ký tự liên tiếp nhau có trong dữ liệu chuỗi đầu vào. Phương pháp xem xét chuỗi đầu vào là một dãy liên tiếp gồm các gram và tính tần suất xuất hiện của *n* gram liên tiếp để làm đặc trưng cho chuỗi và giá trị được lưu trong một vectơ.

Trong luận án, phương pháp n-gram được sử dụng để biểu diễn và xác định giá trị các đặc trưng của một chuỗi mã thực thi trong không gian vectơ. Mỗi phần tử của vectơ đặc trưng thể hiện sự hiện diện hay vắng mặt của n-gram tương ứng trong chuỗi mã thực thi. Phương pháp n-gram đã được chứng minh là có hiệu quả trong việc phát hiện phần mềm độc hại dựa trên chuỗi mã thực thi trong các nghiên cứu [79] [115]. Kang và cộng sự [11] đã trình bày và đánh giá một phương pháp tiếp cận dựa trên các đặc trưng n-gram của mã thực thi sử dụng máy học để xác định và phân loại mã độc trên hệ điều hành Android với độ chính xác F-measure là 98%.

Trong phương pháp n-gram, nếu *n* quá nhỏ (unigram), thông tin thu được sẽ chỉ là tần suất xuất hiện của các mã thực thi đơn lẻ. Nếu *n* quá lớn, số lượng đặc trưng rất lớn, đặc biệt là phần mềm độc hại sử dụng các kỹ thuật chuyển đổi mã thực thi. Có một số

nghiên cứu NCS khảo sát [114] đã chỉ ra rằng 2-gram và 3-gram có thể cho kết quả tốt trong phát hiện mã độc IoT.

Ví dụ, tính n-gram của một chuỗi mã thực thi X *(push sub push sub push call add jz add jz add)* được mô tả trong Bảng 3.2.

Bảng 3.2. N-gram của chuỗi mã thực thi X.

2-gram	push sub	sub push	push call	call add	add jz	jz add	
Tần suất	2	2	1	1	2	2	
3-gram	push sub push	sub push sub	sub push call	push call add	call add jz	add jz add	jz add jz
Tần suất	2	1	1	1	1	2	1

Sau đó, các thuật toán học máy không sâu có giám sát phổ biến trong phát hiện mã độc được luận án lựa chọn để huấn luyện các mô hình phát hiện mã độc IoT dựa trên đặc trưng chuỗi mã thực thi đã xác định giá trị trong không gian vectơ. Trong mô hình phát hiện mã độc IoT đề xuất, ba thuật toán học máy không sâu có giám sát phổ biến trong phát hiện mã độc gồm Support Vectơ Machines, Random Forest, Naive Bayes được lần lượt thử nghiệm để đánh giá hiệu quả đối với thiết bị IoT tài nguyên hạn chế. Trong quá trình thử nghiệm, nghiên cứu sinh sẽ thực nghiệm nhiều lần và tinh chỉnh các tham số của các mô hình học máy để lựa chọn mô hình học máy phù hợp trong xây dựng mô hình phát hiện mã độc IoT đã đặt ra.

Các mô hình phát hiện mã độc IoT dựa trên đặc trưng chuỗi mã thực thi được xây dựng sử dụng lần lượt các thuật toán học máy để xác định hiệu quả dựa trên các tiêu chí đánh giá. Từ đó, lựa chọn mô hình phát hiện mã độc IoT sử dụng học máy và đặc trưng chuỗi mã thực thi phù hợp nhất khi triển khai trong môi trường IoT tài nguyên hạn chế.

Kết quả của giai đoạn này là các tập tin mô hình phát hiện phục vụ quá trình đánh giá, phân lớp tập tin cần xác định nhãn.

3.2.2. Phân lớp tập tin dựa trên mô hình phát hiện

Tương tự quá trình huấn luyện mô hình, việc phân lớp các tập tin kiểm tra dựa trên mô hình phát hiện mã độc IoT được tiến hành thông qua thu thập chuỗi mã thực thi dựa trên từ điển mã thực thi đã xây dựng trong giai đoạn thu thập, lựa chọn đặc trưng mã thực thi. Tập các chuỗi mã thực thi đại diện cho các tập tin kiểm tra được tiền xử lý với phương pháp n-gram và phân lớp với mô hình phát hiện mã độc IoT hiệu quả tương ứng.

3.3. Thử nghiệm và đánh giá hiệu quả của mô hình đề xuất

3.3.1. Môi trường thử nghiệm

Các thử nghiệm của NCS được thực hiện trên cùng một máy tính sử dụng hệ điều hành Windows 10 phiên bản 64-bit, với cấu hình Intel Core i7-6500U, 2,59 GHz, RAM 8GB. Các thực nghiệm đánh giá hiệu quả được tiến hành 10 lần tại các khoảng thời gian

thực hiện khác nhau. Kết quả đánh giá là trung bình của tất cả các lần thử nghiệm.

Để đánh giá hiệu suất của mô hình đề xuất, luận án thử nghiệm mô hình phát hiện sử dụng tất cả 419 mã thực thi trên kiến trúc MIPS và các mô hình khác cùng cách tiếp cận của nhóm tác giả Ding [125] và Phú [117] trên cùng một tập dữ liệu, cùng một hệ thống thử nghiệm như phương pháp của luận án đề xuất.

3.3.2. Tập dữ liệu thử nghiệm

Bộ dữ liệu C500-IoT dataset được công bố bởi Phú và các cộng sự [115] như trong nội dung 2.3.2 đã trình bày trong luận án được sử dụng để thử nghiệm các mô hình phát hiện mã độc IoT dựa trên đặc trưng chuỗi mã thực thi. Để đánh giá các kịch bản thử nghiệm, NCS sử dụng tập dữ liệu mở rộng của tập đã sử dụng trong chương 2 để thử nghiệm hiệu quả của mô hình phát hiện mã độc IoT đơn kiến trúc sử dụng đặc trưng tĩnh của tập tin thực thi gồm 8.904 tập tin ELF trên kiến trúc MIPS (với 4.511 tập tin độc hại và 4.393 tập tin lành tính) trong tập C500-IoT dataset. Tập dữ liệu huấn luyện và kiểm tra được phân chia ngẫu nhiên theo tỉ lệ 70/30 phục vụ đánh giá hiệu quả các mô hình phát hiện.

3.3.3. Kết quả trích xuất và lựa chọn đặc trưng mã thực thi

Kết quả thu thập tập dữ liệu chuỗi mã thực thi thông qua phân tích tĩnh tập tin được thể hiện chi tiết trong Bảng 3.3. Từ kết quả Bảng 3.3 đã chỉ ra rằng việc thu thập chuỗi mã thực thi của một số tập tin mã độc (112 mẫu chiếm 2,5 % tổng số mẫu tập tin mã độc thử nghiệm) còn chưa hiệu quả do tập tin mã độc đã sử dụng các kỹ thuật "pack", chuỗi mã thực thi thu thập được từ các tập tin nhỏ hơn 50. Đây cũng là hạn chế chính của phân tích tĩnh tập tin mã độc nói chung và mã độc IoT nói riêng.

Bảng 3.3. Kết quả thu thập chuỗi mã thực thi từ các tập tin.

Nhãn	**Mã sạch**	**Mã độc**
Số lượng tập tin thực thi ELF	4.393	4.511
Số lượng tập tin thu thập được chuỗi mã thực thi đáp ứng yêu cầu	4.393	4.423

Kết quả lựa chọn các mã thực thi trên kiến trúc vi MIPS với mức độ quan trọng hàng đầu (có giá trị TF-IDF cao nhất) được thể hiện trong Bảng 3.4.

Bảng 3.4. Các mã thực thi có trọng số quan trọng cao nhất trong tập dữ liệu thử nghiệm trên kiến trúc MIPS.

Mức độ quan trọng	Tên mã thực thi	Giá trị TF-IDF	Mô tả	Mức độ quan trọng	Tên mã thực thi	Giá trị TF-IDF	Mô tả
1	bgezal	8,29	Branch On >= 0 And Link	**21**	mfhi	1,34	Move From Hi
2	trap	7,09	Exception And Interrupt Instruction	**22**	bgtz	1,34	Branch if Greater Than Zero
3	sub	5,99	Subtract	**23**	xori	1,29	XOR Immediate
4	addi	5,59	Add Immediate	**24**	sllv	1,27	Shift Left Logical Variable
5	mthi	4,89	Move To Hi	**25**	blez	1,27	Branch if Less Than or Equal to Zero
6	bltzal	4,48	Branch On < 0 And Link	**26**	xor	1,26	Bitwise XOR
7	mtlo	4,31	Move to LO Register	**27**	lb	1,21	Load Byte
8	add	3,89	Add	**28**	bgez	1,19	Branch if Greater Than or Equal to Zero
9	jal	2,63	Jump And Link	**29**	slt	1,18	Set on Less Than
10	srav	2,18	Shift Right Arithmetic Variable	**30**	bltz	1,16	Branch if Less Than Zero
11	lh	1,91	Load Halfword	**31**	ori	1,14	OR Immediate
12	syscall	1,68	System Call	**32**	sra	1,14	Shift Right Arithmetic
13	div	1,62	Divide	**33**	slti	1,12	Set on Less Than Immediate
14	srlv	1,58	Shift Right Logical Variable	**34**	sh	1,12	Store Halfword
15	break	1,53	Breakpoint	**35**	srl	1,11	Shift Right Logical
16	multu	1,52	Unsigned Multiply	**36**	lhu	1,10	Load Halfword Unsigned
17	mult	1,49	Multiply	**37**	lui	1,09	Load Upper Immediate
18	divu	1,47	Unsigned Divide	**38**	or	1,09	Logical OR
19	mflo	1,37	Move From LO Register	**39**	and	1,08	Logical and
20	nor	1,35	Bitwise NOR (NOT-OR)	**40**	sltiu	1,05	Set on Less Than Immediate Unsigned

3.3.4. Kết quả huấn luyện mô hình phát hiện mã độc IoT

Luận án tiến hành thử nghiệm nhiều lần mô hình phát hiện với từng thuật toán học máy có giám sát gồm SVM, RF, NB được cài đặt thông qua ngôn ngữ lập trình Python với thư viện Sklearn[9]. Qua quá trình thử nghiệm nhằm đánh giá tác động của các tham số đến độ chính xác, kích thuộc mô hình, thời gian huấn luyện và phát hiện, các tham số chính cho hiệu quả nhất trong các lần thử nghiệm xây dựng mô hình phát hiện mã độc IoT được mô tả như Bảng 3.5.

Bảng 3.5. Các tham số chính sử dụng trong các thuật toán học máy sử dụng từ thư viện Sklearn.

Thuật toán học máy	**Tham số/ Hàm sử dụng trong thư viện**	**Ý nghĩa**	**Giá trị sử dụng**
SVM	c	Điều chỉnh mức độ chặt chẽ của siêu phẳng	1
	gamma	Điều chỉnh ảnh hưởng của từng điểm dữ liệu	“scale”
	decision_function_shape	Cách xử lý phân loại nhiều lớp	“ovo”
	cache_size	Kích thước bộ nhớ đệm để lưu kernel matrix trong quá trình huấn luyện, nhằm tối ưu hóa hiệu suất	500
	tol	Dung sai cho tiêu chí dừng	2e-3
	break_ties	Nếu bật, mô hình sẽ phá vỡ các trường hợp hòa khi dự đoán	“True”
	random_state	Một giá trị cố định đảm bảo rằng kết quả phân chia dữ liệu train/test sẽ giống nhau ở mọi lần chạy	1
	StandardScaler()	Chuẩn hóa dữ liệu để mỗi đặc trưng có trung bình bằng 0 và độ lệch chuẩn bằng 1	
	PCA()	Giảm số chiều của dữ liệu	
RF	bootstrap	Tính ngẫu nhiên của mẫu được chọn với thay thế (bootstrap)	“True”
	random_state	Hạt giống để tái lập tính ngẫu nhiên trong việc xây dựng cây	2
	Max_features	Số lượng đặc trưng được xem xét khi tìm kiếm phép chia tốt nhất	“sqrt”
	n_jobs		-1
	n_estimators	Số lượng cây trong rừng ngẫu nhiên	50
	criterion	Hàm đánh giá chất lượng của phép chia	'gini'
	max_depth	Độ sâu tối đa của mỗi cây	200

[9] https://scikit-learn.org/stable/

	min_samples_split	Số lượng mẫu tối thiểu cần thiết để một nút có thể chia nhỏ	2
	min_samples_leaf	Số lượng mẫu tối thiểu trong mỗi l	1
NB	priors	Mảng xác suất tiên nghiệm của các lớp	"None"
	var_smoothing	Một phần nhỏ được thêm vào phương sai ước tính của mỗi đặc trưng để tránh việc chia cho 0 trong quá trình tính toán xác suất	1^{e-9}
	test_size	tỷ lệ/ số lượng mẫu được dùng làm tập test khi chia dữ liệu thành hai phần train và test	0.3
	random_state	Một giá trị cố định đảm bảo rằng kết quả phân chia dữ liệu train/test sẽ giống nhau ở mọi lần chạy	42

Các thực nghiệm ngoài việc lựa chọn tham số phù hợp cho các mô hình học máy thì còn giúp đánh giá sự ảnh hưởng của các mã thực thi trong "từ điển opcode". Nếu sử dụng ít mã thực thi xây dựng "từ điển opcode" thì chuỗi mã thực thi thu thập được sẽ ngắn và không đủ để đại diện cho tập tin thực thi. Ngược lại, khi sử dụng quá nhiều mã thực thi làm từ điển trích xuất, chuỗi mã thực thi thu thập được quá dài và có thể xuất hiện nhiều mã thực thi gây nhiễu, hiệu quả của việc giảm số đặc trưng tĩnh cần thu thập không đáng kể. Do đó, với mục tiêu bài toán nghiên cứu phát hiện mã độc IoT cho thiết bị IoT tài nguyên hạn chế, luận án sẽ xem xét lựa chọn số lượng mã thực thi quan trọng phù hợp thông qua các thực hiện để xây dựng từ điển trích xuất mã thực thi của tập tin.

Thông qua việc thực nghiệm và quan sát sự thay đổi các chỉ số độ đo khi sử dụng mã thực thi tăng dần, các độ đo có sự thay đổi nhiều để đánh giá khi bước nhảy các ngưỡng số lượng opcode được chọn là 5. Mặt khác, trong nghiên cứu [26], 14 mã thực thi thường xuyên nhất trong kiến trúc Intel được chọn để xây dựng các mô hình phát hiện mã độc nhóm tác giả đề xuất.Vì vậy, không mất tính tổng quát, để thực nghiệm đánh giá sự ảnh hưởng lớn của số lượng mã thực thi lựa chọn trên kiến trúc MIPS và so sánh sự phù hợp trong trường hợp sự dụng ngưỡng tưng tự với kiến trúc Intel như nghiên cứu [26], NCS xem xét lựa chọn số lượng mã thực thi có mức độ quan trọng cao nhất theo 9 kịch bản thử nghiệm gồm lần lượt 5, 10, 14, 15, 20, 25, 30, 35, 40 mã thực thi có mức độ cao nhất trong Bảng 3.4 kết hợp với ba thuật toán học máy RF, SVM, NB. Kết quả phát hiện mã độc IoT trên kiến trúc MIPS thể hiện trong Bảng 3.6, Bảng 3.7 và Bảng 3.8. Khi số lượng mã thực thi quan trọng hàng đầu được lựa chọn là 20 thì mô hình xây dựng dựa trên thuật toán học máy RF cho độ chính xác cao nhất là 99,8% và F1-Weight là 99,8%.

Bảng 3.6. Kết quả phát hiện mã độc với các mã thực thi có mức độ quan trọng cao nhất sử dụng phương pháp 2-gram.

Số lượng mã thực thi	RF			SVM			NB		
	Accuracy (%)	**F1-Weight (%)**	**Thời gian huấn luyện (giây)**	**Accuracy (%)**	**F1-Weight (%)**	**Thời gian huấn luyện (giây)**	**Accuracy (%)**	**F1-Weight (%)**	**Thời gian huấn luyện (giây)**
5	50,6	36,1	**2,347**	50,3	35,4	**6,287**	49,8	33,7	**1,894**
10	71,9	69,7	2,598	60,9	54,3	8,107	52,8	40,0	1,944
14	98,7	98,7	2,673	98,0	98,0	9,195	56,6	46,6	1,946
15	99,0	99,0	2,866	98,6	98,6	10,294	90,7	90,7	1,970
20	**99,8**	**99,8**	3,746	98,7	98,7	12,584	94,3	94,3	2,346
25	99,7	99,7	4,523	99,1	99,1	15,876	95,2	95,2	2,896
30	99,7	99,7	5,268	**99,3**	**99,3**	20,425	**96,1**	**96,1**	3,805
35	99,6	99,6	8,231	99,2	99,2	29,129	96,0	96,0	4.012
40	99,5	99,5	12,07	99,2	99,2	47,642	96,0	96,0	4,494

Bảng 3.7. Kết quả phát hiện mã độc với các mã thực thi có mức độ quan trọng cao nhất sử dụng phương pháp 3-gram.

Số lượng mã thực thi	RF			SVM			NB		
	Accuracy (%)	**F1-Weight (%)**	**Thời gian huấn luyện (giây)**	**Accuracy (%)**	**F1-Weight (%)**	**Thời gian huấn luyện (giây)**	**Accuracy (%)**	**F1-Weight (%)**	**Thời gian huấn luyện (giây)**
5	50,6	34,4	2,251	50,2	34,2	7,503	50,3	33,9	1,850
10	70,9	68,4	3,457	58,5	49,9	14,494	53,0	40,3	2,056
14	98,7	98,7	4,711	98,0	98,0	17,891	58,1	49,2	2,721
15	99,7	99,7	6,892	98,0	98,0	21,580	95,1	95,0	2,971

20	**99,8**	**99,8**	11,771	98,4	98,4	54,504	97,0	97,0	5,460
25	99,7	99,7	16,865	98,9	98,9	129,231	97,3	97,3	8,128
30	99,7	99,7	25,038	**99,1**	**99,1**	239,36	**98,1**	**98,1**	16,817
35	99,6	99,6	43,128	99,0	99,0	412,433	98,0	98,0	56,198
40	99,5	99,5	121,231	99,0	99,0	881,169	98,0	98,0	129,173

Bảng 3.8. Kết quả phát hiện mã độc với các mã thực thi có mức độ quan trọng cao nhất sử dụng phương pháp 4-gram.

Số lượng mã thực thi	**RF**			**SVM**			**NB**		
	Accuracy (%)	**F1-Weight (%)**	**Thời gian huấn luyện (giây)**	**Accuracy (%)**	**F1-Weight (%)**	**Thời gian huấn luyện (giây)**	**Accuracy (%)**	**F1-Weight (%)**	**Thời gian huấn luyện (giây)**
5	50,3	34,2	2,292	50,2	34,2	8,905	50,2	33,5	1,999
10	69,7	66,8	4,966	57,7	48,5	22,302	53,6	40,8	2,239
14	98,2	98,2	8,348	97,2	97,2	20,301	95,0	95,0	3,531
15	99,7	99,7	13,802	97,9	97,9	48,631	97,0	97,0	5,274
20	99,8	99,8	34,968	98,4	98,4	217,923	98,5	98,5	15,76
25	**99,8**	**99,8**	154,768	98,9	98,9	765,908	98,7	98,7	187,875
30	99,7	99,7	377,251	**99,1**	**99,1**	1470,333	**98,9**	**98,9**	427,198
35	99,7	99,7	709,976	99,0	99,0	2810,123	98,8	98,8	1235,632
40	99,6	99,6	1825,891	99,0	99,0	6548,703	98,8	98,8	3444,284

Kết quả thu được từ các Bảng 3.6, 3.7 và 3.8 đã chỉ ra rằng việc lựa chọn số lượng mã thực thi để xây dựng từ điển trích xuất đặc trưng chuỗi mã thực thi cho các tập tin có thể ảnh hưởng nhiều đến hiệu quả mô hình phát hiện mã độc IoT. Khi lựa chọn quá ít mã thực thi để xây dựng từ điển trích xuất chuỗi đặc trưng mã thực thi sẽ làm mất nhiều thông tin quan trọng về tập tin, chuỗi mã thực thi không đại diện được cho tập tin khi xây dựng các mô hình phát hiện mã độc, độ chính xác trong phát hiện mã độc IoT thấp. Ngược lại, khi lựa chọn quá nhiều mã thực thi khác nhau để thu thập chuỗi đặc trưng mã thực thi cho các tập tin sẽ làm tăng độ dài chuỗi mã thực thi đại diện cho từng tập tin điều này sẽ không loại bỏ được các đặc trưng gây nhiễu. Khi đó, mô hình sẽ giảm độ chính xác và tăng thời gian huấn luyện và xử lý đối với từng tập tin thực thi. Vì vậy, trong bài toán phát hiện mã độc trên thiết bị IoT tài nguyên hạn chế luận án tập trung nghiên cứu, cần lựa chọn số lượng mã thực thi phù hợp để xây dựng từ điển trích xuất chuỗi mã thực thi trong xây dựng mô hình phát hiện đảm bảo hiệu quả về độ chính xác, thời gian và tài nguyên mô hình phát hiện mã độc IoT sử dụng.

Cụ thể, mô hình phát hiện sử dụng thuật toán học máy RF cho kết quả phát hiện mã độc IoT đơn kiến trúc trên kiến trúc MIPS tốt nhất so với các mô hình phát hiện sử dụng các thuật máy học khác được thử nghiệm với độ chính xác là 99,8% cho cả 2-gram, 3-gram và 4-gram khi lựa chọn 20 mã thực thi. Bên cạnh đó, thời gian huấn luyện của các mô hình là khác nhau. Mô hình sử dụng thuật toán NB là mô hình sử dụng ít thời gian huấn luyện nhất. Mô hình sử dụng thuật toán RF vượt trội đối với số lượng mã thực thi nhỏ nhưng nó tiêu tốn bộ nhớ lớn hơn và cần nhiều thời gian tính toán để huấn luyện mô hình. Trong trường hợp tương tự, SVM cần nhiều thời gian tính toán hơn để huấn luyện do cần bộ nhớ lớn. Bằng cách loại bỏ các mã thực thi không liên quan khỏi tập mã thực thi, thời gian phát hiện của các thuật toán học máy và độ phức tạp của không gian có thể giảm đáng kể và có thể tạo ra một mô hình phát hiện mã độc IoT tổng quát hơn. Do đó, một tập đặc trưng tối ưu là cần thiết để xây dựng các mô hình phát hiện mã độc IoT dựa trên học máy hiệu quả.

Bên cạnh đó, thời gian phát hiện trung bình các lần thử nghiệm và kích thước của các mô hình phát hiện mã độc IoT dựa trên học máy khi lựa chọn 20 mã thực thi có mức độ quan trọng cao nhất hiện thể hiện trong Bảng 3.9.

Bảng 3.9. Thời gian phát hiện trung bình và kích thước mô hình phát hiện mã độc IoT kiến trúc MIPS sử dụng 20 mã thực thi có mức độ quan trọng cao nhất.

Thuật toán	**RF**		**SVM**		**NB**	
Phương pháp n-gram	**2-gram**	**3-gram**	**2-gram**	**3-gram**	**2-gram**	**3-gram**
Thời gian phát hiện (giây)	0,0267	0,0309	0,0190	0,0251	0,0101	0,0102
Kích thước mô hình (MB)	1,0	1,3	21,4	1228	0,04	0,63

Từ kết quả thử nghiệm có thể thấy rằng các mô hình áp dụng các phương pháp n-gram với *n* tăng dần thì thời gian phát hiện và kích thước của mô hình có xu hướng tăng đối với tất cả các thuật toán học máy đã thử nghiệm. Thời gian phát hiện và kích thước của mô hình sử dụng thuật toán NB là nhỏ nhất do tính đơn giản và được thiết kế dựa trên xác suất. Thời gian phát hiện của mô hình sử dụng thuật toán RF là nhiều nhất do cần tính toán trên nhiều cây quyết định. Kích thước của mô hình phát hiện sử dụng thuật toán SVM là lớn nhất do phụ thuộc vào số lượng đặc trưng và số lượng điểm dữ liệu huấn luyện. So sánh với các nghiên cứu NCS đã khảo sát trong luận án, thời gian và kích thước của mô hình phát hiện khi áp dụng phương pháp 2-gram cho độ chính xác cao, kích thước mô hình và thời gian phát hiện phù hợp với môi trường IoT. Vì vậy, mô hình phát hiện luận án đề xuất có khả năng tích hợp trong các giải pháp phát hiện mã độc trong môi trường thiết bị IoT tài nguyên hạn chế.

3.3.5. So sánh hiệu quả của mô hình phát hiện mã độc IoT luận án đề xuất với các mô hình khác có liên quan

- Thực nghiệm so sánh hiệu quả với 3 mô hình phát hiện mã độc IoT dựa trên chuỗi mã thực thi trên cùng một tập dữ liệu như của NCS thu thập, cùng một môi trường thiết bị:

Các mô hình phát hiện mã độc IoT dựa trên học máy và đặc trưng chuỗi lời gọi hệ thống gồm: Nghiên cứu của nhóm tác giả Ding [125] sử dụng 400 đặc trưng mã thực thi có trọng số (information gain) cao nhất, phương pháp của nhóm tác giả Trần Nghi Phú [117] sử dụng phương pháp quy hoạch động C500-CFG kết hợp với phương pháp trích chọn đặc trưng n-gram và phương pháp sử dụng "từ điển opcode" đầy đủ gồm tất cả các mã thực thi trên nền tảng MIPS khi chưa áp dụng phương pháp trích chọn luận án đề xuất. Các mô hình học máy này đã chứng minh được hiệu quả trong phát hiện mã độc IoT dựa trên đặc trưng chuỗi mã thực thi trong thời gian qua. Các mô hình phát hiện trên được luận án thực nghiệm lại trên cùng một tập dữu liệu, môi tường để so sánh, đánh giá.. Kết quả so sánh được thể hiện trong Bảng 3.10.

Bảng 3.10. Kết quả so sánh mô hình đề xuất với các mô hình phát hiện mã độc IoT dựa trên chuỗi mã thực thi trên cùng một tập dữ liệu.

Phương pháp n-gram sử dụng	**Độ đo**	**Mô hình Ding đề xuất**	**Mô hình Phú đề xuất**	**Mô hình sử dụng "từ điển opcode" đầy đủ**	**Mô hình luận án đề xuất**
2-gram	Accuracy (%)	97,2	99,0	97,7	99,8
	F1-Score (%)	96,8	98,6	97,6	99,7
	Thời gian phát hiện (giây)	>40	40	0,145	0,0267
3-gram	Accuracy (%)	95,9	98,7	96,5	99,8
	F1-Score (%)	95,6	98,1	96,4	99,7
	Thời gian phát hiện (giây)	> 40	40	0,455	0,0309

Kết quả so sánh cho thấy rằng mô hình luận án đề xuất đã cho kết quả phát hiện mã độc IoT có độ chính xác tốt nhất với các độ đo accuracy là 99,8% và F1-score là 98,7% trong cả hai trường hợp sử dụng phương pháp 2-gram và 3-gram. Kết quả độ đo đã vượt trội hơn nhiều so với phương pháp sử dụng Information Gain để lựa chọn đặc trưng mã thực thi của Ding [125] đã đề xuất. Bên cạnh đó, với việc sử dụng số lượng mã thực thi ít hơn các nghiên cứu khác cũng giúp việc trích xuất và phát hiện mã độc IoT nhanh hơn các mô hình còn lại do hai mô hình còn lại do hạn chế về thời gian và tài nguyên khi xử lý các tập tin có kích thước lớn.

- Đánh giá khả năng phát hiện đối các tập tin không thu thập được đặc trưng chuỗi lời gọi hệ thống từ phân tích động:

Nhằm hướng đến mục tiêu ứng dụng các mô hình luận án đề xuất trong xây dựng giải pháp phát hiện mã độc IoT toàn diện cho các tập tin thực thi. Thực nghiệm tiến hành đánh giá khả năng phát hiện các tập tin không hiệu quả đối với mô hình đề xuất trong chương 2 của luận án này, hai tập tin mã độc phục vụ đánh giá trong chương 2 và chưa có trong tập dữ liệu huấn luyện của mô hình chương 3 có mã hash SHA256 là: "*e9e0fef27f91e1eb8d9a1b937959764fcb141900a3e702bbd58c3384dda35149*" và "*ef41b244e082286231848befd143283cf4450244b59f596dae75f6dfaecc84a6*". Thông tin hai tập tin trên được thu thập từ nguồn VirusShare và được các antivirus gán nhãn là "mã độc" thể hiện trong hình 3.9.

e9e0fef27f91e1eb8d9a1b937959764fcb141900a3e702bbd58c3384dda35149

VirusShare info last updated 2019-01-15 23:51:18 UTC

MD5	003b20e9a44fdc38fc8c763f4acc52f9
SHA1	dcc3a802f363233928f39ed5c96c4e15cb39e27f
SHA256	e9e0fef27f91e1eb8d9a1b937959764fcb141900a3e702bbd58c3384dda35149
SSDeep	3072:5tUXxOK6/6bzimS05letJ8aweG61l1Y5URxRGqJkMipPn:HUX96/1m/5letJ8aAm5URxRGqJkMipPn
Size	182,200 bytes
File Type	ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, not stripped
Mime Type	application/x-executable
Extension	o
TrID	ELF Executable and Linkable format (generic) (100.0%)

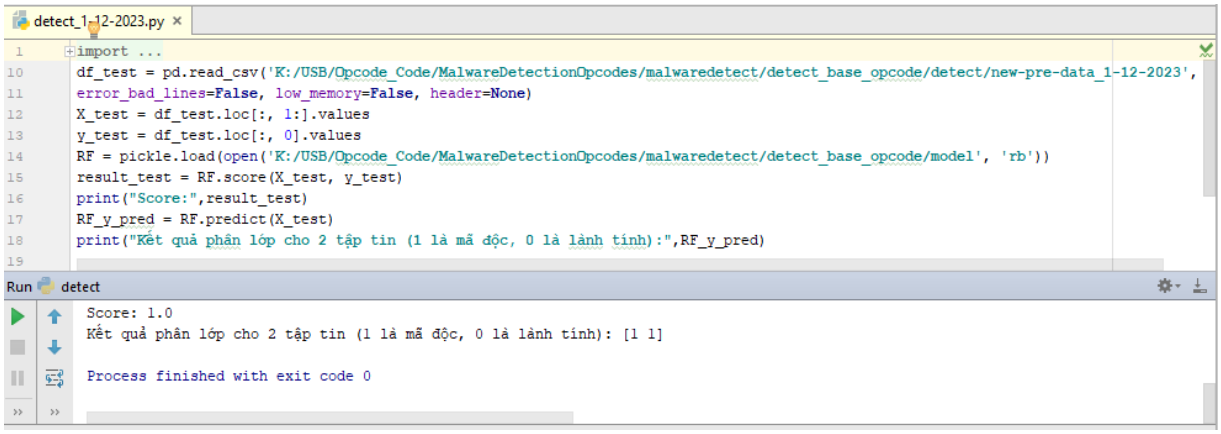
ef41b244e082286231848befd143283cf4450244b59f596dae75f6dfaecc84a6

VirusShare info last updated 2017-08-27 14:10:28 UTC

MD5	d19d5fd8abceddd07506a18829f9dd0c
SHA1	1fd4a6c0a4bbe058b946371e4b8220a32686648d
SHA256	ef41b244e082286231848befd143283cf4450244b59f596dae75f6dfaecc84a6
SSDeep	3072:ndu1kGeNGYRXgVzj0YD56j3HyfiNTqLoJmAlh:nE1gXVgJ0M54HyfiNTqLoJmAlh
Size	156,757 bytes
File Type	ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, not stripped
Mime Type	application/x-executable
Extension	o
TrID	ELF Executable and Linkable format (generic) (100.0%)

Hình 3.9. Thông tin 2 tập tin mã độc phục vụ đánh giá khả năng phát hiện của mô hình sau khi huấn luyện

Kết quả mô hình phát hiện đề xuất đã phân lớp 2 tập tin chính xác là "mã độc" như hình 3.10 dưới đây:



Hình 3.10. Kết quả phân lớp 2 tập tin dựa trên mô hình RF.

Từ thực nghiệm đã chứng minh mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng thu thập từ phân tích tĩnh tập tin đã khắc phục hạn chế và hỗ trợ tốt đối với trường hợp mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng thu thập từ phân tích động đề xuất trong chương 2 của luận án trong một số trường hợp. Đây là cơ sở để xây dựng các giải pháp phát hiện mã độc IoT dựa trên học máy toàn diện trong thực tế. Các mô hình phát hiện mã độc IoT đơn kiến trúc dựa trên đặc trưng thu thập từ phân tích động và phân tích tĩnh có thể hỗ trợ, khắc phục các hạn chế của nhau, giúp giải pháp có thể phát hiện đối với tất cả các trường hợp phân tích một tập tin thực thi.

- Đánh giá sự tương đồng về độ chính xác, thời gian phát hiện của mô hình đề xuất với các mô hình phát hiện sử dụng ít đặc trưng mã thực thi khác:

Việc đánh giá sự tương đồng về độ chính xác, thời gian phát hiện trung bình của mô hình sau khi trích xuất đối với một tập tin thực thi nhằm xem xét sự phù hợp của phương pháp luận án đề xuất với các nghiên cứu liên quan đã khảo sát trong chương 1 về giảm số lượng đặc trưng mã thực thi trong xây dựng mô hình phát hiện. Hiệu quả của mô hình đề xuất khi so sánh với các nghiên cứu khác có liên quan được thể hiện trong Bảng 3.11.

Bảng 3.11. So sánh hiệu quả các mô hình phát hiện sử dụng ít đặc trưng mã thực thi.

Tác giả	**Tập dữ liệu thử nghiệm**	**Số lượng đặc trưng sử dụng**	**Phương pháp học máy**	**Accuracy/ F-measure (%)**	**Thời gian phát hiện (giây)**
Ding [35]	4.600 tập tin mã độc và 4.600 tập tin lành tính trên Windows	400 n-gram có IG cao nhất	Deep Belief Network	95,8/-	-
Kang [11]	1.260 tập tin mã độc và 1.260 tập	56.850 n-gram opcodes (6-gram)	SVM	-/98	0,32

	tin lành tính trên Android	55.856 n-gram opcodes (8-gram)	RF	-/97	0,02
		51.845 n-gram opcodes (5-gram)	NB	-/90	5,02
		37.992 n-gram opcodes (4-gram)	PART	-/96	0,01
Wan [111]	111.610 tập tin mã độc và 111.353 tập tin lành tính ELF	Độ dài chuỗi là 384 và 4-gram	SVM	99,9/-	0,001
		Độ dài chuỗi là 128 và 4-gram	kNN	99,7-	0,53
		Độ dài chuỗi là 768 và 7-gram	NB	99,7/-	0,1
		Độ dài chuỗi là 512 và 4-gram	MLP	99,9/-	0,01
Darabian [46]	247 tập tin mã độc và 269 tập tin lành tính trên ARM	36 đặc trưng opcodes	SVM	99,5/99,5	-
			RF	98,9/98,9	-
			DT	99,8/99,8	-
			kNN	99,4/99,4	-
			AdaBoost	99,5/99,5	-
			MLP	99,1/99,1	-
Yewale [8]	100 tập tin mã độc và lành tính trên Windows	20 opcodes xuất hiện nhiều nhất	RF	97/-	-
			SVM	93/-	-
			DT	80/-	-
			Boost	87/-	-
Mô hình đề xuất	4.511 tập tin mã độc và 4.393 tập tin lành tính trên MIPS	20 opcodes quan trọng nhất	RF	99,8/-	0,02
		30 opcodes quan trọng nhất	NB	96,1/-	0,01
		30 opcodes quan trọng nhất	SVM	99,3/-	0,02

Kết quả cho thấy mô hình sử dụng số lượng mã thực thi ít nhất nhưng độ chính xác và thời gian phát hiện vẫn phù hợp trong môi trường IoT khi so sánh với các mô hình khác đã công bố.

3.4. Kết luận chương 3

Trong chương này, đặc trưng chuỗi mã thực thi của các tập tin trên thiết bị IoT đã được sử dụng và cho thấy hiệu quả tốt trong xây dựng các mô hình phát hiện mã độc IoT dựa trên học máy. Luận án đã đề xuất phương pháp thu thập và lựa chọn đặc trưng chuỗi mã thực thi của tập tin thực thi trên môi trường IoT phục vụ xây dựng mô hình phát hiện mã độc IoT đơn kiến trúc hiệu quả. Các thử nghiệm để đánh giá hiệu quả của phương pháp và đưa ra được "từ điển opcode" hiệu quả cho mô hình phát hiện mã độc IoT trên kiến trúc MIPS với độ chính xác cao nhất là 99,8%. Bên cạnh đó, với việc áp dụng các thuật toán học máy truyền thống giúp giảm độ phức tạp tính toán và thời gian huấn luyện và sử dụng mô hình trong môi trường IoT hạn chế tài nguyên. Mặt khác, các

thực nghiệm đã chứng minh mô hình phát hiện mã độc IoT đã đề xuất dựa trên phân tích tĩnh có thể giải quyết hiệu quả đối với trường hợp các tập tin không thể thu thập thông tin từ phân tích động đã trình bày trong chương 2 của luận án. Với sự phát triển của các mã độc IoT, đây là cơ sở để xây dựng các giải pháp toàn diện trong phát hiện mã độc IoT đơn kiến trúc dựa trên đặc trưng của tập tin thực thi trong thực tế.

Trong phân tích mã độc nói chung và phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng tập tin thu thập từ phân tích động hoặc phân tích tĩnh nói riêng, việc thu thập, trích xuất các đặc trưng động và đặc trưng tĩnh vẫn còn khó khăn trong các trường hợp mã độc sử dụng các kỹ thuật phức tạp cụ thể như kỹ thuật gây rối, che giấu, đa hình, đa nền tảng,... Các mô hình đã đề xuất trong nội dung chương 2 và chương 3 đã có khả năng phát hiện hiệu quả mã độc IoT đơn kiến trúc, tuy nhiên đối với sự phát triển của mã độc trên các thiết bị IoT đa dạng, việc phát hiện các mã độc trên kiến trúc mới và các biến thể mã độc từ các kiến trúc phổ biến trước đây là rất cần thiết. Vì vậy, để hướng đến giải quyết bài toán tổng thể của luận án, mô hình phát hiện mã độc IoT dựa trên học máy được nghiên cứu mở rộng để giải quyết các vấn đề phát hiện mã độc IoT đa kiến trúc và dự đoán mã độc zero-day hoạt động đa kiến trúc trên thiết bị IoT trong nội dung tiếp theo của luận án.

Ý tưởng tiếp cận và các kết quả nghiên cứu trình bày trong chương này đã được NCS công bố tại hội thảo và tạp chí:

- "*Static Feature Selection for IoT Malware Detection*", Hội thảo Nghiên cứu ứng dụng mật mã và an toàn thông tin năm 2022 (Kỷ yếu trên Tạp chí Journal of Science and Technology on Information Security, Special Issue CS (15), 2022).

- "*An Efficient Algorithm to Extract Control Flow-Based Features for IoT Malware Detection*", The Computer Journal, 2020 (SCIE index, Q2).

CHƯƠNG 4: XÂY DỰNG MÔ HÌNH PHÁT HIỆN MÃ ĐỘC IOT ĐA KIẾN TRÚC HIỆU QUẢ SỬ DỤNG ĐẶC TRƯNG CHUYỂN ĐỔI CHÉO KIẾN TRÚC VI XỬ LÝ

4.1. Mở đầu

Về mặt nguồn gốc, Allix và cộng sự [63] đã chỉ ra rằng hầu hết mã độc hiện nay được sinh ra bằng cách sao chép mã nguồn theo các hướng dẫn sẵn có hoặc tạo ra các biến thể từ một mã độc ban đầu. Cùng một mã nguồn mã độc có thể được biên dịch trên nhiều kiến trúc vi xử lý khác nhau để trở thành các mẫu mã độc hoạt động trên nhiều nền tảng kiến trúc vi xử lý. Các dòng mã độc hoạt động đa hệ điều hành và đa kiến trúc vi xử lý xuất hiện ngày càng nhiều và ngày càng nguy hiểm hơn. Các kiến trúc của thiết bị IoT có nguyên lý và cơ chế hoạt động khác nhau, các dòng mã độc có khả năng hoạt động trên thiết bị IoT này sẽ được thiết kế để có thể hoạt động trên nhiều kiến trúc vi xử lý khác nhau một cách linh hoạt và hiệu quả. Vì vậy, việc phát hiện mã độc IoT đa kiến trúc cần có những nghiên cứu và phương pháp phù hợp để nâng cao khả năng phát hiện và ngăn chặn mã độc phá hoại trên thiết bị IoT.

Theo khảo sát của nghiên cứu sinh, trong thời gian qua, hai hướng nghiên cứu phổ biến trong phát hiện mã độc IoT đa kiến trúc dựa trên học máy gồm: Dựa trên kỹ thuật trích chọn đặc trưng đa kiến trúc và dựa trên mô hình phát hiện mã độc IoT chéo kiến trúc. Việc xây dựng các môi trường phân tích động để trích xuất và lựa chọn đặc trưng của tập tin đa kiến trúc đang đặt ra nhiều khó khăn, thách thức. Theo các tìm hiểu của NCS, chưa có sandbox được tích hợp sẵn để có thể thực hiện thu thập các hành vi của tất cả mã độc hoạt động đa kiến trúc vi xử lý trên thiết bị IoT, đặc biệt các mã độc hoạt động trên kiến trúc vi xử lý mới. Bên cạnh đó, hiệu quả từ việc sử dụng phân tích tĩnh để trích xuất và lựa chọn đặc trưng đa kiến trúc đã đem lại hiệu quả trong phát hiện mã IoT đa kiến trúc. Tuy nhiên, các mô hình phát hiện mã độc IoT đa kiến trúc xây dựng dựa trên các đặc trưng này chưa đánh giá khả năng phát hiện, dự báo mã độc mới, mã độc zero-day và giải quyết vấn đề mất cân bằng dữ liệu khi huấn luyện các mô hình phát hiện trên các kiến trúc vi xử lý khác nhau, đặc biệt các kiến trúc vi xử lý mới.

Vì vậy, hướng tiếp cận tăng cường tập dữ liệu đa kiến trúc và xây dựng mô hình phát hiện chéo kiến trúc vi xử lý có thể giải quyết được các hạn chế nêu trên. Theo các khảo sát mô hình phát hiện mã độc IoT đa kiến trúc dựa trên huấn luyện và phát hiện chéo kiến trúc đã trình bày trong mục 1.3.3 của luận án, các nghiên cứu còn tồn tại một số hạn chế sau đây:

Thứ nhất, các độ đo về tiêu chí độ chính xác của mô hình phát hiện mã độc IoT đa kiến trúc còn thấp trong các trường hợp phát hiện chéo kiến trúc vi xử lý giữa tập huấn luyện và tập đánh giá. Đặc biệt, trong các trường hợp huấn luyện trên tập dữ liệu của nền tảng thiết bị IoT phổ biến sử dụng trong các ứng dụng nhúng và thiết bị mạng như

MIPS để phát hiện mã độc hoạt động đa nền tảng trên các thiết bị IoT sử dụng kiến trúc khác chỉ đạt độ chính xác dưới 60%.

Thứ hai, tập dữ liệu thử nghiệm thu thập từ các thiết bị IoT và kiến trúc vi xử lý khác nhau còn hạn chế về số lượng và chưa có sự cân bằng giữa các kiến trúc phổ biến như Intel với các kiến trúc khác như MIPS, ARM, SPARC, PowerPC,…

Thứ ba, số lượng các nghiên cứu trong việc dự đoán, phát hiện mã độc zero-day trên các kiến trúc vi xử lý của thiết bị IoT mới còn hạn chế, các phương pháp, mô hình phát hiện đã đưa ra gặp khó khăn khi áp dụng các tập dữ liệu IoT lớn và chưa hiệu quả với kiến trúc CPU mới, kiến trúc CPU của các thiết bị IoT có nhiều đặc trưng khác biệt.

Thứ tư, các mô hình phát hiện mã độc IoT đa kiến trúc đã được xây dựng trên các đặc trưng tĩnh và đặc trưng động phụ thuộc nhiều vào các dòng thiết bị IoT và kiến trúc bộ vi xử lý phổ biến, nền tảng đã có nhiều số lượng mẫu và tri thức về mã độc trước đó.

Trong xây dựng mô hình học máy nói chung và xây dựng mô hình phát hiện mã độc IoT nói riêng, việc tăng cường dữ liệu sẽ góp phần quan trọng trong nâng cao hiệu quả của mô hình bởi vì một số lý do sau đây:

- *Tăng độ đa dạng*: Dữ liệu phong phú và đa dạng hơn có thể cung cấp thông tin bổ sung và nhiều đại diện hơn về các trường hợp khác nhau, giúp mô hình phát hiện mã độc IoT hiểu rõ hơn về tính đa dạng của dữ liệu thực tế.

- *Phòng tránh overfitting*: Tăng cường dữ liệu có thể giúp ngăn ngừa hiện tượng overfitting khi mô hình phát hiện mã độc IoT quá tập trung vào việc học mẫu riêng biệt trong tập dữ liệu huấn luyện, dẫn đến việc hiệu suất giảm khi xử lý dữ liệu mới.

- *Tăng cường khả năng tổng quát hóa*: Dữ liệu phong phú giúp mô hình phát hiện mã độc IoT có khả năng tổng quát hóa tốt hơn cho các trường hợp mới, các trường hợp chưa có trong dữ liệu huấn luyện.

- *Cải thiện độ chính xác*: Trong một số trường hợp, việc tăng cường dữ liệu có thể cải thiện đáng kể độ chính xác của mô hình phát hiện mã độc IoT, đặc biệt khi mô hình phát hiện đang gặp vấn đề với dữ liệu hạn chế hoặc không cân đối.

- *Xử lý các vấn đề thiếu cân bằng dữ liệu (imbalanced data):* Trong trường hợp dữ liệu không cân bằng giữa các lớp khác nhau, việc tăng cường dữ liệu có thể giúp cân bằng dữ liệu, cải thiện hiệu suất dự đoán trên các lớp thiếu dữ liệu.

Trong xây dựng mô hình phát hiện mã độc IoT đa kiến trúc dựa trên học máy, tăng cường dữ liệu không chỉ cải thiện hiệu suất của mô hình mà còn giúp tạo ra mô hình phát hiện tổng quát hơn và hạn chế ảnh hưởng bởi dữ liệu huấn luyện cụ thể. Từ đó, mô hình có thể học được các tri thức mới, các tri thức từ ngoài tập dữ liệu ban đầu. Để tăng cường tập dữ liệu ngoài thực hiện thông qua thu thập mẫu từ các hệ thống thông tin, việc ứng dụng các mô hình trí tuệ nhân tạo tạo sinh hay phương pháp chuyển đổi dữ liệu, tạo dữ liệu giả là cần thiết đối với các hạn chế trong thu thập dữ

liệu từ thiết bị IoT hạn chế tài nguyên và đa dạng nền tảng.

***Vì vậy**, để nâng cao hiệu quả trong phát hiện mã độc IoT đa kiến trúc dựa trên xây dựng mô hình phát hiện IoT chéo kiến trúc vi xử lý, chương 4 luận án đề xuất phương pháp chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc vi xử lý phục vụ xây dựng mô hình phát hiện mã độc IoT đa kiến trúc dựa trên các thuật toán học máy truyền thống nhằm tăng cường tập dữ liệu đa kiến trúc vi xử lý, tăng độ chính xác và khả năng phát hiện, dự báo mã độc zero-day trên thiết bị IoT sử dụng kiến trúc vi xử lý đa dạng. Mô hình đề xuất có thể thay thế việc cần kết hợp nhiều mô hình phát hiện mã độc IoT đơn kiến trúc để phát hiện mã độc cho một tập tin thực thi và giải quyết vấn đề hạn chế dữ liệu huấn luyện trong môi trường IoT đa dạng kiến trúc vi xử lý.*

4.2. Đề xuất mô hình chuyển đổi đặc trưng của tập tin thực thi

Với hướng tiếp cận xây dựng mô hình phát hiện mã độc IoT chéo kiến trúc vi xử lý, qua các khảo sát trong luận án, đặc trưng dạng chuỗi trích xuất từ phân tích tĩnh được đề xuất sử dụng để thực hiện xây dựng mô hình phát hiện mã độc IoT đa kiến trúc. Phương pháp chuyển đổi đặc trưng chuỗi mã thực thi thu thập từ phân tích tĩnh tập tin chéo kiến trúc vi xử lý được đề xuất để tăng cường tập dữ liệu đa kiến trúc phục vụ xây dựng mô hình phát hiện mã độc IoT. Việc chuyển đổi đặc trưng chuỗi mã thực thi chéo nền tảng kiến trúc giúp hình thức các tập dữ liệu đa kiến trúc vi xử lý, tạo ra mô hình có khả năng phát hiện mã độc cho đặc trưng đầu vào thuộc nhiều nền tảng kiến trúc khác nhau.

Tập các mã thực thi trên kiến trúc vi xử lý khác nhau của thiết bị IoT có sự khác biệt lớn về tên mã thực thi, chức năng và số lượng các mã thực thi. Một số lệnh trên các kiến trúc khác nhau là không tương đương nhau. Ví dụ lệnh MOV trên kiến trúc Intel và ARM không có lệnh tương đương trên kiến trúc MIPS. Sự khác biệt được trình bày trong Bảng 4.1.

Bảng 4.1. So sánh mã thực thi trên một số kiến trúc bộ vi xử lý của thiết bị IoT.

Kiến trúc vi xử lý	**Intel**	**MIPS**	**ARM**	**PowerPC**	**SPARC**
Đặc điểm kiến trúc	Sử dụng kiến trúc CISC (Complex Instruction Set Computer)	Sử dụng kiến trúc RISC (Reduced Instruction Set Computer)	Sử dụng kiến trúc RISC khác	Sử dụng kiến trúc RISC được IBM, Apple, Motorola cùng phát triển	Sử dụng kiến trúc RISC khác
Đặc điểm mã thực thi	Sử dụng một số mã thực thi đa năng, có thể được sử dụng để thực	Sử dụng một số mã thực thi đa năng nhưng ít hơn Intel	Mã thực thi thường sử dụng cho các lệnh đơn giản và sử	Sử dụng một số mã thực thi đa năng nhưng ít hơn Intel	Sử dụng một số mã thực thi đa năng

	hiện nhiều loại lệnh khác nhau		dụng trong các ứng dụng nhúng		
Số lượng tên mã thực thi[10]	Trên 2400 tên mã thực thi khác nhau tuỳ thuộc vào phiên bản kiến trúc	Khoảng 410 tên mã thực thi khác nhau tuỳ thuộc vào phiên bản kiến trúc	Trên 700 tên mã thực thi khác nhau tuỳ thuộc vào phiên bản kiến trúc	Khoảng 1410 tên mã thực thi khác nhau tuỳ thuộc vào phiên bản kiến trúc	Trên 200 tên mã thực thi khác nhau tuỳ thuộc vào phiên bản kiến trúc

Việc chuyển đổi các mã thực thi chéo kiến trúc vi xử lý đã được thực hiện thông qua xây dựng "ngôn ngữ trung gian" dựa trên Vex đã đem lại khả năng phát hiện chéo mã độc IoT trên các nền tảng kiến trúc vi xử lý [116]. Tuy nhiên, phương pháp sẽ gặp khó khăn và không hiệu quả đối với các kiến trúc vi xử lý có ít sự tương đồng và chênh lệch về số lượng mã thực thi. Các phương pháp xây dựng mô hình phát hiện mã độc IoT đơn kiến trúc sẽ không giải quyết được bài toán phát hiện mã độc IoT trên kiến trúc vi xử lý mới và mã độc hoạt động đa kiến trúc. Do đó, cần có phương pháp xử lý dữ liệu chuỗi mã thực thi giữa các kiến trúc vi xử lý để xác định các đặc trưng chuỗi mã thực thi tương đồng giữa các kiến trúc phục vụ quá trình huấn luyện mô hình phát hiện mã độc IoT đa kiến trúc.

Dựa vào cách tiếp cận đặc trưng mã thực thi thu thập từ phân tích tĩnh tập tin là các biểu diễn dạng chuỗi, việc xử lý dữ liệu chuỗi mã thực thi chéo kiến trúc để xác định sự tương đồng sẽ được thực hiện thông qua việc chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc. Điều này tương tự như việc dịch một "câu" từ ngôn ngữ này sang ngôn ngữ khác trong xử lý ngôn ngữ tự nhiên. Trong xử lý ngôn ngữ tự nhiên, việc xây dựng mô hình dịch được tiếp cận theo các phương pháp: Dịch từ sang từ (Word-level Translation), dịch câu sang câu (Phrase-level Translation), dịch dựa trên cấu trúc (Syntax-based Translation), dịch dựa trên ngữ cảnh (Context-based Translation) hoặc kết hợp các phương pháp trên. Mỗi phương pháp có ưu và nhược điểm riêng, tuỳ thuộc vào ngữ cảnh và mục tiêu của việc dịch để có thể lựa chọn phương pháp phù hợp. Đối với các cặp ngôn ngữ có ít dữ liệu song song thì các thuật toán học máy cần tận dụng tốt dữ liệu đơn ngữ để thực hiện việc dịch. Việc sử dụng học không giám sát để xây dựng các mô hình dịch trong ngôn ngữ tự nhiên đã được các nghiên cứu [43], [72] sử dụng và cho hiệu quả tốt. Hoặc tiếp cận theo hướng xây dựng một từ điển song ngữ giữa các cặp ngôn ngữ không có dữ liệu song song trước đó thông qua việc căn chỉnh từ đơn ngữ trong không gian nhúng [1], [73].

Đối với chuyển đổi mã thực thi, theo khảo sát của NCS, hiện nay chưa có công bố

[10] *https://gitlab.com/ngoctoan/caimp/-/tree/main/Opcode%20Extraction?ref_type=heads*

nào về "từ điển ánh xạ mã thực thi" phục vụ chuyển đổi mã thực thi chéo kiến trúc vi xử lý. Vì vậy, việc chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc vi xử lý được luận án tiếp cận theo hướng chuyển đổi một chuỗi không có "từ điển ánh xạ mã thực thi" chung giữa hai kiến trúc vi xử lý nguồn, đích và dựa trên căn chỉnh chuỗi mã thực thi biểu diễn trong không gian nhúng. Việc căn chỉnh chuỗi mã thực thi biểu diễn trong không gian nhúng được thực hiện dựa trên ý tưởng biến đổi hình học biểu diễn chuỗi mã thực thi trên các phép biến đổi cơ sở như phép tịnh tiến, phép quay, phép biến đổi tỉ lệ, phép đối xứng, phép biến dạng. Có hai quan điểm về phép biến đổi hình học gồm: Biến đổi đối tượng (object transformation) là thay đổi tọa độ của các điểm mô tả nó theo một quy tắc nào đó và biến đổi hệ tọa độ (coordinate transformation) là tạo ra một hệ tọa độ mới để tất cả các điểm mô tả đối tượng sẽ được chuyển về hệ tọa độ mới. Luận án tiếp cận chuyển đổi đặc trưng chuỗi mã thực thi trong không gian nhúng theo các phép biến đổi đối tượng. Cơ sở biến đổi hình học đối tượng là phép biến đổi Affine.

Định nghĩa 4.1. *Phép biến đổi hai chiều là phép biến đổi điểm P trong một mặt phẳng thành điểm Q có tọa độ mới theo một quy luật nào đó.*

Về mặt bản chất, một phép biến đổi điểm trong không gian hai chiều là một ánh xạ T được định nghĩa: T [P(x, y)] ↦ P'(x', y'). Ví dụ đối với không gian 2 chiều, các phép biến đổi cơ sở được thể hiện như phép tịnh tiến, phép tỉ lệ, phép biến dạng, phép xoay,.. Mọi phép biến đổi phức tạp đều có thể tạo thành từ các phép biến đổi cơ sở. Phép biến đổi affine trong không gian vectơ có một số tính chất sau đây:

- *Tính chất 1: Bảo toàn đường thẳng:* Một phép biến đổi affine giữ nguyên tính thẳng hàng của các điểm.

- *Tính chất 2: Bảo toàn tỷ lệ đoạn thẳng*: Một điểm nằm giữa hai điểm còn lại theo tỷ lệ cho trước, thì tỷ lệ này được bảo toàn sau phép biến đổi affine.

- *Tính chất 3: Kết hợp tuyến tính:* Tổng hợp của hai phép biến đổi affine là một phép biến đổi affine khác.

- *Tính chất 4: Không bảo toàn gốc toạ độ*: Các quan hệ tương đối giữa các điểm được giữ nguyên nhưng gốc tọa độ có thể được dời đi khi phép biến đổi affine thêm vectơ tịnh tiến.

- *Tính chất 5*: *Không bảo toàn khoảng cách:* Phép biến đổi affine không bảo toàn độ dài đoạn thẳng giữa hai điểm. Các điểm có thể bị co giãn, kéo dãn, hoặc tịnh tiến.

- *Tính chất 6: Không bảo toàn góc:* Góc giữa hai vectơ có thể thay đổi.

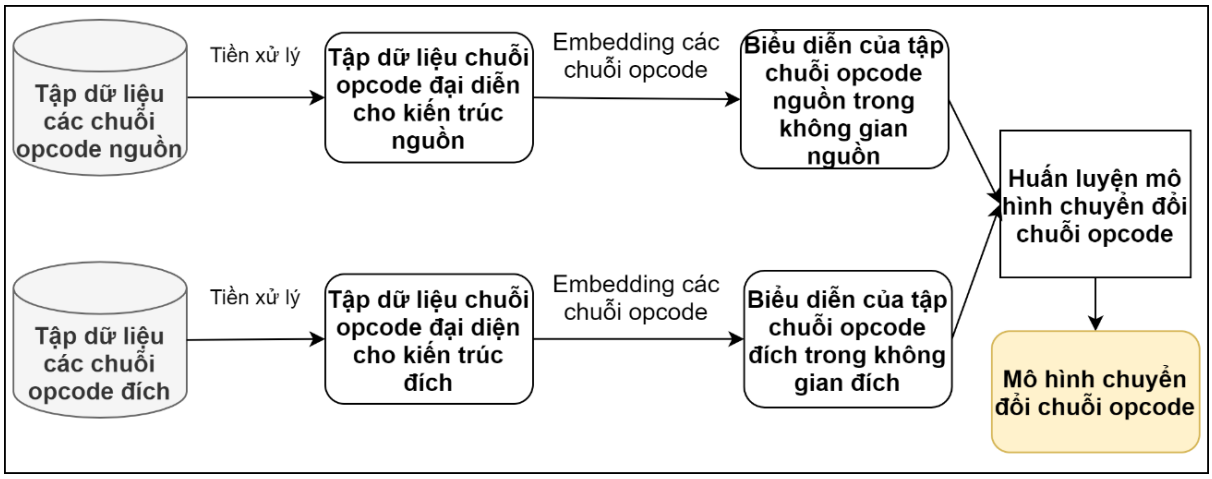
- *Tính chất 7: Biến đổi không gian con:* Phép biến đổi affine đưa các không gian con (như đường thẳng, mặt phẳng) thành các không gian con khác có cùng số chiều.

Thông thường chúng ta cần phải thực hiện một phép biến đổi phức tạp từ nhiều phép biến đổi cơ bản. Việc biến đổi một mã thực thi hoặc cụm mã thực thi được thực hiện dựa

trên các phép biến đổi tổng hợp từ các phép biến đổi affine cơ sở. Phép biến đổi chuỗi mã thực thi sử dụng trong luận án được định nghĩa như sau:

Định nghĩa 4.2. *Phép biến đổi chuỗi mã thực thi là phép ánh xạ toạ độ tập các vectơ biểu diễn mã thực thi trên không gian nhúng (embedding) nguồn thành tập các vectơ khác trong không gian nhúng đích.*

Phép biến đổi chuỗi mã thực thi được xây dựng dựa trên các phép biến đổi affine cơ sở. Quá trình xây dựng mô hình chuyển đổi chuỗi mã thực thi được thể hiện theo sơ đồ hình 4.1.



Hình 4.1. Quá trình xây dựng mô hình chuyển đổi chuỗi mã thực thi chéo kiến trúc.

4.2.1. Biểu diễn mã thực thi trong không gian nhúng

Quá trình bắt đầu với việc tiền xử lý tập dữ liệu các chuỗi mã thực thi thu thập được sau khi trích xuất từ các tập tin trên kiến trúc vi xử lý nguồn và đích. Các chuỗi mã thực thi của mỗi kiến trúc vi xử lý được ghép lại để trở thành các tập dữ liệu mã thực thi đại diện cho các kiến trúc vi xử lý. Điều này giống như tạo ra một "văn bản" trong ngôn ngữ tự nhiên từ các "câu" là các chuỗi mã thực thi. Sau đó, tập dữ liệu chuỗi mã thực thi đại diện cho mỗi kiến trúc được embedding trong không gian vectơ để tạo ra biểu diễn của tập chuỗi mã thực thi trong không gian tương ứng với mỗi kiến trúc vi xử lý. Trong xử lý ngôn ngữ tự nhiên, các phương pháp embedding "từ" phổ biến như Skip-gram, Continuous Bag-of-Words (CBOW), FastText,... đều có những ưu và nhược điểm khác nhau như Bảng 4.2.

Bảng 4.2. Ưu, nhược điểm của một số phương pháp biểu diễn từ trong không gian vectơ.

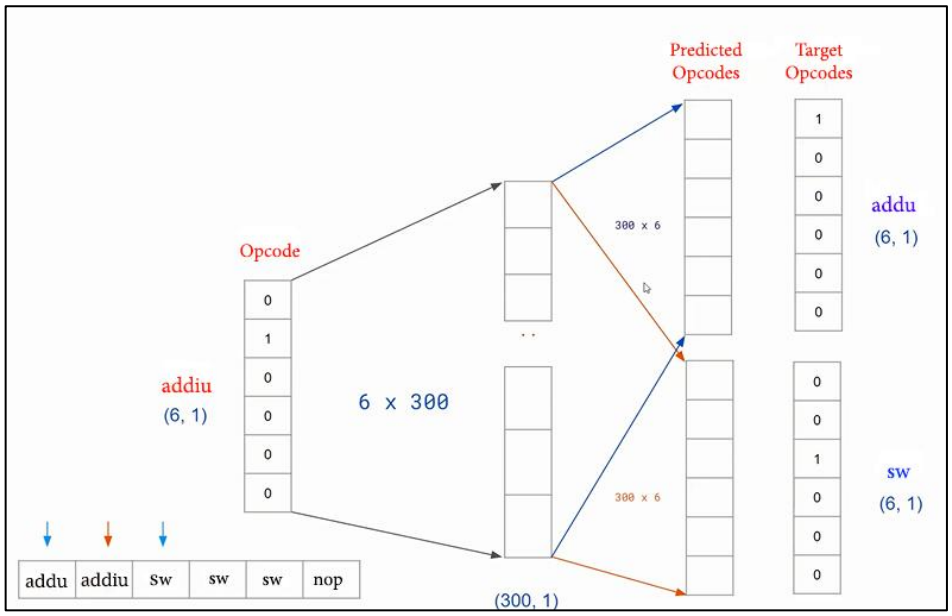
Phương pháp	**Skip-gram**	**CBOW**	**FastText**
Ưu điểm	- Hiệu quả trong việc biểu diễn các từ hiếm, có tần suất xuất hiện thấp. - Dễ dàng xử lý các từ đa nghĩa, từ có nhiều	- Huấn luyện nhanh hơn so với Skip-gram, đặc biệt với các từ phổ biến và có tần suất xuất hiện cao. - Thường tạo ra các	- Có thể tạo ra biểu diễn cho các từ không có trong từ điển bằng cách sử dụng các n-gram, do đó tăng khả năng xử lý các từ hiếm. - Hiệu quả trong việc xử lý

	ngữ cảnh khác nhau.	vectơ nhúng chất lượng tốt cho việc phân loại và dự đoán.	các từ có các thành phần n-gram khái quát.
Nhược điểm	- Yêu cầu nhiều dữ liệu huấn luyện để đạt được hiệu suất tốt, đặc biệt với các từ hiếm. - Hạn chế về mặt tính toán do cần học từng từ một.	- Không hiệu quả với các từ hiếm, có tần suất xuất hiện thấp. - Không xử lý tốt các từ đa nghĩa, từ có nhiều ngữ cảnh khác nhau.	- Tăng kích thước của từ điển và mô hình sẽ làm tăng độ phức tạp tính toán. - Khó khăn trong việc tìm hiểu ý nghĩa của các từ được hình thành từ các thành phần n-gram.

Lựa chọn giữa các phương pháp này thường phụ thuộc vào yêu cầu cụ thể của bài toán cũng như tính chất của dữ liệu. Skip-gram thích hợp cho các tác vụ đòi hỏi biểu diễn chi tiết cho các từ hiếm, trong khi CBOW có thể nhanh chóng học các từ phổ biến. FastText có thể hữu ích khi cần xử lý các từ không có trong từ điển thông thường bằng cách sử dụng thông tin từ n-gram. Vì vậy, luận án sử dụng giải pháp FastText [41] để thực hiện nhúng các chuỗi mã thực thi thành vectơ trong không gian.

FastText là một mô hình embedding từ trong xử lý ngôn ngữ tự nhiên được phát triển bởi Facebook AI Research (FAIR). FastText là một cải tiến của các phương pháp nhúng từ truyền thống như Word2Vec. FastText đã được sử dụng phổ biến trong nhiều ứng dụng bao gồm phân loại văn bản, gợi ý từ, dịch máy và nhiều tác vụ khác trong lĩnh vực xử lý ngôn ngữ tự nhiên. FastText sử dụng kỹ thuật nhúng từ dựa trên n-gram thông qua việc tạo ra các vectơ cho các từ dựa trên các đoạn văn bản con (n-gram), không chỉ sử dụng các từ đơn lẻ. Trong FastText, mỗi từ được biểu diễn dưới dạng một vectơ số học trong không gian *n* chiều. Cách tính toán các giá trị biểu diễn này thường thông qua sử dụng mô hình học sâu như Skip-gram, CBow, … Quá trình biểu diễn từ bắt đầu thông qua việc chia nhỏ từ thành các n-gram (các phần tử con liên tiếp có độ dài n) và sử dụng các vectơ n-gram này để tạo vectơ cho từ hoặc câu. Với mạng Skip-gram các từ có ngữ cảnh giống nhau sẽ được biểu diễn ở gần nhau trong không gian nhúng. Vì vậy, luận án lựa chọn FastText sử dụng mạng skip-gram để huấn luyện biểu diễn mã thực thi (opcode embedding) từ tập các chuỗi mã thực thi.

Ví dụ 4.1: Để biểu diễn một mã thực thi thông qua mạng Skip-gram được tiến hành theo hình 4.2. Mã thực thi “addiu” được biểu diễn trong không gian vectơ bởi một ma trận embedding 6x300 thông qua một mạng noron. Kết quả là hai từ láng giềng gần với mã thực thi “addiu” được xác định thông qua ngữ cảnh là “addu” và “sw”.



Hình 4.2. Biểu diễn mã thực thi trong không gian vectơ thông qua mô hình Skip-gram.

Quá trình xây dựng một vectơ biểu diễn mã thực thi trong FastText sử dụng mạng Skip-gram thường bao gồm các bước sau đây:

- *Bước 1: Xây dựng từ điển mã thực thi:* Từ tập dữ liệu các chuỗi mã thực thi trên kiến trúc vi xử lý, một từ điển mã thực thi được xây dựng dựa trên các mã thực thi duy nhất của kiến trúc vi xử lý tương ứng. Xây dựng các n-gram từ các mã thực thi trong từ điển.

- *Bước 2: Tạo dữ liệu huấn luyện*: Dựa trên kích thước cửa sổ trượt, số chiều của vectơ nhúng và các tham số khác để tạo các cặp mã thực thi hoặc cặp n-gram mã thực thi từ dữ liệu huấn luyện. Chi tiết gồm:

+ Xác định cửa sổ trượt: Đối với mỗi mã thực thi hoặc n-gram trong tập dữ liệu mã thực thi huấn luyện, chọn cửa sổ trượt để xác định ngữ cảnh của mã thực thi. Cửa sổ này xác định phạm vi các mã thực thi xung quanh mà mô hình sẽ cố gắng dự đoán ngữ cảnh.

+ Tạo các cặp mã thực thi: Với mỗi mã thực thi hoặc n-gram mã thực thi, tạo các cặp mã thực thi xung quanh theo cửa sổ trượt. Ví dụ mã thực thi "mov" có thể tạo các cặp "mov – sub" , "mov - s", " mov - jmp",… Mỗi cặp mã thực thi sẽ được sử dụng như một điểm dữ liệu để huấn luyện mô hình.

- *Bước 3: Huấn luyện mô hình:*

+ Các cặp mã thực thi được tạo từ bước 2 được đưa vào mô hình skip-gram để huấn luyện. Quá trình huấn luyện mô hình skip-gram gồm:

(1) Tạo One-Hot Encoding: Mỗi mã thực thi được biểu diễn bằng một vectơ one-hot với độ dài bằng kích thước của từ điển mã thực thi và chỉ có một phần tử bằng 1 ở vị trí tương ứng với mã thực thi đó.

(2) Huấn luyện mạng nơron: Mô hình Skip-gram sử dụng một mạng nơron để học

cách dự đoán các mã thực thi lân cận dựa trên mã thực thi đang xét. Mô hình sẽ dự đoán các mã thực thi lân cận dựa trên hàm softmax để tính xác suất của mã thực thi có thể là lận cận.

Dựa trên sự sai khác giữa dự đoán và thực tế, các trọng số của mạng nơ-ron sẽ được cập nhật theo nguyên tắc lan truyền ngược (backpropagation) để cải thiện dự đoán. Quá trình cập nhật trọng số dựa trên đạo hàm của hàm mất mát so với các trọng số để điều chỉnh các trọng số sao cho sai khác giữa dự đoán và thực tế là nhỏ nhất. Quá trình lan truyền ngược tính toán mức độ lỗi tại lớp đầu ra sau đó lan truyền ngược qua mạng nơ-ron để tính độ lỗi tại các tầng trước đó và cập nhật trọng số dựa trên đạo hàm của lỗi theo từng trọng số.

(3) Tạo vectơ biểu diễn: Sau khi huấn luyện xong, ma trận trọng số giữa đầu vào là mã thi và lớp ẩn là các vectơ biểu diễn cho các mã thực thi trong không gian vectơ. Lớp đầu vào có kích thước bằng số lượng mã thực thi trong từ điển và được biểu diễn bằng vectơ one-hot cho mỗi mã thực thi. Lớp ẩn có kích thước bằng số chiều của biểu diễn vectơ mã thực thi và chứa các biểu diễn của mã thực thi.

+ Cải thiện sự phù hợp của biểu diễn mã thực thi: Mô hình huấn luyện dựa trên skip-gram sẽ cố gắng cập nhật các vectơ biểu diễn mã thực thi sao cho mã thực thi hoặc n-gram mã thực có thể dự đoán được mã thực thi hoặc n-gram mã thực thi xung quanh trong ngữ cảnh tập dữ liệu mã thực thi huấn luyện.

+ Lặp lại quá trình huấn luyện: Quá trình huấn luyện được lặp lại nhiều lần thông qua dữ liệu huấn luyện để cải thiện độ chính xác của mô hình. Số lần lặp lại quá trình huấn luyện có thể thay đổi tùy thuộc vào bộ dữ liệu cụ thể, tỉ lệ học, kích thước của bộ từ điển, kích thước của bộ nhớ mô hình. Việc lựa chọn số lần huấn luyện thông qua siêu tham số “epoch” được điều chỉnh trong quá trình thử nghiệm của luận án.

- *Bước 4: Tạo vectơ biểu diễn mã thực thi*: Sau khi huấn luyện hoàn tất, mô hình sẽ tạo ra các vectơ biểu diễn cho mã thực thi. Vectơ này thường có kích thước cố định và chứa thông tin về ngữ cảnh của mã thực thi trong không gian nhiều chiều. Các mã thực thi được biểu diễn bằng các vectơ có số chiều xác định. Số chiều vectơ nhúng sẽ quyết định độ phức tạp của mô hình và khả năng biểu diễn các đặc trưng ý nghĩa của mã thực thi. Khi số chiều càng cao thì mô hình có khả năng biểu diễn càng chi tiết về mối quan hệ ý nghĩa học được giữa các mã thực thi. Việc lựa chọn số chiều phù hợp cần cân nhắc để đảm bảo sự cân bằng giữa độ phức tạp của mô hình và hiệu suất. Sau quá trình thử nghiệm và so sánh với hiệu quả của một số mô hình sử dụng trong ngôn ngữ tự nhiên, trong mô hình FastText sử dụng của luận án, mỗi mã thực thi được biểu diễn bằng một vectơ số học trong không gian 300 chiều.

Mô hình FastText sử dụng thông tin n-gram mã thực thi thông qua mô hình huấn

luyện skip-gram, trong đó mã thực thi được biểu diễn bằng tổ hợp các n-gram mã thực thi nhúng. Các n-gram mã thực thi nhúng này có thể đại diện cho các phần nhỏ hơn của mã thực thi nhưng không phải là n-gram cụ thể như trong mô hình n-gram truyền thống. Trong FastText, quá trình tạo mã thực thi từ các n-gram là một bước trong quá trình huấn luyện mô hình và không thể tái tạo chính xác mã thực thi từ các n-gram của nó sau khi mô hình đã được huấn luyện. Quá trình này giúp FastText có khả năng xử lý các mã thực thi không có trong từ điển, bởi vì nó sử dụng thông tin từ các phần tử con để tạo ra biểu diễn vectơ cho mã thực thi.

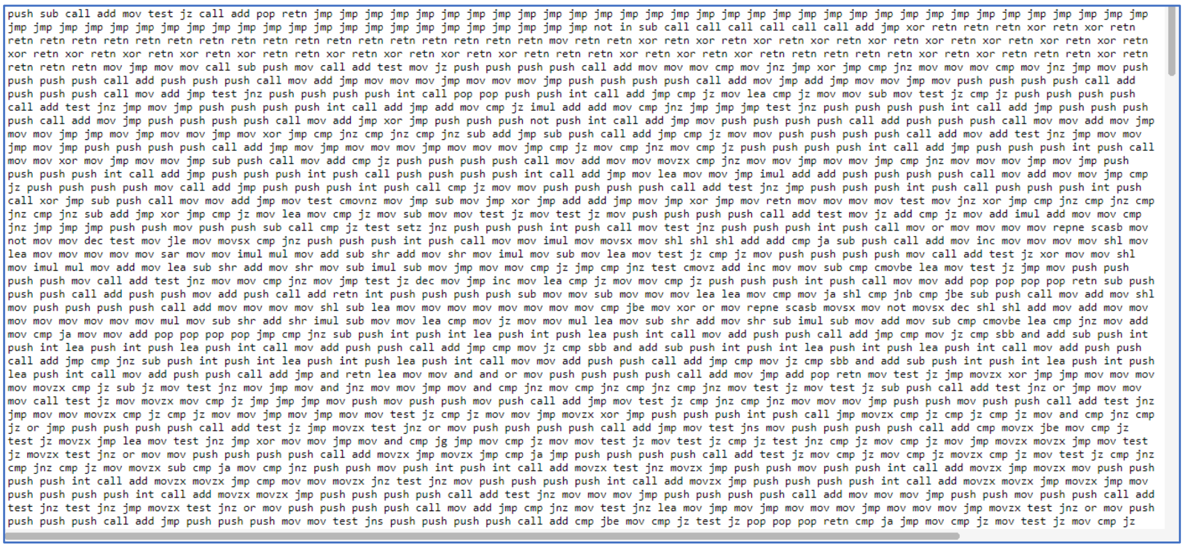
Ví dụ, để biểu diễn một mã thực thi trong FastText dựa trên mô hình skip-gram, các bước sau thường được thực hiện như sau:

(1) Tạo các n-gram: Mã thực thi được chia thành các n-gram (các phần tử con có độ dài n) để mô tả cấu trúc bên trong mã thực thi. Ví dụ: mã thực thi "push" có thể được chia thành các n-gram "pu", "pus", "us", "ush", "sh",.... Trong FastText sử dụng skip-gram [41], *n* được lựa chọn trong khoảng giá trị từ 3 đến 6 để tạo các n-gram từ các mã thực thi.

(2) Biểu diễn vectơ cho mã thực thi: Vectơ biểu diễn cho mã thực thi được tính bằng cách tổng hợp các vectơ của các n-gram đó. Mỗi n-gram được biểu diễn bởi một vectơ số.

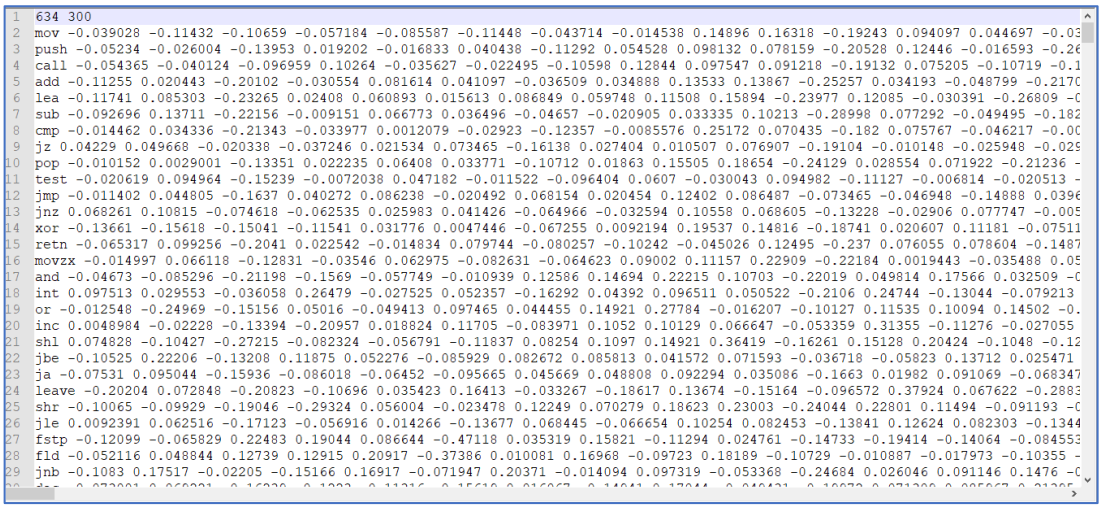
(3) Tính toán vectơ cho mã thực thi (chuỗi mã thực thi): Vectơ của mã thực thi hoặc chuỗi mã thực thi được tạo bằng cách lấy tổng các vectơ của các n-gram tương ứng.

Tập tin chứa tất cả các chuỗi mã thực thi thu thập từ tập dữ liệu kiến trúc Intel phục vụ huấn luyện mô hình FastText thể hiện trong hình 4.3.


```
push sub call add mov test jz call add pop retn jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp
jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp not in sub call call call call call call add jmp xor retn retn retn xor retn xor retn
retn retn retn retn retn retn retn retn retn retn retn retn retn retn retn retn retn mov retn retn xor retn xor retn xor retn xor retn xor retn xor retn xor retn xor retn xor retn
xor retn xor retn xor retn xor retn xor retn retn xor retn xor retn xor retn xor retn retn retn xor retn xor retn xor retn retn retn retn xor retn xor retn retn retn xor retn
retn retn retn mov jmp mov mov call sub push mov call add test mov jz push push push push call add mov mov mov cmp mov jnz jmp xor jmp cmp jnz mov mov mov cmp mov jnz jmp mov push
push push push call add push push push call mov add jmp mov mov mov jmp mov mov mov jmp push push push push call add mov jmp add jmp mov mov jmp mov push push push push call add
push push push call mov add jmp test jnz push push push push int call pop pop push push int call add jmp cmp jz mov lea cmp jz mov mov sub mov test jz cmp jz push push push push
call add test jnz jmp mov jmp push push push push int call add jmp add mov cmp jz imul add add mov cmp jnz jmp jmp jmp test jnz push push push push int call add jmp push push push
push call add mov jmp push push push push call mov add jmp xor jmp push push push not push int call add jmp mov push push push push call add push push push call mov mov add mov jmp
mov mov jmp jmp mov jmp mov mov jmp mov xor jmp cmp jnz cmp jnz cmp jnz sub add jmp sub push call add jmp cmp jz mov mov push push push push call add mov add test jnz jmp mov mov
jmp mov jmp push push push push call add jmp mov jmp mov mov mov jmp mov mov mov jmp cmp jz mov cmp jnz mov cmp jz push push push push int call add jmp push push push int push call
mov mov xor mov jmp mov mov jmp sub push call mov add cmp jz push push push push call mov add mov mov movzx cmp jnz mov mov jmp mov mov jmp cmp jnz mov mov mov jmp mov jmp push
push push push int call add jmp push push push int push call push push push push int call add jmp mov lea mov mov jmp imul add add push push push push call mov add mov mov jmp cmp
jz push push push push mov call add jmp push push push int push call cmp jz mov mov push push push call add test jnz jmp push push push int push call push push push int push
call xor jmp sub push call mov mov add jmp mov test cmovnz mov jmp sub mov jmp xor jmp add add jmp mov jmp xor jmp mov retn mov mov mov mov test mov jnz xor jmp cmp jnz cmp jnz cmp
jnz cmp jnz sub add jmp xor jmp cmp jz mov lea mov cmp jz mov sub mov mov test jz mov test jz mov push push push push call add test mov jz add cmp jz mov add imul add mov mov cmp
jnz jmp jmp jmp push push mov push push sub call cmp jz test setz jnz push push push int push call mov test jnz push push push int push call mov or mov mov mov mov repne scasb mov
not mov mov dec test mov jle mov movsx cmp jnz push push push int push call mov mov imul mov movsx mov shl shl shl add add cmp ja sub push call add mov inc mov mov mov mov shl mov
lea mov mov mov mov mov sar mov mov imul mul mov add sub shr add mov shr mov imul mov sub mov lea mov test jz cmp jz mov push push push push mov call add test jz xor mov mov shl
mov imul mul mov add mov lea sub shr add mov shr mov sub imul sub mov jmp mov mov cmp jz jmp cmp jnz test cmovz add inc mov mov sub cmp cmovbe lea mov test jz jmp mov push push
push push mov call add test jnz mov mov cmp jnz mov jmp test jz dec mov jmp inc mov lea cmp jz mov mov cmp jz push push push int push call mov mov add pop pop pop pop retn sub push
push push call add push push mov add push call add retn int push push push push sub mov mov sub mov mov mov lea lea mov cmp mov ja shl cmp jnb cmp jbe sub push call mov add mov shl
mov push push push push call add mov mov mov mov shl sub lea mov mov mov mov mov mov mov cmp jbe mov xor or mov repne scasb movsx mov not movsx dec shl shl add mov add mov mov
mov mov mov mov mov mul mov sub shr add shr imul sub mov mov lea cmp mov jz mov mov mul lea mov sub shr add mov shr sub imul sub mov add mov sub cmp cmovbe lea cmp jnz mov add
mov cmp ja mov mov add pop pop pop pop jmp cmp jnz sub push int push int lea push int push lea push int call mov add push push call add jmp cmp mov jz cmp sbb and add sub push int
push int lea push int push lea push int call mov add push push call add jmp cmp mov jz cmp sbb and add sub push int push int lea push int push lea push int call mov add push push
call add jmp cmp jnz sub push int push int lea push int push lea push int call mov mov add push push call add jmp cmp mov jz cmp sbb and add sub push int push int lea push int push
lea push int call mov add push push call add jmp and retn lea mov mov and and or mov push push push push call add mov jmp add pop retn mov test jz jmp movzx xor jmp jmp mov mov mov
mov movzx cmp jz sub jz mov test jnz mov jmp mov and jnz mov mov jmp mov and cmp jnz mov cmp jnz cmp jnz cmp jnz mov test jz mov test jz sub push call add test jnz or jmp mov mov
mov call test jz mov movzx mov cmp jz jmp jmp jmp mov push mov push push mov push call add jmp mov test jz cmp jnz cmp jnz mov mov mov jmp push push mov push push call add test jnz
jmp mov mov movzx cmp jz cmp jz mov mov jmp mov jmp mov mov test jz cmp jz mov mov jmp movzx xor jmp push push push int push call jmp movzx cmp jz cmp jz cmp jz mov and cmp jnz cmp
jz or jmp push push push push call add test jz jmp movzx test jnz or mov push push push push call add jmp mov test jns mov push push push push call add cmp movzx jbe mov cmp jz
test jz movzx jmp lea mov test jnz jmp xor mov mov jmp mov and cmp jg jmp mov cmp jz mov mov test jz mov test jz cmp jz test jnz cmp jz mov cmp jz mov jmp movzx movzx jmp mov test
jz movzx test jnz or mov mov push push push push call add movzx jmp movzx jmp cmp ja jmp push push push push call add test jz mov cmp jz mov cmp jz movzx cmp jz mov test jz cmp jnz
cmp jnz cmp jz mov movzx sub cmp ja mov cmp jnz push push mov push int push int call add movzx test jnz movzx jmp push push mov push push int call add movzx jmp movzx mov push push
push push int call add movzx movzx jmp cmp mov mov movzx jnz test jnz mov push push push push int call add movzx jmp push push push int call add movzx movzx jmp movzx jmp mov
push push push push int call add movzx movzx jmp push push push push call add test jnz mov mov mov jmp push push push call add mov mov mov jmp push push mov push push call add
test jnz test jnz jmp movzx test jnz or mov push push push push call mov add jmp cmp jnz mov test jnz lea mov jmp mov jmp mov mov mov jmp mov mov mov jmp movzx test jnz or mov push
push push push call add jmp push push push mov mov test jns push push push push call add cmp jbe mov cmp jz test jz pop pop pop retn cmp ja jmp mov cmp jz mov test jz mov cmp jz
```

Hình 4.3. Tập tin chứa tất cả các chuỗi mã thực thi thu thập từ tập cơ sở dữ liệu trên kiến trúc Intel.

Kết quả thu được sau khi thực hiện biểu diễn 634 mã thực thi trên kiến trúc Intel sử dụng FastText như hình 4.4:



Hình 4.4. Kết quả thu được sau khi nhúng 634 mã thực thi kiến trúc Intel sử dụng FastText.

4.2.2. Huấn luyện các mô hình chuyển đổi chuỗi mã thực thi

Sau khi đã có biểu diễn của các tập chuỗi mã thực thi trong không gian vectơ nhúng, việc huấn luyện để xây dựng mô hình chuyển đổi tốt nhất được thực hiện thông qua ba bước gồm: Xây dựng Bảng ánh xạ mã thực thi, mô hình hoá chuỗi mã thực thi và chuyển đổi ngược lặp đi lặp lại.

Xây dựng Bảng ánh xạ mã thực thi: Việc xây dựng Bảng ánh xạ mã thực thi (từ mã thực thi nguồn đến đích và từ mã thực thi đích đến nguồn) được thực hiện sử dụng phương pháp xác định ánh xạ giữa 2 bộ dữ liệu mã thực thi sao cho các bản chuyển đổi tốt nhất trong một không gian vectơ chung. Phương pháp xác định dựa trên khai thác những mã thực thi tương đồng giữa các không gian vectơ nhúng mã thực thi của mỗi kiến trúc vi xử lý để tìm ra một ánh xạ. Mục tiêu của quá trình là xác định sự liên kết giữa các mã thực thi nguồn và đích dựa trên mô hình xác suất.

Việc xây dựng mô hình chuyển đổi từ chuỗi mã thực thi *i* sang chuỗi mã thực thi *j* sẽ được đánh giá thông qua việc xác định điểm số (score) dựa trên nguyên lý Bayes. Đây là nguyên lý thường được sử dụng trong các bài toán phân lớp và ước lượng trong học máy và xử lý ngôn ngữ tự nhiên để xác định lớp hoặc sự kiện có xác suất cao nhất dựa trên các thông tin quan sát được. Với *P(i|j)* là xác suất có điều kiện của biến cố *i* xảy ra khi *j* đã xảy ra sẽ được xác định dựa trên Bảng ánh xạ chuỗi mã thực thi và *P(j)* là xác suất xảy ra *j* được sử dụng xác định điểm số tạo bởi mô hình hoá chuỗi mã thực thi, điểm số của *j* được xác định theo công thức sau:

$$argmax_j\ (P(j|i)) = argmax_j(P(i|j))P(j) \qquad (4.5)$$

Để tìm ra lớp *j* có xác suất cao nhất dựa trên dữ liệu *i* thông qua việc so sánh xác suất có điều kiện *P(j|i)* của lớp *j* và xác suất hợp nhất *P(i|j)P(j)* của lớp *j*. Luận án chọn lớp *j* có giá trị *P(j|i)* cao nhất, điều này cũng tương đương với việc chọn lớp *j* có giá trị *P(i|j)P(j)* cao nhất.

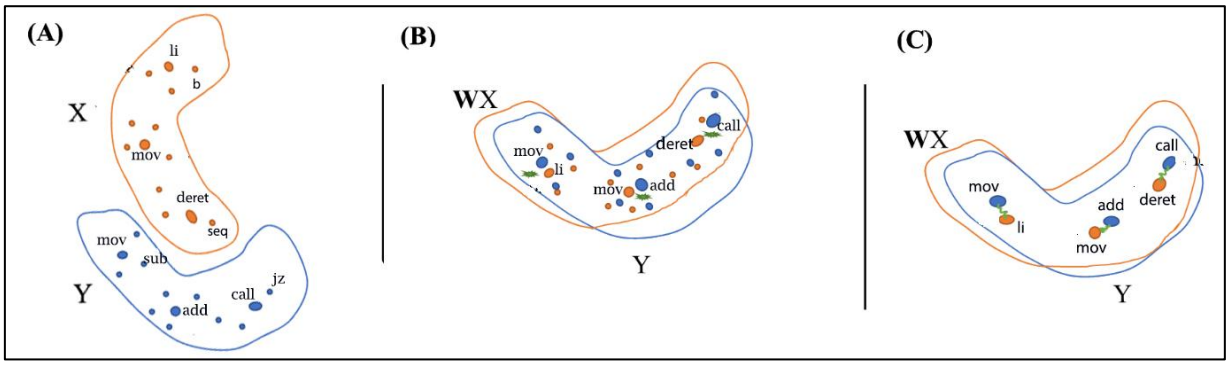
Hai không gian biểu diễn các chuỗi mã thực thi được huấn luyện độc lập cho các chuỗi mã thực thi trên mỗi kiến trúc để khai thác sự tương đồng giữa các mã thực thi. Mục tiêu là tìm ra được một phép biến đổi hình học sao cho kết quả ánh xạ biểu diễn hình học giữa hai bộ nhúng trong không gian nguồn và không gian đích là phù hợp nhất. Để tìm ra phép biến đổi hình học phù hợp, NCS tiến hành lựa chọn các mã thực thi có tên và chức năng giống nhau ở cả hai kiến trúc để làm các điểm "neo". Phép biến đổi hình học sẽ được tìm ra phép toán sao cho có thể biến đổi cụm các điểm neo từ không gian nguồn sang không gian đích, sao cho tổng khoảng cách giữa các cặp điểm neo sau ánh xạ trong không gian nguồn và không gian đích tương ứng là bé nhất. Điều này được thực hiện thông qua việc tìm ra phép biến hình W* biến đổi hình học từ tập hợp các điểm trong không gian X $\{x_1,x_2,\ldots,x_n\}$ sang không gian Y $\{y_1,y_2,\ldots,y_n\}$ sao cho sai số nhỏ nhất. Việc biến đổi hình học được mô tả thông qua bài toán tối ưu hoá phát biểu như sau:

Cho d là một số nguyên dương biểu diễn kích thước của chuỗi mã thực thi được nhúng; $M_d(\mathbb{R})$ *là tập hợp các ma trận có kích thước d* x *d biểu diễn không gian của các ma trận số thực* $\mathbb{R}$*; X và Y là hai ma trận* $[d \times n]$ *được căn chỉnh để chứa nhúng của các mã thực thi trong Bảng chuỗi mã thực thi. Cần tìm một ma trận* $W \in M_d(\mathbb{R})$ *sao cho norm Frobenius của hiệu của hai ma trận* $||WX - Y||_F$ *đạt giá trị nhỏ nhất.*

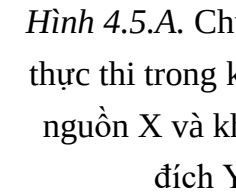
Bài toán được thể hiện thông qua công thức 4.2 dưới đây:

$$W^* = \frac{arg\ min}{W \in M_d(\mathbb{R})} ||WX - Y||_F \quad (4.6)$$

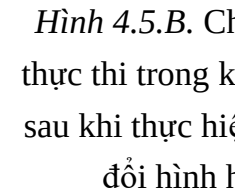
Phép biến đổi hình học trong hai không gian nhúng biểu diễn chuỗi mã thực thi hai kiến trúc vi xử lý được minh hoạ trong hình 4.5. Hai chuỗi mã thực thi trên hai kiến trúc vi xử lý được biểu diễn trên không gian X và không gian Y. Các giá trị điểm neo là các mã thực thi có tên giống nhau hoặc tương đồng được xác định trên các chuỗi mã thực thi không gian X và không gian Y. Một phép biến đổi affine WX đối với chuỗi mã thực thi trên không gian X sang không gian Y được thực hiện sao cho khoảng cách giữa các điểm neo là nhỏ nhất.



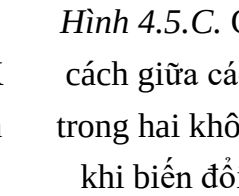
Hình 4.5. Minh hoạ phép biến đổi hình học không gian.



Hình 4.5.A. Chuỗi các mã thực thi trong không gian nguồn X và không gian đích Y.



Hình 4.5.B. Chuỗi các mã thực thi trong không gian X sau khi thực hiện phép biến đổi hình học W.



Hình 4.5.C. Các khoảng cách giữa các điểm neo trong hai không gian sau khi biến đổi hình học

Một phép biến đổi W của bất kỳ mã thực thi nguồn nào đó được xác định dựa trên việc tìm C sao cho hàm cosin của tích vô hướng hai vectơ W x_i và y_j đạt giá trị lớn nhất được định nghĩa theo công thức 4.7 sau đây:

$$C = arg\ max_c \cos(W\ x_i, y_j) \tag{4.7}$$

Luận án sử dụng một công thức thường được sử dụng trong việc dự đoán từ tiếp theo hoặc một phần của văn bản dựa trên ngữ cảnh theo mô hình ngôn ngữ tự nhiên [43]. Bằng cách sử dụng hàm e(x,y) để đo lường sự tương đồng giữa các phần của văn bản và ngữ cảnh với việc sử dụng tham số T để kiểm soát độ phẳng của phân phối xác suất, chúng ta có thể tính xác suất p($d_i|n_j$) dựa trên thông tin ngữ cảnh n_j với các đối tượng d_icần dự đoán. Công thức xác suất thể hiện như sau:

$$P(d_i|n_j) = \frac{e^{\frac{1}{T}cos\,(e(d_i),We(n_j))}}{\sum_k e^{\frac{1}{T}cos\,(e(d_k),We(n_j))}} \tag{4.8}$$

Trong đó: $d_i$là mã thực thi thứ *i* trong mã thực thi đích, n_j là mã thực thi thứ *j* trong mã thực thi nguồn. *T* là một siêu tham số được sử dụng để kiểm soát độ phẳng (sự phân biệt giữa các giá trị xác suất), khi *T* lớn thì phân phối xác suất sẽ mềm mại hơn. *W* là phép xoay để ánh xạ các phần tử trong không gian nguồn thành phần tử trong không gian đích, *e(x)* là nhúng của *x*.

Công thức biến đổi hình học (4.6) có thể rút gọn đưa ra được nghiệm dạng đóng thu được từ việc phân tích giá trị đơn lẻ (Singular Value Decomposition – SVD) của YX^T:

$$W^* = \frac{arg\ min}{W \in M_d(\mathbb{R})} ||\ W\ X - Y||_F = UV^T \tag{4.9}$$

với U,V là các ma trận trực giao, Σ là ma trận đường chéo không vuông với các phần tử trên đường chéo và rank của ma trận $U\Sigma V^T$ = SDV (Y X^T). Trong Python để tính SVD của một ma trận, chúng ta có thể sử dụng module *linalg* trong thư viện *numpy*.

Để cải thiện chất lượng chuyển đổi chuỗi mã thực thi, luận án xác định tỷ lệ vùng tương đồng các mã thực thi láng giềng để chọn ra các cặp chuyển đổi mã thực thi tốt hơn nhằm tạo ra một Bảng chuyển đổi mã thực thi mới có khả năng cao tương đồng cao hơn ở mỗi lần lặp. Bên cạnh đó, luận án chỉ xem xét các láng giềng gần nhau nhất theo tỷ lệ vùng tương đồng giúp làm giảm kích thước của Bảng chuyển đổi mã thực thi được tạo và cải thiện hiệu suất, độ chính xác. Luận án xem xét giá trị tương đồng

trung bình của nhúng mã thực thi nguồn x_i vào lân cận mã thực thi đích của nó dựa trên việc sử dụng hàm $cos(W\ x_i, y_j)$ để đo lường sự tương đồng của $W\ x_i$ và y_j, sau đó trung bình hoá giá trị này thông qua việc chia cho *k* là số lượng phần tử trong tập hợp $N_I(W\ x_i)$. Giá trị tương đồng giữa vectơ $W\ x_i$ (vectơ đầu vào) và một tập hợp $N_I(W\ x_i)$ các vectơ đích y_j mà vectơ $W\ x_i$ tương tác được tính theo công thức sau:

$$r_I(W\ x_i) = \frac{1}{k} \sum_{y_j \in N_I(W\ x_i)} cos(W\ x_i, y_j) \quad (4.10)$$

Tương tự, giá trị tương đồng trung bình của nhúng mã thực thi đích y_j vào lân cận của nó được biểu diễn bằng $r_J(W\ y_j)$. Các đại lượng này được tính toán cho tất cả các vectơ mã thực thi nguồn và đích với việc áp dụng phương pháp xây dựng biểu đồ k-NN của Johnson và cộng sự [60]. Luận án sử dụng chúng để xác định độ tương đồng giữa mã thực thi nguồn được ánh xạ *i* và mã thực thi đích *j* là *S*. Công thức tổng hợp sự tương đồng giữa $W\ x_i$ và y_j được tính dựa trên sự tương đồng trực tiếp (cosin) và hai độ đo tương đồng r_I và r_J. Giá trị *S* có thể được sử dụng để so sánh sự tương đồng giữa các cặp vectơ $W\ x_i$ và y_j trong không gian vectơ *W*. Công thức tổng hợp sự tương đồng *S* được tính theo công thức như sau:

$$\mathrm{S(W}\ x_{i,}y_j) = 2\ cos(W\ x_{i,}y_j) - r_I(W\ x_i) - r_J(W\ y_j) \quad (4.11)$$

Việc xác định phép biến đổi mã thực thi chéo kiến trúc được xây dựng cụ thể như sau:

Các mã thực thi kiến trúc A (với tổng số m mã thực thi khác nhau), kiến trúc B (với n mã thực thi khác nhau) được biểu diễn trong hai không gian vectơ nhúng z chiều tương ứng là $[A]_{mxz}$ *và* $[B]_{nxz}$ *thông qua mô hình embedding FastText dựa trên skip-gram. Cần tìm một phép biến đổi T để ánh xạ các mã thực thi biểu diễn trong* $[A]_{mxz}$ *sang các mã thực thi biểu diễn trong* $[B]_{nxz}$.

Các bước tiến hành:

- Bước 1: Xác định các cặp điểm "neo" là các cặp mã thực thi trên 2 kiến trúc được đánh giá là tương đồng. Tìm phép ánh xạ từ không gian nhúng $[A]_{mxz}$ sang không gian nhúng $[B]_{nxz}$ dựa trên việc tìm ma trận $[R]_{zxz}$ sao cho $||R - [A]^T[B]||$ đạt giá trị nhỏ nhất và $[R]^T[R] = [A]_{zxz}$.

- Bước 2: Tính xác suất chuyển đổi các mã thực giữa 2 kiến trúc để xây dựng Bảng ánh xạ mã thực thi:

+ Xác định khoảng cách trung bình của *k* điểm láng giềng với mỗi mã thực thi từ kiến trúc A sang kiến trúc B:

Đặt [C] = [A]. $[B]^T$ ta có $[C_1]_{mxn}$, sau đó thực hiện lựa chọn mỗi hàng *k* giá trị lớn nhất để có $[E_1]_{mxk}$ và tính khoảng cách trung bình của *k* điểm đó thu được $[E_1]_{mx1}$.

Tương tự, xác định khoảng cách trung bình của *k* điểm láng giềng với mỗi mã thực thi từ kiến trúc B sang kiến trúc A thu được $[E_2]_{nx1}$

+ Tìm vị trí các mã thực thi có khả năng là kết quả của phép chuyển đổi. Số lượng các mã thực thi là *t*.

Đặt $[V_1] = 2[C_1]_{mxn}$ - $[\underbrace{[E_1]\ [E_1]\ \dots\ [E_1]}_{n\ cột}$ - $[\begin{matrix}[E_2]^T \\ [E_2]^T \\ \dots \\ [E_2]^T\end{matrix}]\Big\}\ m\ hàng$, tìm k vị trí có giá trị lớn nhất trong mỗi hàng của $[V_1]$. Kết quả là *t* mã thực thi ($[V'_1]_{mxt}$) có khả năng cao nhất là kết quả của phép chuyển đổi mã thực thi từ [A] sang [B].

Tương tự, tìm được *t* mã thực thi ($[V'_2]_{nxt}$) có khả năng cao nhất là kết quả của phép chuyển đổi mã thực thi từ [B] sang [A].

+ Tính xác suất của từng cặp mã thực thi tương ứng với $[V']$:

Với *i*=1,2,…,m, *j*=1,2,…,n thì $P(x_{ij}) = \frac{e^{x_{ij}}}{\sum e^{x_{ij}}}$ là phân phối softmax để tính xác suất dịch mã thực thi *i* trong [A] sang mã thực thi *j* trong [B]. $[v_{ij}]$.Tuy nhiên, các mã thực thi trong [B] chỉ lấy trong vùng mã thực thi (*t* giá trị) đã chọn ở $[V'_1]_{mxt}$.

Tương tự, xác định xác suất chuyển đổi mỗi mã thực thi từ [B] sang [A]: $P'(x_{ji}) = e^{x_{ji}}$ được sử dụng để đưa ra giá trị không chuẩn hóa (trước khi chuẩn hóa thành phân phối softmax) của đầu ra từ một phần tử trong vectơ đầu ra. Việc sử dụng hàm mũ tự nhiên để tăng độ lớn của giá trị đầu ra này sẽ giúp tăng cường sự phân biệt giữa các giá trị và làm cho lớp hoặc nhãn có giá trị cao hơn trở nên rõ ràng hơn so với các lớp hoặc nhãn khác. Các mã thực thi trong [A] chỉ lấy trong *t* giá trị đã chọn ở $[V'_2]_{nxt}$.

Xác suất $P'(x_{ji})$ và $P(x_{ij})$ được sử dụng và lưu trữ trong "Bảng ánh xạ mã thực thi".

- Bước 3: Sau khi có "Bảng ánh xạ mã thực thi" mô hình sẽ xây dựng dựa trên việc coi chuỗi *n* mã thực thi liên tiếp là những mã thực thi đơn lẻ để thực hiện lặp lại quá trình chuyển đổi với các chuỗi mã thực thi dài hơn.

Ví dụ 4.2: *Xây dựng "Bảng ánh xạ mã thực thi" từ kiến trúc A có 7 mã thực thi khác nhau là a,b,c,d,e,f,g sang kiến trúc B có 5 mã thực thi khác nhau là x,y,z,t,w; số chiều không gian nhúng là k =6.*

Giả sử sau quá trình biểu diễn các mã thực thi của 2 kiến trúc A, B thông qua mô hình FastText sử dụng skip-gram chúng ta thu được:

$$[A]_{7x6} = \begin{matrix}a\\b\\c\\d\\e\\f\\g\end{matrix}\begin{bmatrix}1&2&2&1&2&1\\2&3&4&5&1&1\\1&5&3&2&2&3\\2&1&5&6&7&1\\6&2&1&3&1&2\\3&3&3&4&5&1\\4&3&4&2&1&5\end{bmatrix} \text{ và } [B]_{5x6} = \begin{matrix}x\\y\\z\\t\\w\end{matrix}\begin{bmatrix}1&2&3&4&5&6\\6&3&5&4&4&1\\6&2&1&3&4&3\\3&5&1&2&6&1\\1&1&2&1&2&1\end{bmatrix}$$

Sau khi thực hiện các bước nêu trên với giả thuyết 2 cặp điểm "neo" là (a-x), (b-y) thì kết quả là một "Bảng ánh xạ mã thực thi" như sau:

Kiến trúc A	Kiến trúc B	P'(x_i)	P(x_i)
a	x	0,64	0,65
a	y	0,63	0,64
a	z	0,71	0,72
a	t	0,71	0,71
a	w	0,67	0,67
b	x	0,69	0,68
…	…	…	…
g	w	0,62	0,61

Mô hình hoá chuỗi mã thực thi: Mục tiêu của bước này là từ các dữ liệu chuỗi mã thực thi ban đầu, một mô hình được huấn luyện để học cấu trúc của các chuỗi mã thực thi trên kiến trúc đó. Cấu trúc của mỗi loại chuỗi mã thực thi trên từng kiến trúc vi xử lý là khác nhau. Trong ngôn ngữ tự nhiên, cấu trúc một câu trong mỗi ngôn ngữ là khác nhau, ví dụ trong tiếng Anh thì tính từ đứng trước danh từ, còn trong tiếng Việt tính từ đứng sau danh từ. Trong ngôn ngữ lập trình cũng tương tự, mỗi loại hợp ngữ sẽ có mô hình khác nhau, vì vậy chuỗi mã thực thi trích xuất từ các chương trình thực thi sẽ có những biểu diễn mô hình hoá. Việc xây dựng mô hình hoá chuỗi mã thực thi sẽ giúp sắp xếp thứ tự các mã thực thi cho phù hợp nhất ngay cả khi có từ điển ánh xạ mã thực thi để dịch đúng mã thực thi hoặc chuỗi mã thực thi ngắn tương ứng.

Với số lượng lớn chuỗi mã thực thi trên các kiến trúc khác nhau, chúng ta có thể huấn luyện các mô hình hoá mã thực thi của kiến trúc nguồn và kiến trúc đích. Các mô hình này thể hiện dữ liệu về cách đọc các chuỗi mã thực thi trong mỗi kiến trúc và chứng minh chất lượng của các mô hình chuyển đổi chuỗi mã thực thi thông qua thay thế và sắp xếp lại mã thực thi. Qua quá trình khảo sát của nghiên cứu sinh, hiện chưa có công bố nào về việc mô hình hoá chuỗi mã thực thi cho các kiến trúc vi xử lý. Vì vậy, luận án dựa trên cách tiếp cận các mô hình hoá ngôn ngữ sử dụng trong lĩnh vực xử lý ngôn ngữ tự nhiên.

Trong xử lý ngôn ngữ tự nhiên, quá trình mô hình hoá ngôn ngữ thường được sử dụng để dự đoán xác suất xuất hiện của các từ hoặc cụm từ trong một ngôn ngữ. Các mô hình này cố gắng hiểu cấu trúc, quy luật và xu hướng trong ngôn ngữ để tạo ra các câu dịch từ ngôn ngữ nguồn sang ngôn ngữ đích chính xác. Một số phương pháp mô hình hoá ngôn ngữ được sử dụng trong dịch máy bao gồm :

- *N-gram Language Models*: Sử dụng xác suất đồng thời của các cụm từ (n-gram) để dự đoán từ tiếp theo dựa trên các từ trước đó trong câu. Các mô hình n-gram có thể dự đoán từ tiếp theo dựa trên từ trước đó hoặc cụm từ trước đó thông qua việc tính toán xác suất của các chuỗi từ.

- *Statistical Language Models*: Mô hình dựa trên thống kê, sử dụng kỹ thuật smoothing và interpolation để xử lý vấn đề của các cụm từ chưa xuất hiện trong tập dữ liệu huấn luyện.

- *Neural Language Models:* Dùng các mạng nơ-ron để mô hình hoá ngôn ngữ. Nó thường sử dụng các kiến trúc như Recurrent Neural Networks, Long Short-Term Memory Networks hoặc Transformer-based models như Generative Pre-trained Transformer để dự đoán từ tiếp theo trong câu.

- *Transformer-based Language Models:* Đây là phương pháp sử dụng self-attention và parallel processing để xử lý các từ trong câu hiệu quả với các kiến trúc như Transformer, BERT, GPT và các biến thể khác.

Mỗi phương pháp có ưu điểm và hạn chế riêng. Với yêu cầu bài toán chuyển đổi chuỗi mã thực thi không có nhiều tương đồng giữa các kiến trúc và có khả năng tạo ra chuỗi mã thực thi chưa có trong tập dữ liệu huấn luyện, luận án lựa chọn mô hình Statistical Language Models do Moses đề xuất [89]. Mô hình được xây dựng dựa trên việc xác định tần suất xuất hiện của từng mã thực thi hoặc chuỗi mã thực thi ngắn. Phân bố xác suất P(W), W ∈ V với V là tập hợp tất cả các mã thực thi trên một kiến trúc vi xử lý được tính toán, khi đó P(W) ≥0. Trong đó, xác suất của một chuỗi mã thực thi có độ dài *l* sẽ được tính theo công thức xác suất có điều kiện như sau :

$$P(w_1w_2w_3\ldots w_l) = P(w_1).P(w_2|w_1)\ldots P(w_l|w_1w_2\ldots w_{l-1}) \quad (4.12)$$

Từ đó, chúng ta có thể biết được mã thực thi w_i (i=1,2,3,…) có khả năng xuất hiện ở trong ngữ cảnh nào và cùng xuất hiện với các mã thực thi w_j ($j \neq i$) là bao nhiêu. Xác định được mã thực thi có khả năng xuất hiện cao ở vị trí và ngữ cảnh thì nó sẽ được ưu tiên sử dụng làm kết quả chuyển đổi mã thực thi trong phép chuyển đổi chuỗi mã thực thi.

Chuyển đổi ngược lặp đi lặp lại: Việc học lặp lại là một đặc trưng thường thấy của các mô hình học máy, nó giúp cải thiện phần nào đó hiệu quả của mô hình thu được. Trong dịch máy, việc chuyển đổi ngược lặp đi lặp lại là sự kết hợp mô hình dịch từ ngôn ngữ nguồn sang đích với mô hình dịch ngược từ ngôn ngữ đích sang nguồn. Mục tiêu của mô hình là tạo ra nhiều câu nguồn cho mỗi câu đích trong kho ngữ liệu đơn ngữ. Trong xây dựng mô hình chuyển đổi mã thực thi, việc học lặp lại được tiến hành dựa vào Bảng ánh xạ mã thực thi được khởi tạo từ bước một và mô hình mã thực thi ở phía đích được tạo từ bước hai để xây dựng một mô hình hạt giống. Giá trị hạt giống P(0) được xây dựng dựa trên Bảng chuỗi mã thực thi không tồn tại một cặp từ điển ánh xạ mã thực thi nào trước đó và mô hình mã thực thi ở phía đích được xây dựng dựa trên phương pháp n-gram. Sau đó, mô hình này được sử dụng để dịch từ mã thực thi nguồn sang mã thực thi đích. Khi mã thực thi được tạo ra, mô hình huấn luyện với dữ liệu tương ứng là các chuỗi mã thực thi và sẽ tạo ra mô hình ánh xạ dữ liệu ngược trở lại các mã thực thi ban đầu. Tiếp theo, quá trình tạo và huấn luyện được thực hiện theo hướng ngược lại. Các bước này được thực hiện nhiều lần để đạt được kết quả mong muốn. Quá trình huấn luyện này sẽ giúp mô hình chuyển đổi tối ưu hoá hàm mất mát thông qua việc

điều chỉnh các tham số trong mô hình.

Một mô hình "discriminator" được huấn luyện để phân biệt giữa các phần tử được lấy mẫu ngẫu nhiên từ WX = {W x_1, …, W x_i} và Y. W được huấn luyện để chống lại việc dự đoán chính xác của bộ phân lớp. Trong khi mục đích của discriminator là tối đa hoá khả năng xác định nguồn gốc của một nhúng, thì W lại ngăn chặn bộ phân lớp thực hiện công việc này thông qua việc làm cho WX và Y giống nhau nhất có thể. Để thực hiện điều này, NCS gọi các tham số của bộ phân biệt là θ_D và xem xét xác suất P_{θ_D} (source = 1|t) với vectơ *t* là ánh xạ của một nhúng nguồn theo discriminator. Hàm mất mát của mô hình discriminator có thể được tính toán theo công thức:

$$\mathcal{L}_D(\theta_D|W) = \frac{-1}{m}\sum_{i=1}^{m} log\ P_{\theta_D}\left(source = 0\middle|y_j\right) - \frac{1}{n}\sum_{i=1}^{n} log\ P_{\theta_D}(\text{source} = 1|W_{x_i}) \quad (4.13)$$

Trong huấn luyện mô hình chuyển đổi, W được huấn luyện để discriminator không thể dự đoán được chính xác chuỗi mã thực thi nhúng ban đầu. Hàm mất mát ánh xạ (Mapping loss) được tính toán theo công thức sau đây:

$$\mathcal{L}_W(W|\theta_D) = \frac{-1}{m}\sum_{i=1}^{m} log\ P_{\theta_D}\left(source = 1\middle|y_j\right) - \frac{1}{n}\sum_{i=1}^{n} log\ P_{\theta_D}(\text{source} = 0|W_{x_i}) \quad (4.14)$$

Mô hình chuyển đổi được xây dựng tuân thủ theo quy trình huấn luyện tiêu chuẩn của mạng đối nghịch sâu do Goodfellow và cộng sự [49] đề xuất. Đối với mỗi mẫu đầu vào, discriminator và W được huấn luyện liên tục với các cập nhật gradient ngẫu nhiên nhằm mục đích giảm giá trị $\mathcal{L}_D$ and $\mathcal{L}_W$ tương ứng.

Cụ thể quá trình huấn luyện mô hình chuyển đổi chuỗi mã thực thi giữa hai kiến trúc vi xử lý được trình bày giả mã trong thuật toán 4.1 dưới đây:

Thuật toán 4.1: Huấn luyện mô hình chuyển đổi chuỗi mã thực thi đa kiến trúc.

Đầu vào: *Hai tập dữ liệu chuỗi mã thực thi của hai kiến trúc trong không gian nhúng* (O_n, O_d).

Đầu ra: *Mô hình chuyển đổi chuỗi mã thực thi giữa kiến trúc nguồn và kiến trúc đích.*

 # Xây dựng Bảng ánh xạ mã thực thi

1: Opcode _ translation_table ← $\boldsymbol{f(O_n, O_d | min_{distance}(WO_n, O_d))}$

 # Mô hình hoá chuỗi mã thực thi

2: Opcode_sequences_ Model ← Statistical Language Models by Moses

 # Xác định các giá trị hạt giống $P^0_{n\to d}$ *dựa trên Bảng ánh xạ mã thực thi và mô hình hoá chuỗi mã thực thi*

3: $\boldsymbol{P^0_{n\to d}}$ ← Opcode _ translation_table & Opcode_sequences_ Model

 # Sử dụng giá trị hạt giống để chuyển đổi chuỗi mã thực thi nguồn sang đích

4: $\boldsymbol{Y^0_d \leftarrow P^0_{n\to d}}$

 # Quá trình chuyển đổi ngược lặp lại N lần

5: **For** i=1 **to** N **do**

 # Huấn luyện mô hình chuyển đổi mã thực thi đích sang mã thực thi nguồn

6: Train model $\boldsymbol{P^i_{d\to n} \leftarrow Y^{i-1}_d}$

 # Sử dụng mô hình P để chuyển đổi chuỗi mã thực thi đích

7: $\boldsymbol{Y^i_n \leftarrow P^i_{d\to n}}$

 # Huấn luyện mô hình chuyển đổi mã thực thi nguồn sang mã thực thi đích

8: Train model $\boldsymbol{P^i_{n\to d} \leftarrow Y^i_n}$

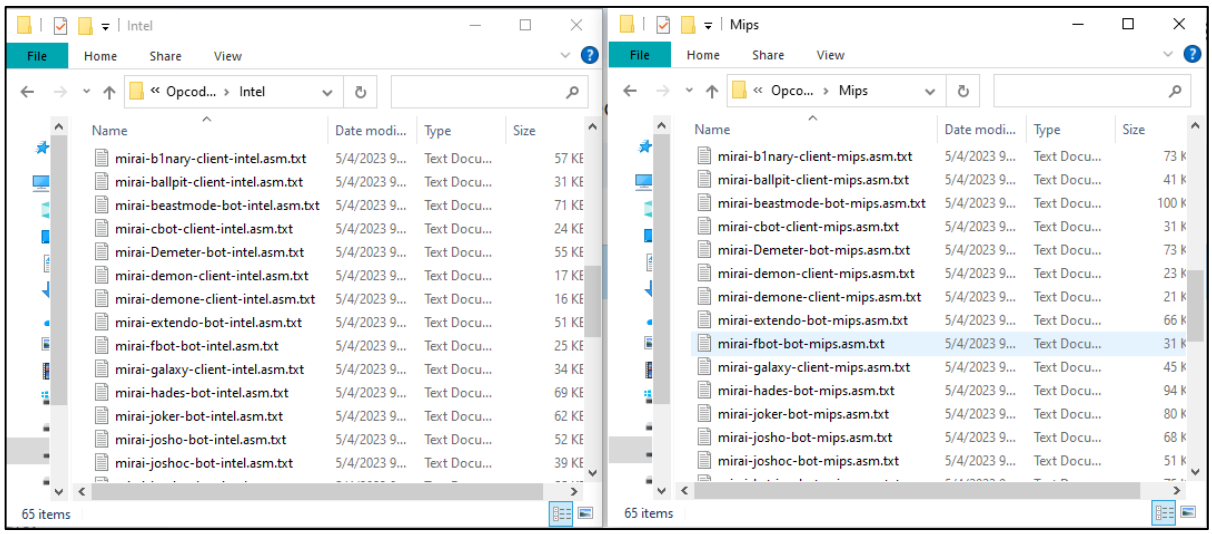
 # Sử dụng giá trị P để chuyển đổi chuỗi mã thực thi nguồn

9: $\boldsymbol{Y^i_d \leftarrow P^i_{n\to d}}$

10: **Endfor**

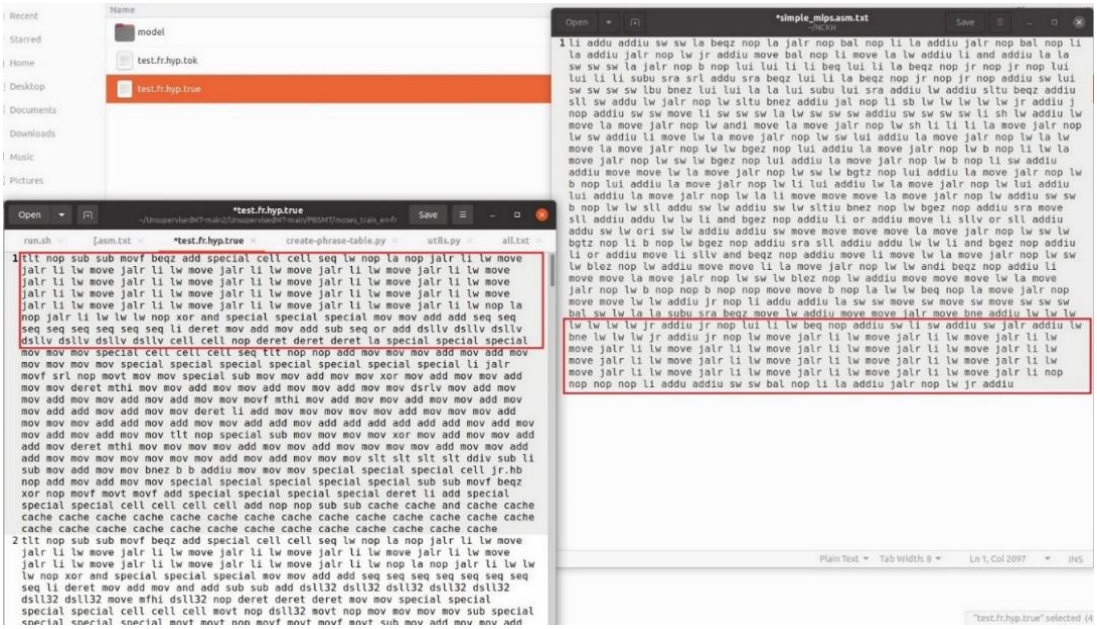
11: **Return** $\boldsymbol{Y^i_d}$

Luận án chưa khảo sát thấy một công trình khoa học đã công bố nào về Bảng ánh xạ hoặc chuyển đổi chuỗi mã thực thi từ kiến trúc vi xử lý các thiết bị IoT khác nhau. Để có cơ sở bước đầu đánh giá hiệu quả của quá trình chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc, dựa trên các khảo sát kỹ thuật tạo mã độc IoT hoạt động đa kiến trúc vi xử lý, nghiên cứu sinh tiến hành thu thập các mã nguồn mã độc IoT đã công bố và các mã nguồn chương trình đơn giản từ đó thực hiện kỹ thuật cross-compiler mã nguồn để thu thập các tập tin thực thi trên các kiến trúc vi xử lý của thiết bị IoT. Các tập tin thực thi trên các kiến trúc đó được thực hiện dịch ngược và phân tích tĩnh thủ công nhằm xác định các vùng chuỗi mã thực thi tương ứng phù hợp trên hai kiến trúc vi xử lý phục vụ điều chỉnh các tham số trong quá trình huấn luyện mô hình chuyển đổi chuỗi mã thực thi chéo kiến trúc.



Hình 4.6. Kết quả dịch ngược các tập tin thực thi trên hai kiến trúc Intel và MIPS sau khi cross-compiler từ một mã nguồn các mã độc IoT.

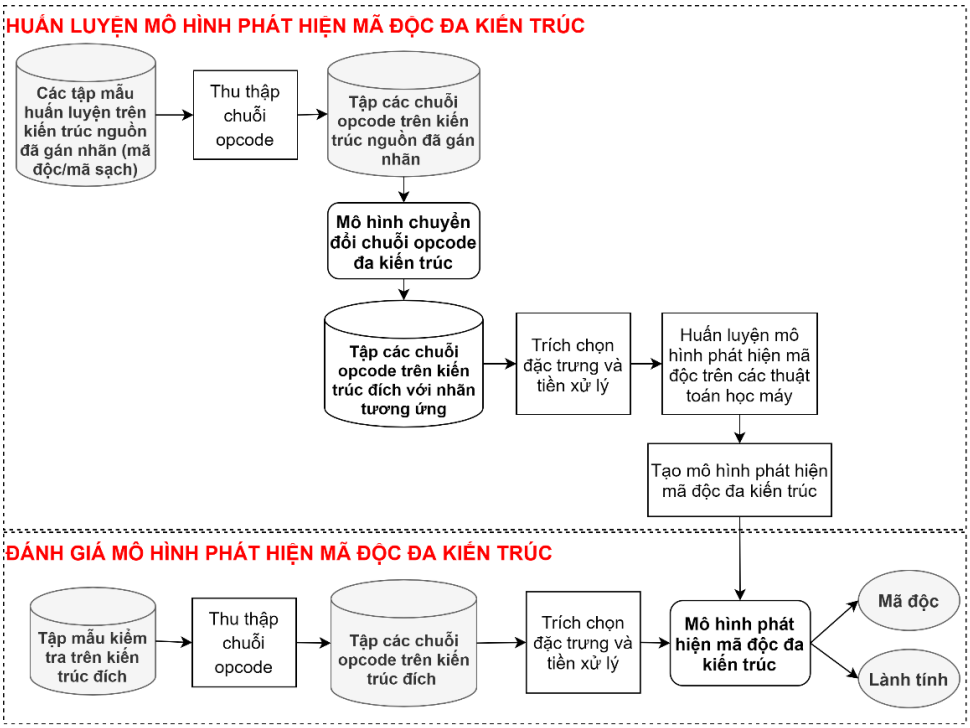
Mặt khác, luận án thực hiện so sánh và đánh giá kết quả chuỗi mã thực thi thu được sau khi chuyển đổi với chuỗi mã thực thi thu thập từ nền tảng ban đầu của cùng một mã nguồn biên dịch để hiệu chỉnh các tham số trong quá trình xây dựng mô hình chuyển đổi chuỗi mã thực thi chéo kiến trúc.



Hình 4.7. Đối sánh kết quả chuyển đổi chuỗi mã thực thi thu với chuỗi mã thực thi thu thập trên kiến trúc ban đầu.

4.3. Xây dựng mô hình phát hiện mã độc IoT đa kiến trúc dựa trên mô hình chuyển đổi đặc trưng đề xuất

Quá trình xây dựng mô hình phát hiện mã độc IoT đa kiến trúc dựa trên phương pháp chuyển đổi đặc trưng chuỗi mã thực thi luận án đề xuất gồm 2 giai đoạn: Huấn luyện mô hình phát hiện mã độc đa kiến trúc và đánh giá mô hình phát hiện. Chi tiết các quá trình xây dựng mô hình phát hiện mã độc IoT đa kiến trúc dựa trên mô hình chuyển đổi đặc trưng chuỗi mã thực thi luận án đề xuất được thể hiện trong hình 4.8.



Hình 4.8. Quá trình xây dựng mô hình phát hiện mã độc đa kiến trúc dựa trên đặc trưng chuỗi mã thực thi.

4.3.1. Huấn luyện mô hình phát hiện mã độc IoT đa kiến trúc dựa trên mô hình chuyển đổi chuỗi mã thực thi

- *Thu thập chuỗi mã thực thi:* Trong xây dựng mô hình phát hiện mã độc chéo kiến trúc vi xử lý, có nhiều phương pháp để trích xuất đặc trưng chuỗi mã thực thi từ phân tích tĩnh tập tin như các khảo sát của NCS trong nội dung 1.3.3 của luận án và nghiên cứu trình bày trong [TC4]. Việc sử dụng các phương pháp trích xuất đặc trưng chuỗi mã thực thi dựa trên CFG trong các nghiên cứu [117], [125], [127] đã đem lại hiệu quả đối với phát hiện chéo mã độc IoT, tuy nhiên độ phức tạp và yêu cầu tài nguyên hệ thống lớn. Vì vậy, để đạt được mục tiêu bài toán đã đặt ra trong chương này và giảm độ phức tạp trong trích xuất đặc trưng chuỗi mã thực thi từ các tập tin, NCS trích xuất chuỗi mã thực thi từ tập tin thực thi ELF thông qua phương pháp đã trình bày trong nội dung 3.2.1 của luận án. Kết quả của giai đoạn này là các tập dữ liệu chuỗi mã thực thi trên kiến trúc vi xử lý có nhãn mã độc hoặc lành tính tương ứng với nhãn của tập tin thực thi ELF.

- *Tăng cường tập dữ liệu chuỗi mã thực đa kiến trúc thông qua mô hình chuyển đổi chuỗi mã thực thi chéo kiến trúc vi xử lý*: Tập các chuỗi mã thực thi cần chuyển đổi trên kiến trúc nguồn đã được gán nhãn sẽ chuyển đổi thành tập các chuỗi mã thực thi trên kiến trúc đích thông qua mô hình chuyển đổi chuỗi mã thực thi chéo kiến trúc tương ứng. Việc chuyển đổi chuỗi mã thực thi chéo kiến trúc được thực hiện nhằm tăng cường tập dữ liệu huấn luyện nhằm xây dựng các mô hình phát hiện mã độc có khả năng phát hiện các mẫu mã độc thực thi trên nhiều nền tảng kiến trúc vi xử lý hoặc tạo ra các tập đặc trưng chuỗi mã thực thi mới của mã độc trên các kiến trúc vi xử lý khác nhau phục vụ xây dựng mô hình phát hiện mã độc zero-day.

- *Trích chọn đặc trưng và tiền xử lý*: Các chuỗi mã thực thi sau khi chuyển đổi bị trống hoặc có độ dài mã thực thi nhỏ hơn 50 (ngắn hơn chuỗi mã thực thi của một tập tin bình thường) sẽ được loại bỏ khỏi tập dữ liệu phục vụ huấn luyện và đánh giá mô hình. Các tập dữ liệu phục vụ xây dựng mô hình học máy phát hiện mã độc IoT hoạt động đa kiến trúc được phân chia theo các kịch bản phát hiện chéo kiến trúc và phát hiện mã độc zero-day. Tiếp theo, phương pháp n-gram được sử dụng để trích chọn các đặc trưng và chuyển đổi đặc trưng chuỗi mã thực thi thành các vectơ để phục vụ huấn luyện mô hình phát hiện mã độc IoT đa kiến trúc.

- *Huấn luyện mô hình phát hiện mã độc dựa trên các thuật toán học máy*: Việc huấn luyện mô hình phát hiện mã độc dựa trên các thuật toán học máy sử dụng đặc trưng chuỗi mã thực thi có thể được thực hiện thông qua các thuật toán học máy truyền thống hoặc mạng học sâu. Mạng học sâu thường được sử dụng trong trường hợp dữ liệu thô ban đầu, số lượng lớn và phức tạp. Còn các thuật toán học máy truyền thống phù hợp đối với các bài toán tài nguyên hạn chế, dữ liệu không quá lớn và trích xuất được đặc

trưng biểu diễn. Vì vậy, với đặc điểm, tính chất của bài toán trong chương 4, việc áp dụng các thuật toán học máy truyền thống sẽ là phù hợp hơn đối với các mạng học sâu và tăng khả năng tích hợp, xử lý trong môi trường IoT hạn chế tài nguyên. Để đánh giá hiệu quả của mô hình phát hiện mã độc IoT đa kiến trúc, các thuật toán học máy có giám sát không sâu truyền thống phổ biến như SVM, RF, NB đã chứng minh hiệu quả trong phát hiện mã độc IoT đơn kiến trúc dựa trên đặc trưng chuỗi mã thực thi trình bày trong chương 3 của luận án tiếp tục được sử dụng để xây dựng các mô hình phát hiện. Các thuật toán học máy có giám sát được luận án lựa chọn với các lý do như sau:

Thứ nhất, dựa trên kết quả khảo sát các nghiên cứu về hiệu quả của một số mô hình phát hiện mã độc IoT sử dụng học máy trong nội dung 1.3 của luận án, các nghiên cứu trong và ngoài nước đã chứng minh hiệu quả tốt với độ chính xác tương đối cao trong phát hiện mã độc nói chung.

Thứ hai, để có cơ sở đánh giá hiệu quả của mô hình đề xuất với các mô hình khác đã công bố, luận án lựa chọn phương pháp phân tích thu thập đặc trưng và thuật toán học máy có điểm tương đồng để đánh giá.

Việc huấn luyện và đánh giá mô hình phát hiện được thực hiện nhiều lần trên cùng một thiết bị và tại các mốc thời gian khác nhau để đánh giá hiệu quả và hiệu năng của mô hình đề xuất.

4.3.2. Đánh giá mô hình phát hiện mã độc IoT đề xuất

Đối với quá trình đánh giá mô hình phát hiện mã độc IoT đề xuất, luận án đánh giá thông qua phương pháp thực nghiệm và tiến hành đánh giá theo các kịch bản đánh giá mô tả trong Bảng 4.3.

Bảng 4.3. Kịch bản đánh giá mô hình phát hiện mã độc IoT đa kiến trúc.

STT	Mã kịch bản đánh giá	Mục tiêu	Phương pháp đánh giá	Cách thức đánh giá
1	DA	Đánh giá hiệu quả của mô hình chuyển đổi trong tăng cường tập dữ liệu	Phương pháp thực nghiệm: Dựa trên việc so sánh kết quả của mô hình phát hiện với tập dữ liệu ban đầu và kết quả của mô hình phát hiện với dữ liệu được bổ sung thêm từ mô hình chuyển đổi chuỗi mã thực thi	Đánh giá dựa trên các tiêu chí độ đo độ chính xác của các mô hình huấn luyện trên tập dữ liệu ban đầu với các mô hình huấn luyện trên tập dữ liệu ban đầu và dữ liệu chuyển đôi .
2	CR	Đánh giá hiệu quả phát hiện	Phương pháp thực nghiệm:	Đánh giá dựa trên các tiêu chí độ đo độ chính

		mã độc IoT đa kiến trúc dựa trên đặc trưng chuyển đổi chéo kiến trúc	Dựa trên các kịch bản sử dụng tập dữ liệu tập tin thực thi ELF từ nhiều nền tảng để huấn luyện và đánh giá mô hình phát hiện.	xác của các mô hình huấn luyện trên các tập dữ liệu đã chuyển đổi chéo kiến trúc và dữ liệu kiến trúc đích.
3	ZR	Đánh giá hiệu quả phát hiện mẫu mã độc IoT zero-day đa kiến trúc dựa trên tập dữ liệu cross-compiler	Phương pháp thực nghiệm: Dựa trên các mã độc biên dịch đa kiến trúc từ cùng một mã nguồn chương trình độc hại.	Đánh giá kết phát hiện mã độc cho các tập tin được cross-compiler trên nhiều kiến trúc vi xử lý từ một mã nguồn mã độc đã công bố và xác định nhãn mã độc trước đó với kết quả gán nhãn với mô hình được huấn luyện dựa trên các tập dữ liệu được tạo ra sau quá trình chuyển đổi đặc trưng chuỗi mã thực thi chéo nền tảng kiến trúc.

Dựa trên yêu cầu đánh giá hiệu quả của các mô hình phát hiện, các cặp dữ liệu huấn luyện và đánh giá được xây dựng để phục vụ các kịch bản thử nghiệm tương ứng. Các tập dữ liệu tập tin thực thi phục vụ kiểm tra, đánh giá hiệu quả trong các kịch bản được tiến hành thu thập chuỗi mã thực thi, trích chọn và tiền xử lý tương tự quá trình huấn luyện mô hình trước đó. Các tiêu chí độ đo độ chính xác cơ bản trong học máy và so sánh kết quả phân lớp tập tin của mô hình với nhãn ban đầu được sử dụng để đánh giá hiệu quả của mô hình phát hiện mã độc IoT đề xuất.

4.4. Thử nghiệm và đánh giá kết quả mô hình đề xuất

4.4.1. Môi trường thử nghiệm

Các thử nghiệm của luận án được thực hiện trên cùng một máy tính sử dụng hệ điều hành Ubuntu 18.04 LTS 64 bit, với cấu hình Intel®Xeon CPU E3-1535M v5 @ 2.90GHz x 8, Quadro M2000M/PCIe/SSE2, RAM 64GB, SSD 500GB. Các thử nghiệm được tiến hành 10 lần tại các mốc thời gian khác nhau. Kết quả đánh giá là trung bình cộng của các giá trị thu được.

Để đánh giá hiệu suất, các phương pháp và mô hình khác cùng hướng tiếp cận đặc trưng của luận án như của Phú và cộng sự [116], Vasan và cộng sự [31] được tiến hành trên cùng một tập dữ liệu tương đồng và thiết bị như các thử nghiệm trong phương pháp của luận án đề xuất.

4.4.2. Tập dữ liệu thử nghiệm

Tập dữ liệu sử dụng để thử nghiệm là các mẫu tập tin thực thi được lựa chọn mở

rộng từ chương 2, chương 3 của luận án và nằm trong tập dữ liệu C500-IoT dataset được thu thập bởi Phú [115] và Trung [85] với 3 nền tảng kiến trúc vi xử lý có số lượng tập tin lớn và có tính đại diện cho các thiết bị IoT sử dụng hệ điều hành Linux nhúng hiện nay. Các tập dữ liệu có sự chênh lệch về số lượng mẫu được gán nhãn trong cùng một kiến trúc và giữa các kiến trúc với nhau. Các tập dữ liệu tập tin lành tính của một số nền tảng kiến trúc còn hạn chế về số lượng do các tập tin lành tính được thu thập từ các nguồn phổ biến gồm bản ảnh phần sụn của thiết bị IoT và máy tính cá nhân như trong các luận án [64], [85], [114]. Việc bóc tách bản ảnh phần sụn và trích xuất các tập tin thực thi của nhiều thiết bị IoT còn hạn chế do cấu tạo phần cứng, khả năng tương tác, giao tiếp với thiết bị IoT thực tế đòi hỏi nhiều thiết bị, công cụ phức tạp. Vì vậy, nghiên cứu sinh thu thập bổ sung các tập tin ELF trên nền các kiến trúc tương ứng của các nhà cung cấp phần mềm tin cậy và đã được đánh giá bởi công cụ quét mã độc trực tuyến Virus Total. Bên cạnh đó, hiện nay để tạo ra các mã độc có khả năng hoạt động trên nhiều kiến trúc vi xử lý, NCS đã thu thập các mã nguồn chương trình mã độc công bố trên các nguồn khác nhau để tiến hành phân tích hành vi mã độc trong mã nguồn và biên dịch tập tin thực thi trên nhiều kiến trúc vi xử lý khác nhau thông qua kỹ thuật cross-compiler phục vụ các kịch bản thử nghiệm. Chi tiết thông tin về tập dữ liệu đa kiến trúc phục vụ thử nghiệm được mô tả trong Bảng 4.4. Trong tập dữ liệu tập tin thực thi thực thi, số lượng tập tin trên hai kiến trúc Intel và MIPS là lớn nhất và đây là các kiến trúc phổ biến trên thiết bị IoT, cũng như có sự khác biệt lớn về đặc điểm các mã thực thi. Tập dữ liệu trên kiến trúc PowerPC có sự mất cân bằng dữ liệu lớn giữa tập tin mã độc và tập tin lành tính. Vì vậy, luận án lựa chọn tập dữ liệu gồm ba kiến trúc Intel, MIPS, PowerPC để thử nghiệm mô hình phát hiện mã độc IoT đề xuất mà không mất tính tổng quát.

Bảng 4.4. Thống kê số lượng mẫu trong tập dữ liệu thử nghiệm.

Kiến trúc vi xử lý	Số lượng mẫu mã độc	Số lượng mẫu lành tính	Số lượng mã độc cross-compiler
Intel 80386	5.460	11.100	65
MIPS	4.604	8.911	65
PowerPC	1.060	60	65

4.4.3. Kết quả xây dựng kịch bản thử nghiệm

Kết quả thu thập chuỗi mã thực thi của các tập dữ liệu thử nghiệm thể hiện trong Bảng 4.5.

Bảng 4.5. Kết quả thu thập chuỗi mã thực thi.

Kiến trúc vi xử lý	Số lượng tập tin ELF		Số lượng tập tin thu thập thành công chuỗi mã thực thi		Độ dài trung bình chuỗi ocpode	
	Mã độc	Lành tính	Mã độc	Lành tính	Mã độc	Lành tính

Intel	5.460	11.100	5.394	11.100	35.291	19.622
MIPS	4.604	8.911	4.592	8.911	31.176	22.345
PowerPC	1.060	60	979	60	9.898	8.186

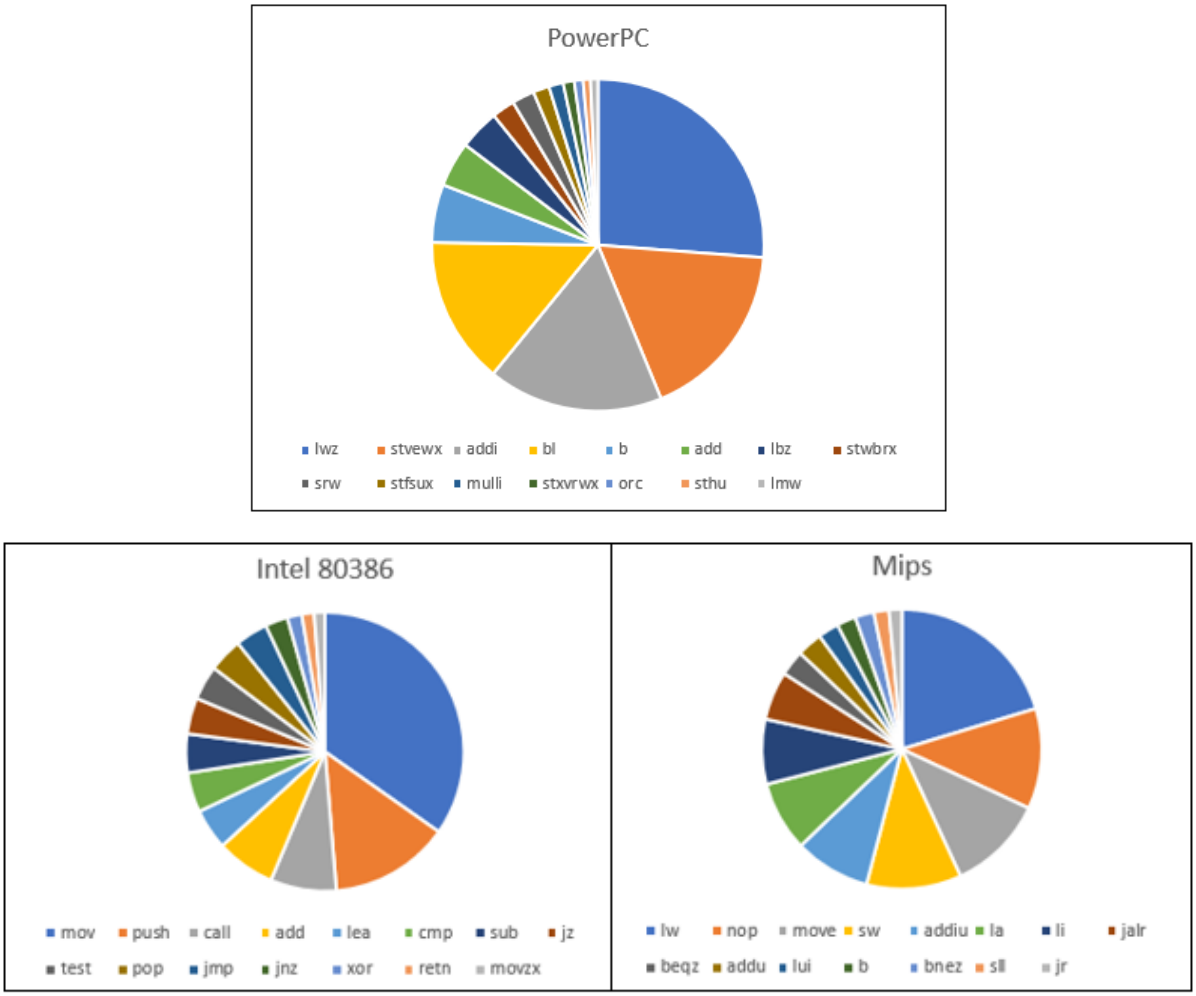
Các chuỗi mã thực thi có độ dài chuỗi nhỏ hơn 100 sinh ra bởi các tập tin sử dụng kỹ thuật làm rối, mã hoá hoặc không thể giải nén sẽ được loại bỏ khỏi tập dữ liệu. Qua phân tích tập dữ liệu thử nghiệm, các mẫu tập tin IoT thử nghiệm trên nền tảng MIPS chủ yếu từ các thiết bị IoT hạn chế tài nguyên và chưa sử dụng các kỹ thuật làm rối phức tạp như làm rối và mã hoá mã thực thi. Mỗi chuỗi mã thực thi thu thập được sẽ đại diện cho một tập tin thực thi ELF trong quá trình xử lý tiếp theo. Kết quả tập dữ liệu chuỗi mã thực thi thu thập được phân theo ngưỡng độ dài tối thiểu thể hiện chi tiết trong Bảng 4.6.

Bảng 4.6. Thống kê so sánh độ dài tối thiểu của chuỗi mã thực thi thu thập được trên các kiến trúc vi xử lý.

Độ dài tối thiểu của chuỗi mã thực thi		**100**	**1000**	**2000**	**6000**	**10000**	**20000**	**50000**	**100000**
Kiến trúc Intel	**Mã độc**	5.375	3.871	3.404	3.011	2.809	1.193	761	637
	Lành tính	10.829	6.544	4.965	2.814	2.037	1.264	596	301
Kiến trúc MIPS	**Mã độc**	4.587	4.270	4.263	4.212	4.124	2.274	326	306
	Lành tính	8.693	5.659	4.615	3.213	2.643	2.047	1.112	497
Kiến trúc PowerPC	**Mã độc**	971	948	944	842	264	9	5	5
	Lành tính	60	38	33	23	19	6	1	0

Kết quả trong Bảng 4.6 cho thấy rằng các tập tin ELF trên các kiến trúc có độ dài chuỗi mã thực thi lớn hơn 2.000 chiếm tỉ lệ cao. Điều này có sự khác biệt lớn đối với độ dài của một câu trong ngôn ngữ tự nhiên. Số lượng các tập tin thu thập được chuỗi mã thực thi nhỏ hơn 100 chiếm tỉ lệ nhỏ (dưới 2%) so với tổng số tập tin ELF được thử nghiệm. Bên cạnh đó, qua phân tích tập dữ liệu mã thực thi thu thập được từ các kiến trúc thử nghiệm, các tập mẫu trên kiến trúc Intel chỉ sử dụng 634 mã thực thi khác nhau và trên tập mẫu của kiến trúc MIPS chỉ sử dụng 260 mã thực thi và trên tập mẫu của kiến trúc PowerPC chỉ sử dụng 176 mã thực thi. Số lượng các mã thực thi được sử dụng phổ biến trên các tập dữ liệu thử nghiệm được thể hiện trong hình 4.9.

Tên và chức năng các mã thực thi phổ biến được sử dụng trên các kiến trúc vi xử lý này có sự khác biệt nhau. Vì vậy, các phương pháp khác thác tần suất xuất hiện để xác định sự tương đồng giữa các mã thực thi sẽ không khả thi.



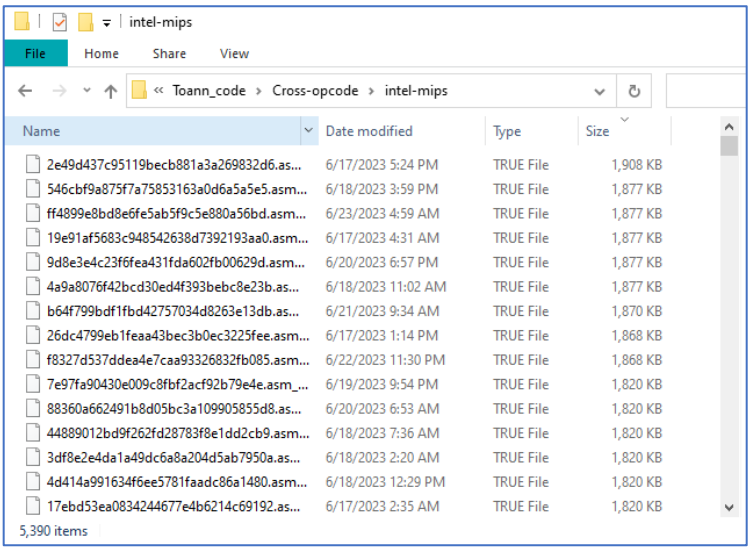
Hình 4.9. Các mã thực thi xuất hiện nhiều nhất trên kiến trúc Intel và MIPS.

Sau đó, các tập dữ liệu chuỗi mã thực thi được sử dụng để xây dựng các mô hình chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc. Kết quả chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc thể hiện như Bảng 4.7, hình 4.11 và hình 4.12.

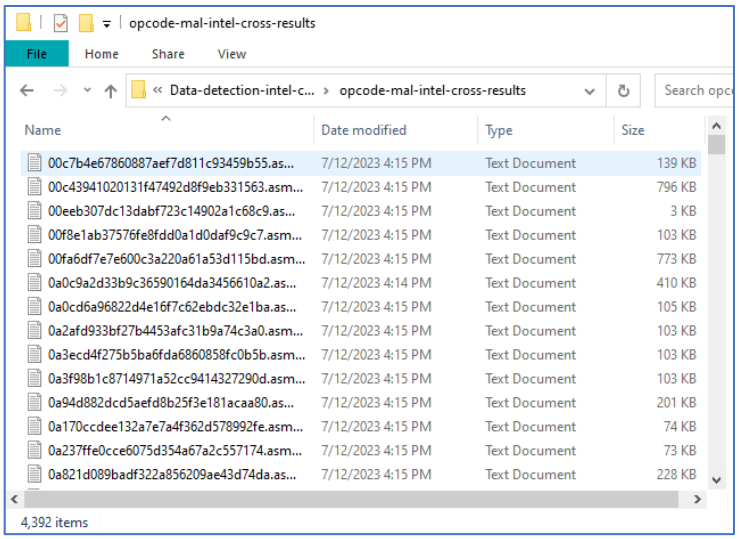
Bảng 4.7. Kết quả chuyển đổi chuỗi mã thực thi chéo kiến trúc vi xử lý.

Kiến trúc vi xử lý nguồn	**Kiến trúc vi xử lý đích**	**Số lượng chuỗi mã thực thi mã độc chuyển đổi thành công**	**Số lượng chuỗi mã thực thi mã độc chuyển đổi không thành công**
Intel	MIPS	5.390	4
Intel	PowerPC	5.350	44
MIPS	Intel	4.392	200
MIPS	PowerPC	4.006	586

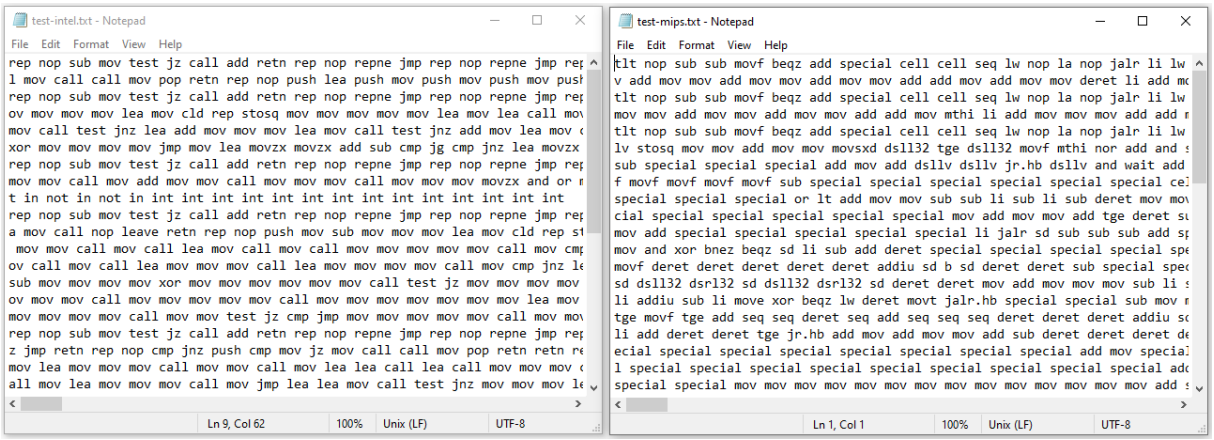
Kết quả các tập tin chứa chuỗi mã thực thi được chuyển đổi thành công thể hiện như trong hình 4.10 và 4.11. So sánh kết quả của một chuỗi mã thực thi tạo ra trên kiến trúc MIPS được chuyển đổi từ chuỗi mã thực thi kiến trúc Intel thể hiện trong hình 4.12.



Hình 4.10. Kết quả các tập tin chứa chuỗi mã thực thi được chuyển đổi từ MIPS sang Intel



Hình 4.11. Kết quả các tập tin chứa chuỗi mã thực thi được chuyển đổi từ Intel sang MIPS



Hình 4.12. So sánh kết quả chuyển đổi một chuỗi mã thực thi tạo ra trên kiến trúc MIPS từ một chuỗi mã thực thi trên kiến trúc Intel.

Cuối cùng, các mô hình chuyển đổi đặc trưng chuỗi mã thực thi chéo kiến trúc được sử dụng để xây dựng các tập dữ liệu phục vụ xây dựng mô hình phát hiện mã độc

IoT đa kiến trúc theo các kịch bản thử nghiệm. Ngoài việc đánh giá hiệu quả khi tăng cường tập dữ liệu. Các kịch bản thử nghiệm tập trung đánh giá mô hình phát hiện mã độc chéo kiến trúc khi huấn luyện trên các kiến trúc vi xử lý phổ biến có số lượng mẫu nhiều như Intel để phát hiện trên các kiến trúc khó thu thập và ít mẫu hơn như MIPS và PowerPC. Bên cạnh đó, để đánh giá khả năng phát hiện mã độc zeroday đa kiến trúc, luận án lựa chọn 10 mã nguồn chương trình mã độc chưa xuất hiện trong tập huấn luyện và kiểm tra chéo kiến trúc để cross-compiler trên các nền tảng kiến trúc khác nhau để phục vụ thử nghiệm. Kết quả xây dựng tập dữ liệu chuỗi mã thực thi phục vụ đánh giá mô hình theo các kịch bản thể hiện chi tiết trong Bảng 4.8.

Bảng 4.8. Kết quả xây dựng các tập dữ liệu chuỗi mã thực thi phục vụ các kịch bản thử nghiệm mô hình phát hiện mã độc IoT đề xuất.

TT	Mã kịch bản	Kiến trúc/Số lượng của tập huấn luyện		Kiến trúc/Số lượng của tập kiểm tra	
		Mã độc	Lành tính	Mã độc	Lành tính
1	DA-01	MIPS/3.674	MIPS/7.128	MIPS/918	MIPS/1.783
2	DA-02	Intel→MIPS/4.312 và MIPS/3.674	MIPS/7.128	Intel→MIPS/1.078 và MIPS/918	MIPS/1.783
3	DA-03	Intel/4.315	Intel/8.880	Intel/1.079	Intel/2.220
4	DA-04	MIPS→Intel/3.514 và Intel/4.315	Intel/8.880	MIPS→Intel/878 và Intel/1.079	Intel/2.220
5	CR-01	Intel→MIPS/5.390	MIPS/7.128	MIPS/4.604	MIPS/1.783
6	CR-02	MIPS→Intel/4.392	Intel/8.880	Intel/5.394	Intel/2.220
7	ZR-01	Intel/5.394	Intel/11.100	Cross-compiler Intel/10	-
8	ZR-02	Intel→MIPS/5.390	MIPS/8.911	Cross-compiler MIPS/10	-
9	ZR-03	Intel→PowerPC/5.350	PowerPC/1060	Cross-compiler PowerPC/10	-

4.4.4. Kết quả đánh giá các mô hình phát hiện mã độc IoT đa kiến trúc đề xuất

Với kết quả đã đạt được trong chương 3 của luận án, để xây dựng mô hình phát hiện hiệu quả trên các kịch bản thử nghiệm, các thuật toán học máy gồm SVM, RF, NB tiếp tục được sử dụng kết hợp với phương pháp n-gram. Các thuật toán học máy SVM, RF, NB được lựa chọn để cài đặt thông qua ngôn ngữ lập trình Python với thư viện Sklearn.

Trong quá trình thực nghiệm các kịch bản thử nghiệm, NCS đã tiến hành thực nghiệm nhiều lần để lựa chọn các siêu tham số phù hợp đối với mỗi thuật toán học máy. Quá trình huấn luyện và kiểm tra khả năng phát hiện của các mô hình, NCS đồng thời tiến hành tinh chỉnh các tham số chính của các thuật toán học máy để tìm ra mô hình phù hợp nhất. Tham số chính được sử dụng trong các thuật toán học máy trong các thực nghiệm sử dụng được mô tả trong Bảng 4.9.

Bảng 4.9. Tham số chính sử dụng trong các thuật toán học máy.

Thuật toán	Tham số	Ý nghĩa tham số	Giá trị
SVM	gamma	Tham số của hàm Radial Basic Function	scale
	decision_function_shape	Cách SVM xác định và xuất kết quả từ hàm quyết định	ovo
	cache_size	Kích thước cache sử dụng để lưu trữ tập dữ liệu trong quá trình huấn luyện	500
	tol	Ngưỡng độ lỗi cho phép trong quá trình tối ưu hóa hàm mất mát	2e-3
	random_state	Khởi tạo quá trình lấy mẫu ngẫu nhiên	1
	test_size	Tỷ lệ tập mẫu sử dụng cho tập kiểm tra	0,2
RF	max_depth	Độ sâu tối đa của cây quyết định	200
	Random_state	Khởi tạo quá trình lấy mẫu ngẫu nhiên	1
	Max_features	Số lượng đặc trưng tối đa được xem xét khi tạo ra một cây quyết định	sqrt
	n_jobs	Số lượng tác vụ sẽ chạy xử lý song song	-1
	n_estimators	Số lượng cây trong tập cây quyết định	50
	test_size	Tỷ lệ tập mẫu sử dụng cho tập kiểm tra	0,2
NB	test_size	Tỷ lệ tập mẫu sử dụng cho tập kiểm tra	0,2
	Random_state	Khởi tạo quá trình lấy mẫu ngẫu nhiên	1

Luận án sử dụng các độ đo Accuracy và F1-score để đánh giá hiệu quả độ chính xác của các mô hình phát hiện mã độc IoT đa kiến trúc. Kết quả thử nghiệm các kịch bản đánh giá kết quả nâng hiệu quả độ chính xác của mô hình sau khi tăng cường dữ liệu hạn chế sự mất cân bằng dữ liệu thể hiện trong Bảng 4.10.

Bảng 4.10. Đánh giá kết quả tăng cường tập dữ liệu của các mô hình sử dụng 2-gram

Mã kịch bản	Mô hình		RF		SVM		NB	
	Huấn luyện	Kiểm tra	Acc (%)	F1-score (%)	Acc (%)	F1-score (%)	Acc (%)	F1-score (%)
DA-01	MIPS	MIPS	99,0	99,0	98,3	98,3	92,7	92,7
DA-02	Intel→MIPS và MIPS	Intel→MIPS và MIPS	99,2	99,1	98,5	98,5	93,5	93,4
DA-03	Intel	Intel	99,1	99,1	96,2	96,1	94,3	94,2
DA-04	MIPS→Intel và Intel	MIPS→Intel và Intel	99,3	99,2	96,9	96,7	94,6	94,3

Kết quả bằng 4.10 cho thấy rằng sau khi tăng cường tập dữ liệu thông qua bổ sung các tập tin chuỗi mã thực thi chuyển đổi chéo kiến trúc vi xử lý có nhãn mã độc thì các mô hình có xu hướng nâng cao độ chính xác do hạn chế được yếu tố mất cân bằng dữ liệu. Tuy nhiên, độ chính xác của các mô hình phát hiện sử dụng học máy truyền thống chưa có sự cải thiện rõ rệt do việc tăng cường dữ liệu huấn luyện sẽ mang lại nhiều ý

nghĩa hơn đối với các mô hình phát hiện mã độc dựa trên các mạng học sâu phức tạp.

Trong chương này, luận án tập trung đánh giá khả năng phát hiện mã độc IoT đa kiến trúc và mã độc zero-day đa kiến trúc dựa trên mô hình chuyển đổi đặc trưng chuỗi mã thực thi. Các kết quả thực nghiệm trong Bảng 4.11, 4.12 và 4.13 đã minh chứng được hiệu quả của phương pháp chuyển đổi đăc trưng. Cụ thể là:

Trong Bảng 4.11, trường hợp huấn luyện trên nền tảng kiến trúc Intel để xây dựng mô hình phát hiện trên nền tảng kiến trúc MIPS cho độ chính xác tốt nhất đạt accuracy là 98,8% và đạt 99,4% trong trường hợp ngược lại huấn luyện trên kiến trúc MIPS để phát hiện trên kiến trúc Intel khi sử dụng thuật toán Random Forest. Tuy nhiên, mô hình phát hiện sử dụng thuật toán Naive Bayes cho kích thước mô hình và thời gian phát hiện tốt nhất, phù hợp cao khi xây dựng trong các giải pháp bảo mật trên môi trường IoT tài nguyên hạn chế mà vẫn có thể đảm bảo độ chính xác phát hiện trên 95%. Vì vậy, đối với từng bài toán cụ thể, yêu cầu về độ chính xác và tài nguyên sử dụng được cân nhắc để lựa chọn mô hình phát hiện phù hợp và hiệu quả nhất. Khi huấn luyện và triển khai ứng dụng các mô hình phát hiện mã độc đa kiến trúc trong môi trường IoT thực tế cần lựa chọn các thuật toán, phương pháp trích chọn đặc trưng n-gram phù hợp để đảm bảo về mặt thời gian và kích thước của mô hình phát hiện triển khai trên các thiết bị IoT.

Bên cạnh đó, thời gian phát hiện của các mô hình học máy khi sử dụng các thuật toán SVM, NB, RF là khác nhau. Thời gian phát hiện của mô hình sử dụng thuật toán NB cho kết quả nhanh nhất do tính đơn giản và được thiết kế dựa trên xác suất. Thời gian phát hiện của mô hình sử dụng thuật toán RF là nhiều nhất do cần tính toán trên nhiều cây quyết định. Tuy nhiên, về mặt tổng thể khi so sánh, đánh giá với các nghiên cứu đã khảo sát trong nội dung 1.3 của luận án này, các mô hình phát hiện mã độc IoT đa kiến trúc trên đảm bảo các độ đo độ chính xác, kích thước mô hình và thời gian phát hiện đáp ứng các yêu cầu ứng dụng triển khai trong các giải pháp bảo mật trên môi trường IoT. Với mục tiêu của luận án và từ kết quả phát hiện mã độc IoT, mô hình phát hiện huấn luyện chéo tập dữ liệu sử dụng thuật toán NB và 2-gram được lựa chọn để thử nghiệm phát hiện mã độc đa kiến trúc và chứng minh khả năng dự đoán mã độc zero-day đa kiến trúc trên các kiến trúc vi xử lý khác nhau. Kết quả phát hiện của mô hình của mô hình khi sử dụng tập dữ liệu trên nền tảng kiến trúc Intel để huấn luyện 3 mô hình phát hiện mã độc trên các nền tảng kiến trúc Intel, MIPS và PowerPC thể hiện trong Bảng 4.12.

Từ kết quả mô tả trong Bảng 4.12 đã chứng minh hiệu quả của mô hình phát hiện mã độc đa kiến trúc trên các thiết bị IoT đa dạng với việc mô hình phát hiện trên nền tảng kiến trúc Intel cho kết quả phân lớp tương đồng với kết quả phân lớp tập tin thực thi của Virus Total khi kết quả phát hiện đạt 10/10 tập tin gán nhãn là mã độc. Bên cạnh đó, mô hình phát hiện trên nảng tảng kiến trúc MIPS có khả năng phát hiện 9/10 tập tin

cross-compiler từ mã nguồn chương trình độc hại là mã độc trong khi các hãng antivirus trên Virus Total không có thông tin về mẫu và kết quả phát hiện trước đó. Một mẫu có mã băm là "5935B26C6B93486EE8CA2BE3571B6FB0" được tạo ra từ một mã nguồn trên nền tảng Intel Virus Total có 2 hãng antivirus gán nhãn là mã độc thì mô hình phát hiện đề xuất đã xác định được nhãn là mã độc trong khi Virus Total chỉ ra rằng không có hãng antivirus nào gán nhãn là mã độc. Đặc biệt, mô hình đề xuất xác định nhãn của 10/10 tập tin thử nghiệm trên nền tảng kiến trúc PowerPC là mã độc trong khi tất cả các hãng antivirus trên Virus Total không có thông tin và chưa gán nhãn cho tập tin được cross-compiler từ mã nguồn chương trình độc hại.

Vì vậy, phương pháp chuyển đổi đặc trưng chuỗi mã thực thi có thể mở ra hướng tiếp cận mới trong phát hiện mã độc zero-day hoạt động đa kiến trúc vi xử lý trên các kiến trúc vi xử lý mới và ít tri thức mã độc trước đó trong tương lai.

Bảng 4.11. Kết quả phát hiện của các mô hình phát hiện mã độc IoT đa kiến trúc sử dụng phương pháp 2-gram.

Mã kịch bản	Mô hình		RF					SVM					NB				
	Huấn luyện	Kiểm tra	Acc (%)	F1-score (%)	Thời gian huấn luyện (giây)	Kích thước model (MB)	Thời gian phát hiện (giây)	Acc (%)	F1-score (%)	Thời gian huấn luyện (giây)	Kích thước model (MB)	Thời gian phát hiện (giây)	Acc (%)	F1-score (%)	Thời gian huấn luyện (giây)	Kích thước model (MB)	Thời gian phát hiện (giây)
CR-01	Intel→MIPS	MIPS	98,8	98,7	4	0,938	0,114	82,8	81,4	60	67	0,041	95,4	95,3	2	0,044	0,008
CR-02	MIPS→Intel	Intel	99,4	99,3	25	0,11	0,144	99,0	98,9	463	59	0,040	97,5	97,3	24	0,24	0,015

Bảng 4.12. Kết quả phát hiện của các mô hình phát hiện mã độc IoT đa kiến trúc sử dụng phương pháp 3-gram.

Mã kịch bản	Mô hình		RF					SVM					NB				
	Huấn luyện	Kiểm tra	Acc (%)	F1-score (%)	Thời gian huấn luyện (giây)	Kích thước model (MB)	Thời gian phát hiện (giây)	Acc (%)	F1-score (%)	Thời gian huấn luyện (giây)	Kích thước model (MB)	Thời gian phát hiện (giây)	Acc (%)	F1-score (%)	Thời gian huấn luyện (giây)	Kích thước model (MB)	Thời gian phát hiện (giây)
CR-01	Intel→MIPS	MIPS	98,5	98,4	221	3,8	0,342	87,0	86,9	2.225	81	0,082	88,5	88,3	178	0,79	0,024
CR-02	MIPS→Intel	Intel	99,2	99,2	1.381	0,45	0,446	97,2	97,1	3.562	69	0,124	94,8	94,6	1344	4,3	0,047

Bảng 4.13. Kết quả phát hiện mã độc IoT được cross-compiler trong các kịch bản ZR với mô hình sử dụng thuật toán NB và 2-gram.

STT	Tên tập tin mã nguồn chương trình	Mã hash MD5 lần lượt trên các nền tảng (Intel, MIPS, PowerPC)	Nhãn do mô hình đề xuất dự đoán	Nhãn do Virustotal cung cấp	
				Số hãng AV gán nhãn mã độc	Số hãng AV gán nhãn lành tính
1	mirai-josho-bot	470335F984A8FEB746562E756DC5B7DC	Mã độc	34	29
		4B6D5DD1E85F15F49A0E249FB6D718C0	Mã độc	Unkonwn	Unkonwn
		39D159E7FF9A03942D0509D58AB0CF26	Mã độc	Unkonwn	Unkonwn
2	elfvirus-virus	DC230881E704A2766D87F7BF1A43085C	Mã độc	2	62
		63CF1601FC98D3089D4F4FC7FD4B35B8	Mã độc	Unkonwn	Unkonwn
		D301F175B74C9F4CB713F9D37395EAA8	Mã độc	Unkonwn	Unkonwn
3	mirai-void-client	093A5F57DF9A0FEAA691BF5F7C254A0C	Mã độc	39	23

		093A5F57DF9A0FEAA691BF5F7C254A0C	Mã độc	Unkonwn	Unkonwn
		46A96AE0E01042DED7586F9DD7E117D8	Mã độc	Unkonwn	Unkonwn
4	mirai-sythe-bot	1DB3006C6FEAD3FF4451BDEA33ACD75A	Mã độc	38	24
		AA1E35895D5C0B0E01DC652170A55066	Mã độc	Unkonwn	Unkonwn
		C73251A3EA4B95574AA37970ED7588E2	Mã độc	Unkonwn	Unkonwn
5	mirai-sora-bot	6A36FCC4307C78E377B8D428579F2F4F	Mã độc	36	26
		A9F865B35A8C13489F81127084D7D696	Mã độc	Unkonwn	Unkonwn
		1B2103985131F568C30111089BFF70EE	Mã độc	Unkonwn	Unkonwn
6	mirai-reaperv2-cnc-client2	39A5172ED61420B8D20592C0A1131E0E	Mã độc	36	25
		5C267536F2BE90AEDE2569D5EA7A6413	Mã độc	Unkonwn	Unkonwn
		3F7CA39E6670FF071B7B58021FD85BB7	Mã độc	Unkonwn	Unkonwn
7	icmp-backdoor-client	079E88B8C03677D7DC1431F2A74F0AAB	Mã độc	9	54
		B7AC6518DE183F83A7B77F4F2FDBF6DD	Mã độc	Unkonwn	Unkonwn
		B13D4346DE039DEFCEEA0A75F22435C8	Mã độc	Unkonwn	Unkonwn
8	Backdoor-linux-Rootin	1DF0EBCE20556E8CE799861BC30B3756	Mã độc	22	40
		3D259F0D2C12A634E8F03B5B8F6BCA01	Mã độc	Unkonwn	Unkonwn
		2FF4ADA779F8D4B724C2737D65494970	Mã độc	Unkonwn	Unkonwn
9	trigemini	66D4EBAA44C96A14A35FB654112E6AF6	Mã độc	Unkonwn	Unkonwn
		9112A73870106A1420C301E71A388025	Mã độc	7	65
		51F38044FB1C1E491D23510C4386F7B6	Mã độc	Unkonwn	Unkonwn
10	backdoor-jimmy-sfe	019461D3FE91682F3E68775B8BCB9DA5	Mã độc	2	61
		5935B26C6B93486EE8CA2BE3571B6FB0	Mã độc	0	59
		97EDD16950923DB07B7C95864853A6FA	Lành tính	Unkonwn	Unkonwn

4.5.5. So sánh hiệu quả với các mô hình khác có liên quan

Để đánh giá hiệu quả của mô hình đề xuất trong phát hiện mã độc IoT đa kiến trúc dựa trên phương pháp phát hiện chéo kiến trúc vi xử lý, NCS tiến hành thực nghiệm và so sánh với một số phương pháp xây dựng mô hình phát hiện mã độc IoT đa nền tảng kiến trúc dựa trên cách tiếp cận huấn luyện và phát hiện chéo kiến trúc đã khảo sát trong nội dung 1.3.2 của luận án. Chi tiết tập dữu liệu các nghiên cứu sử dụng không được tác giả công bố công khai nhưng dựa trên nguồn thu thập và thời gian công bố thì có thể thấy rằng vì vậy, luận án sử dụng Kết quả so sánh được thể hiện trong Bảng 4.13.

Bảng 4. 14. So sánh kết quả phát hiện mã độc đa kiến trúc với các nghiên cứu có liên quan.

Tác giả	**Nguồn thu thập tập dữ liệu thử nghiệm**	**Đặc trưng tĩnh sử dụng**	**Mô hình phát hiện**			**ACC (%)**	**F1-Score (%)**
			Huấn luyện	**Phát hiện**	**Thuật toán**		
Lee [130]	Virus Total	Printable String	X86 + ARM + MIPS	SPARC	RF	98,40	98,41
				X86-64	SVM	96,83	96,87
				PPC	MLP	99,28	99,35
		ELF header	X86 + ARM + MIPS	SPARC	SVM	93,05	92,78
				X86-64		86,94	85,32
				PowerPC		93,79	93,22
		Mã thực thi	X86 + ARM + MIPS	SPARC	RF	53,66	43,68
				X86-64	RF	85,38	84,78
				PowerPC	SVM	35,57	45,87
Phú [116]	C500-IoT: Virus Share, IoTPOT, Detux, Shared	Chuỗi luồng điều khiển thực thi	Intel	MIPS	SVM	95,7	95,5
			MIPS	Intel		58,2	58,1
Vasan [31]	C500-IoT: Virus Share, IoTPOT, Detux, Share	Chuỗi mã thực thi	Intel	MIPS	Hybrid (RNN &CNN)	95,72	93,95
			MIPS	Intel		58,20	56,28
			ARM	X86-64		97,27	97,23
Mô hình đề xuất	C500-IoT: Virus Share, IoTPOT, Detux, Shared	Chuỗi mã thực thi	Intel	MIPS	RF	98,5	98,4
			MIPS	Intel		99,2	99,2

Từ kết quả trong Bảng 4.13 chỉ ra rằng phương pháp đề xuất đã giải quyết được hạn chế của các mô hình hiệu quả khác trong phát hiện mã độc đa kiến trúc dựa trên phát hiện chéo khi có sự khác biệt về đặc trưng mã thực thi trên mỗi kiến trúc vi xử lý. Mô hình phát hiện mã độc đa kiến trúc luận án đề xuất đã cải thiện đáng kể về các độ đo độ chính

xác khi phát hiện chéo kiến trúc vi xử lý. Do các mã độc IoT thử nghiệm hoạt động trên các kiến trúc vi xử lý mới được sử dụng phổ biến như MIPS mang nhiều đặc trưng mã thực thi có tính "di truyền" từ các kiến trúc truyền thống phổ biến như Intel, mô hình luận án đề xuất đã khai khác tốt tính "di truyền" để các mô hình có khả năng phát hiện mã độc IoT chính xác hơn các mô hình của Vasan và Phú đã công bố trong trường hợp huấn luyện trên tập dữ liệu kiến trúc MIPS phục vụ phát hiện trên kiến trúc Intel. Điều này cũng lý giải thực tế hiện nay, các hacker thường sử dụng các kỹ thuật, công cụ để tạo ra các mẫu mã độc và biến thể mã độc mới có khả năng hoạt động trên nhiều thiết bị khác nhau từ một mẫu mã độc (hoặc mã nguồn chương trình độc hại) trước đây trên nền tảng phổ biến của máy tính cá nhân như Intel. Với số lượng mã nguồn chương trình độc hại và mẫu mã độc trên các thiết bị máy tính sử dụng nền tảng vi xử lý phổ biến Intel thì việc cải tiến, biến đổi chúng để tạo ra các biến thể mới có khả năng hoạt động trên các thiết bị IoT đa dạng đã và đang được các hacker tiến hành, đặc biệt ứng dụng trí tuệ nhân tạo để tạo ra các mẫu mã độc mới từ các mẫu đã biết đến trước đây. Vì vậy, bên cạnh việc nâng cao độ chính xác trong phát hiện, mô hình đề xuất còn mở ra khả năng ứng dụng trong xây dựng các giải pháp phát hiện mã độc zero-day, mã độc hoạt động trên các kiến trúc vi xử lý mới được phát triển trong thời gian tới.

Tuy nhiên, trong quá trình thực nghiệm xây dựng mô hình chuyển đổi tập đặc trưng chuỗi mã thực thi giữa các kiến trúc vi xử lý, một số chuỗi mã thực thi của kiến trúc nguồn có độ dài nhỏ, không đa dạng tập mã thực xuất hiện trong chuỗi đã dẫn đến mô hình chuyển đổi chuỗi mã thực thi giữa hai kiến trúc chưa thể chuyển đổi thành công chuỗi mã thực thi do việc biểu diễn và xác định sự tương quan đặc trưng chuỗi mã thực thi chéo kiến trúc vi xử lý còn hạn chế.

4.5. Kết luận chương 4

Trong chương này, luận án đã đề xuất phương pháp chuyển đổi chuỗi mã thực thi chéo kiến trúc vi xử lý để phục vụ xây dựng mô hình phát hiện mã độc IoT đa kiến trúc hiệu quả dựa trên học máy và đặc trưng của tập tin thực thi. Kết quả thực nghiệm đã thể hiện khả năng phát hiện mã độc chéo kiến trúc với độ chính xác tốt, kích thước và thời gian phát hiện của mô hình phù hợp khi triển khai trong môi trường IoT tài nguyên hạn chế. Hơn nữa, phương pháp chuyển đổi chuỗi mã thực thi chéo kiến trúc luận án đề xuất đã chứng minh hiệu quả trong việc phát hiện mã độc hoạt động trên các kiến trúc vi xử lý có sự khác biệt lớn khi so sánh với các mô hình khác có cùng cách tiếp cận. Phương pháp chuyển đổi đặc trưng chuỗi mã thực thi cho phép xây dựng các mô hình dự báo mã độc zero-day trên các kiến trúc vi xử lý khác nhau thông qua xây dựng các tập dữ liệu chuỗi mã thực thi mã độc trên các kiến trúc vi xử lý mới từ các mã độc đã biết.

Tuy nhiên, việc xác định mức độ tương đồng giữa các chuỗi mã thực thi giữa các kiến trúc vi xử lý có khác biệt về tên và chức năng còn hạn chế. Đặc biệt đối với các

thiết bị IoT sử dụng các kiến trúc vi xử lý mới và mã độc tấn công có chủ đích. Trong tương lai, việc tiếp tục nghiên cứu các phương pháp trích xuất đặc trưng mã thực thi và chuyển đổi đặc trưng chéo kiến trúc khác là cần thiết để giải quyết các hạn chế này.

Ý tưởng và kết quả thực nghiệm của mô hình đề xuất trong chương này đã được trình bày, công bố trên các Tạp chí khoa học quốc tế:

- "*CAIMP: Cross-architecture IoT malware detection and prediction based on static feature*", The Computer Journal, 2024; bxae042, https://doi.org/10.1093/comjnl/bxae042 (SCIE index, Q2).

KẾT LUẬN VÀ KIẾN NGHỊ

Sự phát triển của cuộc cánh mạng công nghệ lần thứ 4 với sự bùng nổ của các công nghệ Internet vạn vật, trí tuệ nhân tạo, dữ liệu lớn đã và đang tác động đến mọi mặt của đời sống, xã hội. Các thiết bị IoT đang phát triển vượt bậc về cả số lượng và loại thiết bị. Các hiểm hoạ an toàn thông tin ngày càng gia tăng trên các thiết bị IoT, đặc biệt là mã độc nói chung và mã độc IoT nói riêng đã và đang là hiểm hoạ hàng đầu đối với mỗi hệ thống IoT hiện nay. Bên cạnh đó, sự phát triển của các loại mã độc, hình thành các biến thể và dòng mã độc mới lây nhiễm trên các thiết bị IoT đã tạo ra nhiều thách thức trong việc đảm bảo an ninh, an toàn thông tin. Vì vậy, NCS tập trung tìm hiểu các đặc điểm và xu hướng phát triển của mã độc IoT, các phương pháp, mô hình phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng của tập tin thực thi để làm cơ sở trong nghiên cứu, xây dựng các mô hình phát hiện mã độc IoT hiệu quả.

Nội dung của luận án đã tập trung nghiên cứu các phương pháp phát hiện mã độc IoT dựa trên học máy sử dụng đặc trưng biểu diễn dạng chuỗi thu thập từ phân tích động và phân tích tĩnh tập tin thực thi. Từ đó, luận án nâng cao độ chính xác, giảm số lượng và thời gian trích xuất đặc trưng của tập tin, giảm thời gian phát hiện, nâng cao khả năng phát hiện, dự báo mã độc hoạt động đa kiến trúc, mã độc zero-day của các mô hình phát hiện mã độc IoT dựa trên học máy. Các mô hình phát hiện mã độc trên thiết bị IoT đề xuất trong luận án có tính thực tiễn với việc triển khai ứng dụng một phần trong thực tế tại các đề tài khoa học và công nghệ các cấp [ĐT1], [ĐT2], [ĐT3]. Tuy nhiên, với sự phát triển của các thiết bị và công nghệ IoT, mã độc trên các thiết bị IoT sẽ tiếp tục được xây dựng, cải tiến và lây lan sâu rộng trong thời gian tới. Vì vậy, các nhà nghiên cứu trong và ngoài nước ngày càng quan tâm đến vấn đề phát hiện mã độc hoạt động đơn kiến trúc và đa kiến trúc trên thiết bị IoT. Mặc dù luận án đã đạt được các kết quả nghiên cứu quan trọng về lý luận và thực tiễn trong phát hiện mã độc trên thiết IoT dựa trên học máy và các đặc trưng biểu diễn dạng chuỗi trích xuất từ phân tích động, phân tích tĩnh tập tin nhưng vẫn còn một số vấn đề cần nghiên cứu và cải tiến sau đây:

Thứ nhất, các mô hình đề xuất đang thử nghiệm chủ yếu với bộ dữ liệu IoT do các nhóm tác giả tại Học viện An ninh nhân dân thu thập, tập hợp chủ yếu gồm các mẫu trên nền tảng Intel, MIPS, ARM. Trong thời gian gần đây, nhiều kiến trúc vi xử lý IoT mới như các thiết bị thông minh cầm tay, các thiết bị Raspberry Pi,… đang được phát triển và nhiều biến thể mã độc có thể lây lan trên các thiết bị IoT hạn chế tài nguyên. Số lượng mẫu mã độc trong các thực nghiệm được sử dụng để phù hợp với môi trường thử nghiệm và nhằm đánh giá các mô hình nhưng các thực nghiệm trong tương lai có thể thực hiện với các tập dữ liệu lớn hơn. Do đó, cần thử nghiệm và điều chỉnh tham số các thuật toán học máy của các mô hình đề xuất trong luận án với các tập dữ liệu mới, các tập dữ liệu uy tín khác đã công bố trong thời gian tới. Khi đó, kết quả thu được sẽ tăng độ tin cậy

và hiệu quả của mô hình đã đề xuất trong luận án.

Thứ hai, các mô hình phát hiện mã độc IoT đã đề xuất sử dụng các phương pháp trích xuất đặc trưng của tập tin khác nhau. Mỗi phương pháp trích xuất đặc trưng của tập tin động và tĩnh có những ưu và nhược điểm nhất định. Do đó, nghiên cứu tương lai của luận án có thể kết hợp cả hai đặc trưng biểu diễn dạng chuỗi thu thập từ phân tích động và phân tích tĩnh để xây dựng một mô hình phát hiện mã độc IoT hiệu quả tối ưu nhất dựa trên phương pháp phân tích lai.

Thứ ba, việc sử dụng F-Sandbox để thu thập chuỗi lời gọi hệ thống và các công cụ dịch ngược như IDA Pro để trích xuất chuỗi mã thực thi còn chưa đạt hiệu quả cao trong một số trường hợp. Các mã độc có chức năng phát hiện môi trường phân tích, sử dụng các kỹ thuật làm rối, mã hoá mã thực thi có thể gây khó khăn trong quá trình phân tích động và tĩnh mã độc. Vì vậy, cần có các cải tiến về môi trường sandbox và các phương pháp trích xuất chuỗi mã thực thi để phục vụ xây dựng các mô hình phát hiện mã độc hiệu quả hơn trên các thiết bị IoT.

Thứ tư, các mô hình phát hiện mã độc IoT đề xuất đã giải quyết được vấn đề nâng cao hiệu quả về độ chính xác và số lượng và thời gian thu thập đặc trưng, kích thước, thời gian huấn luyện và phát hiện. Tuy nhiên, một số tiêu chí khác về tính hiệu quả của mô hình học máy như mức độ sử dụng tài nguyên huấn luyện, độ phức tạp của mô hình cũng cần được tiếp tục nghiên cứu, cải tiến để phù hợp với các hệ thống IoT tài nguyên hạn chế.

DANH MỤC CÔNG TRÌNH CỦA TÁC GIẢ

Tất cả các nội dung, kết quả nghiên cứu trình bày trong luận án này đều đã được công bố trên các tạp chí, hội thảo uy tín chuyên ngành trong nước và quốc tế. Cụ thể như sau:

[TC1]: Luong The Dung**, Nguyen Ngoc Toan,** Tran Nghi Phu**,** *CAIMP: Cross-architecture IoT malware detection and prediction based on static feature*, The Computer Journal, 2024;bxae042, https://doi.org/10.1093/comjnl/bxae042 (SCIE index, Q2).

[TC2]: Nguyen Ngoc Toan, Luong The Dung, Dang Quang Thang, *Static Feature Selection for IoT Malware Detection*, Journal of Science and Technology on Information Security, Special Issue CS (15) 2022, https://doi.org/10.54654/isj.v1i15.844, ISSN 2615 -9570.

[TC3]: Toan Nguyen Ngoc, Dung Luong The, Phu Tran Nghi, A *novel approach to detect IoT malware by System calls and Long Short-term Memory model*, Journal of Theoretical and Applied Information Technology, 31st August 2021 -- Vol. 99. No. 16 – 2021 (SCOPUS,Q4), ISSN: 1992-8645.

[TC4]: Tran Nghi Phu, Nguyen Dai Tho, Le Huy Hoang, **Nguyen Ngoc Toan**, Nguyen Ngoc Binh, *An Efficient Algorithm to Extract Control Flow-Based Features for IoT Malware Detection*, The Computer Journal, Volume 64, Issue 4, April 2020, Pages 599–609, https://doi.org/10.1093/comjnl/bxaa087. (SCIE index, Q2).

Một số công trình tham khảo khác đã công bố:

[HT1]: Ngoc Toan Nguyen, Xuan Tuan Le; The Dung Luong, *An Ensemble Method for Sentiment Classification of Long Vietnamese Documents*, 2022 RIVF International Conference on Computing and Communication Technologies (RIVF), Ho Chi Minh City, Vietnam, 2022, pp. 428-433, doi: 10.1109/RIVF55975.2022.10013802. (SCOPUS index), ISSN: 2162-786X.

Bên cạnh đó, trong thời gian thực hiện luận án, một phần kết quả nghiên cứu được ứng dụng trong các đề tài nghiên cứu khoa học NCS là đồng tác giả sau đây:

[ĐT1]: Thành viên Đề tài khoa học và công nghệ cấp Bộ "*Nghiên cứu xây dựng hệ thống phân tích, phát hiện mã độc và lỗ hổng bảo mật trong phần sụn (firrmware) của một số thiết bị mạng*", đã nghiệm thu năm 2022.

[ĐT2]: Thành viên chính Đề tài khoa học và công nghệ cấp Bộ "*Nghiên cứu xây dựng hệ thống diễn tập phòng thủ và tấn công mạng phục vụ công tác đào tạo, huấn luyện đảm bảo an toàn thông tin mạng*", đã nghiệm thu năm 2022.

[ĐT3]: Thư ký khoa học Đề tài khoa học và công nghệ cấp cơ sở "*Nghiên cứu phát triển công cụ giám sát máy chủ tài Học viện An ninh nhân dân*", đã nghiệm thu năm 2022.

TÀI LIỆU THAM KHẢO

[1] A. Conneau, G. Lample, M.A. Ranzato, L. Denoyer, H. Jégou, "Word translation without parallel data", arXiv preprint arXiv:1710.04087, 2017.

[2] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares", USENIX, Proceedings of the 23rd USENIX Conference on Security Symposium, p.95–110, 2014.

[3] A. Damodaran, F.D. Troia, C.A. Visaggio, T.H. Austin, M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection", Journal of Computer Virology and Hacking Techniques, 13, 1-12, 2017.

[4] A. Dehghantanha, A. Azmoodeh , K.K.R. Choo, "Robust Malware Detection for Internet Of (Battlefield) Things Devices Using Deep Eigenspace Learning", IEEE Transactions on Sustainable Computing, 4 (1) ISSN 2377-3782 , p.88–95, 2018.

[5] A. H. Celdrán, P. M. S. Sánchez, M. A. Castillo, G. Bovet, G. M. Pérez, B. Stiller, "Intelligent and behavioral-based detection of malware in IoT spectrum sensors", International Journal of Information Security, 2022.

[6] A. O. Prokofiev, Y. S. Smirnova, and V. A. Surov, "A method to detect Internet of Things botnets", in 2018 IEEE Conference of Russian Young 121 Researchers in Electrical and Electronic Engineering (EIConRus), Moscow, 2018.

[7] A. P. Felt, M. Finifter, E. Chin, S. Hanna, D. Wagner, "A survey of mobile malware in the wild", The Security and Privacy in Smartphones and Mobile Devices (SPSM), pp.3-14, 2011.

[8] A. Yewale, M. Singh, "Malware detection based on opcode frequency", in Proc. 2016 Int. Conf. Adv. Commun. Control Comput. Technol. ICACCCT, 2016.

[9] Atitallah, S. Ben, M. Driss, I. Almomani, "A novel detection and multi-classification approach for IoT-malware using random forest voting of fine-tuning convolutional neural networks", Sensors 22.11 (2022): 4302, 2022.

[10] Aohui Wang et al., "An Inside Look at IoT Malware", International Conference on Industrial IoT Technologies and Applications, Wuhu, China: Springer, pp176–186, 2017.

[11] B. Kang, S.Y. Yerima, S. Sezer, K. McLaughlin, "N-gram Opcode Analysis for Android Malware Detection", International Journal on Cyber Situational Awareness, Vols. Vol. 1, No. 1, pp.231-255, 2016.

[12] B. Xue, M. Zhang, W.N. Browne, "A comprehensive comparison on evolutionary feature selection approaches to classification", Int. J. Comput. Intell. Appl, 2015.

[13] C. Guarnieri, A. Tanasi, J. Bremer, M. Schloesser, "The cuckoo sandbox", 2012.

[14] C. Kolias, G. Kambourakis, A. Stavrou, J. Voas, "DDoS in the IoT: Mirai and other botnets", IEEE Computer Society, vol. 50, pp.80–84, 2017.

[15] C. Lévy-Bencheton, E. Darra, G. Tétu, G. Dufay, and M. Alattar, 'Security and resilience of smart home environments good practices and recommendations', Eur. Union Agency Netw. Inf. Secur. ENISA Heraklion Greece, 2015.

[16] C. Li, Q. Lv, N. Li, Y. Wang, D. Sun, Y. Qiao, "A novel deep framework for dynamic

malware detection based on API sequence intrinsic features", Computers & Security, 116, 102686, 2022.

[17] C. W. Tien, S. W. Chen, T. Ban, S.Y. Kuo, "Machine Learning Framework to Analyze IoT Malware Using ELF and Opcode Features", Digital Threats: Research and Practice, 1(1), pp.1-19, 2020.

[18] C. Willems, T. Holz, F. Freiling, "Toward automated dynamic malware analysis using cwsandbox", IEEE Secur. Priv., Vols. vol. 5, no. 2, 2007.

[19] C. Y. Wu, T. Ban, S. M. Cheng, T. Takahashi, D. Inoue, "IoT malware classification based on reinterpreted function-call graphs", Computers & Security, 125, 103060, 2023.

[20] Capstone, URL: https://www.capstone-engine.org/, (Accessed 14 October 2022).

[21] Chaganti, Rajasekhar, Vinayakumar Ravi, and Tuan D. Pham, "Deep learning based cross architecture internet of things malware detection and classification", Computers & Security 120 (2022): 102779, 2022.

[22] Chandriah, Kiran Kumar, and Raghavendra V. Naraganahalli. "RNN/LSTM with modified Adam optimizer in deep learning approach for automobile spare parts demand forecasting." Multimedia Tools and Applications 80.17 (2021): pp. 26145-26159, 2021.

[23] Cilimkovic, Mirza. "Neural networks and back propagation algorithm." Institute of Technology Blanchardstown, Blanchardstown Road North Dublin 15.1, 2015.

[24] Congressional Research Service, [Online] Available: https://sgp.fas.org/crs/misc/IF11239.pdf, (Accessed 5 June 2022)

[25] Costin, Andrei, "Large Scale Security Analysis of Embedded Devices' Firmware", Thesis of Doctor, Paris Institute of Technology, France, 2016.

[26] D. Bilar, "Opcodes as predictor for malware", Int. J. Electronic Security and Digital Forensics, Vol. 1, No. 2, 2007.

[27] D. Carlin, A. Cowan, P. O'Kane, S. Sezer., "The effects of traditional anti-virus labels on malware detection using dynamic runtime opcodes", IEEE Access 5, pp.17742–17752, 2017.

[28] D. Gibert, C. Mateu, J. Planes, R. Vicens., "Classification of malware by using structural entropy on convolutional neural networks", in In: IAAI Conference on Artificial Intelligence, 2018.

[29] D. S. Berman, A. L. Buczak,, J. S. Chavis, C. L. Corbett, "A Survey of Deep Learning Methods for Cyber Security", Information. 10. 122. 10.3390/info10040122, 2019.

[30] D. Uhrıcek, "LiSa–Multiplatform Linux Sandbox for Analyzing IoT Malware," http://excel.fit.vutbr.cz/submissions/2019/058/58.pdf, 2019.

[31] D. Vasan, M. Alazab, S. Venkatraman, J. Akram, Z. Qin, "MTHAEL: Cross-architecture IoT malware detection based on neural network advanced ensemble learning", IEEE Transactions on Computers 69.11 (2020), pp.1654-1667, 2020.

[32] D. Yuxin, Z. Siyi, "Malware detection based on deep learning algorithm", Neural Comput, Appl. 31 (2), pp.461–472, 2019.

[33] Daniel Uhrıcek, "LiSa – Multiplatform Linux Sandbox for Analyzing IoT Malware",

Excel@FIT, 2019.

[34] Detux, "Detux: The Multiplatform Linux Sandbox," https://github.com/detuxsandbox/detux, 2020 (Accessed 12 February 2022).

[35] Ding Yuxin and Zhu Siyi, "Malware detection based on deep learning algorithm". Neural Comput. Appl. 31, 2 (February 2019), pp. 461–472, 2019.

[36] E. Cozzi, M. Graziano, Y. Fratantonio, D. Balzarotti, "Understanding Linux Malware", IEEE Symposium on Security and Privacy, pp.870–884, 2018.

[37] E. M. Dovom, A. Azmoodeh, A. Dehghantanha, D. E. Newton, R. M. Parizi, H. Karimipour, "Fuzzy Pattern Tree for Edge Malware Detection and Categorization in IoT", J. Syst. Archit, 2019.

[38] E. S. Alomari, R. R. Nuiaa, Z. A. A. Alyasseri, H. J. Mohammed, N. S. Sani, M. I. Esa, B. A. Musawi, "Malware Detection Using Deep Learning and Correlation-Based Feature Selection", Symmetry 15.1 (2023): 123, 2023.

[39] E. Zhu, J. Zhang, J. Yan, K. Chen, C. Gao, "N-gram MalGAN: Evading machine learning detection via feature n-gram". Digital communications and networks, 8(4), pp.485-491, 2022.

[40] F. A. Narudin, A. Feizollah, N. B. Anuar, A. Gani, “Evaluation of machine learning classifiers for mobile malware detection”, Soft Comput20.(1), pp.343–357, 2016.

[41] FastText, "GitHub - facebookresearch/fastText: Library for fast text representation and classification," (Accessed 10 September 2022).

[42] G. Bendiab, S. Shiaeles, A. Alruban, N. Kolokotronis, "IoT Malware Network Traffic Classification using Visual Representation and Deep Learning", in IEEE Conference on Network Softwarization (NetSoft), Ghent, Belgium, 2020.

[43] G. Lample, M. Ott, A. Conneau, L. Denoyer, M.A. Ranzato, "Phrase-based & neural unsupervised machine translation", arXiv preprint arXiv:1804.07755, 2018.

[44] G. Zhao, K. Xu, L. Xu, B. Wu, "Detecting apt malware infections based on malicious dns and traffic analysis", IEEE Access 3, pp.1132–1142, 2015.

[45] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Mohaisen, "Analyzing and Detecting Emerging Internet of Things Malware: A Graph-based Approach", IEEE Internet Things J. Prepr, vol. 6, pp.8977–8988, 2019.

[46] H. Darabian, A. Dehghantanha, S. Hashemi, S. Homayoun, K.K.R. Choo, , "An opcode-based technique for polymorphic Internet of Things malware detection", Concurrency and Computation: Practice and Experience, 32(6), e5173, 2020.

[47] H. HaddadPajouh, A. Dehghantanha, R. Khayami, K. R. Choo, "A Deep Recurrent Neural Network Based Approach for Internet of Things Malware Threat Hunting", Future Gener. Comput. Syst., vol. 85, pp.88–96, 2018.

[48] I Janiesch, Christian, Patrick Zschech, Kai Heinrich, "Machine learning and deep learning", Electronic Markets 31.3, pp. 685-695, 2021.

[49] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, Y. Bengio, "Generative adversarial networks", Communications of the ACM 63.11, pp.139-144, 2020.

[50] IDA pro. URL: https://www.hex-rays.com. (Accessed 11 October 2022).

[51] IoT, "Internet of Things-IOT : Definition , Characteristics , Architecture , Enabling Technologies , Application & Future Challenges," vol. 6(5), no. https://doi.org/10.4010/2016.1482, 2016.

[52] IoT OWASP TOP 10, URL: https://owasp.org/www-chapter-toronto/assets/slides/2019-12-11-OWASP-IoT-Top-10---Introduction-and-Root-Causes.pdf, (Accessed 12 October 2022)

[53] ITU, [Online] Available: https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx. (Accessed 9 January 2022).

[54] J. Abawajy, A. Darem and A. Alhashmi, "Feature Subset Selection for Malware Detection in Smart IoT Platforms", 2021.

[55] J. Aycock, "Computer Virus and Malware", Springer, 2006.

[56] J. Uitto, S. Rauti, S. Laurén, V. Leppänen, "A survey on anti-honeypot and anti-introspection methods", In Recent Advances in Information Systems and Technologies: Volume 2 5, Springer International Publishing, pp. 125-134, 2017.

[57] J. Su, D.V. Vargas, S. Prasad, D. Sgandurra, Y. Feng, K. Sakurai, "Lightweight Classification of IoT Malware based on Image Recognition", 2018 IEEE 42nd Annu. Comput. Softw. Appl. Conf. COMPSAC, vol. 2, pp.664–669, 2018.

[58] J.Li, D. Xue, W. Wu, J. Wang, "Incremental Learning for Malware Classification in Small Datasets", Security and Communication Networks 2020, 2020.

[59] Jeon, Jueun, Jong Hyuk Park, Young-Sik Jeong, "Dynamic analysis for IoT malware detection with convolution neural network model", IEEE Access 8 (2020), pp.96899-96911, 2020.

[60] Johnson, Jeff, Matthijs Douze, and Hervé Jégou, "Billion-scale similarity search with gpus", IEEE Transactions on Big Data 7.3, pp.535-547, 2019.

[61] K. Angrishi, "Turning Internet of Things (IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets", ArXiv170203681v1 CsNI, pp. 13, 2017.

[62] K. Monnappa, "Automating linux malware analysis using limon sandbox", in Black Hat Eur, 2015.

[63] Kevin Allix, Quentin Jerome, Tegawendes F. Bissyande, Jacques Klein, Radu State and Yves Le Traon, "A Forensic Analysis of Android Malware – How Is Malware Written and How It Could Be Detected?", IEEE 38th Annual Computer Software and Applications Conference, Vasteras, Sweden, pp. 384-393, 2014.

[64] Lê Hải Việt, Luận án tiến sĩ "Nghiên cứu xây dựng hệ thống V-Sandbox trong phân tích và phát hiện mã độc IoT Botnet", Học viện KHCN, Viện Hàn lâm khoa học Việt Nam, 2022.

[65] Le Hai Viet, Ngo Quoc Dung, "V-Sandbox for Dynamic Analysis IoT Botnet", IEEE Access,1-11. 10.1109/ACCESS.2020.3014891, 2020.

[66] Le Quoc, Tomas Mikolov, “Distributed representations of sentences and documents”, In Proceedings of the 31 st International Conference on Machine Learning, pp.1188–1196, 2014.

[67] Liberty, Edo, Kevin Lang, and Konstantin Shmakov. "Stratified sampling meets machine learning." International conference on machine learning. PMLR, 2016.

[68] Lương Thế Dũng, Hoàng Thanh Nam, "Giáo trình Mã độc", Học viện Kỹ thuật mật mã, Hà Nội, Việt Nam, 2013.

[69] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification", in CODASPY 16. In: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, ACM, New York, NY, USA, 2016.

[70] M. Alhanahnah, Q. Lin, Q. Yan, "Efficient signature generation for classifying cross-architecture IoT malware", 2018 IEEE conference on communications and network security (CNS), 2018.

[71] M. Ali, S. Shiaeles, G. Bendiab, B. Ghita, "MALGRA: Machine learning and N-gram malware feature extraction and detection system", Electronics, 9(11), 1777, 2020.

[72] M. Artetxe, G. Labaka, E. Agirre, K. Cho, "Unsupervised neural machine translation", arXiv preprint arXiv:1710.11041, 2017.

[73] M. Artetxe, G. Labaka, E. Agirre, Learning bilingual word embeddings with (almost) no bilingual data, In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 451-462, 2017.

[74] M. Asam, S. H. Khan, A. Akbar, S. Bibi, T. Jamal, A. Khan, "IoT malware detection architecture using a novel channel boosted and squeezed CNN", Scientific Reports 12.1 (2022), pp.1-12, 2022.

[75] M. Dimjašević, S. Atzeni, I. Ugrina, Z. Rakamaric, "Evaluation of Android Malware Detection Based on System Calls", in The 2016 ACM on International Workshop on Security and Privacy Analytics (IWSPA '16).Association for Computing Machinery, New York, NY, USA, 2016.

[76] M. Ficco, "Detecting IoT Malware by Markov Chain Behavioral Models", pp.229-234, 2019.

[77] M. Ghiasi, A. Sami, Z. Salehi, "Dynamic vsa: a framework for malware detection based on register contents", Eng. Appl. Artif. Intell. 44, pp.111–122, 2015.

[78] M. K. Alzaylaee, S. Y. Yerima, S. Sezer, "Dl-droid: Deep learning based android malware detection using real devices", Computers & Security, 89:101663, February 2020, 2020.

[79] M. Mas'ud, S. Sahib, M. Abdollah, S. Selamat and C. Huoy, "A comparative study on feature selection method for N-gram mobile malware detection", Int. J. Netw. Secur, pp.727–733, 2017.

[80] M. Shobana, S. Poonkuzhali, "A novel approach to detect IoT malware by system calls using Deep learning techniques", in In 2020 International Conference on Innovative Trends in Information Technology (ICITIIT), IEEE, pp.1-5,2020.

[81] M. Stamp, M. Alazab, A. Shalaginov, "Malware Analysis Using Artificial Intelligence and Deep Learning", Berlin/Heidelberg, Germany: Springer, 2021.

[82] M. Tomas, K. Chen, G. Corrado, J. Dean, "Efficient estimation of word

representations in vectơ space", Proceedings of the International Conference on Learning Representations (ICLR 2013) URL: https://arxiv.org/abs/1301.3781, 2013.

[83] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince and F. Herrera, "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 4, pp. 463-484, 2012.

[84] N. A. M. Ariff, M. Z. Mas' ud, N. Bahaman, E. Hamid, N. A. Anuar, "Ensemble method for mobile malware detection using n-gram sequences of system calls", International Journal of Communication Networks and Information Security, 13(2), pp.236-241, 2021.

[85] Nguyễn Huy Trung, Luận án tiến sĩ "Nghiên cứu đề xuất đặc trưng đồ thị PSI trong phát hiện mã độc Botnet trên các thiết bị IoT", Học viện KHCN, Viện Hàn lâm khoa học Việt Nam, 2021.

[86] Nguyen Long Giang, Braulio Dumba, Ngoc Quoc Dung, Le Hai Viet, Nguyen Ngoc Tu, "A collaborative approach to early detection of IoT Botnet", Computers & Electrical Engineering Journal, ISSN: 0045-7906, 2021.

[87] Nguyễn Ngọc Toàn, Luận văn thạc sỹ " Nghiên cứu giải pháp hỗ trợ phân tích mã độc cho thiết bị IoT (Internet of Things)", Học viện Kỹ thuật mật mã, 2019.

[88] OllyDBG. URL: https://www.ollydbg.de/ (Accessed 16 January 2023).

[89] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, E. Herbst, "Moses: Open source toolkit for statistical machine translation", Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions, 2007.

[90] P.Y.M. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, C. Rossow, "IoTPOT: A novel honeypot for revealing current IoT threats", J Inf Process, 24.(3), pp. 522–533, 2016.

[91] Padawan Sandbox, "Padawan Sandbox". https://padawan.s3.eurecom.fr/about (Accessed 8 Mar 2020).

[92] Q. D. Ngo, H.T. Nguyen, V.H. Le, D.H. Nguyen, "A survey of IoT malware and detection methods based on static features", ICT Express, 6(4), pp.280-286, 2020.

[93] Qaiser, Shahzad, Ali, Ramsha, "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents", International Journal of Computer Applications, 181. 10.5120/ijca2018917395, 2018.

[94] R. Canzanese, S. Mancoridis, M. Kam, "System Call-based Detection of Malicious Processes", https://doi.org/10.1109/QRS.2015.26, pp.119-124, 2015.

[95] R. Pascanu, J.W. Stokes, H. Sanossian, M. Marinescu, A. Thomas, "Malware classification with recurrent networks", in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, pp. 1916-1920, 2015.

[96] Raymond Canzanese. Detection and Classification of Malicious Processes Using System Call Analysis. Doctor of Philosophy, Drexel University, 2015.

[97] REMnux, "REMnux: A free Linux Toolkit for Reverse-Engineering and Analyzing

Malware," https://remnux.org/ , 2023, (Accessed 10 March 2022).

[98] .Mathew, J., and M. A. Ajay Kumara. "API call based malware detection approach using recurrent neural network—LSTM." Intelligent Systems Design and Applications: 18th International Conference on Intelligent Systems Design and Applications (ISDA 2018) held in Vellore, India, December 6-8, 2018, Volume 1. Springer International Publishing, 2020.

[99] S. Li, Q. Zhou, R. Zhou, Q. Lv, "Intelligent malware detection based on graph convolutional network", The Journal of Supercomputing, 78(3), pp.4182-4198, 2022.

[100] S. Maniath, A. Ashok, P. Poornachandran, V. G. Sujadevi, P. Sankar A.U. and S. Jan, "Deep learning LSTM based ransomware detection," 2017 Recent Developments in Control, Automation & Power Engineering (RDCAPE), Noida, India, pp.442-446, 2017.

[101] S. S. Hansen, T. M. T. Larsen, M. Stevanovic and J. M. Pedersen, "An approach for detection and family classification of malware based on behavioral analysis," 2016 International Conference on Computing, Networking and Communications (ICNC), Kauai, HI, USA, pp.1-5, 2016.

[102] S. Sriram, R. Vinayakumar, M. Alazab, K. Soman, "Network Flow based IoT Botnet Attack Detection using Deep Learning", pp.189–194, 2020.

[103] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, T. Yagi, "Malware detection with deep neural network using process behavior", in 2016 IEEE 40th annual computer software and applications conference (COMPSAC), Vol. 2, IEEE, pp. 577-582, 2016.

[104] S. Yadav and S. Shukla, "Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification," 2016 IEEE 6th International Conference on Advanced Computing (IACC), Bhimavaram, India, pp. 78-83, 2016.

[105] Sagi, Omer, and Lior Rokach. "Ensemble learning: A survey." Wiley interdisciplinary reviews: data mining and knowledge discovery 8.4 (2018): e1249, 2018.

[106] Sepp Hochreiter, Jürgen Schmidhuber, "Long Short-Term Memory", Neural Comput 1997; 9 (8): pp.1735–1780, 1997.

[107] Sonicwall, Available: https://www.sonicwall.com/2023-cyber-threat-report/,(Accessed 10 July 2022).

[108] SW. Kim, JM. Gil, "Research paper classification systems based on TF-IDF and LDA schemes", Hum, Cent, Comput, Inf, Sci. 9, 30 (2019).

[109] T. Islam, S.S.M.M. Rahman, M.A. Hasan, A.S.M.M Rahaman,M.I. Jabiullah, "Evaluation of N-gram based multi-layer approach to detect malware in Android", Procedia Computer Science, 171, pp.1074-1082, 2020.

[110] T. Apostolopoulos, V. Katos, K.K.R. Choo, C. Patsakis, "Resurrecting anti-virtualization and anti-debugging: Unhooking your hooks." Future Generation Computer Systems 116, pp.393-405, 2021.

[111] T.L. Wan et al., "Efficient Detection and Classification of Internet-of-Things Malware Based on Byte Sequences from Executable Files," in IEEE Open Journal of the Computer Society, vol. 1, pp.262-275, 2020.

[112] T.L. Wan, T. Ban, S.M. Cheng, Y.T. Lee, B. Sun, R. Isawa, T. Takahashi, D. Inoue, ‘‘Efficient detection and classification of Internet-of-Things malware based on byte sequences from executable files,’’ IEEE Open J. Comput. Soc., vol.1, pp.262–275, 2020.

[113] Tống Anh Tuấn, Luận án tiến sĩ "Nghiên cứu cải tiến một số mô hình học máy và học sâu áp dụng cho bài toán phân loại DGA Botnet", Học viện KHCN, Viện Hàn lâm khoa học Việt Nam, 2023.

[114] Trần Nghi Phú, Luận án tiến sĩ "Phân tích tự động mã độc trong các thiết bị nhúng trên nền Linux", Đại học công nghệ, Đại học Quốc gia Hà Nội, 2020

[115] Tran Nghi Phu, Hoang Dang Kien, Ngo Quoc Dung, Nguyen Dai Tho, "A Novel Framework to Classify Malware in MIPS Architecture-Based IoT Devices”, Hindawi Security and Communication Networks Volume 2019, Article ID 4073940, 2019.

[116] Tran Nghi Phu, Le Huy Hoang, Nguyen Ngoc Toan, Nguyen Dai Tho, Nguyen Ngoc Binh, "CFDVex: A Novel Feature Extraction Method for Detecting Cross-Architecture IoT Malware", In SoICT’ 19: The Tenth International Symposium on Information and Communication Technology, December 4–6, 2019, Hanoi - Ha Long Bay, Vietnam. ACM, New York, NY, USA, 2019.

[117] Tran Nghi Phu, Le Huy Hoang, Nguyen Ngoc Toan, Nguyen Dai Tho, Nguyen Ngoc Binh, "C500-CFG: A Novel Algorithm to Extract Control Flow-based Features for IoT Malware Detection", in 19th International Symposium on Communications and Information Technologies (ISCIT), Ho Chi Minh city, 2019.

[118] Tsujitani, Masaaki, and Yusuke Tanaka. "Cross-validation, bootstrap, and support vectơ machines." Advances in Artificial Neural Systems 2011.1: 302572, 2011.

[119] V. Bolón-Canedo, N. Sánchez-Maroño and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data", Knowl. Inf. Syst, pp.483–519, 2013.

[120] V. Sihag, M. Vardhan, P. Singh, G. Choudhary, S. Son, "De-LADY: Deep learning based Android malware detection using Dynamic features", J. Internet Serv. Inf. Secur., 11(2), pp.34-45, 2021.

[121] Yingying Liu, and Yiwei Wang. "A robust malware detection system using deep learning on API calls." 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). IEEE, 2019.

[122] W. Jung, H. Zhao, M. Sun, G. Zhou, "IoT botnet detection via power consumption modeling", Smart Health, vol. 15, 2020.

[123] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, C. Talhi, "An anomaly detection system based on variable N-gram features and one-class SVM", Information and Software Technology, 91, pp.186-197, 2017.

[124] Wendell E. Carter, [Online] Available: https://myamend.com/linux-takes-lead-in-iot-market-keeping-80-market-share/, (Accessed 1 July 2022).

[125] Y. Ding, W. Dai, S. Yan, Y. Zhang, "Control Flow-Based Opcode Behavior Analysis

for Malware Detection", Computers & Security 44, 2014.

[126] Y. Ding, X. Yuan, K. Tang, X. Xiao, Y. Zhang, "A fast malware detection algorithm based on objective-oriented association mining", Computers & Security, Volume 39, Part B, ISSN 0167-4048, vol. 39, pp.315-324, 2013.

[127] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, G. Vigna, "Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware", Internet Society, NDSS Symposium, pp.1-15, 2015.

[128] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent pe-malware detection system based on association mining", J. Comput. Virol. 4 (4), pp.323–334, 2008.

[129] Y. Ye, T. Li, D. Adjeroh, S.S. Iyengar, "A survey on malware detection using data mining techniques", ACM Comput Surv CSUR. Số 50.(3), pp.1-40, 2017.

[130] Y.T. Lee, T. Ban, T.L. Wan, S.M. Cheng, R. Isawa, T. Takahashi, and D. Inoue, ‘‘Cross platform IoT-malware family classification based on printable strings’’ in Proc IEEE 19th Int Conf Trust, Secur. Privacy Comput. Commun, pp. 775–784, 2020.

[131] Ying, Xue. "An overview of overfitting and its solutions." Journal of physics: Conference series. Vol. 1168. IOP Publishing, 2019.

[132] Yu Bo et al., "A survey of malware behavior description and analysis", Front Inf Technol Electron Eng, in 19.(5), pp.583–603, 2018.

[133] Z. Fuyong, Z. Tiezhu, "Malware detection and classification based on n-grams attribute similarity", in 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), vol. 1.2017, pp.793–796, 2017.

[134] Z. Moti, S. Hashemi, H. Karimipour, A. Dehghantanha, A. N. Jahromi, L. Abdi, F. Alavi, "Generative adversarial network to detect unseen internet of things malware", Ad Hoc Networks 122 (2021): 102591, 2021.

[135] Z. Salehi, A. Sami, M. Ghiasi, "Maar: robust features to detect malicious activity based on api calls, their arguments and return values", Eng. Appl. Artif. Intell. 59, pp.93–102, 2017.

[136] Zhao, Yang, Alifu Kuerban, "MDABP: A Novel Approach to Detect Cross-Architecture IoT Malware Based on PaaS", Sensors 23.6, 2023.