How Machine Learning Is Solving the Binary Function Similarity Problem.

Cách Máy học giải quyết vấn đề Tương tự Hàm nhị phân.

Tóm tắt

* Khả năng tính toán độ tương tự chính xác giữa hai đoạn mã nhị phân đóng một vai trò quan trọng trong nhiều vấn đề khác nhau. Các cộng đồng nghiên cứu như bảo mật, phân tích ngôn ngữ lập trình và học máy đã làm việc trên chủ đề này trong hơn năm năm, với hàng trăm bài báo được xuất bản về chủ đề này. Một người có thể kỳ vọng rằng, đến thời điểm này, sẽ có thể trả lời một số câu hỏi nghiên cứu vượt ra ngoài các kỹ thuật cụ thể rất cụ thể được trình bày trong các bài báo, nhưng có tính tổng quan đối với toàn lĩnh vực nghiên cứu. Rất tiếc, chủ đề này bị ảnh hưởng bởi một số thách thức, từ vấn đề tái sản xuất đến tính mờ của kết quả nghiên cứu, làm trở ngại cho tiến bộ ý nghĩa và hiệu quả.
* Trong bài báo này, chúng tôi bắt đầu bằng cách hệ thống hóa các nghiên cứu hiện có. Sau đó, chúng tôi xác định một số phương pháp liên quan, đại diện cho một loạt các giải pháp được đề xuất gần đây bởi ba cộng đồng nghiên cứu khác nhau. Chúng tôi thực hiện triển khai lại các phương pháp này và tạo ra một tập dữ liệu mới (với các tệp nhị phân được biên dịch với các trình biên dịch khác nhau, cài đặt tối ưu hóa và cho ba kiến trúc khác nhau), cho phép chúng tôi thực hiện một so sánh công bằng và có ý nghĩa. Nỗ lực này cho phép chúng tôi trả lời một số câu hỏi nghiên cứu vượt ra ngoài những gì có thể được suy ra từ việc đọc các bài báo nghiên cứu cá nhân. Bằng việc phát hành toàn bộ khung công cụ mô-đun và các tập dữ liệu của chúng tôi (kèm theo tài liệu liên quan), chúng tôi cũng hy vọng truyền cảm hứng cho công việc tương lai trong lĩnh vực nghiên cứu thú vị này.

1. ***Giới thiệu***

* Tương tự hàm nhị phân là vấn đề đưa ra từ hai biểu diễn nhị phân của hai hàm và tạo ra một giá trị số học biểu thị "tương tự" giữa chúng. Vấn đề này rất khó giải quyết trong trường hợp chung. Trên thực tế, phần mềm thường được biên dịch với các bộ công cụ khác nhau, các tối ưu hóa và cờ biên dịch khác nhau và, trong một số tình huống như các thiết bị IoT, phần mềm được biên dịch cho các kiến trúc khác nhau, làm cho các phương pháp tương tự nhị phân thông thường không hiệu quả.
* Tương tự hàm nhị phân đóng một vai trò quan trọng trong các lĩnh vực nghiên cứu về bảo mật hệ thống khác nhau, vì nhiều vấn đề nghiên cứu đòi hỏi đo lường tương tự hàm như một khối xây dựng cốt lõi. Ví dụ, những người đảo ngược thường xử lý các nhị phân đã bị cắt bỏ có được kết nối tĩnh (do đó không có các ký hiệu), và các phương pháp tương tự mã nhị phân có thể được sử dụng để phù hợp một hàm không xác định với các hàm (đã được đánh nhãn) trong cơ sở dữ liệu được tạo trước đó, tiết kiệm hàng giờ công sức đảo ngược. Tương tự nhị phân cũng quan trọng để phát hiện và vá lỗi trong các thư viện bên thứ ba. Trong trường hợp này, dựa vào một hàm có lỗ hổng, các kỹ thuật tương tự giúp tìm ra các trường hợp của cùng một hàm trong một hoặc nhiều mã nhị phân, cho phép xác định và vá lỗi nhanh chóng hơn. Ngoài ra, vấn đề này cũng liên quan đến việc tạo khác biệt nhị phân và phân tích bản vá, trong đó hai nhị phân với nhiều hàm phải được so sánh với nhau và trong phân tích dòng dẫn phần mềm hoặc gom nhóm phần mềm độc hại, người phân tích quan tâm đến xác định các hàm chung qua nhiều mẫu phần mềm độc hại và nhóm chúng lại dựa trên sự tương tự của chúng.
* Tính quan trọng và liên quan của vấn đề này được phản ánh qua tài liệu có sẵn: các nhà nghiên cứu trong nhiều lĩnh vực khác nhau, bao gồm bảo mật hệ thống, ngôn ngữ lập trình và học máy, đã xuất bản một số lượng đáng kinh ngạc các bài báo (thường là tại các hội nghị uy tín của từng lĩnh vực) để đề xuất các phương pháp mới cho tương tự nhị phân. Cuộc cạnh tranh này đã dẫn đến sự tiến triển rất nhanh chóng của các phương pháp hiện có và sự phát triển và tinh chỉnh tiến bộ của nhiều giải pháp.
* Một người có thể mong đợi rằng thể lượng công việc quan trọng này sẽ đủ để trả lời một số câu hỏi nghiên cứu quan trọng. Ví dụ: Các phương pháp khác nhau so sánh như thế nào khi được đánh giá với cùng một tập dữ liệu và sử dụng các chỉ số đo lường như nhau? Đóng góp chính của các giải pháp máy học mới so với các phương pháp băm mờ đơn giản là gì? Vai trò của các tập hợp đặc trưng khác nhau là gì? Các phương pháp khác nhau hoạt động tốt hơn ở các nhiệm vụ khác nhau? So sánh qua kiến trúc khác nhau khó giải quyết hơn là làm việc trên một kiến trúc duy nhất? Có bất kỳ dòng nghiên cứu cụ thể nào mà có vẻ hứa hẹn là một hướng đi tiềm năng cho việc thiết kế các phương pháp mới? Rất tiếc, chúng tôi thấy rằng tập hợp tài liệu nghiên cứu đã xuất bản hiện tại không thể trả lời những câu hỏi này, do một số thách thức lớn.
* Thách thức đầu tiên là sự không thể tái tạo hoặc sao chép kết quả trước đó. Mặc dù điều này là một vấn đề phổ biến trong lĩnh vực bảo mật, lĩnh vực tương tự nhị phân là một ví dụ tốt về vấn đề này. Chỉ có 12 trong số 61 giải pháp được báo cáo trong cuộc khảo sát của Haq và đồng nghiệp [27] đã phát hành công cụ của họ cho các nhà nghiên cứu khác. Và ngay cả khi các tài liệu có sẵn, chúng thường không chính xác (tức là không thực hiện đúng cùng một giải pháp được mô tả trong bài báo), không hoàn chỉnh (tức là thiếu các thành phần quan trọng, ví dụ như trích xuất đặc trưng) hoặc mã có thể không chạy trên các tập dữ liệu khác so với tập dữ liệu được sử dụng bởi tác giả. Vì việc tái thực hiện các kỹ thuật trước đó rất phức tạp và cực kỳ tốn thời gian, mỗi giải pháp thường chỉ được so sánh với một vài kỹ thuật trước đó, mà đôi khi thậm chí không được thiết kế để giải quyết cùng một vấn đề (ví dụ: tìm kiếm mã vs. sự khác biệt nhị phân), và trong một số trường hợp cực đoan, so sánh chỉ được thực hiện với một bài báo trước đó từ cùng tác giả. Sự thiếu tính tái sản xuất còn quan trọng hơn đối với các phương pháp học máy, nơi các lựa chọn thực hiện, siêu tham số và phương pháp đào tạo và kiểm tra ảnh hưởng mạnh đến kết quả. Thông thường, không rõ liệu các mô hình đã được phát hành có nên sử dụng nguyên vẹn hay cần phải đào tạo lại để tái tạo kết quả tương tự trên các tập dữ liệu khác.
* Thách thức thứ hai là kết quả đánh giá thường mờ nhạt. Các giải pháp khác nhau thường được tùy chỉnh cho các mục tiêu hơi khác nhau (ví dụ: tìm kiếm lỗ hổng vs. tìm các mẫu mã độc tương tự), trong các cài đặt khác nhau (ví dụ: so sánh giữa các trình biên dịch vs. kiến trúc khác nhau), sử dụng khái niệm khác nhau về tương tự (cùng một mã vs. cùng một ý nghĩa), và hoạt động ở các mức độ khác nhau (ví dụ: các đoạn mã nhỏ vs. toàn bộ chức năng). Các thí nghiệm cũng được thực hiện trên các tập dữ liệu có kích thước và bản chất khác nhau (ví dụ: firmware vs. tiện ích dòng lệnh), và kết quả được báo cáo bằng cách sử dụng các chỉ số đo lường khác nhau (ví dụ: đường cong ROC vs. top-n vs. MRR10). Do đó, ngay cả các con số cơ bản được báo cáo trong mỗi bài báo cũng không thể so sánh trực tiếp. Do đó, khi kết quả vượt qua các công trình trước đó, không rõ liệu điều này chỉ xảy ra trong kịch bản được chọn hay cũng xảy ra trong các trường hợp sử dụng khác. Để làm cho mọi thứ trở nên tồi tệ hơn, các bài báo thường lờ đi chi tiết về cách các chức năng được lọc và cách các cặp tích cực và tiêu cực được chọn cho việc đào tạo, làm cho việc tái tạo luồng công việc nguyên vẹn ngay cả với cùng một tập mã trở nên khó khăn.
* Lưu ý rằng những công trình này thường được xây dựng trên các luồng công việc không đơn giản, ví dụ như các bộ công cụ để xác định ranh giới chức năng, giải mã mã nguồn và trích xuất đồ thị luồng điều khiển. Các phương pháp có sẵn rất ít sử dụng các bộ công cụ khác nhau và chúng được xây dựng trên các luồng công việc khác nhau. Do đó, việc xác định mức độ đáng tin cậy của các giai đoạn "không liên quan" ban đầu trong luồng công việc thực sự ảnh hưởng đến độ tin cậy của phương pháp tổng thể là một thách thức. Nói cách khác, thường không rõ liệu kết quả ưu việt của một phương pháp cụ thể có liên quan đến những đóng góp được giới thiệu như là mới mẻ hay có liên quan đến các yếu tố khác.
* Tác động kết hợp của hai thách thức đầu tiên đã tạo ra một lĩnh vực vô cùng phân mảnh, nơi tồn tại hàng chục kỹ thuật nhưng không có một sự hiểu rõ ràng về những gì hoạt động (hoặc không hoạt động) trong các tình huống cụ thể. Điều này dẫn đến khó khăn trong việc hiểu hướng đi của nghiên cứu về tương tự nhị phân và lý do tại sao. Mỗi giải pháp mới đưa ra một kỹ thuật phức tạp hơn hoặc một kết hợp mới của nhiều kỹ thuật, và khó để xác định liệu điều này được thúc đẩy bởi các giới hạn thực sự của các phương pháp đơn giản hơn hay do nhu cầu thuyết phục các nhà đánh giá về tính mới mẻ của từng công trình. Sự phân mảnh này thường dẫn đến những dòng nghiên cứu song song và riêng biệt, trong đó mỗi người đều khẳng định có giải pháp tốt nhất. Sự phân mảnh này cũng dẫn đến các bài báo với các đánh giá và phương pháp không tối ưu. Ví dụ, các bài báo mạnh về khía cạnh phân tích chương trình có thể thiếu ứng dụng của các kỹ thuật học máy tiên tiến. Các giải pháp dựa trên học máy là xu hướng hiện tại, nhưng chúng thường mù quáng áp dụng các kỹ thuật từ các lĩnh vực khác, làm cho việc đánh giá tổng thể tiến bộ và đổi mới trong lĩnh vực trở nên khó khăn hơn.
* Các đóng góp của chúng tôi trong bài báo này là thực hiện một nghiên cứu đo lường hệ thống đầu tiên trong lĩnh vực nghiên cứu này. Đầu tiên, chúng tôi khám phá nghiên cứu hiện có và nhóm mỗi giải pháp dựa trên phương pháp đã áp dụng, tập trung đặc biệt vào các kỹ thuật thành công gần đây dựa trên học máy. Sau đó, chúng tôi chọn, so sánh và thực hiện mười phương pháp đại diện và các biến thể có thể có của chúng. Các phương pháp này đại diện cho một loạt xu hướng và bao gồm ba cộng đồng nghiên cứu khác nhau: cộng đồng bảo mật máy tính, phân tích ngôn ngữ lập trình và cộng đồng học máy. Để làm cho so sánh của chúng tôi có ý nghĩa, các bản thực hiện của chúng tôi được xây dựng dựa trên một khung làm việc chung (ví dụ: chúng tôi trích xuất các đặc trưng thô bằng cách sử dụng cùng một bộ công cụ cơ bản, trong khi các công trình trước đó dựa vào các bộ công cụ khác nhau). Nếu bản triển khai ban đầu có sẵn, chúng tôi bao gồm bản triển khai mô hình cốt lõi trong một mã nguồn chung để huấn luyện và kiểm tra và mở rộng hỗ trợ cho các kiến trúc và độ rộng bit bị thiếu. Cuối cùng, chúng tôi tận dụng lập trình song song và kỹ thuật mã hóa dữ liệu hiệu quả để tránh những chỗ chai có thể ảnh hưởng tiêu cực đến hiệu suất mô hình.
* Bằng cách thực hiện lại các phương pháp khác nhau - và không nhất thiết là "các bài báo" - chúng tôi cô lập các "nguyên thủy" hiện có và đánh giá chúng khi sử dụng độc lập hoặc kết hợp với nhau, để có cái nhìn và chỉ ra các yếu tố quan trọng bị ẩn trong sự phức tạp của các công trình trước đó, và để trả lời các câu hỏi nghiên cứu mở. Để làm cho công sức đánh giá này có thể so sánh hơn, chúng tôi cũng đề xuất một tập dữ liệu mới mà chúng tôi sử dụng như một bài đo chung với các khía cạnh thay đổi như họ tạo, tối ưu hóa và kiến trúc.
* Chúng tôi lưu ý rằng nghiên cứu của chúng tôi tập trung vào đánh giá các kỹ thuật chính đã được đề xuất cho đến nay mà không cố gắng tái tạo kết quả chính xác được báo cáo trong các bài báo tương ứng. Trong khi một số trong số các bản triển khai của chúng tôi được dẫn xuất từ mã được phát hành bởi các tác giả khi có sẵn, các bản triển khai khác đã được phát triển từ đầu với mục tiêu có một mã nguồn và luồng công việc duy nhất có thể cô lập kỹ thuật khỏi các yếu tố còn lại có thể ảnh hưởng đến kết quả.
* Việc đánh giá của chúng tôi nhấn mạnh một số nhận thức thú vị. Ví dụ, chúng tôi phát hiện ra rằng trong khi các phương pháp đơn giản (ví dụ: fuzzy hashing) hoạt động tốt cho các cài đặt đơn giản, chúng thất bại khi xử lý các kịch bản phức tạp hơn (chẳng hạn như tập dữ liệu chéo kiến trúc hoặc tập dữ liệu mà nhiều biến số thay đổi cùng một lúc). Trong số các mô hình học máy, các mô hình dựa trên Graph Neural Network đạt được kết quả tốt nhất trong gần như tất cả các nhiệm vụ và nhanh nhất khi so sánh với thời gian suy luận. Một phát hiện thú vị khác là nhiều bài báo gần đây đều có độ chính xác rất giống nhau khi được thử nghiệm trên cùng một tập dữ liệu, mặc dù có một số tuyên bố cải tiến so với tình trạng nghệ thuật.
* Mặc dù chúng tôi không khẳng định rằng mã hoặc tập dữ liệu của chúng tôi tốt hơn hoặc đại diện hơn các công trình trước đó, chúng tôi công bố toàn bộ khung modul của chúng tôi, bản triển khai lại của tất cả các phương pháp đã chọn, tập dữ liệu đầy đủ và hướng dẫn chi tiết về cách tạo lại và điều chỉnh nó.1 Bằng cách cho phép cộng đồng thử nghiệm với các thành phần riêng lẻ và so sánh trực tiếp chúng với nhau, chúng tôi hy vọng khuyến khích và giúp đỡ công việc của các nhà nghiên cứu tương lai quan tâm đến tiếp cận lĩnh vực nghiên cứu tích cực này.

1. ***Vấn đề Tương đồng Hàm Nhị phân***

* Trong dạng đơn giản nhất, tương đồng hàm nhị phân nhằm vào việc tính toán giá trị số học thể hiện "tương đồng" giữa một cặp hàm trong biểu diễn nhị phân của chúng, tức là chuỗi byte (tức là mã máy) tạo thành nội dung của hàm, được tạo ra bởi trình biên dịch. Lưu ý rằng, trong bài báo này, chúng tôi tập trung vào các phương pháp sử dụng hàm làm đơn vị mã lệnh, tuy nhiên các nhà nghiên cứu cũng đã nghiên cứu các kỹ thuật tập trung vào các trừu tượng cấp thấp hơn (ví dụ: basic blocks) hoặc cấp cao hơn (ví dụ: chương trình hoàn chỉnh). Thuật ngữ "tương đồng" được hiểu theo nhiều cách, tùy thuộc vào ngữ cảnh. Trong bài báo này, chúng tôi coi hai hàm là "tương đồng" nếu chúng được biên dịch từ cùng mã nguồn, độc lập với trình biên dịch, phiên bản của nó, các cờ biên dịch của nó, hoặc thậm chí kiến trúc mà hàm đã được biên dịch thành (ví dụ: x86, ARM). Do đó, theo định nghĩa của chúng tôi, hai hàm "tương đồng" có thể có biểu diễn nhị phân khác nhau đáng kể - và đây là điều làm cho vấn đề nghiên cứu này thú vị và thách thức.
* Tương đồng hàm nhị phân đã được nghiên cứu trong hơn một trăm bài báo. Để phức tạp hơn, hầu hết các phương pháp hiện có không thể được phân loại thành một loại kỹ thuật duy nhất, vì chúng thường được xây dựng dựa trên các thành phần khác nhau. Do đó, trong phần này, chúng tôi tập trung vào các khối xây dựng khác nhau mà các phương pháp này được cấu thành, bằng cách xem xét trước các kỹ thuật để tính toán tương đồng, sau đó đối diện các loại dữ liệu đầu vào mà các phương pháp này có thể sử dụng.

***2.1 Đo lường Tương đồng Hàm***

* So sánh trực tiếp và gián tiếp. Chúng ta có thể nhóm các kỹ thuật để đo lường tương đồng hàm vào hai nhóm chính. Lớp giải pháp đầu tiên thực hiện một so sánh trực tiếp của các cặp hàm, hoặc bằng cách xem xét dữ liệu đầu vào nguyên gốc hoặc bằng cách thực hiện trích xuất đặc trưng. Những giải pháp này thường cần học rằng hai giá trị có vẻ không liên quan có thể biểu thị các hàm tương đồng, hoặc ngược lại, những giá trị gần nhau không nhất thiết phải biểu thị một điều tương tự. Điều này xảy ra khi các đặc trưng được trích xuất từ hàm nhị phân không thể được so sánh trực tiếp bằng cách sử dụng các phép đo tương đồng cơ bản vì chúng có thể không được đại diện trong một không gian tuyến tính hoặc không có trọng số tương đương trên điểm số tương đồng. Do đó, các nhà nghiên cứu đã đề xuất sử dụng các mô hình học máy để xác định xem hai hàm có tương đồng nhau dựa trên một tập hợp đặc trưng đã trích xuất làm đầu vào. Có nhiều phương pháp tiếp cận này bằng cách sử dụng các mạng Bayesian [2], mạng neural tích chập [44], Mạng phù hợp đồ thị (GMN) [40], mạng neural feed-forward thông thường [67], hoặc sự kết hợp của chúng [37]. Trong các trường hợp này, mô hình được sử dụng để đưa ra điểm số tương đồng giữa một cặp hàm.
* Để tìm các hàm tương đồng, những phương pháp này cần tìm kiếm trong toàn bộ tập dữ liệu và so sánh các đặc trưng của hàm được truy vấn với mỗi mục nhập trong tập dữ liệu, điều này không phải là một giải pháp có khả năng mở rộng. Vì lý do này, nhiều phương pháp thực hiện các chiến lược chỉ số hóa để tiền lọc các ứng viên tương đồng tiềm năng với các kỹ thuật như cấu trúc dữ liệu dựa trên cây [55, 68], lược đồ tăng cường cục bộ [15, 22, 32, 56, 61] (tìm kiếm hàng xóm gần nhất gần đúng), bộ lọc bloom [35], bộ lọc trước tùy chỉnh dựa trên dữ liệu số đơn giản hơn [6, 20], kỹ thuật phân cụm [81], hoặc thậm chí các phương pháp tìm kiếm phân tán như map-reduce [15].
* Lớp thứ hai của các giải pháp thực hiện các kỹ thuật so sánh gián tiếp. Những phương pháp này ánh xạ các đặc trưng đầu vào vào một biểu diễn “rút gọn” chiều thấp có thể được so sánh dễ dàng với nhau bằng cách sử dụng một phép đo khoảng cách như khoảng cách Euclidean hoặc cosin. Những giải pháp này cho phép so sánh một-nhiều hiệu quả. Ví dụ, nếu một hàm mới cần được so sánh với toàn bộ tập dữ liệu, ta có thể đầu tiên ánh xạ mỗi hàm trong kho lưu trữ vào biểu diễn chiều thấp tương ứng của nó (đây là một hoạt động một lần), sau đó thực hiện cùng một hoạt động trên hàm mới và cuối cùng so sánh những biểu diễn này bằng cách sử dụng các kỹ thuật hiệu quả như gần nhất hàng xóm gần đúng.
* Fuzzy hashes và embeddings. Một ví dụ phổ biến về biểu diễn chiều thấp là một fuzzy hash. Fuzzy hash được tạo ra bởi các thuật toán khác với các hash mật mã truyền thống vì chúng được thiết kế cố ý để ánh xạ các giá trị đầu vào tương tự thành các hash tương tự. Pagani et al. [58] đã nghiên cứu về các giới hạn của các fuzzy/locality sensitive hash truyền thống được tính toán trên các thực thi thô, kết luận rằng các biến đổi nhỏ trong các byte thô của đầu vào có thể ảnh hưởng đáng kể đến các hash được tạo ra. Tuy nhiên, ngay cả khi fuzzy hash thông thường có thể không phù hợp cho tương đồng hàm, một số phương pháp (như FunctionSimSearch [18]) đã đề xuất các kỹ thuật băm chuyên dụng hơn để so sánh hai hàm.
* Một hình thức phổ biến khác của biểu diễn chiều thấp dựa trên embeddings. Thuật ngữ này phổ biến trong cộng đồng machine-learning, đề cập đến không gian chiều thấp trong đó các đầu vào tương tự về mặt ngữ nghĩa được ánh xạ thành các điểm gần nhau, bất kể những khác biệt về biểu diễn ban đầu của các đầu vào đó như thế nào. Mục tiêu của các mô hình machine-learning là học cách tạo ra các embeddings tối đa hóa sự tương đồng giữa các hàm tương tự và giảm thiểu nó cho các hàm khác biệt. Trong tài liệu, chúng ta có thể xác định hai loại chính của embeddings: những embeddings cố gắng tóm tắt mã của mỗi hàm và những embeddings cố gắng tóm tắt cấu trúc đồ thị của chúng.
* Nhúng mã (Code embeddings). Nhiều nhà nghiên cứu đã thử áp dụng các kỹ thuật xử lý Ngôn ngữ Tự nhiên (NLP) hiện có để giải quyết vấn đề tương đồng hàm nhị phân bằng cách xem mã assembly như là văn bản. Các phương pháp này xử lý các chuỗi token (ví dụ: lệnh, mnemonic, operand, lệnh được chuẩn hóa) và xuất ra một embedding cho mỗi khối mã, một embedding cho mỗi lệnh hoặc cả hai. Một nhóm đầu tiên của các phương pháp (ví dụ: Asm2Vec [14] và [64]) dựa trên word2vec [52, 53], một kỹ thuật nổi tiếng trong lĩnh vực NLP. Mặc dù các mô hình này không được thiết kế để tạo ra nhúng chéo kiến trúc, chúng có thể được huấn luyện trên các bộ chỉ thị khác nhau cùng một lúc, học cú pháp của các ngôn ngữ khác nhau (nhưng không thể ánh xạ ngữ nghĩa qua các ngôn ngữ) hoặc có thể được áp dụng trên một ngôn ngữ trung gian. Dòng thứ hai của các giải pháp dựa trên các mô hình mã hóa giải mã seq2seq [69], cho phép ánh xạ ngữ nghĩa từ các kiến trúc khác nhau vào cùng một không gian nhúng, từ đó học tương đồng chéo kiến trúc [49, 80, 82]. Một loại mô hình thứ ba được xây dựng trên nền tảng BERT [12], mô hình tiền huấn luyện hàng đầu trong NLP dựa trên transformer [71]. Ví dụ, OrderMatters [78] sử dụng mô hình BERT được tiền huấn luyện trên bốn tác vụ để tạo ra nhúng basic block, trong khi Trex [60] sử dụng transformer phân cấp và tác vụ Masked-Language-Modeling để học ngữ nghĩa thực thi chương trình xấp xỉ và sau đó chuyển giao kiến thức đã học để xác định các hàm có ngữ nghĩa tương tự.
* Nhúng mã assembly thường bị ảnh hưởng bởi số lượng các lệnh khác nhau mà chúng có thể xử lý (vấn đề out-of-vocabulary - OOV), và số lượng lệnh tối đa có thể được cung cấp như đầu vào cho mô hình. Kết quả là, một số phương pháp tính toán các nhúng cấp lệnh [14, 16, 64], nhúng basic block [16, 78, 80, 82], hoặc nhúng hàm [14, 49, 60]. Nhúng cấp lệnh hoặc basic block đôi khi được sử dụng để tính toán tương đồng hàm bằng cách sử dụng các thuật toán khác như Longest Common Subsequence [82], hoặc chúng được sử dụng như một phần của các mô hình phức tạp hơn như được trình bày trong phần tiếp theo.
* Nhúng đồ thị (Graph embeddings). Một hướng nghiên cứu khác dựa trên các phương pháp học máy tính toán nhúng cho các đồ thị. Những phương pháp này rất phù hợp để nắm bắt các đặc trưng dựa trên đồ thị kiểm soát chương trình, tự nhiên là chéo kiến trúc. Những nhúng này có thể được tạo ra bằng các thuật toán tùy chỉnh [24, 44] hoặc bằng các kỹ thuật học máy phức tạp hơn, chẳng hạn như Graph Neural Network (GNN) [25, 40, 45, 76, 78, 79]. Một số phương pháp gần đây từ cộng đồng học máy đề xuất các biến thể của GNN, chẳng hạn như GMN. Những biến thể này có thể tạo ra các nhúng so sánh trong không gian vector [40, 43], với đặc tính là những nhúng này mã hóa thông tin từ hai đồ thị được cung cấp làm đầu vào cho mô hình.
* Phương pháp nhúng đồ thị cũng thường mã hóa thông tin từ mỗi basic block (khối cơ bản) vào nút tương ứng của đồ thị để thêm tính mềm dẻo hơn. Ví dụ, một số giải pháp tính toán một tập hợp các thuộc tính cho mỗi nút, dẫn đến các đồ thị kiểm soát chương trình được gọi là Attributed Control-Flow Graphs (ACFG), có thể được thiết kế thủ công [24, 76] hoặc được học tự động một cách không giám sát [45]. Các tác giả khác sử dụng các lớp tính toán nhúng khác bằng cách sử dụng một số kỹ thuật đã được thảo luận ở trên (ví dụ, ở cấp độ basic block) [45, 78, 79].

***2.2 Biểu diễn chức năng***

* Các hàm nhị phân chính là các luồng byte tương ứng với mã máy và dữ liệu cụ thể cho từng kiến trúc. Bắt đầu từ dữ liệu đầu vào nguyên thô này, các nhà nghiên cứu đã sử dụng nhiều cách để trích xuất thông tin cấp cao có thể được sử dụng để xác định liệu hai hàm có xuất phát từ cùng một nguồn mã nguồn hay không. Danh sách này, được sắp xếp theo mức trừu tượng tăng dần, bao gồm các thông tin sau đây.
* Dữ liệu nguyên thô. Một số giải pháp sử dụng trực tiếp thông tin nhị phân nguyên thô làm điểm khởi đầu cho phương pháp đo đo sự tương đồng (ví dụ, Catalog1 [74]) hoặc kết hợp byte nguyên thô với các thông tin khác như control-flow graph (CFG) hoặc call graph (CG) [44].
* Hợp ngữ. Các chỉ thị hợp ngữ, được thu thập bằng cách sử dụng disassembler, có thể hữu ích khi các hoạt động có thể được mã hóa bằng nhiều cách khác nhau tùy thuộc vào kích thước chỉ thị hoặc toán hạng của nó (ví dụ, trong kiến trúc x86/64, một chỉ thị mov có thể được mã hóa bằng cách sử dụng một số byte opcode khác nhau [33]). Các phương pháp như Asm2Vec [14] và Trex [60] có lợi từ mức trừu tượng này bằng cách sử dụng các chỉ thị được disassemble làm đầu vào, trong khi các phương pháp khác tính toán các số liệu khác như "số lượng chỉ thị hợp ngữ số trong một hàm cụ thể" [24, 25, 76].
* Hợp ngữ đã chuẩn hóa. Hợp ngữ thường mã hóa các giá trị hằng số (ví dụ: toán hạng tức thì và địa chỉ tuyệt đối hoặc tương đối), dẫn đến một số lượng rất lớn các kết hợp tiềm năng của các hoạt động và toán hạng. Chuẩn hóa hợp ngữ được sử dụng trong [22, 45, 49, 64, 80, 82] để trừu tượng một phần sự biến đổi này, giảm kích thước từ vựng và hội tụ tất cả các biến thể có thể của cùng một hoạt động thành một biểu diễn duy nhất.
* Biểu diễn trung gian. Một số phương pháp làm việc trên một mức trừu tượng cao hơn bằng cách chuyển từ biểu diễn nhị phân thành biểu diễn trung gian (IR). Sử dụng IR mang lại một số lợi ích: (i) nó có thể thống nhất biểu diễn của các chỉ thị có cùng nghĩa nhưng khác nhau cú pháp, (ii) nó có thể trừu tượng hóa các yếu tố không liên quan của các kiến trúc khác nhau và (iii) nó cho phép áp dụng các kỹ thuật phân tích chương trình để đơn giản hóa (và hội tụ) một số kiến trúc mã. Các công trình hiện có đã sử dụng một số IR khác nhau, chẳng hạn như LLVM [10, 23, 25], VEX [6, 10, 30, 67] và IDA microcode [78, 79].
* Cấu trúc. Nhiều phương pháp cố gắng bắt lấy cấu trúc bên trong của một hàm cụ thể hoặc vai trò mà một hàm đóng trong chương trình tổng thể. Để bắt lấy cấu trúc bên trong của một hàm, nhiều phương pháp [3, 18, 32, 56] trích xuất (nội-sư-ly) Control-Flow Graph (CFG). Một số phương pháp làm giàu CFG với dữ liệu thu được từ các khối cơ bản, tức là Attributed Control-Flow Graph (ACFG) [18, 20, 24, 25, 40, 51, 76, 78, 79, 81], hoặc các loại đồ thị hoặc thông tin khác thu được từ hàm (ví dụ: đồ thị dòng đăng ký [1]) hoặc ngữ cảnh của nó trong nhị phân (đồ thị gọi [44, 68]). Cuối cùng, một số phương pháp chỉ tận dụng cấu trúc được cung cấp bởi CFG để tính toán các tính năng thay thế - chẳng hạn như tracelets (chuỗi các khối cơ bản liên tiếp trong CFG [11, 56]).
* Phân tích luồng dữ liệu. Cài đặt một biểu thức số học ở mức hợp ngữ có thể sử dụng các biểu thức khác nhau để thực hiện cùng một ngữ nghĩa. Để giải quyết các tình huống này, các công trình trước đó đã đề xuất trước tiên tính toán các chương trình slices dựa trên các phụ thuộc dữ liệu, sau đó chuẩn hóa và sử dụng chúng như các tính năng để bắt lấy hành vi của một hàm [9, 67]. Các tài liệu khác, chẳng hạn như Vulseeker [25], sử dụng các cạnh luồng dữ liệu giữa các khối như một tính năng bổ sung.
* Phân tích động. Một số phương pháp dựa vào phân tích động [19], ví dụ như bằng cách thực hiện các cặp hàm và trích xuất các tính năng từ mối quan hệ giữa các đầu vào và đầu ra [30, 62]. Các phương pháp khác đơn giản là trích xuất các tính năng ngữ nghĩa dựa trên các trạng thái thực thi hoàn toàn hoặc một phần của chương trình [29, 34, 39, 54, 72], trong khi các phương pháp khác sử dụng kỹ thuật giả lập [77] hoặc hỗn hợp [31, 60].
* Thực hiện và phân tích biểu tượng. Ngược lại với việc thực hiện động cụ thể, một số phương pháp dựa vào việc thực hiện biểu tượng để bắt toàn bộ hành vi của hàm đang được phân tích và xác định mối quan hệ giữa các đầu vào và đầu ra của nó trong tất cả các đường dẫn có thể [6, 46, 55].

1. ***Tiếp cận được chọn lọc***

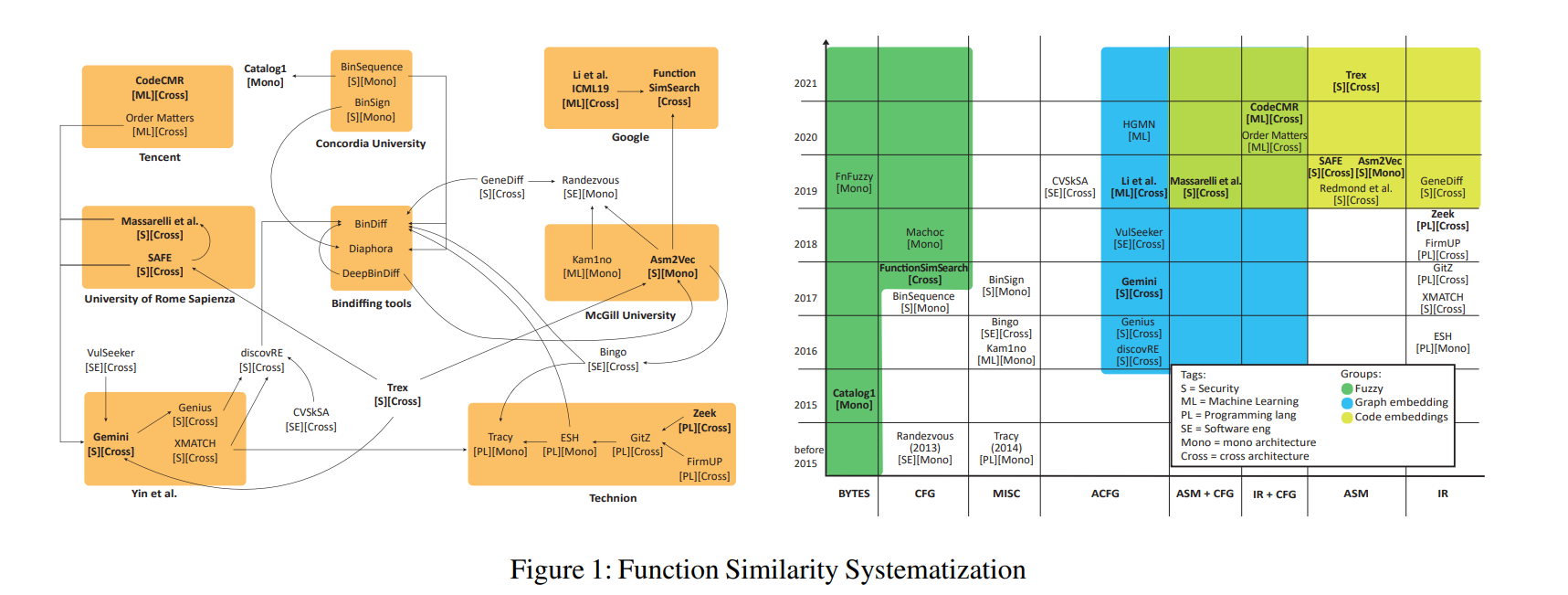
* Một trong những đóng góp chính của công việc của chúng tôi là cung cấp một thực hiện tham chiếu cho một số phương pháp chính và so sánh chúng bằng cách thực hiện các thí nghiệm trên một tập dữ liệu chung và toàn diện. Lý tưởng nhất là đánh giá càng nhiều phương pháp càng tốt, nhưng rõ ràng là không khả thi để cài đặt lại tất cả chúng. Ngoài ra, quan trọng là hiểu rằng, mặc dù có hàng trăm bài báo về chủ đề này, nhiều trong số chúng trình bày những biến thể nhỏ của cùng một kỹ thuật và số lượng các giải pháp mới là đáng kể thấp.
* Trong phần này, chúng tôi trước tiên thảo luận về tiêu chí chọn lọc của chúng tôi, sau đó giới thiệu tập các phương pháp mà chúng tôi đã cài đặt và đánh giá.

***3.1 Tiêu chí chọn lọc***

* Chúng tôi sử dụng các tiêu chí sau để chọn lọc các phương pháp thích hợp cho đánh giá của chúng tôi:
* ***Khả năng mở rộng và ứng dụng thực tiễn:*** Chúng tôi quan tâm đến các phương pháp có tiềm năng mở rộng cho các tập dữ liệu lớn và có thể áp dụng cho các tình huống thực tế. Vì vậy, chúng tôi không đánh giá các phương pháp chậm và chỉ tập trung vào các phương pháp trực tiếp so sánh, chẳng hạn như các phương pháp dựa trên phân tích động, thực thi biểu tượng hoặc các thuật toán liên quan đến đồ thị có độ phức tạp cao.
* ***Tập trung vào các phương pháp đại diện, không tập trung vào các bài báo cụ thể***: Có nhiều công trình nghiên cứu đề xuất chỉ là các biến thể nhỏ của cùng một phương pháp - ví dụ như tái sử dụng các kỹ thuật trước đó trong khi thay đổi một chút các đặc trưng được sử dụng. Điều này thường dẫn đến độ chính xác tổng thể tương tự, làm cho chúng ít thú vị cho cuộc so sánh của chúng tôi.
* ***Bao gồm các cộng đồng nghiên cứu khác nhau:*** Các đóng góp nghiên cứu về vấn đề tương tự hàm nhị phân đến từ các cộng đồng nghiên cứu khác nhau và cả từ cả học thuật và công nghiệp. Tuy nhiên, thường xuyên xảy ra trường hợp các bài báo nghiên cứu trong cùng một cộng đồng chỉ được đánh giá so sánh với các đề xuất từ cùng một cộng đồng hoặc, đôi khi, chỉ so sánh với các công trình trước đó của cùng tác giả. Do đó, trong đánh giá của chúng tôi, chúng tôi muốn bao gồm các nghiên cứu đại diện từ các cộng đồng nghiên cứu về hệ thống an ninh, phân tích ngôn ngữ lập trình và học máy.
* ***Ưu tiên các xu hướng mới nhất:*** Mặc dù các đóng góp đầu tiên trong lĩnh vực nghiên cứu này đã xuất hiện từ hơn một thập kỷ trước, gần đây đã có một sự bùng nổ quan tâm. Hơn nữa, hầu hết các công trình gần đây này sử dụng một cách nào đó các kỹ thuật dựa trên học máy. Các kỹ thuật này, trong một cách nào đó, đã được báo cáo vượt qua tất cả các phương pháp trước đây. Một số nhà nghiên cứu đã đề xuất rằng các phương pháp cơ bản hoạt động cũng tốt như các kỹ thuật học máy, nhưng đánh giá của chúng tôi cho thấy rằng điều này chỉ xảy ra khi xem xét các tình huống đánh giá đơn giản. Do đó, trong khi chúng tôi xem xét các loại phương pháp khác nhau, chúng tôi ưu tiên những xu hướng nghiên cứu mới hơn, hứa hẹn hơn cả.

***3.2 Các Phương Pháp Được Lựa Chọn***

* Trong phần 2, chúng tôi đã trình bày các loại dữ liệu đầu vào mà các nhà nghiên cứu đã trích xuất qua nhiều năm cũng như các phương pháp khả thi để tính đo độ tương đồng giữa các hàm. Tuy nhiên, chỉ có một phần nhỏ trong số hàng trăm bài báo được xuất bản trong thập kỷ qua đáp ứng được các tiêu chí được mô tả trên. Dựa trên phân tích của chúng tôi, chúng tôi đã xác định 30 kỹ thuật, được biểu thị trong Hình 1, và sau đó chọn ra 10 giải pháp đại diện để nghiên cứu.
* Biểu đồ bên trái của Hình 1 hiển thị các phương pháp được phân cụm theo nhóm nghiên cứu tương ứng. Các nhóm này đến từ cả học thuật và công nghiệp, trong đó cả Google và Tencent đều rất tích cực trong lĩnh vực này. Các đoạn mũi tên biểu thị các giải pháp so sánh kết quả của mỗi bài báo với các kết quả trước đó. Ví dụ, mũi tên giữa Gemini và Genius nghĩa là các kết quả của Gemini đã được so sánh bởi các tác giả với kết quả đã thu được bởi Genius (cả hai đều từ cùng một nhóm nghiên cứu). Phần bên phải của Hình 1 thể hiện dòng thời gian xuất bản trên trục Y và các loại dữ liệu đầu vào khác nhau trên trục X. Các phương pháp sau đó được phân cụm vào ba nhóm chính dựa trên các cách khác nhau để tính đo độ tương đồng, bao gồm fuzzy hashes, graph embeddings và code embeddings.
* Cả hai hình đều sử dụng các thẻ (trong ngoặc vuông) để xác định cộng đồng nghiên cứu ( [S] bảo mật, [PL] ngôn ngữ lập trình, [ML] học máy và [SE] kỹ thuật phần mềm). Chúng tôi cũng sử dụng các thẻ [Mono] và [Cross] để xác định xem các phương pháp đề xuất tập trung vào các kịch bản đơn lẻ hay đa kiến trúc.



* Dù biểu đồ trong Hình 1 không phải là toàn diện và chỉ hiển thị các bài báo chúng tôi đã chọn, nó một lần nữa cho thấy cách mà một số bài báo so sánh chỉ với một tập hạn chế các phương pháp trước đó. Có những thông điệp thú vị khác mà chúng ta có thể rút ra từ các đồ thị này. Đầu tiên, các công cụ binary diffing được nhóm lại trong hộp giữa [13, 16, 83] đều được thiết kế cho so sánh trực tiếp hai binary (ví dụ, chúng sử dụng call graph) và tất cả đều tập trung vào kiến trúc đơn lẻ. Tuy nhiên, một số bài báo đề xuất các giải pháp đa kiến trúc và tương đồng hàm so sánh kết quả của họ với các công cụ này. Điều này rõ ràng là một vấn đề có thể dẫn đến những kết luận sai lầm và cho thấy một số sai sót trong thực nghiệm và chiến lược đánh giá không phù hợp.
* Thứ hai, đồ thị cho thấy các cộng đồng nghiên cứu khác nhau thường khá kín kẽ và họ hiếm khi so sánh với các bài báo từ lĩnh vực khác. Điều này là một hạn chế rõ ràng đối với việc đẩy mạnh nghiên cứu về độ tương đồng hàm và chúng tôi hy vọng bài báo này có thể khuyến khích sự hợp tác giữa các lĩnh vực khác nhau. Cuối cùng, chúng ta có thể nhận ra các bài báo có tính bước ngoặt như Gemini [76] và discovRE [20] đã được tái triển khai và kiểm tra kỹ lưỡng trong các nghiên cứu khác. Các công trình này đã rõ ràng truyền cảm hứng cho các nhà nghiên cứu khác để cải tiến trạng thái hiện tại.
* Hình ảnh dòng thời gian bên phải cho thấy một xu hướng rõ ràng: độ phức tạp của các giải pháp và việc sử dụng học máy đã tăng theo thời gian. Chúng tôi đã sử dụng thông tin này và các mối quan hệ được miêu tả trong hình ảnh để chọn ra mười giải pháp hàng đầu hiện tại, có tính mẫu mực, và gần đây. Đồng thời, chúng tôi đã cố gắng tối đa hóa sự biến động giữa các cộng đồng nghiên cứu.
* Ví dụ, chúng tôi đã chọn Gemini [76] nhưng không phải là Genius và discovRE vì Gemini đã vượt trội hơn Genius [24] trong tất cả các thử nghiệm của nó và Genius lại vượt trội hơn discovRE [20]. Do đó, chúng tôi tin tưởng rằng Gemini cũng vượt trội hơn discovRE. Dựa trên các lựa chọn tương tự, chúng tôi đã chọn Asm2Vec [14], một giải pháp đã cho thấy độ chính xác tốt hơn so với Bingo [6]. Chúng tôi cũng giữ lại một số công trình mà khi nhìn vào đầu tiên có thể có vẻ đã lỗi thời (ví dụ, Catalog1 [74]) vì chúng tôi tin rằng chúng đại diện cho việc nghiên cứu sự phát triển của lĩnh vực này, có kết quả thú vị và phản ánh tiêu chí lựa chọn của chúng tôi. Cuối cùng, trong một số trường hợp, chúng tôi có thể đã chọn nhiều hơn một công trình trong một danh mục cụ thể. Điều này có thể xảy ra khi có những công trình đồng thời và không rõ từ văn học là công trình nào có hiệu suất tốt hơn. Trong phần còn lại của phần này, chúng tôi sẽ mô tả ngắn gọn mười giải pháp được chọn.
* Bytes fuzzy hashing: Catalog1. Catalog1 [74] là một phương pháp fuzzy hashing dựa trên MinHash Locality Sensitive Hashing [4]. Thuật toán nhận dữ liệu đầu vào là các byte của hàm và tạo ra một chữ ký có độ dài cố định, đây là một cách triển vọng để so sánh các hàm từ cùng một kiến trúc.
* CFG fuzzy hashing: FunctionSimSearch. FunctionSimSearch [18] sử dụng thuật toán SimHash [7] để tính toán một fuzzy hash kết hợp các đồ thị con (graphlets) được trích xuất từ CFG, các từ chỉ thị (mnemonics) và các giá trị tức thì (immediate values) từ mã assembly. Phương pháp này có khả năng xuyên kiến trúc do tính năng dựa trên CFG. Công cụ này được phát triển bởi một nhà nghiên cứu trong ngành công nghiệp.
* Attributed CFG và GNN: Gemini. Gemini [76] sử dụng mạng neural đồ thị (GNN) (Structure2vec [8]) để tính toán embedding của hàm bắt đầu từ ACFG (control-flow graph với các thuộc tính cấp basic-block). Phương pháp này đánh dấu một bước tiến vượt trội, vì nó là một phương pháp đầu tiên sử dụng GNN với kiến trúc Siamese [5] để học sự tương đồng hàm. Đây rõ ràng là một sự tiến hóa so với các giải pháp dựa trên ACFG cơ bản (Genius [24]) và nó cũng hiệu quả hơn các phương pháp khác sử dụng dữ liệu CFG như Bingo [6], Binsign [56], Kam1no [15], hoặc Tracy [11].
* Attributed CFG, GNN và GMN: Li et al. 2019. Phương pháp được giới thiệu trong [40] được đề xuất bởi các nhà nghiên cứu trong ngành công nghiệp (DeepMind và Google) từ cộng đồng học máy và nó giới thiệu một mô hình ghép đồ thị mới để tính toán sự tương đồng giữa các cặp đồ thị. Tác giả đã khám phá tương đồng hàm như một trong những trường hợp sử dụng thực tế. Phương pháp này đề xuất hai mô hình tiên tiến từ cộng đồng học máy mà trước đây chưa được nghiên cứu bởi các nhà nghiên cứu về bảo mật hệ thống. Hơn nữa, bài báo cũng cho thấy kết quả triển vọng.
* IR, phân tích data flow và mạng neural: Zeek. Zeek [67] thực hiện phân tích data flow (slicing) trên mã lệnh được đưa lên (VEX IR) ở cấp basic-block và tính toán strands. Sau đó, một mạng neural toàn bộ hai lớp fully-connected được huấn luyện để học nhiệm vụ tương đồng xuyên kiến trúc. Phương pháp này là đề xuất tiên tiến nhất kết hợp các biểu diễn trung gian, phân tích data flow và học máy. Công trình này vượt trội so với các nghiên cứu trước đó từ cùng tác giả.
* Nhúng mã assembly: Asm2Vec. Mô hình Asm2Vec [14] xuất phát từ biến thể PV-DM của paragraph2vec [38], là một sự mở rộng của mô hình gốc word2vec [53]. Asm2Vec giới thiệu việc chia nhỏ chi tiết hơn cấp lệnh và xây dựng embedding để vượt qua vấn đề out-of-vocabulary (OOV) với các chỉ thị assembly. Phương pháp này hoàn toàn không giám sát, và đạt được kết quả hàng đầu trong các thí nghiệm đơn kiến trúc.
* Nhúng mã assembly và bộ mã tự chú ý: SAFE. SAFE [49] sử dụng bộ mã tự chú ý tự động cho câu từ Lin et al. [42] để học nhúng hàm xuyên kiến trúc. Phương pháp này đại diện cho các bộ mã tự chú ý NLP từ mô hình seq2seq, và, khác với Asm2Vec, nó được thiết kế đặc biệt để học tương đồng xuyên kiến trúc.
* Nhúng mã assembly, CFG và GNN: Massarelli et al., 2019. Massarelli et al. [45] sử dụng cùng mô hình Structure2vec GNN của Gemini [76], nhưng thay đổi các đặc trưng cấp block, chuyển từ các đặc trưng được thiết kế thủ công sang các đặc trưng không giám sát. Phương pháp này thú vị vì nó là một bước tiến của Gemini và kết hợp các lợi ích của việc nhúng cấp chỉ thị, bộ mã tự chú ý cấp basic-block và GNN.
* CodeCMR/BinaryAI. Mô hình được trình bày trong [79] cung cấp nền tảng cho framework BinaryAI [70] cho việc so khớp mã nguồn nhị phân cấp hàm. Chúng tôi chỉ tập trung vào phần xử lý hàm trong định dạng nhị phân. Mô hình kết hợp biểu diễn trung gian với một bộ mã tự chú ý NLP để nhận được nhúng cấp basic-block và một GNN để thu được nhúng đồ thị. Hai LSTMs mã hóa chuỗi và dữ liệu số nguyên từ hàm. Nhúng hàm là sự kết hợp của ba nhúng trên và mô hình nhị phân được huấn luyện end-to-end. Chúng tôi lưu ý rằng công trình này của Tencent là theo sau công trình trước đó của tác giả [78] và khi liên hệ, tác giả giải thích cách mô hình mới chính xác hơn.
* Trex. Trex [60] là một công trình gần đây dựa trên transformer phân cấp và micro-traces. Công trình này mang đến một thành phần động giúp trích xuất các micro-traces của hàm và điều này rất quan trọng để học ngữ nghĩa của các hàm. Tác giả tiền huấn luyện mô hình ML trên các micro-traces này và chuyển giao kiến thức đã học để so khớp các hàm tương tự ngữ nghĩa. Giai đoạn so khớp dựa hoàn toàn vào các đặc trưng tĩnh trong khi giả lập để tạo ra các micro-traces chỉ cần thực hiện trong giai đoạn tiền huấn luyện. Phương pháp này vượt trội trong việc xuyên kiến trúc và được xây dựng trên transformer, mô hình học sâu tiên tiến nhất trong NLP.

***4.1 Đánh giá***

* Một trong những mục tiêu của nghiên cứu này là thực hiện so sánh công bằng giữa các phương pháp khác nhau. Vì vậy, chúng tôi đã triển khai mỗi giai đoạn của đánh giá một cách thống nhất, bao gồm phân tích nhị phân, trích xuất đặc trưng và triển khai máy học. Bằng cách này, chúng tôi có thể tạo ra một cơ sở chung để thực hiện so sánh ý nghĩa và công bằng giữa các phương pháp khác nhau.
* Đối với giai đoạn phân tích nhị phân, chúng tôi sử dụng IDA Pro 7.3 [28], trong khi đối với trích xuất đặc trưng, chúng tôi dựa vào một tập hợp các tập lệnh Python sử dụng các API IDA Pro, Capstone [63], và NetworkX [26]. Chúng tôi triển khai tất cả các mô hình mạng neural trong Tensorflow 1.14, với ngoại lệ duy nhất của Trex [60], mô hình được xây dựng trên nền tảng Fairseq [57], một bộ công cụ mô hình chuỗi cho PyTorch. Cuối cùng, chúng tôi sử dụng Gensim 3.8 [65] để triển khai Asm2Vec [14] và chạy các mô hình nhúng chỉ thị [45, 49].
* Trong nhiều trường hợp, chúng tôi đã thu được ít nhất một phần của mã nguồn gốc của các công trình nghiên cứu cơ bản [17, 21, 48, 50, 59, 74, 75]. Rất tiếc, ngay cả khi một phần của mã có sẵn, nó thường được điều chỉnh cho bộ dữ liệu mà tác giả sử dụng trong bài báo của họ, và chúng tôi đã phải đưa ra nỗ lực triển khai đáng kể để thực thi đúng trên một tập hợp khác các trường hợp thử nghiệm. Trong quá trình này, chúng tôi đã áp dụng một triển khai thống nhất để giảm thiểu sự khác biệt đánh giá và chúng tôi đã giới thiệu một số tối ưu mã. Khi mã không có sẵn, chúng tôi đã liên hệ với các tác giả, nhưng chúng tôi nhận được hoặc không có câu trả lời hoặc hỗ trợ giới hạn. Hai phương pháp, Zeek [67] và Asm2Vec [14], đã được triển khai lại hoàn toàn, trong khi CodeCMR đã được kiểm tra bởi các tác giả do độ phức tạp cao của mô hình và một số biến ẩn không được thảo luận trong bài báo.
* Thông tin kỹ thuật bổ sung về tất cả các triển khai của chúng tôi, cùng với thông tin liên quan đến nỗ lực liên hệ các tác giả tương ứng và các cân nhắc liên quan đến việc sử dụng các mô hình đã được huấn luyện trước, có sẵn tại [47].
* Chúng tôi thực hiện tất cả các thí nghiệm trên một máy trạm được trang bị Ubuntu 18.04, Intel Xeon Gold 6230 (80 lõi ảo @ 2,10 GHz), 128GB RAM DDR4 và một GPU Nvidia RTX2080 Ti (1350MHz, 11GB GDDR6, 13,45 TFLOPS FP32).

***4.2 Bộ dữ liệu***

* Chúng tôi đã tạo hai bộ dữ liệu mới, Dataset-1 và Dataset-2, nhằm chụp được tính phức tạp và biến đổi của phần mềm thực tế, đồng thời bao gồm các thách thức khác nhau của tính tương đồng chức năng nhị phân: (i) nhiều bộ biên dịch và phiên bản, (ii) nhiều tối ưu hóa bộ biên dịch, (iii) nhiều kiến trúc và độ rộng bit, và (iv) phần mềm thuộc các loại khác nhau (tiện ích dòng lệnh so với ứng dụng giao diện người dùng đồ họa). Chúng tôi sử dụng Dataset-1 để huấn luyện các mô hình máy học và cả hai bộ dữ liệu để kiểm tra các phương pháp đã đánh giá.
* Dataset-1. Dataset-1 gồm bảy dự án mã nguồn mở phổ biến: ClamAV, Curl, Nmap, OpenSSL, Unrar, Z3 và Zlib. Sau khi biên dịch, chúng tôi thu được 24 thư viện riêng biệt. Mỗi thư viện được biên dịch bằng hai gia đình trình biên dịch, GCC và Clang, với bốn phiên bản khác nhau mỗi gia đình, bao gồm các phiên bản chính từ năm 2015 đến 2021 (chi tiết bổ sung về các dự án mã nguồn mở và phiên bản trình biên dịch được cung cấp tại [47].) Mỗi thư viện được biên dịch cho ba kiến trúc khác nhau, x86-64, ARM và MIPS, trong các phiên bản 32 bit và 64 bit (tổng cộng có 6 kết hợp kiến trúc), và 5 cấp độ tối ưu hóa O0, O1, O2, O3 và Os.
* Theo định nghĩa của chúng tôi về tính tương đồng chức năng, chúng tôi đã vô hiệu hóa việc nội suy chức năng để so sánh các chức năng xuất phát từ cùng một mã nguồn chính xác: việc nội suy chức năng thực tế là thêm mã vào mã nguồn gốc, và điều này có thể gây ô nhiễm kết quả và dẫn đến những kết luận sai lầm.
* Tổng cộng, bộ dữ liệu này bao gồm 5.489 nhị phân, với trung bình 228 kết hợp cho mỗi dự án nhị phân, và tổng cộng 26,8 triệu chức năng. Theo tiêu chí áp dụng bởi một số bài báo tiêu biểu [20, 24], chúng tôi lọc bỏ các chức năng có ít hơn năm khối cơ bản (18,2 triệu). 80% trong số các chức năng được lọc có hai khối cơ bản, và 93% trong số này có ít hơn 30 chỉ thị hợp ngữ.
* Các 8,6 triệu chức năng còn lại là điểm khởi đầu để xây dựng tập dữ liệu huấn luyện, xác thực và kiểm tra. Phần Phụ lục bao gồm thông tin bổ sung về số khối cơ bản và số chỉ thị cho các chức năng đã được chọn.
* Dataset-2. Dataset-2 được xây dựng trên các nhị phân được phát hành bởi các tác giả của bài báo rất gần đây là Trex [60]. Cụ thể, chúng tôi đã chọn 10 thư viện trong số 13 để tránh sự trùng lặp với Dataset-1: Binutils, Coreutils, Diffutils, Findutils, GMP, ImageMagick, Libmicrohttpd, LibTomCrypt, PuTTy và SQLite. Tập dữ liệu chứa các nhị phân đã được biên dịch cho các kiến trúc x86, x64, ARM 32 bit và MIPS 32 bit, 4 cấp độ tối ưu hóa (O0, O1, O2, O3) và GCC-7.5. Dataset-2 bổ sung cho Dataset-1 của chúng tôi với mục đích i) xác nhận kết quả của các mô hình trong Dataset-1 trên một bộ sưu tập các nhị phân đa dạng và lớn hơn, và ii) bao gồm việc so sánh với phương pháp Trex mới nhất. Trong thực tế, Trex không thể được tiền huấn luyện và điều chỉnh trên Dataset-1, vì bộ mô phỏng được phát hành bởi các tác giả [59] chỉ hỗ trợ một tập hợp con của các kết hợp kiến trúc và độ rộng bit. Hơn nữa, như đã chi tiết trong tài liệu trực tuyến của chúng tôi tại [47], việc đào tạo lại Trex vô cùng đắt đỏ, và chúng tôi đã chọn sử dụng cùng một mô hình như các tác giả trong các thí nghiệm của họ trên cùng một tập nhị phân. Chúng tôi công nhận lợi thế tiềm năng của Trex trong đánh giá này so với các mô hình khác được huấn luyện trên Dataset-1 (do đó là trên một tập hợp lớn các kiến trúc), và chúng tôi sẽ cân nhắc điều này trong cuộc thảo luận của chúng tôi.
* Sẵn có tập dữ liệu. Vì lợi ích của cộng đồng và để giảm thiểu công việc trong tương lai trong lĩnh vực này, chúng tôi đang công bố toàn bộ tập dữ liệu cho công chúng, có sẵn tại [47]. Chúng tôi cũng công bố các kịch bản và bản vá mà chúng tôi đã sử dụng để biên dịch chúng, để các nghiên cứu viên trong tương lai có thể tạo lại tập dữ liệu và xây dựng dựa trên công việc của chúng tôi.

***4.3 Cài đặt thực nghiệm***

* Chúng tôi đã tiến hành các thí nghiệm một cách kỹ lưỡng để đánh giá độ chính xác của các phương pháp được lựa chọn và một số biến thể của chúng. Với mục đích này, chúng tôi xác định sáu tác vụ khác nhau để đánh giá: (1) XO: các cặp hàm có các tối ưu hóa khác nhau, nhưng cùng một trình biên dịch, phiên bản trình biên dịch và kiến trúc. (2) XC: các cặp hàm có các trình biên dịch, phiên bản trình biên dịch và tối ưu hóa khác nhau, nhưng cùng một kiến trúc và độ rộng bit. (3) XC+XB: các cặp hàm có các trình biên dịch, phiên bản trình biên dịch, tối ưu hóa và độ rộng bit khác nhau, nhưng cùng một kiến trúc. (4) XA: các cặp hàm có các kiến trúc và độ rộng bit khác nhau, nhưng cùng một trình biên dịch, phiên bản trình biên dịch và tối ưu hóa. (5) XA+XO: các cặp hàm có các kiến trúc, độ rộng bit và tối ưu hóa khác nhau, nhưng cùng một trình biên dịch và phiên bản trình biên dịch. (6) XM: các cặp hàm đến từ các kiến trúc, độ rộng bit, trình biên dịch, phiên bản trình biên dịch và tối ưu hóa ngẫu nhiên.
* Ba tác vụ đầu tiên đánh giá các phương pháp cho những trường hợp sử dụng giới hạn chỉ đến một kiến trúc duy nhất, bao gồm một số ứng dụng thực tế của tính tương tự chức năng trong phân tích phần mềm độc hại và đảo mã hợp tác. Tác vụ thứ tư có ý nghĩa trong việc phân tích các hình ảnh firmware được biên dịch chéo bằng cách luôn sử dụng cùng một trình biên dịch và các tùy chọn trình biên dịch. Tác vụ thứ năm được thiết kế để hỗ trợ Dataset-2, được biên dịch với chỉ một trình biên dịch và phiên bản trình biên dịch. Cuối cùng, tác vụ cuối cùng là khó nhất và bao gồm các so sánh trên toàn bộ tập dữ liệu. Trong đánh giá của chúng tôi, chúng tôi cũng xem xét ba tập dữ liệu con cho XM: XM-S, XM-M và XM-L, bao gồm các chức năng có kích thước nhỏ (ít hơn 20 khối cơ bản), trung bình (từ 20 đến 100), và lớn (hơn 100 khối).

***4.3 Cài đặt thực nghiệm (tiếp theo)***

* Mỗi tác vụ được đánh giá dựa trên hai bài kiểm tra khác nhau: (i) diện tích dưới đường cong (AUC) của đường cong hoạt động của người nhận (ROC), đây là một chỉ số tổng hợp cho hiệu suất của mô hình trên tất cả các ngưỡng phân loại có thể có, và (ii) hai phép đo xếp hạng thông dụng, trung bình nghịch đảo (MRR) và độ nhớ (Recall@K) tại các ngưỡng K khác nhau. Các phép đo xếp hạng hữu ích để đánh giá hiệu suất mô hình trong các ứng dụng nơi cần phải tìm kiếm các hàm ứng viên trong cơ sở dữ liệu lớn, chẳng hạn như trong trường hợp sử dụng nghiên cứu lỗ hổng.
* Đối với bài kiểm tra đầu tiên, chúng tôi xây dựng một tập dữ liệu gồm 50k cặp hàm dương và 50k cặp hàm âm cho mỗi tác vụ, với tổng cộng 700k cặp hàm trên Dataset-1 và Dataset-2. Đối với bài kiểm tra xếp hạng, chúng tôi đã chọn 1,400 cặp hàm dương và 140k cặp hàm âm, tức là 100 cặp hàm âm cho mỗi cặp hàm dương.
* Tổng cộng, các bài kiểm tra của chúng tôi bao gồm 438,981 hàm nhị phân duy nhất, với điều kiện có ít nhất năm khối cơ bản. Trong mỗi tác vụ, các cặp được lấy mẫu ngẫu nhiên theo các ràng buộc tương ứng (ví dụ, trong tối ưu hóa chéo, các cặp dương và âm thuộc cùng một kiến trúc).
* Để làm cho việc đánh giá của chúng tôi ý nghĩa hơn, chúng tôi đã quyết định loại bỏ các hàm trùng lặp (bằng cách kiểm tra tên và băm của các hàm). Chúng tôi cũng chú ý rằng khía cạnh "lựa chọn cặp" này là quan trọng cho một đánh giá đúng đắn. Rất tiếc, một số công trình gần đây sử dụng các cách khác nhau để lựa chọn các cặp như vậy, làm cho nhiệm vụ đào tạo của họ trở nên dễ dàng hơn hoặc khó khăn hơn đáng kể. Ví dụ, trong [36], các nhà nghiên cứu đã tạo ra các cặp âm trong khi giữ nguyên các tùy chọn biên dịch. Trong một ví dụ khác (VulSeeker [25]), các cặp âm được tạo ngẫu nhiên và không đòi hỏi kiểm tra bất kỳ tùy chọn nào về trình biên dịch cũng như sự hiện diện của hàm mẫu. Những không đồng nhất này làm suy yếu khả năng so sánh công bằng, và thường được bỏ qua.
* Theo hướng thực tiễn thông thường trong lĩnh vực nghiên cứu này, chúng tôi trình bày đánh giá của các phương pháp được lựa chọn theo cách thông dụng nhất, sử dụng hai tập dữ liệu được thiết kế đặc biệt. Chúng tôi nhấn mạnh rằng đánh giá trực tiếp các phương pháp trong một ứng dụng bảo mật thường khó khăn, đặc biệt là trong trường hợp phát hiện lỗ hổng và phân cụm phần mềm độc hại, vì nó yêu cầu tạo một cơ sở chính xác, đây cũng chính là vấn đề mà tính tương tự chức năng đang cố gắng giải quyết từ đầu.

***4.4 So sánh giữa các phương pháp fuzzy-hashing***

* Phần này bàn về cuộc điều tra chi tiết của hai phương pháp dựa trên fuzzy-hashing: Catalog1 [74] và FunctionSimSearch [18].
* Catalog1 sử dụng bytes gốc làm đặc trưng đầu vào và kích thước chữ ký khác nhau (tức là số lượng hàm băm): chúng tôi hiển thị kết quả của hai biến thể, một với kích thước 16 và một với kích thước 128. Ngược lại, FunctionSimSearch (FSS) sử dụng một sự kết hợp của các đồ thị con (G), mnemonics (M) và immediates (I): chúng tôi đã thực hiện các thử nghiệm khác nhau bằng cách kích hoạt từng loại đặc trưng đầu vào một cách tăng dần, bao gồm cả sự kết hợp tuyến tính có trọng số của chúng w, như chúng tôi đã tìm thấy trong phiên bản gốc (G:1.00, I:4.00, M:0.05).
* Do các phương pháp fuzzy-hashing không bị ảnh hưởng bởi giai đoạn đào tạo, chúng tôi đã sử dụng chúng để thực hiện đánh giá được hướng dẫn về cách các biến biên dịch ảnh hưởng đến việc so sánh hàm nhị phân. Do đó, đối với các phương pháp này, chúng tôi trước tiên thực hiện nhiều thí nghiệm trong đó chúng tôi biến đổi một biến (tức là gia đình trình biên dịch, phiên bản, tùy chọn, kiến trúc và bitness) trong khi giữ các biến còn lại không đổi. Kết quả, trong Bảng 1, cho thấy rõ ràng rằng, khi chỉ xem xét một biến tự do tại một thời điểm, ngay cả các phương pháp đơn giản như fuzzy hashes cũng rất hiệu quả: "raw" bytes được xác nhận là các đặc trưng tốt cho các so sánh cùng kiến trúc, trong khi graphlets hiệu quả trong các so sánh chéo kiến trúc. Đối với Catalog1, càng lớn kích thước chữ ký, càng tốt hiệu suất, nhưng chúng bị giới hạn bởi tổng số hàm băm được bao gồm trong việc thực hiện.

A table with numbers and letters

Description automatically generated

* Sau đó, chúng tôi đã đánh giá hai phương pháp này với sáu nhiệm vụ được trình bày trước đó. Bảng 2 và 4 cho thấy kết quả trên Dataset-1 và Dataset-2: có nhiều biến tự do cùng một lúc là một vấn đề khó hơn và các phương pháp đơn giản không còn hiệu quả nữa. Trong nhiệm vụ XC (Bảng 2), Catalog1 và FSS có cùng AUC. Đối với FSS, cấu hình chỉ có graphlets (G) là tốt nhất trong tất cả các nhiệm vụ trừ XC và XO, nơi việc sử dụng graphlets kết hợp với mnemonics (G+M) có AUC cao hơn. Hơn nữa, FSS cũng có vẻ hoạt động tốt hơn trên các hàm lớn hơn, điều này có thể do số lượng graphlets khác nhau lớn hơn có thể được trích xuất.
* Cuối cùng, trong nhiệm vụ XA, độ chính xác của FSS giảm khi sử dụng các đặc trưng bổ sung như mnemonics và immediates, và bất ngờ là sự kết hợp tuyến tính có trọng số của ba đặc trưng này không cho kết quả tốt hơn so với các cấu hình cơ bản khác.
* Catalog1 nhanh hơn trong hai phương pháp, trong khi FSS chậm hơn khoảng 3 lần do giai đoạn trích xuất đặc trưng kéo dài hơn.

A table with numbers and symbols

Description automatically generated

***4.5 So sánh các mô hình học máy***

* Chúng tôi đánh giá tất cả các phương pháp đã chọn bằng cách sử dụng một tập dữ liệu huấn luyện chung được trích xuất từ Bộ dữ liệu 1 (ngoại trừ Trex [60]) và sử dụng tiêu chí tương tự cho nhiệm vụ XM để tạo các mẫu tích cực và tiêu cực. Tuy nhiên, quan trọng để lưu ý rằng có thể cải thiện kết quả của mỗi nhiệm vụ bằng cách sử dụng dữ liệu huấn luyện cụ thể cho từng nhiệm vụ. Chúng tôi đã thực hiện đánh giá này nhưng chúng tôi đã bỏ qua các kết quả do nhận thấy việc huấn luyện trên dữ liệu phổ quát nhất (XM) đã đạt được hiệu suất tổng quát gần như tốt nhất cho mỗi nhiệm vụ.
* So sánh các mô hình học máy, đặc biệt là các mạng neural sâu, là một nhiệm vụ khó khăn do có nhiều biến số có thể ảnh hưởng đến kết quả cuối cùng, bao gồm cài đặt và cấu hình mô hình (ví dụ: số lớp hoặc loại mạng neural tuần tự), các siêu tham số khác nhau (ví dụ: tỷ lệ học và kích thước lô), hàm mất mát, bộ tối ưu và số lượng epoch huấn luyện. Để đảm bảo sự đồng nhất trong việc so sánh, tất cả các mô hình được huấn luyện với cùng một dữ liệu được tạo ngẫu nhiên từ 256.625 hàm nhị phân duy nhất. Hơn nữa, chúng tôi đã thực hiện các thí nghiệm sâu rộng để đánh giá các bộ tính năng khác nhau, các cấu hình mô hình khác nhau, siêu tham số và hàm mất mát. Kết quả cho mỗi mô hình có thể được cải thiện bằng cách sử dụng phương pháp tìm kiếm lưới phức tạp và các kết quả chúng tôi trình bày có thể được sử dụng như một điểm khởi đầu cho các công trình tương lai.
* Bảng 3 và Bảng 4 cho thấy kết quả của các mô hình đã thử nghiệm và các biến thể tương ứng của chúng trên hai bộ dữ liệu. Bảng 8 bao gồm một số thông tin chung về các mô hình và quá trình huấn luyện của chúng, bao gồm số lượng tham số, kích thước lô, số lượng epoch và thời gian huấn luyện cho mỗi epoch.
* Kết quả cho thấy trong số các mô hình tạo ra biểu diễn vector cho một hàm (tức là nhúng), GNN từ [40] đạt được giá trị tốt nhất trong tất cả các chỉ số và tất cả các nhiệm vụ. Chúng tôi cũng nhận thấy là hầu hết các mô hình học máy có hiệu suất tương tự nhau khi so sánh trên AUC, nhưng khác nhau khi so sánh trên các chỉ số xếp hạng (MRR10 và recall@1), như được thể hiện trong Hình 2. Sau đó, liên quan đến các mô hình nhúng khác, SAFE [49] cung cấp AUC tốt hơn so với GNN với các tính năng không giám sát [45], và trong một cấu hình cụ thể, AUC của nó nhỉnh hơn cả Gemini [76]. Đối với các phương pháp thực hiện so sánh trực tiếp, GMN từ [40] là mô hình có hiệu suất tốt nhất trong tất cả các nhiệm vụ, trong khi Zeek có một AUC hơi thấp hơn (ngoại trừ các hàm lớn), nhưng MRR10 và recall@1 thấp hơn rất nhiều.
* Bây giờ, chúng tôi sẽ thảo luận chi tiết về tám quan sát từ kết quả của chúng tôi.
* So sánh Trex. Kết quả của chúng tôi cho thấy, trong nhiệm vụ XO, Trex có cùng AUC và tương tự MRR10 và recall@1 như Asm2Vec, cũng như các biến thể word2vec khác, tức là hơi thấp hơn GNN và GMN. Điều này củng cố khẳng định rằng các mô hình ngôn ngữ mạnh mẽ trong các nhiệm vụ cùng kiến trúc. Tuy nhiên, khác với Asm2Vec, Trex duy trì hiệu suất tương tự trong nhiệm vụ XA và XA+XO, cho thấy sức mạnh của bộ biến đổi trong cài đặt (cross language) chéo kiến trúc. Chúng tôi xác nhận rằng Trex cải thiện hiệu suất của SAFE trong nhiệm vụ XO (0,94 so với 0,90) và trong nhiệm vụ XA+XO (0,94 so với 0,91). Các thử nghiệm của chúng tôi cũng cho thấy việc suy luận cho Trex chạy trên GPU (chi tiết hơn trong [47]) nhanh hơn so với việc triển khai Asm2Vec đa tiến trình của chúng tôi (3,92 giây so với 8,51 giây cho 100 hàm), nhưng chậm hơn SAFE chạy trên CPU (3,92 giây so với 1,46 giây cho 100 hàm).
* So sánh các GNN khác nhau. Cả Gemini [76] và Li et al. [40] đều sử dụng GNN với các biến thể khác nhau của lớp truyền thông điệp và tổng hợp [8, 41]. Sử dụng kết quả từ các dòng thứ ba và thứ tư trong Bảng 3 và Bảng 4, chúng tôi so sánh hai biến thể này sử dụng cả tính năng khối cơ bản (túi từ (BoW) của các opcode) và không có tính năng. Kết quả cho thấy biến thể GNN của Li et al. cung cấp sự cải thiện đáng kể so với GNN (s2v), một trong những biến thể được sử dụng bởi Gemini, trong tất cả các nhiệm vụ. Tuy nhiên, thời gian thực thi của hai biến thể vẫn tương tự nhau (1,48 giây so với 1,40 giây khi không có tính năng).
* So sánh các tập tính năng khác nhau trong GNN (s2v). Gemini [76] sử dụng mô hình GNN được gọi là Structure2vec (s2v) [8] với các tính năng được thiết kế thủ công. Mục tiêu của chúng tôi là hiểu rõ tính quan trọng của những tính năng này so với việc không có một vectơ đặc trưng hoặc sử dụng một tập tính năng khác như túi từ (BoW) của các opcode. Kết quả trong Bảng 3 và Bảng 4 cho thấy rằng tính năng được thiết kế thủ công chỉ hoạt động tốt hơn đối với các hàm nhỏ và lớn trong nhiệm vụ XA, và BoW của các opcode có hiệu suất tương tự trong tất cả các chỉ số khác nhau, thậm chí có độ nhớ tốt hơn cho các giá trị K khác nhau, như được hiển thị trong Hình 2. Ngoài ra, thời gian thực thi khác nhau (1,66 giây so với 7,18 giây), do giai đoạn trích xuất tính năng lâu hơn trong Gemini. Điều này có nghĩa là tính năng phức tạp hơn và khó khăn hơn để trích xuất không nhất thiết phải vượt trội hơn một biểu diễn cơ bản hơn. Một BoW của 200 opcode có 20 lần số lượng tính năng của Gemini, điều này dẫn đến một ma trận đầu vào lớn hơn cho mạng nơ-ron nút của GNN. Chúng tôi cũng đã kiểm tra một BoW của 1024 opcode nhưng kết quả không cải thiện đáng kể, điều này có nghĩa là những tính năng bổ sung đó không đóng góp đáng kể vào biểu diễn của hàm.
* Cuối cùng, chúng tôi muốn kiểm tra xem việc sử dụng nhúng lệnh như tính năng GNN có giúp tăng AUC không, như được trình bày trong [45]. Kết quả của chúng tôi cho thấy nhúng lệnh trên mã hợp lệ hóa của hệ thống không có AUC cao hơn so với BoW của các opcode hoặc các tính năng được thiết kế thủ công (chỉ có bộ mã hóa khối cơ bản RNN đạt được AUC tương tự), MRR10 và recall@1 thấp hơn và thời gian huấn luyện tăng đáng kể.

A table with numbers and letters

Description automatically generated

A graph of different colored lines

Description automatically generated

* Mô hình hàm sử dụng trình mã hóa văn bản. SAFE [49] sử dụng một bộ mã hóa câu dựa trên nhúng lệnh, và AUC tốt hơn GNN (s2v) với tính năng không giám sát [45]. So với Gemini [76], AUC tương tự, nhưng MRR10 và recall@1 thấp hơn. SAFE hoạt động tốt hơn trên các hàm nhỏ và kết quả cũng cải thiện khi tăng độ dài tối đa của lệnh từ 150 lên 250 (Bảng 3). Tuy nhiên, SAFE phải đối mặt với thách thức của các từ nằm ngoài từ vựng (OOV). Như đã giới thiệu ở phần 2, các phương pháp khác như InnerEye [82] hoặc Mirror [80] áp dụng các kỹ thuật chuẩn hóa mã hợp lệ khác nhau để giảm thiểu vấn đề này. Để minh họa thách thức này, chúng tôi đã đo lường tác động của các lệnh OOV trong SAFE và chúng tôi quan sát rằng x86-64 là kiến trúc bị ảnh hưởng nhiều nhất bởi vấn đề OOV (ít hơn 30% các hàm không có từ OOV), có lẽ do bộ lệnh CISC của nó, tiếp theo là MIPS và cuối cùng là ARM, với hơn 40% các hàm không có từ OOV nào.

A table of numbers and symbols

Description automatically generated

* Asm2Vec và các mô hình paragraph2vec khác. Bảng 3 và Bảng 4 cho thấy kết quả so sánh của Asm2Vec [14] với các biến thể PV-DM và PV-DBOW của paragraph2vec [38]. Tất cả ba mô hình đều có hiệu suất tương tự nhau, và so với GNN [40], sử dụng một phương pháp đơn kiến trúc cũng không mang lại bất kỳ lợi ích nào. Chúng tôi lưu ý rằng kết quả bị ảnh hưởng mạnh mẽ bởi một số yếu tố, bao gồm kích thước từ vựng lệnh, số lượng lần đi bộ ngẫu nhiên và một số chi tiết thực thi. Trong quá trình huấn luyện, chúng tôi đã chọn 1 triệu mã thông báo (trong tổng số 1.9 triệu) với tần suất tối thiểu là 5, trong đó hầu hết chúng là các bù trừ số hoặc địa chỉ thập lục phân. Giảm ngưỡng này làm cải thiện kết quả, nhưng cũng làm tăng kích thước từ vựng và thời gian huấn luyện. Vào thời gian suy luận, chúng tôi không thay đổi từ vựng, mặc dù các phương pháp không giám sát này có thể hưởng lợi từ một từ vựng suy luận mới mà không làm mất tính hợp lệ của kết quả. Trong các thử nghiệm của chúng tôi, tất cả ba mô hình đều sử dụng cùng một từ vựng. Chúng tôi cũng lưu ý rằng tất cả ba biến thể đều đạt được AUC cao trên các kịch bản trong đó chỉ có một biến số tự do, ví dụ, nhiệm vụ XO trong Bảng 4, nhưng AUC giảm khi nhiều biến biên dịch được xem xét cùng nhau, ví dụ, nhiệm vụ XC trong Bảng 3, trong đó trình biên dịch, phiên bản của nó và các tối ưu hóa thay đổi.
* So sánh hiệu suất. Chúng tôi cũng theo dõi tính hiệu quả của các phương pháp này đối với việc huấn luyện (Bảng 8) và thời gian thử nghiệm, tức thời gian suy luận (Bảng 4). Chúng tôi tập trung vào thảo luận về thời gian suy luận, vì huấn luyện các mô hình chủ yếu là một công việc một lần duy nhất. Đối với thời gian suy luận, SAFE [49] dường như là nhanh nhất trong số các mô hình máy học với 1.46 giây để xử lý 100 hàm. GMN và GNN từ [40] có thời gian chạy tương tự và thuộc nhóm thấp nhất, tuy nhiên GMN chỉ xử lý một cặp hàm đầu vào. GNN (s2v) với các tính năng của Gemini [76] chậm hơn 4 lần so với phiên bản có tính năng opcode: lý do là thời gian chuẩn hóa đặc trưng dài hơn. Thời gian suy luận của Zeek [67] cũng bị ảnh hưởng bởi thời gian chuẩn hóa và xử lý đặc trưng dài hơn và là chậm nhất trong số các phương pháp. GNN (s2v) với các tính năng không giám sát chậm hơn trong biến thể RNN, do thời gian suy luận dài hơn do tính phức tạp bổ sung của mô hình. Tương tự, Trex [60] bị ảnh hưởng bởi thời gian suy luận dài do tính phức tạp của mô hình NLP. Cuối cùng, Asm2Vec [14] nằm trong nhóm chậm nhất, vì nó yêu cầu 10 epoch của suy luận để trích xuất các nhúng hàm mới. Đáng chú ý, Asm2Vec chậm hơn các mô hình paragraph2vec [38] khác do xây dựng nhúng lệnh cụ thể.
* Chúng tôi nhận thức rằng loại hình đánh giá này không phù hợp với những nỗ lực tái thực hiện chúng tôi đã thực hiện cho các phương pháp còn lại. Tuy nhiên, chúng tôi tin rằng CodeCMR là một phương pháp khá thú vị và triển vọng, và chúng tôi cho rằng đưa nó vào bài báo của chúng tôi là đáng giá. Chúng tôi cũng lưu ý rằng một lựa chọn khác có thể là cố gắng tái thực hiện phương pháp này, nhưng chúng tôi tin rằng nó sẽ rất khó khăn để đảm bảo tính chính xác của tái thực hiện do sự phức tạp cao của hệ thống và nhiều biến số "ẩn" không được thảo luận trong bài báo.
* Chúng tôi sẽ trình bày những thông tin đáng chú ý từ kết quả đánh giá của chúng tôi. Vì mục tiêu cuối cùng của mô hình của CodeCMR là ghép cặp mã nhị phân với mã nguồn, để huấn luyện và kiểm tra dữ liệu của chúng tôi, họ cô lập phần của mô hình chỉ xử lý dữ liệu nhị phân của hàm. Chúng tôi đã cung cấp cho họ một phiên bản đã tiền xử lý của một phần của tập dữ liệu của chúng tôi, bao gồm các hàm ARM và x86, 32 bit và 64 bit, đã tiền xử lý bằng IDAPro với HexRays decompiler.2 Dữ liệu trích xuất bao gồm một CFG có thuộc tính với các hướng dẫn microcode IDA, số nguyên ctree và các chuỗi từ dữ liệu hàm. Để so sánh cơ sở, chúng tôi đã chạy mô hình GNN từ [40], sử dụng túi từ của 200 mã opcode phổ biến nhất và túi từ của 80 hướng dẫn microcode IDA làm tính năng khối cơ bản.

A screenshot of a computer

Description automatically generated

A graph of a number of results

Description automatically generated

* Kết quả được trình bày trong Bảng 5. Mô hình GNN sử dụng túi từ của các hướng dẫn microcode IDA có AUC cao hơn so với mô hình GNN sử dụng túi từ của các mã opcode, nhưng mô hình thứ hai có recall cao hơn cho các giá trị K lớn (Hình 3). Nói chung, tất cả các số liệu cho mô hình BinaryAI/CodeCMR đều cao hơn so với các mô hình được kiểm tra còn lại. Nếu những kết quả này được xác minh bởi các nghiên cứu độc lập trong cộng đồng, điều này có thể là một hướng nghiên cứu rất triển vọng. Trong thực tế, mô hình BinaryAI/CodeCMR giới thiệu một số đổi mới. Trước tiên, nó kết hợp nhiều thành phần xây dựng trong một mô hình duy nhất (tức là, một bộ mã hóa NLP, một GNN và hai LSTM) và huấn luyện tất cả mọi thứ chung bằng cách sử dụng chiến lược end-to-end. Thứ hai, các tác giả đã chỉ ra trong bài báo của họ rằng chiến lược huấn luyện (ví dụ, sử dụng lấy mẫu dựa trên khoảng cách [73]) và hàm mất mát (ví dụ, mất mát triplet [66]) đóng một vai trò quan trọng và có thể mang lại cải thiện hiệu suất đáng kể.
* Ghi chú về Kim et al. Trong một bài báo gần đây được đăng trên Arxiv của Kim et al. [36], các tác giả đề xuất một mô hình có tính khả giải và cho thấy rằng kỹ thuật lập chỉ mục thủ công có thể đạt được kết quả tương đương với "các mô hình tối tân," như Vulseeker [25]. Đánh giá của họ chỉ xem xét một thay đổi biến duy nhất mỗi lần (ví dụ, chỉ có sự thay đổi của trình biên dịch, trong khi kiến trúc và mức tối ưu hóa cố định). Điều này là một cài đặt đơn giản hơn so với sáu nhiệm vụ đánh giá của chúng tôi, nơi như đã được trình bày trong Bảng 1, ngay cả các phương pháp fuzzy-hash đơn giản cũng hiệu quả. Hơn nữa, bài báo thiếu bất kỳ đánh giá có ý nghĩa nào so với các kỹ thuật tối tân. Ví dụ, so sánh với đường cong ROC của Vulseeker là lý thuyết, được thực hiện trên một tập dữ liệu khác nhau và không cần huấn luyện lại mô hình. Một khía cạnh khác làm cho việc so sánh này rất thách thức là cách các cặp dương tính và tiêu cực được chọn, như đã đề cập trong Phần 4.3.

***4.6 Ứng dụng phát hiện lỗ hổng***

* Như một ví dụ về ứng dụng bảo mật, chúng tôi đã thử nghiệm tất cả các mô hình trên một nhiệm vụ phát hiện lỗ hổng. Để làm điều này, chúng tôi đã chọn mười hàm có lỗ hổng từ OpenSSL 1.0.2d, bao gồm tổng cộng tám CVEs (Các lỗ hổng công nghệ thông tin chung). Làm mục tiêu, chúng tôi đã chọn các thư viện libcrypto được nhúng trong hai hình ảnh firmware: Netgear R7000 (kiến trúc ARM 32 bit) và TP-Link Deco M4 (kiến trúc MIPS 32 bit). Thông tin chi tiết về việc lỗ hổng nào ảnh hưởng đến từng hình ảnh firmware được bao gồm trong [47]. Chúng tôi biên dịch mười hàm có lỗ hổng cho bốn kiến trúc (x86, x64, ARM 32 bit, MIPS 32 bit) và thực hiện đánh giá xếp hạng, tương tự như những gì chúng tôi đã trình bày trong các kiểm tra trước đó. Khi đánh giá kết quả phát hiện lỗ hổng, chúng tôi chỉ sử dụng các hàm truy vấn có lỗ hổng cụ thể cho một hình ảnh firmware cụ thể. Kết quả được hiển thị trong Bảng 7: chúng tôi sử dụng MRR10 (độ chính xác xếp hạng trung bình đầu tiên) làm chỉ số so sánh để đánh giá cách mỗi mô hình xếp hạng hàm mục tiêu có lỗ hổng cho mỗi hàm truy vấn. Không ngạc nhiên, mô hình GMN [40] với các tính năng opcode là mô hình có hiệu suất tốt nhất, tuy nhiên nó yêu cầu phân tích từng cặp hàm, giới hạn tính mở rộng của phương pháp. Trex [60] và biến thể GNN của Li et al. [40] cung cấp kết quả thứ hai tốt nhất. Tuy nhiên, các mô hình FSS với trọng số tùy chỉnh lại có giá trị MRR10 cao nhất đối với so sánh x64 vs. Netgear R7000. Chúng tôi đã sử dụng các trọng số được gửi kèm trong mã nguồn, đã được tối ưu hóa cho so sánh OpenSSL. Điều này chứng tỏ rằng quá trình tối ưu hóa mà FSS thực hiện có các ứng dụng thực tế, tuy nhiên nó không mở rộng sang các cấu hình khác. Bảng 7 cũng cho thấy so sánh giữa các kiến trúc khác nhau, đặc biệt cột ARM32 cho Netgear và cột MIPS32 cho TP-Link cho thấy so sánh cùng kiến trúc. Firmware Netgear R7000 được biên dịch cho kiến trúc ARM 32 bit, trong khi TP-Link Deco-M4 cho kiến trúc MIPS 32 bit: điều này giải thích tại sao Asm2Vec có giá trị MRR10 cao trong các cột tương ứng. Cuối cùng, Bảng 6 chứa kết quả xếp hạng thực tế của các hàm có lỗ hổng cho hình ảnh Netgear R7000, cho thấy rằng các giá trị MRR10 khá cao có thể ẩn trong thực tế các hạng xếp hạng khá thấp.

A table of number and text

Description automatically generated

1. Thảo luận

* Chúng tôi bây giờ rút ra một số kết luận từ các kết quả trước đó và trả lời một số câu hỏi nghiên cứu.
* Kết quả cho thấy rằng mô hình học máy duy nhất, GNN từ Li et al. [40], vượt trội hơn tất cả các biến thể khác trong sáu nhiệm vụ được đánh giá, đạt được hiệu suất tương tự phiên bản GMN ít mở rộng được. Các mô hình dựa trên nhúng [45, 49, 60, 76] hiển thị độ chính xác thấp hơn nhưng tương tự nhau. Zeek [67], một phương pháp so sánh trực tiếp, có AUC cao hơn đối với các hàm lớn. Asm2Vec [14] không cải thiện bất kỳ mô hình nào hơn và các phương pháp fuzzy hashing không hiệu quả khi nhiều biến biên dịch thay đổi cùng lúc.
* Đó là những đóng góp chính của các giải pháp học máy mới so với các phương pháp fuzzy hashing đơn giản hơn? Các mô hình học sâu cung cấp một cách hiệu quả để học một biểu diễn hàm (tức là nhúng), tạo ra sự phân tách không gian giữa các lớp hàm khác nhau. Khác với các phương pháp fuzzy hashing, các mô hình học máy đạt độ chính xác cao ngay cả khi nhiều biến biên dịch thay đổi cùng lúc và họ được hưởng lợi từ lợi thế của các tập dữ liệu đào tạo lớn được xây dựng trên cơ sở một định nghĩa đáng tin cậy về tùy chọn biên dịch. Việc sử dụng một kiến trúc Siamese [40, 45, 49, 76] kết hợp với hàm mất mát dựa trên biên [40, 79] đã giới thiệu những cải tiến đáng kể trong kết quả. Hơn nữa, GNN [40, 45, 76, 79] là một bộ mã hàm hiệu quả có thể được sử dụng kết hợp với các bộ mã hướng dẫn khác [45, 79].

A table of data with numbers

Description automatically generated

A screenshot of a data

Description automatically generated

* Vai trò của các tập đặc trưng khác nhau là gì? Kết quả cho thấy việc lựa chọn kiểu mô hình học máy, đặc biệt là GNN, và hàm mất mát quan trọng không kém việc lựa chọn tập đặc trưng vào mô hình. Sử dụng các đặc trưng khối cơ bản (ví dụ như ACFG) mang lại kết quả tốt hơn, nhưng sự khác biệt giữa các đặc trưng được thiết kế thủ công và các đặc trưng đơn giản hơn, chẳng hạn như túi từ của các opcode cơ bản, là rất nhỏ. Ngạc nhiên thay, các nhúng hướng dẫn [45] không nâng cao hiệu suất của các mô hình GNN, tuy nhiên chúng tôi nghĩ rằng cần thực hiện nhiều thử nghiệm để đánh giá các kết hợp khác. Zeek [67] cho thấy thông tin dòng dữ liệu có thể nâng cao kết quả, đặc biệt là đối với các hàm lớn. Cuối cùng, các phương pháp fuzzy hashing nhạy cảm hơn đối với loại đặc trưng, do thiếu giai đoạn huấn luyện.
* Các phương pháp khác nhau có hiệu quả khác nhau ở các nhiệm vụ khác nhau? Đặc biệt, liệu so sánh giữa các kiến trúc khác nhau có khó khăn hơn so với làm việc với một kiến trúc duy nhất? Đánh giá của chúng tôi cho thấy hầu hết các mô hình học máy thực hiện rất tương tự trong tất cả các nhiệm vụ được đánh giá, cả trong cùng một kiến trúc và giữa các kiến trúc khác nhau. Hơn nữa, không cần phải huấn luyện chúng trên một nhiệm vụ cụ thể, vì việc sử dụng dữ liệu nhiệm vụ thông thường nhất (XM) cho phép đạt được hiệu suất tổng thể gần như tốt nhất cho mỗi nhiệm vụ. Điều này không áp dụng cho các phương pháp fuzzy hashing. Ví dụ, các đặc trưng graphlet của FunctionSimSearch có hiệu suất tương tự trong tất cả các nhiệm vụ, nhưng việc kết hợp chúng với các đặc trưng khác làm giảm AUC trong một số nhiệm vụ. Tuy nhiên, không phải tất cả các phương pháp có thể được sử dụng trong so sánh giữa các kiến trúc khác nhau: Asm2Vec [14] và hai mô hình paragraph2vec [38] bị hạn chế chỉ đối với các so sánh cùng kiến trúc, do tiếp cận huấn luyện không giám sát cụ thể, cũng như Catalog1 [74].
* Có một hướng nghiên cứu cụ thể có vẻ hứa hẹn là điều hướng tương lai để thiết kế các kỹ thuật mới. Kết quả cho thấy các mô hình học sâu có tính mở rộng và độ chính xác phù hợp cho các nhiệm vụ tương tự hàm khác nhau, đặc biệt là nhờ khả năng học một biểu diễn hàm thích hợp cho nhiều nhiệm vụ. Mặc dù các mô hình GNN đã cung cấp kết quả tốt nhất, nhưng vẫn có hàng chục biến thể khác cần được thử nghiệm. Ngoài ra, kết hợp GNN với các bộ mã hóa hướng dẫn lập trình lắp ráp là một hướng nghiên cứu triển vọng khác [45, 78, 79]. Nhiều công trình trước đây tập trung vào việc chọn các đặc trưng và cấp độ trừu tượng của đặc trưng khác nhau, nhưng các mô hình học máy gần đây hơn tập trung vào việc sử dụng mã lắp ráp đã được chuẩn hóa hoặc biểu diễn trung gian, khai thác sức mạnh của việc học biểu diễn. Tác động của việc kết hợp biểu diễn trung gian và thông tin dataflow cũng cần được nghiên cứu. Hơn nữa, chúng tôi đã quan sát thấy rằng việc chọn các đặc trưng và mô hình học máy không phải là những yếu tố duy nhất ảnh hưởng đến hiệu suất của một phương pháp. Một số các yếu tố bổ sung như chiến lược huấn luyện và các hàm mất mát chỉ mới được thảo luận qua loa trong quá khứ và chỉ mới được khám phá gần đây. Li et al. [40] giới thiệu hai hàm mất mát thay thế, một hàm dựa trên khoảng cách Euclidean và một hàm tương đồng Hamming xấp xỉ để tối ưu hóa quá trình tìm kiếm hàng xóm hiệu quả. Tiếp nối theo cùng hướng, các nghiên cứu gần đây của CodeCMR [79] cho thấy cải tiến đáng kể trong kết quả nhờ việc áp dụng phương pháp lấy mẫu có trọng số chuẩn (một dạng lấy mẫu có trọng số khoảng cách [73]) kết hợp với hàm mất mát ba điểm [66].

Tóm tắt

* Bài báo này thực hiện một nghiên cứu đo lường đầu tiên bao gồm hơn năm năm nghiên cứu về tính tương tự của hàm nhị phân. Chúng tôi đã xác định một số thách thức trong lĩnh vực nghiên cứu này và làm cho việc so sánh có ý nghĩa trở nên khó khăn, nếu không phải là hoàn toàn không thể. Công việc của chúng tôi nhằm giải quyết khoảng cách này và giúp cộng đồng có được sự rõ ràng trong lĩnh vực nghiên cứu này.
* Chúng tôi hy vọng rằng việc công bố tất cả các hiện thực, tập dữ liệu và kết quả gốc của chúng tôi sẽ giúp cộng đồng có một điểm tham chiếu để bắt đầu xây dựng các phương pháp mới và sẽ được khuyến khích để đánh giá chúng trong một khung làm việc chung để nhận biết rõ hơn các khía cạnh mới thực sự cải thiện tình trạng nghệ thuật, và những khía cạnh nào chỉ xuất hiện như vậy.