Practical Binary Code Similarity Detection with BERT-based Transferable Similarity Learning.

Phát hiện tương đồng mã nhị phân thực tế với Học Tương Đồng Chuyển Giao Dựa Trên BERT.

TÓM TẮT

* Phát hiện tương đồng mã nhị phân (BCSD) đóng vai trò cơ sở cho một loạt các ứng dụng, bao gồm việc sao chép phần mềm, phân loại phần mềm độc hại và khám phá lỗ hổng đã biết. Tuy nhiên, việc suy luận về ý nghĩa ngữ cảnh của một mã nhị phân là thách thức do thiếu thông tin ngữ nghĩa có sẵn trong mã nguồn. Những tiến bộ gần đây khai thác những lợi ích của một kiến trúc học sâu để hiểu rõ hơn về ngữ nghĩa mã nguồn cơ bản và những lợi ích của kiến trúc Siamese trong việc phát hiện BCSD tốt hơn.
* Trong bài báo này, chúng tôi đề xuất BinShot, một kiến trúc học tương đồng dựa trên BERT có tính chuyển giao cao để thực hiện BCSD hiệu quả. Chúng tôi giải quyết vấn đề phát hiện tương đồng mã thông qua việc học một lần (trường hợp đặc biệt của việc học vài lần). Để làm được điều này, chúng tôi áp dụng một vectơ khoảng cách có trọng số với hàm mất mát entropy nhị phân lên BERT. Với nguyên mẫu của BinShot, kết quả thử nghiệm của chúng tôi chứng minh tính hiệu quả, tính chuyển giao và tính thực tế của BinShot, một kiến trúc có khả năng phát hiện tương đồng của các chức năng chưa từng thấy trước. Chúng tôi chứng minh rằng BinShot vượt trội hơn các phương pháp tiến tiến trước đây cho BCSD.

1 GIỚI THIỆU

* Ngày nay, phần mềm có thể coi là đóng một vai trò then chốt trong hoạt động của hàng tỷ hệ thống máy tính xung quanh chúng ta, bao gồm máy tính để bàn, laptop, máy trạm, điện thoại di động, máy chủ đám mây và các thiết bị IoT (Internet of Things - Internet của các đối tượng). Mỗi chương trình bao gồm một tập hợp các chỉ thị, quy trình và dữ liệu liên quan đến một nhiệm vụ cụ thể, hầu hết trong đó (ví dụ như ứng dụng sẵn có) được phân phối dưới dạng mã nhị phân có thể thực thi. Mặc dù sự gia tăng về phức tạp và kích thước của các chương trình, quá trình phát triển phần mềm đã tiến hóa theo một cách đơn giản với một loạt các thư viện, bộ công cụ, mô-đun và khung ứng dụng đã được thiết kế tốt có sẵn. Trong khi việc tái sử dụng mã giúp tăng năng suất cao và bảo trì mã thuận tiện, nó đi kèm với các mối đe dọa không lường trước như vi phạm bản quyền [90], sự phổ biến của biến thể phần mềm độc hại [5] và việc tái tạo mã nguồn có lỗ hổng [43].
* Phát hiện tương đồng mã nhị phân (BCSD) là nhiệm vụ xác định xem một cặp đoạn mã (tức là hàm assembly) có tương đồng khi mã nguồn gốc không có sẵn. Nhiệm vụ này đóng vai trò cơ sở cho một loạt các ứng dụng khác nhau, bao gồm phát hiện mã nhân bản [19, 20, 23, 24, 26, 40, 46, 63, 65, 79, 88, 92, 96, 97, 100], phát hiện phần mềm độc hại [1, 6, 8, 44, 53, 55, 98], phân loại gia đình phần mềm độc hại [38, 47, 59], khám phá lỗ hổng đã biết [10, 81, 93] và vá mã nguồn [29, 39, 94]. Tuy nhiên, việc suy luận về ý nghĩa ngữ cảnh gốc từ một mã nhị phân bị loại bỏ thông tin ngữ nghĩa là một vấn đề phức tạp vì ○1 quá trình biên dịch phức tạp loại bỏ rất nhiều thông tin ngữ nghĩa (ví dụ: tên biến, cấu trúc, kiểu dữ liệu, hệ thống lớp) với các biến đổi lớn, ○2 hình thức của các mã nhị phân (tương đương ngữ nghĩa) có thể thay đổi tùy theo trình biên dịch, phiên bản trình biên dịch, tối ưu hóa trình biên dịch, kiến trúc hoặc thậm chí là các kỹ thuật làm mờ mã, và ○3 việc chứng minh rằng hai chương trình tùy ý là tương đương chức năng là không thể giải quyết được (tức là Vấn đề Dừng [83]).
* Để giải quyết vấn đề BCSD từ tính chất động của quá trình sinh mã nhị phân như vậy, các công trình ban đầu áp dụng các phương pháp tĩnh với các thuật toán đa dạng, bao gồm phát hiện đồ thị đồng đẳng [3, 25, 101], thực thi biểu tượng [30, 63] hoặc phân tích luồng dữ liệu [19, 20, 74], giúp trích xuất thông tin hữu ích như chỉ thị, luồng điều khiển và giá trị dữ liệu. Một hướng khác sử dụng phương pháp động [9, 26, 45, 47, 66, 67, 73, 89] bằng cách quan sát việc thực thi mã nhị phân để hiểu rõ hơn về ngữ nghĩa của mã; tuy nhiên, nó thường gặp khó khăn về tính mở rộng kém và phạm vi mã không đầy đủ.
* Trong khi đó, với sự phát triển của các kỹ thuật máy học (ML), đã có nhiều phương pháp dựa trên ML được đề xuất [23, 24, 28, 60, 64, 65, 84, 92, 95, 100] trong lĩnh vực BCSD. Hơn nữa, những tiến bộ gần đây (ví dụ: học sâu) trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP) đã truyền cảm hứng cho việc suy luận mã máy có thể được máy hiểu được [23, 24, 65, 100], điều này đã cho thấy kết quả đáng chú ý lặp đi lặp lại. Ví dụ, InnerEye [100] xem xét một chỉ thị hợp ngữ, một khối cơ bản và một hàm như một từ, một câu và một đoạn văn, tương ứng. Gần đây, kiến trúc BERT [22] và các biến thể của nó [54, 61, 78] đã giúp tăng khả năng suy luận về ngữ cảnh cơ bản. Một số phương pháp [51, 57, 73, 88, 96] sử dụng BERT cho các nhiệm vụ hạ nguồn dòng (downstream tasks) khác nhau, bao gồm BCSD.
* Bản chất của vấn đề BCSD với các phương pháp dựa trên ML là học một phép đo tương đồng từ dữ liệu để mô hình xác định phù hợp mức độ tương đồng của một mẫu mới (chưa từng thấy). Vì số lượng mã nhị phân tương đồng ngữ nghĩa bị giới hạn trong quá trình học, học một phép đo tương đồng là một phương pháp thích hợp (ví dụ: mạng nơ-ron Siamese) cho một vấn đề phân loại 𝑁-way 𝑘-shot (tức là 𝑁 lớp với 𝑘 ví dụ cho mỗi lớp). Một số phương pháp trước đó [10, 57, 60, 64, 65, 73, 84, 88, 92, 95, 96, 100] sử dụng mạng nơ-ron Siamese [4, 11], mà tính toán một điểm tương đồng bằng một hàm khoảng cách thể hiện sự gần gũi của hai vectơ (ví dụ: nhúng chức năng). Tiến xa hơn, chúng tôi xem xét việc xác định mã có cùng ngữ nghĩa như một vấn đề phân loại 𝑁-way one-shot (tức là học một lần), một trường hợp đặc biệt (𝑘 = 1) của vấn đề phân loại 𝑁-way 𝑘-shot (tức là học vài lần trong §2). Đặc biệt, nó phù hợp với tình huống thực tế tìm kiếm sự hiện diện của một chức năng cụ thể trong cơ sở dữ liệu đã xây dựng trước.
* Trong bài báo này, chúng tôi đề xuất BinShot, một kiến trúc học tương đồng chuyển giao dựa trên BERT (với mạng nơ-ron Siamese) cho BCSD hiệu quả. Mặc dù khái niệm về BERT và Siamese phổ biến cho việc phát hiện tương đồng mã nhị phân, phát hiện của chúng tôi cho thấy cả hàm khoảng cách và hàm mất mát trong một kiến trúc có ảnh hưởng đáng kể đến hiệu suất của một mô hình. Đặc biệt, đóng góp chính của chúng tôi là gợi ý học một vectơ khoảng cách có trọng số với hàm mất mát entropy nhị phân (được gợi ý bởi Koch và cộng sự [50]), trong khi các mô hình hiện có chỉ học một khoảng cách duy nhất [10, 57, 60, 65, 73, 88, 92, 96, 100]. Tận dụng BERT, chúng tôi tiến hành đào tạo hai giai đoạn: tiền đào tạo để xây dựng một mô hình chung với mã hợp ngữ, tiếp theo là điều chỉnh lại (fine-tuning) để xây dựng một mô hình hạ nguồn cho nhiệm vụ BCSD. Sự áp dụng của BERT theo bản chất tăng cường tính ổn định của mô hình đối với các chức năng chưa từng thấy trước bằng cách học các mối quan hệ giữa mã hợp ngữ (tức là tiền đào tạo), sau đó điều chỉnh nó phù hợp với nhiệm vụ BCSD (tức là điều chỉnh lại). Lưu ý rằng một trong các hàm mất mát phổ biến là hàm mất mát tương phản [11] đã được sử dụng trong các phương pháp học trước đó [10, 57, 60, 65, 92, 100].
* Chúng tôi đã triển khai một nguyên mẫu đầy đủ của BinShot bao gồm bốn thành phần: ○1 bộ tiền xử lý để chuẩn bị cho quá trình học, ○2 máy tiền đào tạo để xây dựng một mô hình mã máy, ○3 bộ điều chỉnh lại để xây dựng một mô hình tương đồng mã, và ○4 bộ dự đoán để phát hiện tương đồng nhị phân với một đoạn mã cụ thể (tức là hàm). Chúng tôi xây dựng 1.400 mã nhị phân từ nguồn mã đa dạng, bao gồm khoảng 1,77 triệu hàm. Đánh giá của chúng tôi về cả hiệu suất và khả năng chuyển giao chứng tỏ rằng BinShot vượt qua các mô hình cơ sở hiện đại trước đây để phát hiện tương đồng mã nhị phân, bao gồm Gemini [92], Asm2Vec [23], PalmTree [57] và DeepSemantic [51]. Hơn nữa, chúng tôi đã thiết lập một tình huống thực tế để phát hiện các chức năng có lỗ hổng. Ngạc nhiên thay, các phương pháp trước đây [23, 51, 57, 92] tiết lộ độ chính xác khá thấp (< 15%) do số lượng dương tính sai lớn, khiến tính thực tế của chúng còn bị đặt dấu hỏi. Cuối cùng, chúng tôi minh họa cách BinShot có thể phân biệt một hàm nhị phân khỏi một hàm (không) tương tự.

Dưới đây là tóm tắt về đóng góp của bài báo của chúng tôi:

* Chúng tôi đề xuất BinShot học một vectơ khoảng cách có trọng số với hàm mất mát entropy nhị phân trên kiến trúc BERT Siamese cho BCSD.
* Chúng tôi thiết kế và triển khai nguyên mẫu của BinShot. Chúng tôi đã công khai mã nguồn của BinShot1 để thúc đẩy lĩnh vực BCSD trong tương lai.
* Chúng tôi đánh giá BinShot để trình bày hiệu suất, khả năng chuyển giao và tính thực tế thông qua việc so sánh nó với các mô hình cơ sở hiện đại.
* Chúng tôi chứng minh rằng BinShot có khả năng đại diện cho các nhúng tương tự tốt trong không gian vector thông qua việc trực quan hóa.

2 NỀN TẢNG

* BERT. BERT [22] là một kiến trúc máy học dựa trên transformer xuất phát từ lĩnh vực xử lý ngôn ngữ tự nhiên (NLP). Kiến trúc Transformer [86] đã đạt được sự phổ biến cao trong các nhiệm vụ xử lý văn bản với các kết quả đáng chú ý. Nó áp dụng ○1 cơ chế tự chú ý nhằm suy luận thông tin ngữ cảnh cho mỗi từ (xem xét một vị trí) trong câu đầu vào và ○2 thiết kế cho phép xử lý song song trong quá trình đào tạo trên một lượng lớn dữ liệu, làm giảm thời gian đào tạo so với việc xử lý tuần tự như Long Short-Term Memory (LSTM) [37]. Kiến trúc BERT sử dụng sự lựa chọn thiết kế cẩn thận từ bộ mã hóa của Transformer (ví dụ, tự chú ý đa đầu vào) vào các ứng dụng hạ nguồn khác nhau với hai giai đoạn chính (tức là tiền đào tạo và điều chỉnh lại). Đầu tiên, tiền đào tạo của BERT nhằm mục tiêu thu thập thông tin ngữ cảnh cả ở cấp từ vựng và cấp câu, xây dựng một mô hình chung chứa các vectơ nhúng của mỗi từ vựng. Một biểu diễn nhúng đầu vào cuối cùng xem xét một vị trí (ví dụ, vị trí của một từ trong một câu) và một phân đoạn (ví dụ, một câu mà một từ thuộc về) cũng như chính từ vựng đó. Để thu thập ngữ cảnh ẩn từ một câu, tiền đào tạo thực hiện hai nhiệm vụ phụ: mô hình ngôn ngữ ẩn (MLM) và dự đoán câu tiếp theo (NSP). MLM ngẫu nhiên ẩn đi một phần từ vựng trong một câu đã cho (ví dụ, 15% trong hệ thống BERT gốc), khai thác dữ liệu không được gán nhãn (tức là vị trí bị ẩn đi) để tạo ra các nhãn (tức là các từ vựng gốc). Ký tự đặc biệt [CLS] được đặt ở đầu mỗi đầu vào, trong đó [SEP] đại diện cho một phân tách từ vựng để hai câu có thể được nối lại thành một đầu vào duy nhất. Trong khi đó, NSP dự đoán xem câu tiếp theo có liên quan đến câu hiện tại hay không. Lưu ý rằng tiền đào tạo không yêu cầu một quá trình gán nhãn mất công (tức là học không giám sát), tạo ra một mô hình chung có thể được sử dụng như một tầng kiến thức cơ bản để xây dựng từ đó. Thứ hai, giai đoạn điều chỉnh lại của BERT cải thiện mô hình chung bằng cách đào tạo lại trọng số với một tập dữ liệu khác cho một nhiệm vụ được định nghĩa cụ thể của người dùng (tức là học có giám sát), cho phép điều chỉnh thành các biểu diễn ngữ cảnh theo nhiệm vụ. Trong bài báo này, chúng tôi xây dựng một mô hình BERT được tiền đào tạo cho biểu diễn mã nhị phân và một mô hình đã điều chỉnh lại cho BCSD như một nhiệm vụ hạ nguồn.

A diagram of a neural network

Description automatically generated

* Mạng nơ-ron Siamese. Một mạng nơ-ron Siamese [4] ban đầu được giới thiệu để xác minh chữ ký, trong đó nhiều mạng con đồng nhất trích xuất các đặc trưng từ nhiều đầu vào khác nhau trong quá trình đào tạo. Ý nghĩa cốt lõi của mạng nơ-ron Siamese là tính toán một hàm khoảng cách thể hiện cách hai vectơ đặc trưng gần nhau; tức là tập hợp các đầu vào tương tự nên được mã hóa sao cho chúng kề nhau càng gần càng tốt, trong khi đó, tập hợp các đầu vào không tương tự nên được đặt càng xa càng tốt. Không giống như các phương pháp truyền thống (ví dụ, máy vector hỗ trợ) yêu cầu có các nhãn đã biết trước thời gian đào tạo cho một vấn đề phân loại, kiến trúc này thích hợp cho những trường hợp có một số lượng lớn các danh mục không biết trước với một số lượng giới hạn các tập dữ liệu trong quá trình đào tạo [49]. Cụ thể hơn, chúng ta có thể định nghĩa hình thức một phép đo tương đồng (𝐹𝑊 ) giữa hai đầu vào (𝑋𝑖 và 𝑋𝑗
* ) bằng một hàm khoảng cách (𝐷), trong đó 𝐺𝑊 (𝑋) đại diện cho một vectơ đầu ra trong mạng nơ-ron được tham số hóa bằng trọng số (𝑊 ) như sau:
* 
* Hàm khoảng cách trong Phương trình 1 giữa hai vectơ 𝑛 chiều có thể thay đổi [68], bao gồm norm L1, norm L2 hoặc khoảng cách cosine. Phép đo tương đồng có thể được học bằng cách tìm 𝑊 sao cho giả thiết hàm mất mát nhỏ nhất trên một tập dữ liệu đào tạo. Với một hàm khoảng cách, một mạng nơ-ron Siamese xem các khoảng cách kết quả như mức độ tương đồng của hai đầu vào. Mạng nơ-ron Siamese phụ thuộc vào một hàm mất mát đã được định nghĩa trước cũng như một hàm khoảng cách được chọn từ đó. Dưới đây là ví dụ về một trong những hàm mất mát phổ biến (Hình 1), hàm mất mát tương phản (𝐿𝑐 trong Phương trình 2). Một hàm mất mát tương phản giả thiết khoảng cách nhỏ nhất cho các cặp tích cực (positive) và lớn nhất cho các cặp tiêu cực (negative). Lưu ý rằng 𝑌 đại diện cho một nhãn (0 hoặc 1) cho cặp đầu vào (𝑋𝑖, 𝑋𝑗).
* 
* Học ít dữ liệu (Few-shot Learning). Học ít dữ liệu khác biệt với học có giám sát cổ điển ở chỗ nó nhằm mục tiêu học cách học (ví dụ, hai đối tượng giống nhau) thay vì để một mô hình nhận diện trực tiếp một nhãn (ví dụ, một đối tượng là con chó). Ý tưởng chính [4, 11, 50, 80] bắt nguồn từ các nhiệm vụ thị giác máy tính, lấy cảm hứng từ khả năng của con người học cách nhận biết một đối tượng mới dựa trên thông tin đã được thu thập trước đó với một số lượng giới hạn các ví dụ. Nói chung, việc phân loại 𝑁 lớp với 𝑘 mẫu cho mỗi lớp (tức là một số lượng mẫu nhỏ) được biết đến là một vấn đề phân loại 𝑁-way 𝑘-shot. Đơn giản, tập dữ liệu được chia thành hai tập đối nghịch hoàn toàn: tập hỗ trợ (𝑁 × 𝑘) cho quá trình đào tạo và tập truy vấn cho quá trình kiểm tra, tiếp theo là học tương đồng giữa các vectơ đặc trưng. Một mạng nơ-ron Siamese là một trong những công cụ hỗ trợ học ít dữ liệu, điều này phù hợp tốt với BCSD. Lý tưởng, một mô hình học được từ các cặp hàm khác có thể nhận biết thành công khoảng cách từ một cặp hàm cụ thể (có thể là chưa từng thấy) với mô hình.

3 PHÁT HIỆN SỰ TƯƠNG TỰ TRONG MÃ NHỊ PHÂN

* Định nghĩa vấn đề và Phạm vi. Một nhiệm vụ BCSD được định nghĩa là xác định sự gần gũi của hai hoặc nhiều mã máy (tức là mã hợp ngữ) khi mã nguồn không khả dụng. Nhiệm vụ này có thể thay đổi tùy theo các khía cạnh khác nhau của so sánh mã. Ví dụ, độ tinh vi của một đoạn mã có thể là toàn bộ chương trình, hàm, khối cơ bản, hoặc thậm chí là lệnh. Người ta có thể xem nhiệm vụ phát hiện như một vấn đề tìm kiếm mã, trong đó số lượng đoạn mã có thể là một-một, một-nhiều, hoặc nhiều-nhiều. Hơn nữa, mã nhị phân có thể chứa ngữ nghĩa tương đương với các biến đổi mã đa dạng từ các trình biên dịch khác nhau, các tùy chọn/trình biên dịch/phiên bản tối ưu hóa khác nhau, kiến trúc khác nhau và sự làm rối. Trong khi đó, nó có thể đại diện cho ngữ nghĩa tương tự với các cập nhật khác biệt từ các phiên bản mã nguồn khác nhau hoặc các sửa lỗi (patch). Trong công việc này, chúng tôi xử lý việc phát hiện các đoạn mã tương tự mà sự biến đổi chỉ xuất phát từ sự kết hợp của trình biên dịch và mức tối ưu hóa khác nhau cho kiến trúc x86\_64. Chúng tôi tìm kiếm cặp hàm nhị phân tương tự từ một tập hàm đã chuẩn bị (thiết lập một-đa) trong thực tế.
* Những Thách thức Còn Mở. Tính động của quá trình sinh mã nhị phân đa dạng không thể tránh khỏi làm cho BCSD trở nên khá khó khăn. Không thể chứng minh rằng hai chương trình tùy ý là tương đương về chức năng và ngữ nghĩa (tức là Vấn đề Dừng [83]). Lưu ý rằng phương pháp luận của chúng tôi không phụ thuộc vào các biến đổi khác từ tùy chọn hoặc phiên bản trình biên dịch khác; tuy nhiên, chúng tôi cố tình không xem xét các sự khác biệt về ngữ nghĩa từ các phiên bản mã nguồn khác (ví dụ, cập nhật, sửa lỗi) do thiếu các chỉ số định lượng cho khoảng cách ngữ nghĩa. Một thách thức khác xuất phát từ các kỹ thuật làm rối mã mà thường làm mờ đi ngữ nghĩa gốc bằng cách đóng gói, chèn mã ngẫu nhiên, sử dụng máy ảo tùy chỉnh, v.v.

4.1 Tổng quan về BinShot

* Hình 2 minh họa toàn bộ luồng làm việc của BinShot, chủ yếu bao gồm bốn thành phần như sau:
* Bộ tiền xử lý. Với một tập hợp các mã nhị phân, chúng tôi chuẩn bị một tập dữ liệu có thể được đưa vào mô hình BERT để đào tạo (○1). Sử dụng một công cụ đảo ngược kỹ thuật, chúng tôi thu thập tất cả các lệnh đã được phân tích ngược. Tiếp theo, chúng tôi chuyển đổi một lệnh ngây thơ thành một dạng gọn gàng (Bảng 4 trong Phụ lục A), xây dựng cơ sở dữ liệu các hàm đã được chuẩn hóa (NFs) với các quy tắc chuẩn hóa được đề xuất bởi DeepSemantic [51].
* Bộ tiền đào tạo. Khi tất cả các NFs đã sẵn sàng, chúng tôi xây dựng một mô hình BERT chung với quá trình tiền đào tạo, có khả năng phát ra một vectơ nhúng cho mỗi hàm (○2). Lưu ý rằng một mô hình BERT chung học được mối quan hệ giữa các lệnh, điều này làm cho mô hình cuối cùng mạnh mẽ hơn đối với các hàm chưa từng thấy. Tương tự như các mô hình ngôn ngữ tiền đào tạo trong xử lý ngôn ngữ tự nhiên (NLP), mô hình này cho ngôn ngữ hợp ngữ cho phép tái sử dụng cho các nhiệm vụ cụ thể khác nhau.
* Bộ điều chỉnh. Tiếp theo, chúng tôi thực hiện việc đào tạo lại mô hình đã được tiền đào tạo (tức là điều chỉnh) để phù hợp với mục đích của chúng tôi; BCSD (○3). Để làm điều này, chúng tôi chuẩn bị một tập dữ liệu khác với các cặp hàm và nhãn tương ứng cho một nhiệm vụ BCSD. Mô hình được điều chỉnh dự đoán xem một hàm có tương tự với một hàm khác không.
* Trình dự đoán. Cuối cùng, bằng cách sử dụng BERT đã được điều chỉnh, chúng tôi xây dựng một cơ sở dữ liệu chứa tất cả các vectơ hàm của chúng tôi để so sánh thêm (○4). Một mô hình hạ tầng lấy cặp hàm làm đầu vào, tìm kiếm bất kỳ hàm tương tự nào với một loạt dự đoán.

A diagram of a computer program

Description automatically generated with medium confidence

4.2 Tiền xử lý: Chuẩn bị cho việc học

* Bước tiền xử lý bao gồm một quá trình chuyển đổi cần thiết để đưa tập hợp rộng các tệp nhị phân thực thi được thu thập (Xem Bảng 1) vào mạng nơ-ron, bao gồm việc phân mảnh và chuẩn hóa.
* Quá trình phân mảnh (Disassembly Process). Trước hết, chúng tôi thực hiện việc phân mảnh mọi lệnh trong một vùng mã của mỗi tệp thực thi bằng một công cụ đảo ngược tiên tiến (ví dụ, IDA Pro [35]). Chúng tôi đã tạo ra toàn bộ bộ sưu tập mã nhị phân với thông tin gỡ lỗi có sẵn (ví dụ, xác định ranh giới hàm) để huấn luyện một mô hình tốt hơn với quá trình phân mảnh chính xác. Lưu ý rằng chúng tôi giả định rằng mã nhị phân đã bị loại bỏ thông tin cho các bước kiểm tra tiếp theo.
* Chuẩn hóa lệnh cân đối. Đưa các lệnh máy tính vào mạng nơ-ron như chúng đang tồn tại sẽ gây ra vấn đề ngoài từ vựng (OOV) do số lượng các biểu thức có thể (tức là mã opcode, các toán hạng hoặc sự kết hợp của chúng) sẽ vô cùng lớn. Ví dụ, một giá trị trực tiếp bốn byte hoặc một địa chỉ tương đối như một toán hạng có thể chứa 2^32 - 1 (tức là khoảng bốn tỷ) biểu thức khác nhau. Một số lượng lớn OOV có thể dẫn đến việc thất bại trong việc tạo vectơ nhúng đầu vào có ý nghĩa. Liên quan đến điều này, chúng tôi áp dụng chiến lược chuẩn hóa lệnh cân đối được giới thiệu bởi DeepSemantic [51], chiến lược này đảm bảo sự cân bằng giữa việc biểu diễn mã nhị phân sao cho vẫn bảo tồn ngữ nghĩa gốc trong khi duy trì một lượng biểu thức hợp lý để tránh OOV (Độc giả quan tâm có thể tham khảo Phụ lục A).

4.3 Tiền đào tạo: Mô hình cho Hợp ngữ

* BinShot áp dụng một mô hình dựa trên BERT (nhưng không có NSP) theo cấu trúc huấn luyện hai bước; bước đầu tiên là tiền đào tạo (○2 trong Hình 2) có nhiệm vụ huấn luyện một mô hình chung cho mã hợp ngữ.
* Nhiệm vụ MLM (Masked Language Modeling). Chúng tôi sử dụng chiến lược giống như BERT gốc, thay thế 15% các biểu thức đầu vào (lệnh) bằng biểu tượng mask (tức là biểu thức [MASK]). Các tham số của nhiệm vụ MLM, 𝜃𝑚, được tối ưu hóa bằng cách giải quyết vấn đề tối ưu hóa sau:
* A close-up of a white background

  Description automatically generated
* trong đó 𝑡, 𝑇, 𝑦 và 𝑦ˆ lần lượt thể hiện một biểu thức, tập hợp các biểu thức, biểu thức gốc trước khi mask và biểu thức được dự đoán cho nhiệm vụ MLM. 𝐺𝑚 (𝑋) biểu thị vectơ đầu ra của một lớp liên kết đầy đủ từ một bộ phân loại MLM với hàm đầu vào 𝑋. Ở mức cao, nhiệm vụ MLM học cách dự đoán một biểu thức thích hợp để thay thế cho biểu thức bị mask, giúp bắt được ngữ cảnh giữa các lệnh riêng lẻ.
* Các lựa chọn thay thế cho nhiệm vụ NSP. Chúng tôi loại bỏ nhiệm vụ NSP khỏi kiến trúc BERT gốc vì, khác với một câu kế tiếp trong NLP, mối quan hệ giữa các hàm được xác định bởi một cuộc gọi hàm thay vì vị trí của chúng, khiến cho việc dự đoán hàm kế tiếp (tức là NSP) trở nên vô nghĩa. Mặc dù chúng tôi không trực tiếp xem xét một cuộc gọi hàm, quá trình chuẩn hóa lệnh cân đối ngầm hiệu quả với một cuộc gọi libc quan trọng bằng cách xác định nó là một từ riêng biệt. Nếu không, quá trình chuẩn hóa nhận biết cuộc gọi thư viện bên ngoài (ví dụ, trong toàn bộ PLT; bảng liên kết thủ tục, và GOT; bảng bù liên kết toàn cầu) hoặc cuộc gọi hàm nội bộ (ví dụ, các hàm khác được định nghĩa trong một tệp thực thi). Ngoài ra, nó xem cuộc gọi gián tiếp (ví dụ, gọi reg8) hoặc cuộc gọi đệ quy (ví dụ, gọi chính nó) là một lệnh phân biệt để tạo ra sự suy luận ngữ cảnh tốt hơn.

A screen shot of a computer

Description automatically generated

4.4 Fine-tuner: Mô hình cho Sự tương đồng mã

* Dựa trên một mô hình BERT chung với một lượng lớn các tệp nhị phân, chúng tôi xác định một nhiệm vụ phụ (○3 trong Hình 2); BCSD. Để làm được điều này, chúng tôi sử dụng một mạng nơ-ron Siamese vào một bộ phân loại, học cách tính một vectơ khoảng cách có trọng số từ một tập dữ liệu có nhãn (tức là (NF1, NF2, {0,1}) với 1 cho cặp tương tự và 0 cho một cặp không tương tự). Hình 3 minh họa kiến trúc Siamese cho bộ điều chỉnh tốt hơn trong BinShot.
* Phương trình dưới đây thể hiện cách tính một vectơ khoảng cách (𝐹 (𝑋𝑖, 𝑋𝑗)) giữa hai vectơ nhúng chức năng dựa trên mô hình BERT (𝐵𝐸𝑅𝑇 (𝑋)) trong đó 𝐷 đại diện cho hàm khoảng cách như sai số tương ứng (k=1) hoặc sai số bình phương (k=2).
* A math equation with black text

  Description automatically generated with medium confidence
* Sau đó, bộ phân loại nhị phân của chúng tôi học một vectơ khoảng cách có trọng số bằng hàm mất mát nhị phân cross entropy sau đây:
* A close-up of a number

  Description automatically generated
* trong đó Y là nhãn cho (𝑋𝑖, 𝑋𝑗). 𝐹𝐶(𝑋) = 𝑊𝑋 + 𝑏 biểu thị vectơ đầu ra của một lớp liên kết đầy đủ với trọng số (𝑊), và 𝜎 là hàm Sigmoid (𝜎(𝑧) = 1/(1+𝑒^𝑧)), có giá trị từ 0 đến 1 cho xác suất của sự tương đồng mã cho một cặp chức năng (𝑝(𝑋𝑖, 𝑋𝑗)).
* Hàm Mất Mát. Đáng chú ý là chúng tôi cẩn thận chọn một hàm mất mát thích hợp cho nhiệm vụ BCSD. Một trong các tình huống thông thường là tìm kiếm một chức năng cụ thể hoặc một loạt các chức năng cụ thể theo quan tâm của người dùng. Giả sử mỗi chức năng có một mẫu duy nhất, nhiệm vụ xác định mã tương tự ngữ nghĩa được điều chỉnh với một bài toán phân loại một lần một cách 𝑁 cách (𝑘 = 1) trong một bài toán phân loại 𝑁 cách 𝑘 lần (Xem §2). Thực chất, chúng tôi sử dụng học khoảng cách có trọng số với kiến trúc Siamese được đề xuất bởi học một lần [50], định nghĩa hàm mất mát là mất mát nhị phân cho phân loại nhị phân. Cùng theo đó, chúng tôi giải quyết một vấn đề tối ưu với hàm mất mát (Phương trình 6), tìm kiếm các tham số tối ưu hóa (𝜃𝑑) cho một mô hình phụ thuộc xuống. Trong khi đó, các mô hình BCSD dựa trên học dựa trên lớp học trước đó [10, 57, 60, 65, 92, 100] thường sử dụng mất mát tương phản 𝐿𝑐 [11]. Chúng tôi kiểm nghiệm theo kinh nghiệm rằng BinShot được trang bị với hàm mất mát của chúng tôi hoạt động tốt hơn cho suy diễn của một chức năng không nhìn thấy (Xem §6) so với các mô hình BCSD tiên tiến khác.

4.5 Dự đoán cho BCSD

* Với mô hình đã điều chỉnh tốt cho BCSD, một bộ dự đoán (○4 trong Hình 2) nhận một cặp chức năng làm đầu vào; một là chức năng mục tiêu để so sánh, và một là từ cơ sở dữ liệu. Lưu ý rằng một quy trình tìm kiếm có thể được tăng tốc bằng cách i) tiền tính toán nhúng chức năng trong cơ sở dữ liệu và ii) chèn các vectơ chức năng không nhìn thấy vào cơ sở dữ liệu. Khi bộ dự đoán thu thập hai vectơ chức năng với 𝐵𝐸𝑅𝑇 (𝑋𝑖) và 𝐵𝐸𝑅𝑇 (𝑋𝑗), nó tính một xác suất theo Phương trình 5 và Phương trình 7. BinShot được đào tạo để điều chỉnh đầu ra thành 0 (âm) hoặc 1 (dương) khi áp dụng hàm Sigmoid để có đầu ra cuối cùng. Do đó, chúng tôi đặt 𝑝 = 0.5 làm ngưỡng để xác định xem hai chức năng có tương tự hay không.

1. THỰC HIỆN

* Phân Tích Tĩnh Binary. Chúng tôi tận dụng một trong những công cụ kỹ thuật đảo ngược binary hàng đầu, IDA Pro 7.6 [35], để thực hiện quá trình giải tháo các binary và trích xuất thông tin tĩnh hữu ích từ chúng. Chúng tôi viết một đoạn mã với các API IDAPython tích hợp [34] có thể trích xuất danh sách các hàm lập trình assembly, các tham chiếu chéo (tức là đồ thị cuộc gọi), tên các phần, các tham chiếu chuỗi và các cuộc gọi thư viện bên ngoài, giúp dễ dàng thực hiện việc chuẩn hóa hướng dẫn sau này. Các công cụ đảo ngược binary khác như angr [2], Ghidra [70] hoặc radare2 [77] cũng đủ để phục vụ mục đích phân tích tĩnh của chúng tôi.
* Thực Hiện BinShot. Chúng tôi phát triển BinShot bằng PyTorch [76], một trong những framework phổ biến nhất cho Machine Learning. Tại trái tim của BinShot, chúng tôi thực hiện nguyên mẫu sử dụng BERT [22, 32, 41, 91] và một mạng neural Siamese [50]. Đối với hàm khoảng cách được mô tả ở §2, chúng tôi áp dụng một sai số bình phương tạo ra một vector khoảng cách như đầu ra vì thực nghiệm của chúng tôi cho thấy tổng thể hiệu suất với sai số bình phương hơi tốt hơn so với sai số tuyệt đối. Chúng tôi chọn một cách hoài nghi các siêu tham số sau đây cho thấy hiệu suất tốt nhất: 256 chiều cho nhúng hướng dẫn, 128 lớp ẩn, tám lớp chú ý và đầu vào (cho cấu trúc Transformer), độ dài tối đa của 256 mã thông báo và ba lớp mã hóa mã thông báo. Chúng tôi sử dụng trình tối ưu Adam [48] với tốc độ học là 0.0005 và thuật toán dropout [36] cho cả việc tiền huấn luyện và điều chỉnh tốt. Tỷ lệ dropout là 0.2 cho các lớp mã thông báo và 0.1 cho các lớp khác. Chúng tôi sử dụng một chiến lược tiếp tục tăng tốc học tập tuyến tính [62] và một chiến lược cắt gradient [72] cho kết quả huấn luyện ổn định. Số lượng tham số để xây dựng mô hình BinShot là 6,815,362. Chúng tôi huấn luyện BinShot trong 20 epochs với một kích thước batch là 32.

A screenshot of a graph

Description automatically generated

6. ĐÁNH GIÁ

* Các nghiên cứu trước về BCSD sử dụng các phương pháp dựa trên ML chủ yếu tập trung vào hiệu suất (tức là độ chính xác, điểm F1) và hiệu quả (tức là hiệu suất thời gian chạy). Trong bài báo này, chúng tôi cố gắng đánh giá BinShot từ khía cạnh tính thực tế bằng cách lập luận về quá trình tạo dữ liệu và tính hữu ích của kết quả bằng cách xem xét kỹ các trường hợp thú vị cũng như các phương pháp đo lường trước. Để thực hiện điều này, chúng tôi xác định hai tập kiểm tra khác nhau để so sánh BinShot với các mô hình cơ sở. Ngoài ra, chúng tôi cho thấy một nhóm vector nhúng hàm tương tự thực sự nằm gần nhau trong không gian bằng một kỹ thuật trực quan hóa (tức là t-SNE [85]).
* Môi trường. Chúng tôi đánh giá BinShot trên máy chủ trang bị hai bộ vi xử lý Intel Xeon Gold 6226R (có tổng cộng 32 lõi) chạy ở tốc độ 2.90 GHz, 256 GB RAM và hai thẻ GPU NVIDIA RTX A6000. Lưu ý rằng chúng tôi chỉ sử dụng một thẻ GPU để đánh giá hiệu suất.
* Các Phương pháp Đánh giá. Chúng tôi sử dụng các phương pháp đánh giá sau cho BinShot: độ chính xác (P), độ bao phủ (R), điểm F1 (F) và độ chính xác (A), trong đó 𝑇 𝑃, 𝐹𝑃, 𝑇 𝑁 và 𝐹𝑁 lần lượt là số trường hợp dương thực sự, sai dương, đúng âm và sai âm.
* A math equation with black text

  Description automatically generated with medium confidence

6.1 Tập dữ liệu

* Tập dữ liệu nhị phân. Bảng 1 tóm tắt tập dữ liệu của chúng tôi để đánh giá. Chúng tôi thu thập các ứng dụng đa dạng trong tập dữ liệu của mình: các tệp nhị phân thực thi từ các tiện ích GNU, SPEC2006, SPEC2017 và các chương trình thế giới thực theo lựa chọn của chúng tôi. Các tiện ích GNU bao gồm binutils (v2.26), coreutils (v8.30), diffutils (v2.8) và findutils (v4.7.0) đã được đánh giá rộng rãi trong các nghiên cứu trước đó [3, 9, 19–21, 23, 24, 26, 28, 51, 57, 60, 65, 73, 74, 89, 93, 100]. SPEC2006 và SPEC2017 cung cấp một loạt các bộ kiểm tra chuẩn về tính năng tiêu tốn CPU. Chúng tôi cũng thu thập 11 chương trình thế giới thực có sẵn là các dự án mã nguồn mở phổ biến trên Github, bao gồm BusyBox (v1.34.1) [7], Libgmp (v6.2.1) [31], ImageMagick (v7.0.10) [42], Libcurl (v7.78.0) [18], LibTomCrypt (v1.18.2) [58], OpenSSL (v1.1.1f) [71], SQLite (v3.30.1) [82], zlib (v1.2.8) [99], PuTTYgen (v0.76) [75], Nginx (v1.16.1) [69], và vsftpd (v3.0.3) [87]. Với hai trình biên dịch khác nhau (GCC v5.4, Clang v6.0.1) và bốn cấp tối ưu trình biên dịch khác nhau (O0-O3), chúng tôi đã tạo ra tổng cộng 1.400 tệp nhị phân thực thi.
* Cải tiến Tập dữ liệu. Chúng tôi cải tiến tập dữ liệu của mình (○3 trong Hình 4) trước khi tạo các cặp hàm tương tự và khác nhau bằng cách loại bỏ một số hàm cụ thể tùy theo kích thước của một hàm (tức là số lượng lệnh bên trong). Cụ thể, chúng tôi thiết lập một ngưỡng cho hai giá trị ngoại lai: hàm quá nhỏ khi số lượng lệnh nhỏ hơn hoặc bằng 𝑚, và hàm quá lớn khi số lượng lệnh lớn hơn 𝑛. Lý do đằng sau quyết định này là i) hàm quá nhỏ có thể không thường xuyên trả về kết quả có ý nghĩa về mặt ngữ nghĩa, ii) khả năng mục tiêu của một hàm (ví dụ: chứa lỗ hổng) quá đơn giản, và iii) BERT có giới hạn trong việc xử lý chuỗi quá dài (tức là hàm lớn) trong quá trình huấn luyện. Trong thí nghiệm của chúng tôi, chúng tôi đặt 𝑚 = 5 và 𝑛 = 250 trong BinShot, có thể điều chỉnh. Trong tập dữ liệu của chúng tôi, tỷ lệ hàm quá lớn và hàm quá nhỏ lần lượt là 4.66% và 16.47%.

A diagram of a data processing process

Description automatically generated

* Tạo dữ liệu huấn luyện. Hình 4 mô tả toàn bộ quy trình tạo tập dữ liệu (○2) từ tập dữ liệu nhị phân (○1), được sử dụng trong kiến trúc BERT. Mượn ý tưởng chiến lược chuẩn hóa hợp lý của các lệnh [51], tổng cộng 117.655.422 lệnh (từ 1.770.675 hàm) dẫn đến 18.449 từ vựng (§4.2). Vì mỗi từ vựng được xem xét như một token, chúng ta có 18.454 token, bao gồm năm token đặc biệt của BERT: [SOS]; bắt đầu của một hàm, [EOS]; kết thúc của một hàm, [UNK]; token không xác định, [MASK]; ký hiệu mask, và [PAD]; ký hiệu đệm. Nhớ lại rằng một bộ tiền huấn luyện xây dựng một mô hình BERT chung với tập dữ liệu đã qua xử lý trước (tức là các NFs). Khi quá trình chuẩn hóa lệnh hoàn thành, chúng tôi tạo ra một tập các cặp hàm tương tự và khác nhau để fine-tuning. Chúng tôi định nghĩa một cặp hàm là tương tự khi hai tên hàm giống nhau từ một tệp nhị phân được xây dựng bằng trình biên dịch hoặc cấp độ tối ưu hóa khác nhau. Chúng tôi loại bỏ các trường hợp mà cặp thân hàm giống nhau sau khi chuẩn hóa lệnh, vì điều này gây ra một khoảng cách bằng 0 trong mạng neural Siamese, gây trở ngại cho quá trình huấn luyện (ví dụ: Hình 8 trong Phụ lục A). Ví dụ, hàm openat\_safer (từ mkdir) biên dịch bằng gcc và O1 và hàm tương tự được biên dịch bằng clang và O2 sẽ giống nhau miễn là hai thân hàm sau khi chuẩn hóa là khác biệt. Chúng tôi sử dụng mẫu âm tiết (với tỷ lệ 1:1) cho các cặp hàm khác nhau, chọn các hàm khác nhau từ các tệp nhị phân khác nhau. Trong thí nghiệm của chúng tôi, chúng tôi tạo tổng cộng 5.259.310 cặp hàm (○4 trong Hình 4). Chúng tôi chia các cặp toàn bộ thành ba nhóm không chồng lấn cho tập dữ liệu huấn luyện, xác thực và thử nghiệm với tỷ lệ 8:1:1, trong khi duy trì nửa và nửa cho các cặp tương tự và khác nhau.
* OOV. Trong cài đặt của chúng tôi, có tổng cộng 18.280 token duy nhất trong tập dữ liệu huấn luyện và 16.166 token trong tập dữ liệu thử nghiệm. Chỉ có 64 token được phát hiện ở trong tập thử nghiệm (tức là không có trong tập huấn luyện), trong đó OOV chiếm 0.40% tổng số token.

A graph with black text

Description automatically generated

6.2 Hiệu suất:

* Trong phần này, chúng tôi đánh giá tính hiệu quả của BinShot so với các phương pháp tiếp cận BCSD trước đây sử dụng các kỹ thuật học sâu. Bảng 2 tóm tắt hàm khoảng cách, hàm mất mát và kiến trúc cho từng phương pháp, điều này có tác động lớn đến hiệu suất.
* Mô hình cơ sở. Như một mô hình cơ sở, chúng tôi đánh giá các mô hình BCSD tiên tiến của thời đại, bao gồm Gemini [92], Asm2Vec [23], PalmTree [57] và DeepSemantic [51]. Trong trường hợp của PalmTree, chúng tôi sử dụng mô hình đã được tiền huấn luyện và được công khai [56] để xây dựng một mô hình downstream với các tập dữ liệu của chúng tôi để so sánh. Chúng tôi tuân theo các siêu tham số từ các phiên bản gốc trừ khi có ghi chú khác.
* Biến thể BinShot. Để hiển thị tính hiệu quả của hàm mất mát, chúng tôi tạo thêm một biến thể của BinShot, gọi là BinShot-CTR, sử dụng một hàm mất mát tương phản (Phương trình 2) với khoảng cách L2 làm hàm khoảng cách (Phương trình 1).
* Ngưỡng tương tự. Chúng tôi chọn một ngưỡng dự đoán tương tự tối đa hóa F1 score trong trường hợp của Gemini [92], Asm2Vec [23], PalmTree [57] và BinShot-CTR (Phụ lục B).
* Đánh giá tập thử nghiệm. Nhớ lại rằng tập thử nghiệm (§6.1) chứa cùng tỷ lệ cặp hàm tương tự và khác nhau. Hình 5 thể hiện tính hiệu quả của BinShot, so sánh với bốn mô hình cơ sở (Gemini [92], Asm2Vec [23], PalmTree [57], DeepSemantic [51]) và biến thể của BinShot (BinShot-CTR). Chúng tôi đo lường độ chính xác, độ chính xác, độ phủ và F1 score với i) toàn bộ tập dữ liệu và ii) các tập dữ liệu được tách ra thành tất cả 36 kết hợp khác nhau giữa các trình biên dịch và cấp độ tối ưu hóa khác nhau (ví dụ, (gcc-O0, gcc-O0), (gcc-O0, clang-O0), . . . , (clang-O3, clang-O3)). Lưu ý rằng tập dữ liệu thứ hai được thiết kế để hiểu rõ hơn về tác động của các cấu hình khác nhau (tức là các trình biên dịch chéo, cấp độ tối ưu hóa chéo) đối với việc suy ra sự tương tự mã nguồn. Hình 5a rõ ràng cho thấy rằng BinShot vượt qua tất cả các mô hình tiên tiến khác cho BCSD. Tương tự, Hình 5b chứng minh rằng hiệu suất của BinShot duy trì ổn định với sự biến động thấp bất kể bất kỳ kết hợp nào của trình biên dịch và cấp độ tối ưu hóa. Chúng tôi giả định rằng việc học một vectơ khoảng cách có trọng số giúp BinShot vượt qua các mô hình khác vì BinShot có thể hiểu quan hệ giữa tất cả các yếu tố của vectơ khoảng cách, tạo điều kiện cho quá trình suy luận phức tạp. Trong khi đó, các mô hình khác đơn giản hóa mối quan hệ giữa hai hàm dựa trên một giá trị vô hướng (tức là khoảng cách từ vectơ đặc trưng của chúng). Cũng lưu ý rằng sự chênh lệch hiệu suất giữa BinShot và BinShot-CTR đến từ các hàm khoảng cách khác nhau chứ không phải từ các hàm mất mát.

A screenshot of a graph

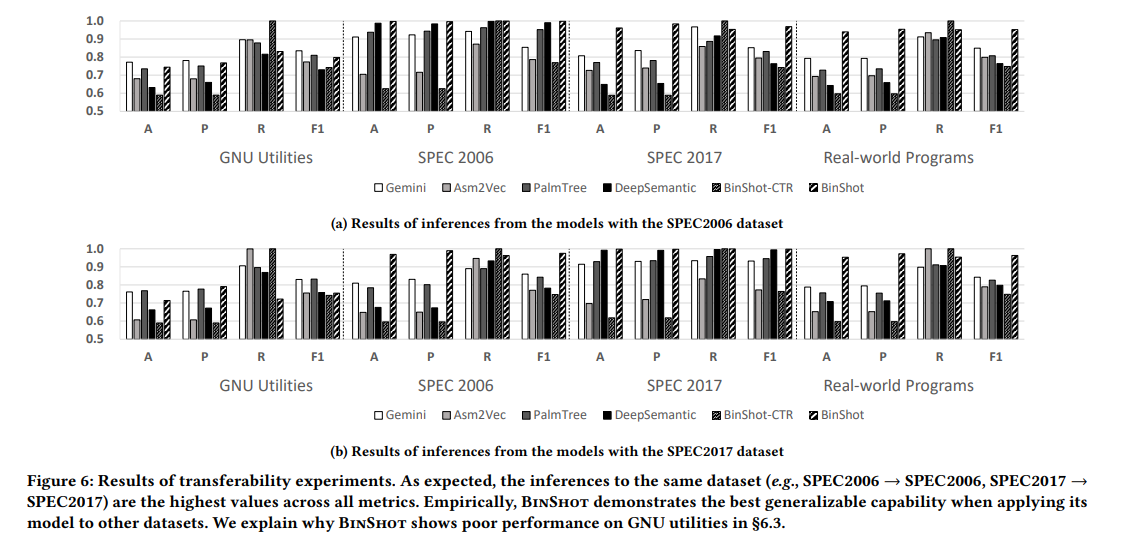
Description automatically generated

6.3 Khả năng chuyển giao

* Phần này mô tả các kết quả thực nghiệm về khả năng chuyển giao của mô hình; cụ thể, chúng tôi đặt ra các câu hỏi nghiên cứu sau: ○1 mô hình học từ tập dữ liệu 𝑋 có khả năng suy ra sự tương tự nhị phân trong tập dữ liệu 𝑌 ra sao? và ○2 mô hình tiên tiến nào hoạt động tốt nhất? Sự chuyển giao cao có nghĩa là mô hình có khả năng tổng quát hóa hoặc mở rộng. Lưu ý rằng thử nghiệm về khả năng chuyển giao của chúng tôi cố gắng trả lời rằng một mô hình cụ thể có thể có khả năng tổng quát hóa khi áp dụng nó vào một nhóm binary hoàn toàn khác biệt (thay vì phân tách thành các tập huấn luyện, xác nhận, thử nghiệm) trong thực tế.
* Thiết lập thử nghiệm. Chúng tôi tiến hành thử nghiệm khác sử dụng một tập dữ liệu nhị phân từ SPEC2006 và SPEC2017 (Bảng 1), xây dựng hai mô hình được điều chỉnh lại với BinShot vì chúng chứa số lượng hàm lớn nhất có thể được tổng quát hóa thêm. Tương tự, sử dụng Gemini [92], Asm2Vec [23], PlamTree [57], DeepSemantic [51] và BinShot-CTR, chúng tôi xây dựng mười hai mô hình khác nhau với cùng tập dữ liệu (SPEC2006, SPEC2017). Sau đó, chúng tôi cho từng mô hình suy ra sự tương tự với các tập dữ liệu không thấy (tức là GNU utilities, chương trình thế giới thực). Hình 6 tóm tắt kết quả suy luận của các mô hình khác nhau, bao gồm độ chính xác, độ chính xác, độ phủ và F1 score. Như thấy trong cả Hình 6a và Hình 6b, chúng tôi quan sát rằng BinShot chủ yếu đạt được hiệu suất tốt hơn so với các mô hình khác ngoại trừ GNU utilities. Nói cách khác, độ chính xác và F1 score của mô hình BinShot cao hơn so với bốn mô hình cơ sở và biến thể BinShot; ví dụ, F1 score cho chương trình thế giới thực (thanh bên phải trong Hình 6b) khi áp dụng học chuyển giao từ SPEC2017 cho thấy hiệu suất tốt nhất trong số tất cả.
* Kết quả. Chúng tôi thu được những hiểu biết sau từ các kết quả thực nghiệm của chúng tôi: ○1 một mô hình được điều chỉnh lại với BinShot có thể áp dụng cho các tập dữ liệu khác (không thấy) nói chung, cho thấy học khoảng cách có trọng số với mạng neural Siamese hoạt động theo đúng ý đồ, ○2 tập huấn luyện và kiểm tra được phân phối đều dựa trên Hình 6a và Hình 6b, giống với Hình 5b, và ○3 các hàm từ SPEC2006 không có nhiều hàm từ SPEC2017 bởi vì những suy luận song hành (ví dụ, mô hình dựa trên SPEC2006 đối với suy luận từ SPEC2017, mô hình dựa trên SPEC2017 đối với suy luận từ SPEC2006) không có nhiều sự khác biệt so với các suy luận cho các chương trình thế giới thực. Cuối cùng, BinShot cho thấy hiệu suất tương đối kém trên GNU utilities. Qua một cuộc điều tra cẩn thận, chúng tôi phát hiện ra rằng GNU utilities có khá nhiều hàm chia sẻ qua các binary khác nhau. Điều này bởi vì coreutils có một thư viện tĩnh (libcore.a) chung khi biên dịch các chương trình tiện ích khác nhau, chứa những hàm (đã được chuẩn hóa) giống nhau trong nhiều binary (Xem Phụ lục D). Lưu ý rằng kết quả của chúng tôi tương thích với quan sát từ Koo et al. [52].

6.4 Phát hiện Hàm Có Lỗ Vulnerable

* Trong phần này, chúng tôi thiết lập một kịch bản thực tế để phát hiện một hàm có lỗ trong một tệp nhị phân để thể hiện tính thực tiễn của BinShot.
* Thiết lập kịch bản. Bảng 3 hiển thị ba chương trình chứa chín hàm có lỗ cùng sáu lỗ và lỗ tiếp xúc (CVEs) phổ biến. Mỗi chương trình có tám biến thể được tạo ra bởi hai trình biên dịch khác nhau (tức là GCC v5.4, Clang v6.0.1) và bốn cấp tối ưu trình biên dịch khác nhau (tức là O0–O3). Chúng tôi sắp xếp ba giả định sau đây gần với một tình huống thực tế: ○1 cơ sở dữ liệu của các hàm có lỗ (được biên dịch bởi GCC) mà chúng tôi quan tâm đã được chuẩn bị; ○2 tệp truy vấn đã bị cắt bỏ, và được biên dịch bằng Clang; ○3 một người cố gắng tìm một hàm có lỗ trong một tệp truy vấn. Lưu ý rằng các phương pháp trước đây [9, 10, 21, 23, 26–28, 60, 65, 73, 74, 92, 93, 95, 100] chỉ truy vấn một hàm để xác định liệu nó có chứa lỗ hay không. Tuy nhiên, truy vấn một hàm có lỗ có thể gặp khó khăn về độ chính xác trong trường hợp có nhiều phản hồi dương sai (ví dụ, nói rằng tất cả các hàm đều có lỗ). Thay vào đó, chúng tôi cho phép người dùng tìm tất cả các hàm có lỗ bằng cách truy vấn toàn bộ tệp nhị phân để đạt được kết quả chính xác. Hơn nữa, một CVE có thể bao gồm nhiều hàm trong thực tế.



A close-up of a chart

Description automatically generated

* Chỉ số Đánh giá. Trong kịch bản trên, các cặp hàm tương tự hiếm hơn rất nhiều so với các cặp không tương tự. Không thể tránh khỏi, sự không cân đối này có thể làm méo mó một số chỉ số thậm chí cho một mô hình cực kỳ chính xác, như tỷ lệ nhớ và F1 score. Giả sử chúng ta có 𝑆 mẫu dương và 𝐷 mẫu âm, và một mô hình đạt đến 𝑇 𝑃 = 0.99 × � giả định rằng 𝑆 ≃ 0.003 × 𝐷. Khi đó, độ chính xác, nhớ và F1 score lần lượt là khoảng 0.23, 0.99 và 0.37. Như đã thể hiện, một lượng lớn mẫu âm có thể méo mó độ chính xác và F1 score ngay cả cho một mô hình chính xác, do đó chúng tôi chỉ bao gồm độ chính xác và nhớ để đánh giá với một tệp truy vấn.
* Kết quả. Chúng tôi đánh giá bốn mô hình cơ sở, BinShot và BinShot-CTR với 12 tệp truy vấn từ ba chương trình được biên dịch với Clang O0-O3. Bảng 3 tóm tắt kết quả phát hiện một hàm có lỗ cùng với các hàm trong cơ sở dữ liệu của chúng tôi. Lưu ý rằng chúng tôi loại bỏ một số hàm có lỗ (ví dụ, crypto\_recv, ctl\_putdata-O3, configure-O1) vì công cụ đảo ngược (IDA [35]) không thể nhận biết ranh giới của chúng trong một tệp nhị phân bị cắt bỏ. Thú vị là, với ngưỡng trong §6.2, tất cả các mô hình ngoại trừ DeepSemantic [51] đều phát hiện ra tất cả các hàm có lỗ trong các tệp truy vấn. Tuy nhiên, chỉ có BinShot đạt được độ chính xác cao (tức là trung bình 88,2%) trong khi tất cả các mô hình khác đều cho thấy độ chính xác khá thấp (tức là dưới 50%), điều này làm cho hiệu quả của chúng khi áp dụng chúng trong một kịch bản thực tế trở nên đáng ngờ. Điều này chủ yếu bởi vì ngay cả một dương sai (so với tất cả các so sánh trong cơ sở dữ liệu) trả về kết quả là một hàm có lỗ, làm giảm độ chính xác tổng thể. Ngoài ra, chúng tôi điều tra trường hợp của NTP trong Bảng 3; ranh giới của một hàm khác với thực tế, dẫn đến độ chính xác tương đối thấp (79,4%).

6.5 Hình ảnh hóa hàm nhúng

* Một trong những phương tiện phổ biến để hiểu một mô hình phân loại trên lượng lớn dữ liệu là thông qua hình ảnh hóa. Vì mô hình của chúng ta dựa trên phân loại nhị phân, nó đòi hỏi kiểm tra xem các nhúng cho một cặp hàm tương tự có đủ gần nhau trong khi những cái khác không giống nhau đủ xa. Tuy nhiên, việc biểu diễn mỗi vectơ trong một không gian chiều cao (ví dụ, 128 chiều trong thí nghiệm của chúng tôi) là khó khăn. Chúng tôi sử dụng phương pháp t-SNE (t-distributed Stochastic Neighbor Embedding) [85] cung cấp phép giảm chiều phi tuyến tính, trong đó dữ liệu tương tự được gom cách nhau trong không gian chiều thấp hơn. Như trong Hình 7, chúng tôi chọn năm chức năng khác nhau đã bị phân loại sai (tức là sai âm) bởi các mô hình cơ sở ở §6.2, mà BinShot đã phân loại đúng (tức là đúng âm). Lưu ý rằng Hình 7 hiển thị tám nhúng khác nhau cho mỗi hàm lắp ráp (với một biểu tượng duy nhất) vì mỗi chức năng đã được biên dịch với sự kết hợp của hai trình biên dịch và bốn cấp tối ưu hóa. Do đó, các vectơ biểu diễn các hàm tương tự dự kiến ​​được đặt ở gần nhau. Hình 7 hình ảnh một cách trực quan các vectơ biểu diễn các hàm tương tự được tạo ra bởi BinShot, chúng thực sự được tổ chức gom nhóm lại. Thú vị, các chức năng được hình ảnh hóa ngoại trừ hmac\_key (đánh dấu X màu xanh lá cây) có hai nhóm riêng biệt: một nhóm từ gcc và một nhóm từ clang.

6.6 Hiệu suất thời gian chạy

* Trong phần này, chúng tôi thực hiện hai thí nghiệm để đánh giá hiệu suất thời gian chạy của BinShot và các mô hình cơ sở khác cho việc suy ra sự tương đồng nhị phân (tức là phân loại nhị phân của một cặp chức năng đã cho). Trước tiên, chúng tôi đã lấy mẫu 16.595 cặp chức năng (0,28%; mức tin cậy 99%) từ tất cả 5.884.690 cặp bao gồm tập huấn luyện, xác nhận và kiểm tra (§6.1). Kết quả là, Gemini [92], Asm2Vec [23], PalmTree [57], DeepSemantic [51], BinShot-CTR và BinShot mất lần lượt 0,10, 81,94, 1,33, 1,34, 1,30 và 1,32 ms (miligiây) cho mỗi cặp chức năng, tương ứng. Ngoại trừ Gemini (thời gian chạy nhanh nhất) và Asm2Vec (thời gian chạy chậm nhất), tất cả các mô hình còn lại đều tương tự nhau. Tiếp theo, chúng tôi giả định kịch bản (§4.5) khi ai đó cố gắng tìm xem bất kỳ chức năng nào trong một chương trình mục tiêu có một chức năng tương tự trong cơ sở dữ liệu chứa 100 hàm nhúng của chúng tôi. Chúng tôi đã xây dựng trước cơ sở dữ liệu lưu trữ một chuỗi (NF, nhúng NF) như một cặp (khóa, giá trị) để tra cứu nhanh. Sau đó, chúng tôi thu thập 823 chức năng (sau quá trình tinh chỉnh theo §6.1) từ thư viện OpenSSL như một mục tiêu. Với các thiết lập trên, chúng tôi đo thời gian chạy cho cả việc tạo nhúng chức năng (tức là 823) và suy ra sự tương tự (tức là 82.300 = 100 × 823) với năm phương pháp cơ sở, bao gồm BinShot. Lưu ý rằng việc tạo nhúng chế độ ưu tiên hơn các so sánh bằng cách tra cứu. Kết quả là, Gemini [92], Asm2Vec [23], PalmTree [57], DeepSemantic [51], BinShot-CTR và BinShot mất lần lượt 1,16, 6.743,66, 29,03, 1,51, 1,45 và 1,54 giây, tương ứng. Trong khi Gemini [92] xếp hạng đầu, BinShot hiển thị hiệu suất thời gian chạy tương đối so với các mô hình còn lại.

7 CÔNG TRƯỞNG LIÊN QUAN

* Lĩnh vực về Phát hiện Sự tương đồng trong Mã nhị phân (BCSD) đã có rất nhiều nghiên cứu liên quan [33]. Chúng tôi phân loại các nghiên cứu trước đây thành ba loại: phương pháp tĩnh [1, 3, 19–21, 25, 27, 30,46, 55, 63, 74, 93, 101], phương pháp động [9, 26, 45, 47, 66, 67, 73, 89], và

phương pháp dựa trên học máy (ML) [10, 23, 24, 28, 51, 57, 60, 64, 65, 73, 84, 92, 95, 96, 100].

* Phương pháp Tĩnh. Trong thời kỳ của các kỹ thuật không dựa trên học máy, các phương pháp tĩnh chính trong lĩnh vực BCSD sử dụng phát hiện đồ thị đồng cấu [3, 25, 101], thực thi biểu tượng [30, 63], phân tích luồng dữ liệu [19, 20, 74], hoặc các phương pháp khác [1, 21, 27, 46, 55, 93]. BinDiff [25, 101] sử dụng phát hiện đồ thị đồng cấu trên cả đồ thị cuộc gọi và đồ thị luồng kiểm soát (CFG), và sau đó BinSlayer [3] cải tiến BinDiff bằng cách bổ sung thuật toán Hungarian cho việc ghép đồ thị hai phía. BinHunt [30] và CoP [63] sử dụng thực thi biểu tượng tĩnh để hiểu ý nghĩa của các mã nhị phân. Cụ thể, BinHunt sử dụng chứng minh định lý để trích xuất ý nghĩa của mã nhị phân, trong khi CoP tìm kiếm dãy con chung dài nhất để tính độ tương đồng. Lee et al. [55] xác định sự tương đồng của một cặp mục tiêu bằng cách sử dụng thông tin cấu trúc của đồ thị cuộc gọi và các n-gram của hướng dẫn nhớ mã lệnh. Tracelet [21] chia đồ thị luồng kiểm soát (CFG) của một hàm thành các dãy con cố định với chiều dài thực thi để đo sự tương đồng với các dãy con đó. Pewny et al. [74] đề xuất một công cụ thu thập cặp đầu vào/đầu ra (I/O) bằng cách cung cấp đầu vào ngẫu nhiên cho một đoạn mã. Sử dụng cặp I/O này, nó lấy giá trị băm đại diện cho một khối cơ bản, sau đó thực hiện việc ghép đồ thị với các giá trị đại diện. OpSeq [1] tạo chữ ký của mã độc Android bằng cách phân tích dãy mã lệnh chuẩn hóa trong một mô-đun chức năng nhạy cảm. discovRE [27] giới thiệu một phương pháp so sánh tương đồng nhanh bằng cách lọc các đặc trưng số học và cấu trúc từ các đặc trưng đã được tạo trước trước khi khớp chức năng. ESH [19] và GitZ [20] phân rã mã mục tiêu thành các đoạn nhỏ thông qua phân tích luồng dữ liệu, áp dụng lý thuyết thống kê để tiến xa hơn trong BCSD. BinXary [93] phân tích chữ ký của một bản vá bằng cách so sánh một chương trình có lỗ hỏng với một chương trình đã được vá. Trong khi đó, BinKit [46] phát hành bộ thử nghiệm công cộng quy mô lớn đầu tiên cho phép người dùng sao chép một nhiệm vụ BCSD cùng với công cụ KitNib [46] để xây dựng thực thể mẫu.
* Phương pháp Động. Các phương pháp động cho BCSD nhằm mục đích phân tích ý nghĩa của các mã nhị phân bằng cách chạy các tập tin thực thi [9, 26,
* 45, 47, 66, 67, 73, 89]. Các kỹ thuật khác nhau đã được giới thiệu, chẳng hạn như so sánh cặp đầu vào/đầu ra [9, 26, 45, 89], thực thi biểu tượng động [66, 67], và học ý nghĩa của các đường thực thi với mạng nơ-ron [73]. BLEX [26] thực thi hai chức năng mục tiêu với các giá trị đầu vào được chọn mẫu ngẫu nhiên để so sánh hành vi của chúng do các đầu vào gây ra. BinSim [67] sử dụng các cuộc gọi hệ thống hoặc cuộc gọi API để tách các đoạn mã tương ứng và sau đó kiểm tra tính tương đương của chúng bằng thực thi biểu tượng. Kim et al. [47] đề xuất một phép tính tương đồng dựa trên việc khớp chức năng với cả phân tích tĩnh (chẳng hạn CFG, N-gram) và phân tích động (chẳng hạn dấu vết hướng dẫn). TREX [73] sử dụng một mô hình mạng nơ-ron, được huấn luyện với các dấu vết thực thi và mã hợp ngữ đa dạng. Tuy nhiên, các phương pháp động đòi hỏi một nhiệm vụ tốn thời gian và phiền toái để thực thi các tập tin thực thi nhiều lần, hạn chế khả năng mở rộ của chúng.
* Phương pháp Tĩnh Dựa trên Học Máy. Trong thập kỷ qua, khi lĩnh vực Học Máy đã được phát triển đáng kể, việc áp dụng các kỹ thuật dựa trên Học Máy vào BCSD [10, 23, 24, 28, 60, 64, 65, 84, 92, 95, 100] đã trở nên phổ biến. Để học các đặc trưng của một chức năng từ một đồ thị cuộc gọi hàm (CFG), Genius [28] sử dụng một thuật toán di truyền và phân cụm phổ. Trong khi đó, Gemini [92] và VulSeeker [10] sử dụng một kiến trúc Siamese dựa trên mạng nhúng đồ thị. Asteria [95] sử dụng cấu trúc cây cú pháp trừu tượng (AST) của chức năng mục tiêu từ việc phân tích dựng lại, sau đó huấn luyện một mô hình dựa trên Tree-LSTM dựa trên một kiến trúc Siamese. Với mô hình và tập hợp các hàm được gọi từ mục tiêu, Asteria đạt được một điểm tương đồng. 𝛼Diff [60] sử dụng một phương pháp khác sử dụng dãy byte gốc của các chức năng để huấn luyện một kiến trúc Siamese dựa trên mạng nơ-ron tích chập. Gần đây, sự tiến bộ trong kỹ thuật xử lý ngôn ngữ tự nhiên (NLP) đã kích hoạt các ứng dụng [23, 24, 65, 100] trong lĩnh vực ngôn ngữ máy tính sử dụng nhiều kỹ thuật khác nhau để huấn luyện ngôn ngữ lập trình hợp ngữ. Asm2Vec [23] và DeepBinDiff [24] thực hiện học không giám sát bằng cách huấn luyện bối cảnh của các hướng dẫn. InnerEye [100] và SAFE [65] sử dụng một mạng Siamese dựa trên NLP. Công việc gần nhất với BinShot có thể là công trình của Luo et al. [64] và BinDiffNN [84], nơi họ sử dụng một phương pháp học một vector khoảng cách có trọng số. Tuy nhiên, Luo et al. [64] trích xuất thủ công các đặc trưng cho việc nhúng chức năng, trong khi BinDiffNN [84] sử dụng mạng nơ-ron dựa trên attention. Hơn nữa, khác với chúng tôi, họ không thảo luận kỹ về sự khác biệt giữa kiến trúc Siamese của họ với các công trình trước đó.

A screenshot of a computer screen

Description automatically generated

* Ứng dụng của BERT. Với sự ra đời của BERT [22], có một xu hướng nghiên cứu nổi bật trong lĩnh vực Xử lý Ngôn ngữ Tự nhiên với các mô hình ngôn ngữ dựa trên transformer [54, 61, 78]. Sự phổ biến của BERT đã tiếp cận vào lĩnh vực phân tích mã nhị phân bằng cách xây dựng một mô hình ngôn ngữ hợp ngữ đại diện [51, 57, 73, 88, 96]. Yu et al. [96], TREX [73] và jTrans [88] tập trung hoàn toàn vào BCSD, trong khi các nghiên cứu khác [51, 57] sử dụng mô hình dựa trên BERT cho các nhiệm vụ thứ cấp khác nhau. Yu et al. [96] thu thập một mô hình BERT được huấn luyện trước với thông tin được trích xuất từ CFG, tạo ra các nhúng mã thông tin ngữ nghĩa và các nhúng khối, sau đó dự đoán xem hai tập tin nhị phân có tương tự nhau hay không. TREX [73] sử dụng MLM để học các giá trị động với micro-traces (một hình thức dấu vết động thiếu ràng buộc) của một chức năng và sau đó điều chỉnh mô hình thứ cấp để thực hiện nhiệm vụ tương đồng chức năng của tập tin nhị phân. jTrans [88] đề xuất một BERT được điều chỉnh để đại diện cho các chỉ thị nhảy để nhận biết đích của các nhảy. Tác giả sử dụng hàm mất mát triplet [80], một hàm mất mát phổ biến khác trong mạng nơ-ron Siamese, để huấn luyện mô hình thứ cấp của họ. Trong khi đó, DeepSemantic [51] và PalmTree [57] áp dụng một mô hình tổng quát cho ngôn ngữ hợp ngữ trong các nhiệm vụ khác. DeepSemantic [51] thực hiện hai nhiệm vụ thứ cấp: BCSD và phân loại cho một trình biên dịch hoặc mức tối ưu. Tương tự, PalmTree [57] áp dụng mô hình tổng quát cho năm nhiệm vụ thứ cấp: phát hiện ngoại lệ, tìm kiếm khối cơ bản, BCSD, suy luận chữ ký loại chức năng và phân tích tập giá trị. Chúng tôi sử dụng mô hình dựa trên BERT cho BinShot; tuy nhiên, chúng tôi tập trung nhiều hơn vào cách huấn luyện mạng nơ-ron Siamese để cải thiện BCSD.

8 KẾT LUẬN

* Trong bài báo này, chúng tôi đề xuất BinShot, một mô hình dựa trên kiến thức BERT với cấu trúc Siamese để phát hiện sự tương tự mã nhị phân thông qua việc học một vectơ khoảng cách có trọng số. Chúng tôi sử dụng BERT để học ngữ nghĩa của ngôn ngữ hợp ngữ và sử dụng hàm mất mát nhị phân của mã nhị phân để cải thiện kết quả. Chúng tôi đã triển khai mẫu thử của BinShot, kết quả cho thấy BinShot mạnh mẽ hơn so với các mô hình tối ưu hiện có trong BCSD. Kết quả thử nghiệm của chúng tôi chứng tỏ hiệu suất của BinShot vượt trội hơn so với các phương pháp tiên tiến, thể hiện tính hiệu quả và khả thi của nó.
* ACKNOWLEDGMENTS
* Chúng tôi xin cảm ơn các nhà phê bình vô danh và người hướng dẫn của chúng tôi, Aisha Ali-Gombe, vì những đóng góp xây dựng. Công việc này được hỗ trợ bởi Quỹ Nghiên cứu Quốc gia Hàn Quốc (NRF), được tài trợ bởi chính phủ Hàn Quốc (MSIT) (NRF-2020R1A2B5B03095204) và Chương trình Nghiên cứu Khoa học Cơ bản thông qua khoản tài trợ từ NRF do Bộ Giáo dục của Chính phủ Hàn Quốc tài trợ (Số 2022R1F1A107437311). Ngoài ra, nó cũng được hỗ trợ một phần bởi Viện Kế hoạch & Đánh giá Công nghệ Thông tin & Truyền thông (IITP), được tài trợ bởi chính phủ Hàn Quốc (MSIT) (Số 2020-0-01840; Phân tích kỹ thuật truy cập và thu thập dữ liệu người dùng trên điện thoại thông minh, Số 2022-0-01199; Trường Cao đẳng Chuyển đổi An ninh (Đại học SungKyunKwan), Số 2022-0-00688; Nền tảng Trí tuệ Nhân tạo thích ứng hoàn toàn và phản ánh các thay đổi Chính sách Riêng tư), MSIT (Bộ Khoa học và Công nghệ Thông tin), Hàn Quốc, theo chương trình hỗ trợ Trung tâm Nghiên cứu Công nghệ Thông tin (IITP) (IITP-2022-2020-0-01602) được giám sát bởi IITP, Trung tâm Nghiên cứu Bán dẫn Liên đại học (ISRC) và Chương trình BK21 FOUR của Chương trình Giáo dục và Nghiên cứu Cho những Người tiên phong Công nghệ Thông tin trong Tương lai, Đại học Quốc gia Seoul vào năm 2022. Mọi ý kiến, kết quả và kết luận hoặc đề xuất được thể hiện trong tài liệu này là của tác giả và không nhất thiết phản ánh quan điểm của nhà tài trợ.