jTrans: Jump-Aware Transformer for Binary Code Similarity Detection.

jTrans: Jump-Aware Transformer cho việc phát hiện sự tương đồng trong mã nhị phân.

1. Mở đầu:

* Phát hiện sự tương đồng trong mã nhị phân (BCSD) có ứng dụng quan trọng trong nhiều lĩnh vực như phát hiện lỗ hổng, phân tích thành phần phần mềm và đảo ngược mã. Các nghiên cứu gần đây đã chỉ ra rằng mạng neural sâu (DNN) có thể hiểu các hướng dẫn hoặc biểu đồ dòng điều khiển (CFG) của mã nhị phân và hỗ trợ BCSD. Trong nghiên cứu này, chúng tôi đề xuất một phương pháp mới dựa trên Transformer, gọi là jTrans, để học các biểu diễn của mã nhị phân. Đây là giải pháp đầu tiên nhúng thông tin dòng điều khiển của mã nhị phân vào các mô hình ngôn ngữ dựa trên Transformer, bằng cách sử dụng một biểu diễn mới có ý thức về các điểm nhảy trong các đoạn mã đã phân tích và một nhiệm vụ tiền huấn luyện được thiết kế mới. Ngoài ra, chúng tôi đã phát hành một tập dữ liệu lớn mới với các mã nhị phân, gọi là BinaryCorp, được đa dạng nhất đến thời điểm hiện tại. Kết quả đánh giá cho thấy jTrans vượt trội hơn so với các phương pháp tiên tiến hiện có trên tập dữ liệu khó khăn hơn này lên đến 30,5% (từ 32,0% lên đến 62,5%). Trong nhiệm vụ thực tế tìm kiếm lỗ hổng đã biết, jTrans đạt được tỷ lệ nhớ cao hơn gấp đôi so với các phương pháp tiên tiến hiện có.

1. GIỚI thiệu:

* Phát hiện sự tương đồng mã nhị phân (BCSD), có thể xác định mức độ tương đồng giữa hai đoạn mã nhị phân, là một kỹ thuật cơ bản hữu ích cho một loạt ứng dụng, bao gồm phát hiện lỗ hổng đã biết [8-11, 18, 20, 21, 24, 33, 40, 49, 50, 55, 59], phát hiện và phân nhóm mã độc hại [4] [30, 31, 37], phát hiện vi phạm bản quyền phần mềm [41, 42, 52], phân tích các bản vá [32, 35, 60] và phân tích chuỗi cung ứng phần mềm [27].
* Trước khi sử dụng học máy trong lĩnh vực này, các giải pháp BCSD truyền thống phụ thuộc nặng vào các đặc trưng cụ thể của mã nhị phân, ví dụ như các đồ thị dòng điều khiển (CFG) của các hàm, để thu thập kiến thức cú pháp của các chương trình. Các giải pháp như BinDiff [64], BinHunt [23] và iBinHunt [46] sử dụng các kỹ thuật đồ thị đồng cấu để tính toán sự tương đồng của các CFG của hai hàm. Tuy nhiên, phương pháp này không chỉ tốn thời gian mà còn dễ thay đổi, vì CFG có thể thay đổi dựa trên tối ưu hóa trình biên dịch. Các nghiên cứu như BinGo [5] và Esh [8] đạt được tính ổn định cao hơn đối với các thay đổi CFG bằng cách tính toán sự tương đồng của các đoạn CFG. Tuy nhiên, các phương pháp này dựa trên các đặc trưng được tạo thủ công, gặp khó khăn trong việc thu thập ngữ nghĩa chính xác của mã nhị phân. Kết quả là, các giải pháp này thường có độ chính xác tương đối thấp.
* Với sự phát triển nhanh chóng của kỹ thuật học máy, hầu hết các giải pháp BCSD tiên tiến hiện tại đều dựa trên học máy. Nói chung, các giải pháp này nhúng mã nhị phân mục tiêu (ví dụ: các hàm) thành các vector và tính toán sự tương đồng của các hàm trong không gian vector. Một số giải pháp, chẳng hạn như Asm2Vec [14] và SAFE [43], mô hình ngôn ngữ hợp ngữ (của mã máy) bằng các mô hình ngôn ngữ được lấy cảm hứng từ xử lý ngôn ngữ tự nhiên (NLP). Các nghiên cứu khác sử dụng mạng neural đồ thị (GNNs) để học biểu diễn của các đồ thị dòng điều khiển (CFGs) và tính toán sự tương đồng của chúng [59]. Một số nghiên cứu kết hợp cả hai phương pháp, học biểu diễn của các khối cơ bản bằng các kỹ thuật NLP và xử lý tiếp các đặc trưng khối cơ bản trong một CFG bằng GNN, chẳng hạn như [44, 62]. Mặc dù hiệu suất của chúng đã được cải thiện, nhưng các phương pháp hiện có vẫn có một số hạn chế.
* Thứ nhất, việc mô hình hóa hợp ngữ dựa trên NLP chỉ xem xét thứ tự tuần tự của các chỉ thị và mối quan hệ giữa chúng; thông tin về việc thực thi chương trình thực tế (ví dụ: luồng điều khiển) không được xem xét. Kết quả là, các phương pháp chỉ dựa vào NLP sẽ thiếu hiểu biết ngữ nghĩa về các mã nhị phân đã được phân tích, và cũng không thích ứng tốt với các thay đổi có thể đáng kể trong mã nguồn là kết quả của tối ưu trình biên dịch.
* Thứ hai, dựa hoàn toàn vào CFG (Control Flow Graphs) bỏ qua ý nghĩa ngữ nghĩa của các lệnh trong mỗi khối cơ bản. Genius [21] và Gemini [59] đề xuất mở rộng CFG bằng các đặc trưng được trích xuất thủ công (ví dụ: số lượng lệnh). Tuy nhiên, các đặc trưng như vậy vẫn chưa đủ để đầy đủ hiểu được ngữ nghĩa của mã. Hơn nữa, những giải pháp này thông thường sử dụng GNN (Graph Neural Networks) để xử lý CFGs, nhưng GNN chỉ bắt được thông tin cấu trúc. GNNs cũng được biết đến là khá khó đào tạo và áp dụng song song, điều này hạn chế sự ứng dụng của chúng trong thế giới thực.
* Thứ ba, các bộ dữ liệu mà các giải pháp hiện tại được đào tạo và đánh giá trên không đủ lớn và/hoặc đa dạng. Do thiếu một bộ kiểm tra đánh giá lớn chung, mỗi nghiên cứu tạo ra bộ dữ liệu riêng của mình, thường từ các kho lưu trữ nhỏ như GNUtils, coreutils và openssl. Những bộ dữ liệu nhỏ này có các mẫu mã code tương tự nhau và do đó thiếu sự đa dạng, điều này có thể dẫn đến các mô hình bị động và một ấn tượng sai lệch về hiệu suất cao. Hơn nữa, việc đánh giá các giải pháp hiện tại thường không phản ánh các trường hợp sử dụng trong thế giới thực. Đa số các nghiên cứu không tiến hành thí nghiệm trên một lượng lớn các hàm ứng cử viên, điều này phổ biến trong thế giới thực. Dưới điều kiện thực tế hơn, hiệu suất của nhiều giải pháp SOTA giảm đáng kể, như chúng tôi thể hiện trong kết quả thí nghiệm của chúng tôi ở Phần 6.
* Trong bài báo này, chúng tôi giới thiệu jTrans, một mô hình mới dựa trên Transformer được thiết kế để giải quyết những vấn đề đã nêu trên và hỗ trợ việc phát hiện tương đồng nhị phân trong thế giới thực. Chúng tôi kết hợp các mô hình xử lý ngôn ngữ tự nhiên (NLP), giúp nắm bắt ngữ nghĩa của các lệnh, cùng với CFGs, giúp nắm bắt thông tin luồng điều khiển, để suy luận về biểu diễn của mã nhị phân. Vì công trình trước đó [62] đã chỉ ra rằng việc kết hợp đơn giản các đặc trưng dựa trên NLP và GNN không đạt được kết quả tối ưu, chúng tôi đề xuất kết hợp thông tin luồng điều khiển vào kiến trúc Transformer. Chúng tôi tin rằng đây là lần đầu tiên được thực hiện.
* Chúng tôi điều chỉnh kiến trúc Transformer để nắm bắt thông tin luồng điều khiển, bằng cách chia sẻ các tham số giữa việc nhúng mã thông báo và việc nhúng vị trí cho mỗi điểm nhảy của các lệnh. Chúng tôi sử dụng các tác vụ học không giám sát để tiền đào jTrans, nhằm học ngữ nghĩa của các lệnh và thông tin luồng điều khiển của các chức năng nhị phân. Tiếp theo, chúng tôi điều chỉnh lại jTrans đã tiền đào để phù hợp với các chức năng có ý nghĩa tương tự về ngữ nghĩa. Lưu ý rằng phương pháp của chúng tôi có khả năng kết hợp các đặc trưng từ mỗi khối cơ bản bằng cách sử dụng các mô hình ngôn ngữ mà không cần dựa vào GNNs để duyệt qua CFG tương ứng.
* Ngoài phương pháp mới của chúng tôi, chúng tôi giới thiệu một bộ dữ liệu lớn và đa dạng, được gọi là BinaryCorp, được trích xuất từ các kho lưu trữ chính thức của ArchLinux [2] và Arch User Repository (AUR) [3]. Bộ dữ liệu mới của chúng tôi cho phép giảm thiểu tình trạng overfitting và thiếu sự đa dạng mà đặc trưng cho các bộ dữ liệu hiện có. Chúng tôi tự động thu thập tất cả các dự án c/c++ từ các kho lưu trữ này, bao gồm phần lớn các phần mềm mã nguồn mở phổ biến, và xây dựng chúng với các tối ưu hóa trình biên dịch khác nhau để tạo ra các tệp nhị phân khác nhau. Theo kiến thức tốt nhất của chúng tôi, đây là bộ dữ liệu chương trình nhị phân lớn nhất và đa dạng nhất cho các nhiệm vụ BCSD (Binary Code Similarity Detection) đến nay.
* Chúng tôi triển khai một mô hình nguyên mẫu của jTrans và đánh giá nó trên các vấn đề BCSD trong thế giới thực, trong đó chúng tôi chỉ ra rằng jTrans vượt trội đáng kể so với các giải pháp SOTA, bao gồm Gemini [59], SAFE [43], Asm2Vec [14], GraphEmb [44] và OrderMatters [62]. Khi sử dụng BinaryCorp với đầy đủ kích thước trong nhiệm vụ tìm chức năng khớp trong các nhóm có 10.000 chức năng, gần giống với các tình huống trong thế giới thực, jTrans xếp hạng chức năng khớp đúng nhất với điểm tương đồng cao nhất (được ký hiệu là Recall@1) với xác suất trung bình là 62,5%, trong khi giải pháp SOTA tốt nhất chỉ đạt 32,0%. Đối với tình huống ít thực tế hơn (và dễ hơn) với nhóm có 32 chức năng, phương pháp của chúng tôi vượt trội hơn đối thủ gần nhất 10,6% (từ 84,3% lên 94,9%) cho cùng chỉ số Recall@1. Hơn nữa, khi được đánh giá trên một nhiệm vụ tìm kiếm lỗ hổng thế giới thực, jTrans đạt được điểm gọi lại gấp đôi so với các giải pháp SOTA.
* Tóm lại, nghiên cứu của chúng tôi đóng góp những điều sau đây:
* Chúng tôi đề xuất một mô hình mới dựa trên Transformer có ý thức về nhảy, jTrans, là giải pháp đầu tiên nhúng thông tin luồng điều khiển vào Transformer. Phương pháp của chúng tôi có thể học biểu diễn mã nhị phân và hỗ trợ BCSD trong thế giới thực. Chúng tôi công bố mã nguồn của jTrans tại <https://github.com/vul337/jTrans>.
* Chúng tôi tạo ra một bộ dữ liệu lớn, đa dạng và đúng đắn mới, BinaryCorp, cho nhiệm vụ BCSD. Theo kiến thức tốt nhất của chúng tôi, BinaryCorp là bộ dữ liệu đa dạng nhất cho đến thời điểm hiện tại và nó có thể giảm thiểu đáng kể các vấn đề overfitting của các bộ kiểm tra trước đây.
* Chúng tôi tiến hành các thí nghiệm mở rộng và chỉ ra rằng mô hình của chúng tôi có thể vượt trội đáng kể so với các phương pháp SOTA.

1. ĐỊNH NGHĨA BÀI TOÁN

* BCSD là một nhiệm vụ cơ bản để tính toán độ tương đồng của hai hàm nhị phân. Nó có thể được sử dụng trong ba loại tình huống như đã thảo luận trong [26], bao gồm:
* (1) Một-đến-một (OO): Ở đây, chúng ta trả về điểm tương đồng của một hàm nguồn đến một hàm đích duy nhất.
* (2) Một-đến-nhiều (OM): Trong trường hợp này, một nhóm các hàm đích sẽ được sắp xếp dựa trên điểm tương đồng của chúng đến một hàm nguồn duy nhất.
* (3) Nhiều-đến-nhiều (MM): Ở đây, một nhóm các hàm sẽ được chia thành các nhóm dựa trên độ tương đồng.
* Mà không mất tính tổng quát, trong nghiên cứu này, chúng tôi tập trung vào các nhiệm vụ OM. Lưu ý rằng, chúng ta có thể giảm các bài toán OM thành các bài toán OO bằng cách đặt kích thước của các hàm đích là 1. Chúng ta cũng có thể mở rộng các bài toán OM thành các bài toán MM, bằng cách xem mỗi hàm trong nhóm là hàm nguồn và giải quyết nhiều bài toán OM. Để giữ cho trình bày rõ ràng, chúng tôi đưa ra một định nghĩa hình thức của các vấn đề như sau:

A diagram of a computer

Description automatically generated

Hình 1: Một ví dụ về luồng điều khiển của một hàm nhị phân. Phần bên trái là mã lắp ráp dạng thứ tự tuyến tính với các địa chỉ nhảy, và phần bên phải là đồ thị luồng điều khiển tương ứng.

* Định nghĩa 2.1 (Hàm). Trong nghiên cứu này, chúng tôi hiểu một hàm là một tập hợp các lệnh được sắp xếp trong các chương trình nhị phân, được biên dịch từ một hàm mã nguồn (có thể chứa các hàm được nhúng vào). Do đó, hàm có ý nghĩa cụ thể và ràng buộc nội tại. Đặc biệt, một hàm có một đồ thị luồng điều khiển đại diện cho thông tin luồng điều khiển của nó, như được thể hiện trong Hình 1.
* Định nghĩa 2.2 (BCSD Task). Cho trước một hàm nguồn 𝑓𝑞 và một nhóm các hàm 𝑃, nhiệm vụ phát hiện độ tương đồng nhị phân là tìm ra các hàm top-k {𝑓1, 𝑓2, . . . , 𝑓𝑘 | 𝑓𝑖 ∈ 𝑃} được xếp hạng theo điểm tương đồng.
* Mục tiêu của BCSD là phát triển một giải pháp để tính toán điểm tương đồng của hai hàm, trong đó hai hàm được biên dịch từ hai hàm mã nguồn có cùng hoặc tương đồng đến một mức độ nào đó (ví dụ: một hàm là phiên bản được vá lỗi của hàm khác) nên có độ tương đồng.

3 CÔNG TRÌNH LIÊN QUAN

3.1 Các Phương pháp BCSD không dựa trên Máy học

* Trước khi áp dụng máy học, các kỹ thuật tương đồng mã nhị phân truyền thống bao gồm các phương pháp tĩnh và động. Dưới giả định rằng mã tương đồng logic có cùng hành vi chạy, các phương pháp phân tích động đo lường sự tương đồng mã nhị phân bằng cách phân tích các đặc trưng động được tạo thủ công. Loại giải pháp này bao gồm các công trình như BinDiff [16], BinHunt [23], iBinHunt [46] và Genius [21], dựa trên lý thuyết đồ thị CG/CFG (Control Flow Graph) đồng dạng (GI) [16, 22]. Những công trình này so sánh sự tương đồng của hai hàm nhị phân bằng cách sử dụng thuật toán khớp đồ thị. ESH [8] sử dụng trình chứng minh định lý để xác định hai khối cơ bản có tương đương hay không. Tuy nhiên, phương pháp này không áp dụng cho trường hợp tối ưu hóa trình biên dịch khác nhau do chia khối cơ bản. BinGo [5], Blex [17] và Multi-MH [49] sử dụng các giá trị được chọn ngẫu nhiên để khởi tạo ngữ cảnh của hàm, sau đó so sánh sự tương đồng bằng cách thu thập các giá trị vào/ra. Hạn chế chính của các phương pháp động này là chúng không phù hợp cho việc phát hiện tương đồng mã nhị phân quy mô lớn. Điều này do chúng tốn kém tính toán và yêu cầu thời gian chạy lâu dài để phân tích toàn bộ mã nhị phân.
* Các phương pháp tĩnh cho BCSD dựa trên việc xác định các khác biệt cấu trúc trong mã nhị phân. Các phương pháp như BinClone [19], ILine [34], MutantX-S [31], BinSign [47] và Kam1n0 [13] sử dụng các toán tử hoặc chỉ thị được phân loại như các đặc trưng tĩnh cho việc tính toán tương đồng mã nhị phân. Tracelet [11] và BinSequence [33] so sánh sự tương đồng của hai hàm nhị phân dựa trên khoảng cách chỉnh sửa giữa các chuỗi chỉ thị. TEDEM [50] và XMATCH [20] tính toán tương đồng mã nhị phân bằng cách sử dụng khoảng cách chỉnh sửa đồ thị/cây của các cây biểu diễn khối cơ bản. Mặc dù các phương pháp tĩnh hiệu quả hơn các phương pháp động, nhưng chúng thường đạt độ chính xác thấp hơn, vì chúng chỉ nắm bắt thông tin cấu trúc và cú pháp của mã nhị phân, bỏ qua ý nghĩa và mối quan hệ giữa các chỉ thị.

A diagram of a function

Description automatically generated

Hình 2: Khi sử dụng nhúng (embeddings) cho việc phát hiện tương đồng mã nhị phân, hàm truy vấn và các hàm ứng cử viên trong nhóm hàm được ánh xạ vào không gian vector ngữ nghĩa. Các nhúng của hàm truy vấn, hàm tương đồng và hàm không tương đồng được ký hiệu là 𝑒𝑞, 𝑒𝑝 và 𝑒𝑛, tương ứng.

3.2 Các Phương pháp BCSD dựa trên Học máy

* Nghiên cứu về BCSD dựa trên học máy đã được truyền cảm hứng từ sự phát triển gần đây trong xử lý ngôn ngữ tự nhiên (NLP) [39, 45, 56], trong đó sử dụng các vectơ giá trị thực được gọi là nhúng (embeddings) để mã hóa thông tin ngữ nghĩa của các từ và câu. Xây dựng dựa trên những kỹ thuật này, các nghiên cứu trước đó [14, 15, 24, 40, 43, 44, 51, 59, 61–63] đã áp dụng các phương pháp học sâu vào việc phát hiện tương đồng mã nhị phân. Ý tưởng chung của nhiều nghiên cứu này là nhúng các hàm nhị phân thành các vectơ số và sau đó sử dụng khoảng cách vector để xấp xỉ sự tương đồng giữa các hàm nhị phân khác nhau. Như được thể hiện trong Hình 2, các phương pháp này sử dụng các thuật toán huấn luyện học sâu để làm cho khoảng cách vector của các hàm nhị phân logic tương tự gần nhau hơn.
* Hầu hết các phương pháp dựa trên học sâu sử dụng mạng Siamese [6], yêu cầu phải có ánh xạ của các hàm nhị phân tương đương trong dữ liệu huấn luyện. Ví dụ, 𝛼Diff [40] học các nhúng của hàm nhị phân trực tiếp từ dãy byte gốc bằng mạng nơ-ron tích chập (CNN) [38]. INNEREYE [63] và RLZ2019 [51] xem các chỉ thị như là từ và các khối cơ bản như là các câu, và sử dụng word2vec [45] và LSTM [29] để học các nhúng của khối cơ bản. SAFE [43] sử dụng phương pháp tương tự để học các nhúng của các hàm nhị phân, trong khi Gemini [59], VulSeeker [24], GraphEmb [44] và OrderMatters [62] sử dụng các mạng nơ-ron đồ thị (GNNs) để xây dựng mô hình nhúng đồ thị cho việc học đồ thị luồng điều khiển được gán thuộc tính (ACFG) của các hàm nhị phân. Gemini và VulSeeker mã hóa các khối cơ bản với các đặc trưng được chọn thủ công, trong khi GraphEmb và OrderMatters sử dụng mạng nơ-ron để học các nhúng của các khối cơ bản. Một phương pháp khác được đề xuất bởi DEEPBINDIFF [15] và Codee [61], sử dụng mạng nơ-ron để học các nhúng của các dãy chỉ thị được tạo ra thay vì nhúng ACFG của các hàm nhị phân.
* Cũng đã có nghiên cứu về học không giám sát (học mà không cần nhãn) trong lĩnh vực BCSD. Một giải pháp tiêu biểu là Asm2Vec [14]. Phương pháp này cũng tạo ra các dãy chỉ thị với đồ thị luồng điều khiển (CFG), nhưng không dựa vào ánh xạ thực địa của các hàm nhị phân tương đương. Asm2Vec sử dụng một thuật toán không giám sát để học nhúng của các hàm nhị phân. Tuy nhiên, hiệu suất của nó không tốt bằng các phương pháp học có giám sát hiện đại.
* Tổng thể, các phương pháp dựa trên học được phù hợp cho việc phát hiện tương đồng mã nhị phân quy mô lớn, vì các chức năng mã nhị phân có thể được chuyển đổi thành các vectơ. Sau đó, tương đồng có thể được tính toán bằng cách sử dụng khoảng cách vector, điều này là hiệu quả tính toán. Tuy nhiên, các kỹ thuật hiện có có những hạn chế. Một số phương pháp [24, 59] bỏ qua ý nghĩa của các chỉ thị và khối cơ bản, vì họ chỉ sử dụng các đặc trưng được chọn thủ công để biểu diễn khối cơ bản. Các phương pháp khác [14, 15, 40, 43, 51, 61, 63] bỏ qua một số hoặc tất cả thông tin cấu trúc của các hàm nhị phân, vì họ không sử dụng thông tin luồng điều khiển, hoặc tạo ra các dãy chỉ thị với CFG bằng cách sử dụng random walk. Cuối cùng, các phương pháp như [44, 62] học các nhúng của khối cơ bản và sử dụng GNN để học nhúng của đồ thị luồng điều khiển được gán thuộc tính (ACFG) của các hàm nhị phân. Mặc dù các phương pháp này hiệu quả trong một số tình huống, nhưng chúng bỏ qua sự xuất hiện chung giữa các chỉ thị giữa các khối cơ bản.

**4. PHƯƠNG PHÁP**

* 1. Tổng quan
* jTrans dựa trên kiến trúc Transformer-Encoder.
* Thay đổi đầu tiên mà chúng tôi đề xuất cho kiến trúc Transformer nhằm giúp jTrans nắm bắt tốt hơn các mối quan hệ nhảy của mã, tức là thông tin luồng điều khiển. Để làm được điều này, chúng tôi trước tiên tiền xử lý mã lắp ráp của các chương trình nhị phân đầu vào để chứa thông tin luồng điều khiển của chương trình. Tiếp theo, chúng tôi điều chỉnh việc nhúng các mã thông báo cá nhân của Transformer sao cho các vị trí nguồn và đích của các nhảy là "ngữ nghĩa" tương tự nhau.
* Thay đổi thứ hai mà chúng tôi đề xuất liên quan đến quá trình huấn luyện của mô hình đề xuất của chúng tôi. Xem xét sự tương đồng giữa ngôn ngữ tự nhiên và các chương trình trong luồng dữ liệu, chúng tôi đã chọn sử dụng phương pháp huấn luyện Transformer hiệu quả thông thường là Masked Language Model (MLM) [12]. Huấn luyện dựa trên MLM đòi hỏi mô hình dự đoán nội dung của các mã thông báo được che giấu dựa trên nội dung của các mã thông báo lân cận của chúng, từ đó buộc mô hình phát triển hiểu biết bối cảnh về mối quan hệ giữa các chỉ thị.
* Hơn nữa, để khuyến khích mô hình học cách các nhảy được tích hợp vào mã, chúng tôi đề xuất một tác vụ huấn luyện phụ mới mà yêu cầu mô hình dự đoán mục tiêu của một chỉ thị nhảy. Tác vụ này, chúng tôi gọi là “Jump Target Prediction” (JTP), đòi hỏi hiểu biết sâu về ngữ nghĩa của mã, và như được thể hiện trong Phần 6.4, nó đóng góp đáng kể vào hiệu suất của mô hình của chúng tôi.

4.2 Mô hình biểu diễn hàm nhị phân

* jTrans dựa trên kiến trúc BERT [12], đây là mô hình ngôn ngữ được huấn luyện trước tối tân trong nhiều tác vụ xử lý ngôn ngữ tự nhiên (NLP). Trong jTrans, chúng tôi tuân theo cùng một phương pháp chung được sử dụng bởi BERT để mô hình hóa văn bản, tức là tạo ra một nhúng (embedding) cho mỗi token (tức là từ), và sử dụng cơ chế chú ý (attention) mạnh mẽ của BERT để hiệu quả mô hình mã nhị phân.
* Tuy nhiên, mã nhị phân khác biệt với ngôn ngữ tự nhiên trong một số khía cạnh. Đầu tiên, có quá nhiều từ vựng (ví dụ: hằng số và chữ số) trong mã nhị phân. Thứ hai, trong mã nhị phân có các chỉ thị nhảy. Đối với một chỉ thị nhảy, chúng tôi ký hiệu mã thông báo của toán hạng của nó là mã thông báo nguồn, xác định địa chỉ của địa chỉ của chỉ thị nhảy. Để đơn giản, chúng tôi ký hiệu mã thông báo kí ức của chỉ thị đích là mã thông báo đích và biểu diễn cặp nhảy này là <mã thông báo nguồn, mã thông báo đích>.

Do đó, chúng tôi phải giải quyết hai vấn đề để áp dụng BERT:

* Các mã thông báo nằm ngoài từ vựng (OOV). Giống như trong lĩnh vực NLP, chúng tôi cần huấn luyện jTrans trên một từ vựng cố định có chứa các mã thông báo phổ biến nhất trong tập dữ liệu đã phân tích. Các mã thông báo không được bao gồm trong từ vựng cần được biểu diễn một cách sao cho Transformer có thể xử lý chúng hiệu quả.
* Mô hình các chỉ thị nhảy. Sau quá trình tiền xử lý, mã nhị phân chỉ còn lại một số thông tin cho mã thông báo nguồn và mã thông báo đích của cặp nhảy. BERT khó có thể suy ra mối liên hệ giữa chúng. Vấn đề này còn được gia tăng thêm bởi khoảng cách có thể lớn giữa mã thông báo nguồn và mã thông báo đích, làm cho việc suy luận ngữ cảnh càng khó khăn hơn.

Chúng tôi đề xuất giải quyết những thách thức này như sau.

4.2.1 Tiền xử lý các chỉ thị.

* Để giảm thiểu vấn đề mã thông báo nằm ngoài từ vựng (OOV), chúng tôi sử dụng công cụ phân tích mã nguồn IDA Pro 7.5 [28], công cụ tiên tiến nhất trong lĩnh vực này, để phân tích các chương trình nhị phân đầu vào và thu được các dãy chỉ thị lắp ráp. Sau đó, chúng tôi áp dụng các chiến lược mã hóa sau để chuẩn hóa mã lắp ráp và giảm kích thước từ vựng:

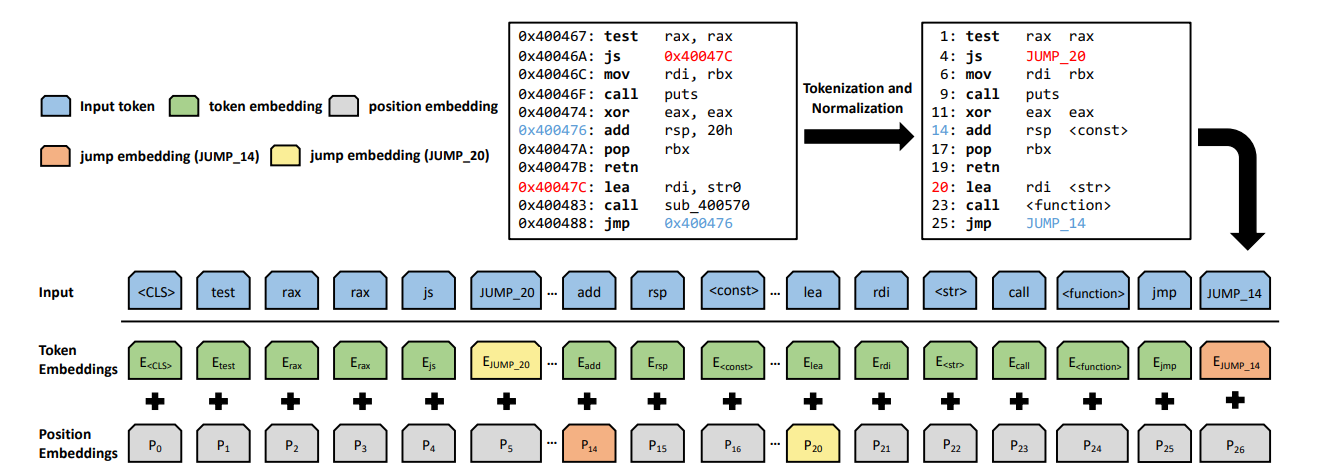
(1)Chúng tôi sử dụng các từ gọi lệnh (mnemonics) và toán hạng như các mã thông báo.

(2) Chúng tôi thay thế các chuỗi chữ số bằng một mã thông báo đặc biệt <str>.

(3) Chúng tôi thay thế các giá trị hằng số bằng một mã thông báo đặc biệt <const>.

(4) Chúng tôi giữ lại tên của các cuộc gọi hàm bên ngoài và nhãn như các mã thông báo, và thay thế tên của các cuộc gọi hàm nội bộ bằng <function>1.

(5) Đối với mỗi cặp nhảy, chúng tôi thay thế mã thông báo nguồn của nó (là địa chỉ tuyệt đối hoặc tương đối của địa chỉ đích của nhảy) bằng một mã thông báo JUMP\_XXX, trong đó XXX là thứ tự của mã thông báo đích của cặp nhảy này, ví dụ 20 và 14 trong Hình 3. Như vậy, chúng tôi có thể loại bỏ tác động của các địa chỉ cơ sở ngẫu nhiên của các tệp nhị phân.



Hình 3: Biểu diễn đầu vào của jTrans. Mã lắp ráp gốc được mã hóa thành các mã thông báo (token) và chuẩn hóa. Sau đó, mỗi mã thông báo được chuyển đổi thành một nhúng mã thông báo (token embedding) và một nhúng vị trí (position embedding), trong khi nhúng đầu vào cuối cùng của nó là tổng của hai nhúng này. Đối với mỗi cặp nhảy, nhúng mã thông báo nguồn của nó (ví dụ: EJUMP\_14), còn được gọi là nhúng nhảy (jump embedding), chia sẻ tham số với nhúng vị trí của mã thông báo đích của nó (ví dụ: P14).

4.2.2 Mô hình các chỉ thị nhảy.

* Tiếp theo, chúng tôi giải quyết vấn đề biểu diễn các chỉ thị nhảy một cách sao cho jTrans có thể ngữ cảnh hóa tốt hơn tính hai phần của chúng (và nắm bắt toàn bộ thông tin luồng điều khiển như một thể). Chúng tôi chọn sử dụng các mã hóa vị trí, một phần không thể thiếu của kiến trúc Transformer. Những mã hóa này cho phép mô hình xác định khoảng cách giữa các mã thông báo. Logic ngầm của biểu diễn này là khoảng cách lớn giữa các mã thông báo thường chỉ ra sự ảnh hưởng tương đối yếu. Tuy nhiên, các chỉ thị nhảy kết nối các khu vực trong mã có thể cách xa nhau. Do đó, chúng tôi điều chỉnh cơ chế mã hóa vị trí để phản ánh hiệu quả của các chỉ thị nhảy.
* Các thay đổi của chúng tôi đối với mã hóa vị trí được thiết kế để phản ánh thực tế rằng mã thông báo nguồn và mã thông báo đích của các chỉ thị nhảy không chỉ gần nhau như hai mã thông báo liên tiếp (do thứ tự thực thi), mà còn có một mối quan hệ ngữ cảnh mạnh. Chúng tôi đạt được mục tiêu này thông qua chia sẻ tham số: đối với mỗi cặp nhảy, nhúng mã thông báo nguồn (xem 𝐸JUMP\_14 trong Hình 3) được sử dụng như mã hóa vị trí của mã thông báo đích (xem 𝑃14).
* Biểu diễn này đạt được hai mục tiêu quan trọng. Thứ nhất, nhúng chung cho phép Transformer nhận dạng mối liên hệ ngữ cảnh giữa các mã thông báo nguồn và mã thông báo đích. Thứ hai, mối liên hệ ngữ cảnh mạnh này được duy trì trong suốt quá trình huấn luyện, vì các tham số chung được cập nhật đồng thời cho cả hai mã thông báo. Lưu ý rằng chúng tôi chỉ tập trung vào các chỉ thị nhảy trực tiếp trong jTrans. Chúng tôi giả định rằng thông tin luồng điều khiển được đưa ra bởi các nhảy gián tiếp sẽ cải thiện hiệu suất của jTrans. Tuy nhiên, việc nhận dạng các mục tiêu của các nhảy như vậy là một thách thức mở rộng phổ biến và nằm ngoài phạm vi của công việc hiện tại của chúng tôi. Nếu có một giải pháp cho việc nhận dạng các mục tiêu của nhảy gián tiếp được đề xuất trong tương lai, chúng tôi có thể nhúng phép gán của nhảy gián tiếp với sự kết hợp của các nhúng vị trí của tất cả các mục tiêu nhảy.

4.2.3 Cơ sở lý do cho phương pháp đề xuất của chúng tôi.

* Bằng cách chia sẻ tham số giữa các mã thông báo nguồn và mã thông báo đích của cặp nhảy, chúng tôi tạo ra một độ tương đồng cao trong biểu diễn của chúng. Kết quả là, mỗi khi cơ chế chú ý của jTrans gán một trọng số chú ý cao cho một trong các mã thông báo này (tức là xác định rằng nó quan trọng đối với việc hiểu / phân tích mã nhị phân), nó cũng sẽ tự động gán trọng số chú ý cao cho đối tác của chúng. Biểu diễn này đảm bảo rằng cả hai phần của chỉ thị nhảy - và các chỉ thị gần chúng trong mã - sẽ được bao gồm trong quá trình suy luận.
* Bây giờ chúng tôi cung cấp một phân tích hình thức về sự tương đồng mã thông báo của cặp nhảy và chứng minh rằng sự tương đồng trong cặp này cao hơn bất kỳ mã thông báo nào trong số họ với bất kỳ mã thông báo khác. Đối với một hàm nhị phân cụ thể f = [𝑥1, · · · , 𝑥𝑛], 𝑥𝑖 là mã thông báo thứ 𝑖 của f. Tất cả các mã thông báo sẽ được chuyển đổi thành các vectơ nhúng hỗn hợp {𝐸(𝑥1), · · · , 𝐸(𝑥𝑛)} trước khi được đưa vào jTrans, trong đó mỗi nhúng 𝐸(𝑥𝑖) có thể được biểu diễn dưới dạng tổng của nhúng mã thông báo 𝐸𝑥𝑖 và nhúng vị trí 𝑃𝑖. Chúng tôi áp dụng cơ chế tự chú ý đa đầu (multi-head self-attention) [57] cho các vectơ nhúng hỗn hợp {𝐸(𝑥1), · · · , 𝐸(𝑥𝑛)}. Chúng tôi ký hiệu nhúng của lớp 𝑚 là 𝐸𝑚 = [𝐸𝑚 (𝑥1), · · · , 𝐸𝑚 (𝑥𝑛)], chúng tôi đầu tiên chiếu nhúng 𝑚-th thành 𝑄𝑚, 𝐾𝑚 và 𝑉𝑚, tương ứng. Sau đó, chúng tôi sử dụng cơ chế tự chú ý theo tỉ lệ để có được ma trận chú ý Attention(𝑄𝑚, 𝐾𝑚, 𝑉𝑚).

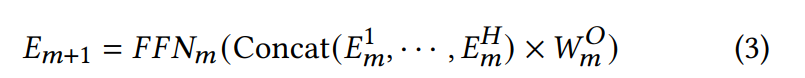
A mathematical equations on a white background

Description automatically generated

* The 𝑊 𝑄 𝑚 ∈ R 𝑑model×𝑑𝑘 , 𝑊 𝐾 𝑚 ∈ R 𝑑model×𝑑𝑘 , 𝑊𝑉 𝑚 ∈ R 𝑑model×𝑑𝑘 are affine transformation matrices of the 𝑚-th layer, 𝑑model is the dimension of the embedding vector. Softmax( 𝑄𝑚𝐾 𝑇 √ 𝑚 𝑑𝑘 ) is the attention weight matrix. We denote the updated embedding by head ℎ as

A close up of a word

Description automatically generated

* Assume we have 𝐻 attention head, we get updated embedding 𝐸𝑚+1 as follows, 𝑊 𝑂 𝑚 ∈ R 𝑑𝑘𝐻×𝑑model is the output transformation matrix of the 𝑚-th layer, 𝐹 𝐹𝑁𝑚 is the feed-forward network of the 𝑚-th layer.
* 
* Đầu ra cuối cùng của jTrans là lớp cuối cùng của mô hình. Chúng tôi tạo ra nhúng hàm 𝐸𝑓 như sau, trong đó 𝑊 𝐹 ∈ R𝑑model×𝑑𝑓 là ma trận biến đổi đầu ra, 𝑑𝑓 là kích thước của nhúng hàm, và 𝐸𝑁 (𝐶) đại diện cho nhúng của <CLS>.
* A black text on a white background

  Description automatically generated
* Tiếp theo, chúng tôi trình bày cách jTrans cung cấp thông tin luồng điều khiển của chương trình. Xem xét ba mã thông báo 𝑖, 𝑗, 𝑙 trong hàm đã cho, các nhúng tương ứng là 𝐸𝑖 , 𝐸𝑗 và 𝐸𝑙 . Giả sử rằng chúng ta có một mối quan hệ nhảy giữa 𝑖 và 𝑗, 𝑖 là mã thông báo nguồn và 𝑗 là mã thông báo đích. Và 𝑙 là bất kỳ mã thông báo nào khác trong hàm. Ký hiệu trọng số chú ý của mã thông báo 𝑖 đến mã thông báo 𝑗 là 𝐴𝑖𝑗 = 𝑄𝑖𝐾 𝑇 𝑗 √ 𝑑𝑘 . Chúng tôi chứng minh rằng kỳ vọng toán học của 𝐴𝑖𝑗 trừ đi 𝐴𝑖𝑙 là dương. Điều này có thể được biểu thị như sau:
* 
* Phương trình này cho thấy rằng 𝑖 thông thường chú ý nhiều hơn đến 𝑗 hơn là 𝑙. Đây là giải thích nội tại về nhúng nhảy. Chứng minh chi tiết được trình bày trong Phụ lục 8.

A close-up of a computer

Description automatically generated

4.3 Tiền huấn luyện jTrans

* Kiến trúc BERT, trên đó jTrans được dựa, sử dụng hai nhiệm vụ học không giám sát để tiền huấn luyện. Nhiệm vụ đầu tiên là mô hình ngôn ngữ bị che giấu (MLM), trong đó BERT được giao nhiệm vụ xây dựng lại các mã thông báo bị che giấu ngẫu nhiên. Nhiệm vụ học không giám sát thứ hai được thiết kế để cải thiện khả năng ngữ cảnh của BERT bằng cách yêu cầu nó xác định xem hai câu có liên tiếp hay không. Chúng tôi xây dựng trên quá trình huấn luyện tổng thể của BERT, đồng thời thực hiện điều chỉnh cụ thể cho miền: chúng tôi giữ nhiệm vụ MLM, nhưng thay thế nhiệm vụ thứ hai bằng một nhiệm vụ chúng tôi gọi là dự đoán nhiệm vụ nhảy (JTP). Như được thể hiện trong Phần 4.3.2, mục tiêu của nhiệm vụ JTP là cải thiện khả năng hiểu ngữ cảnh của jTrans về các hướng dẫn nhảy.

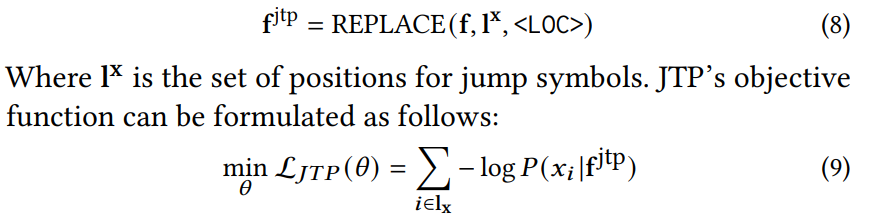
4.3.1 Nhiệm vụ Mô hình Ngôn ngữ bị che giấu (MLM).

* Nhiệm vụ MLM của chúng tôi tương đối giống với nhiệm vụ được đề xuất trong [12], và jTrans sử dụng các quy trình che giấu của BERT: 80% số mã thông báo được chọn ngẫu nhiên bị thay thế bằng mã thông báo che giấu (chỉ ra rằng chúng cần được xây dựng lại), 10% bị thay thế bằng các mã thông báo ngẫu nhiên khác, và 10% không thay đổi. Theo ký hiệu trong phần 4.2.3, chúng tôi định nghĩa hàm f = [𝑥1, · · · , 𝑥𝑛], trong đó 𝑥𝑖 là mã thông báo thứ 𝑖 của f và 𝑛 là số lượng mã thông báo. Trước tiên, chúng tôi chọn một tập hợp ngẫu nhiên các vị trí trong f để che giấu (tức là mx).
* 
* Based on these definitions, the MLM objective of reconstructing the masked tokens can be formulated as follows:
* A black text on a white background

  Description automatically generated
* where mx contains the indices of the masked tokens.
* Một ví dụ về quá trình che giấu được trình bày trong Hình 4. Chúng tôi che giấu các mã thông báo rsp, rdi và call, và yêu cầu jTrans xây dựng lại chúng. Để thành công, mô hình của chúng tôi phải học cú pháp cơ bản của lệnh lắp ráp và thông tin ngữ cảnh của nó. Để thành công xây dựng lại mã thông báo rdi, ví dụ, yêu cầu mô hình học các quy ước gọi hàm của chức năng, trong khi mã thông báo rsp đòi hỏi một sự hiểu biết về việc thực hiện liên tục.

4.3.2 Dự đoán Mục tiêu Nhảy.

* Nhiệm vụ JTP được định nghĩa như sau: cho một mã thông báo nguồn nhảy được chọn ngẫu nhiên, mô hình của chúng tôi được yêu cầu dự đoán mã thông báo mục tiêu tương ứng của nó. Nhiệm vụ này, thậm chí còn khó đối với các chuyên gia con người, đòi hỏi mô hình của chúng tôi phải phát triển một hiểu biết sâu về CFG. Điều này dẫn đến hiệu suất cải thiện cho jTrans, như chúng tôi sẽ thấy sau này trong phần 6.4. JTP được thực hiện bằng cách chọn một tập hợp ngẫu nhiên các mã thông báo nguồn nhảy có sẵn. Sau đó, các mã thông báo này được thay thế bằng mã thông báo <LOC>.

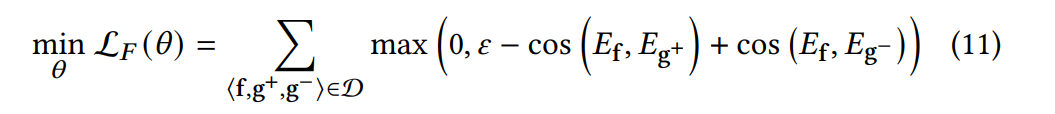


* Một ví dụ về nhiệm vụ JTP được trình bày trong Hình 5, trong đó chúng tôi thay thế mã thông báo JUMP\_20 bằng <LOC> và yêu cầu mô hình dự đoán chỉ mục. Một phân tích về hiệu suất của jTrans trong nhiệm vụ JTP được trình bày trong Bảng 6, cho thấy mô hình của chúng tôi đạt độ chính xác là 92,9%. Hơn nữa, một nghiên cứu chặt rút để đánh giá đóng góp của JTP đối với hiệu suất tổng thể của jTrans được trình bày trong Phần 6.4. Kết quả này cho thấy rõ ràng rằng nhiệm vụ huấn luyện này cải thiện khả năng của jTrans trong việc học các dòng điều khiển của các chức năng được phân tích.
* Hàm mất mát tổng thể của jTrans trong giai đoạn tiền huấn luyện là tổng của hàm mục tiêu MLM và JTP:
* (Loss)\_jTrans = (Loss)\_MLM + (Loss)\_JTP
* Trong đó:
* (Loss)\_MLM là hàm mục tiêu cho nhiệm vụ masked language model,
* (Loss)\_JTP là hàm mục tiêu cho nhiệm vụ jump target prediction.
* A black text with a white background

  Description automatically generated

4.4 Sau khi hoàn thành giai đoạn tiền huấn luyện không giám sát:

* Chúng tôi tiến hành điều chỉnh mô hình để thực hiện nhiệm vụ học giám sát về phát hiện sự tương đồng giữa các hàm. Mục tiêu của chúng tôi là huấn luyện jTrans để tối đa hóa sự tương đồng giữa các cặp hàm tương tự, đồng thời giảm thiểu sự tương đồng cho các cặp không liên quan. Như được trình bày trong Phần 4.2.3, chúng tôi sử dụng Phương trình 4 để biểu diễn hàm 𝑓. Phương pháp đo lường được chọn của chúng tôi để tính toán sự tương đồng giữa các hàm là sự tương đồng Cosine.
* Chúng tôi sử dụng các ký hiệu sau: cho F và G là một tập hợp các hàm nhị phân, và tập hợp các hàm tương tự (được gọi là "đất trời"). Đối với bất kỳ hàm truy vấn 𝑓 ∈ F, cho g
* ∈ G là một hàm tương tự với 𝑓 (ví dụ: được biên dịch từ cùng mã nguồn). Hơn nữa, cho g
* ∈ G là một hàm tùy ý, không liên quan đến 𝑓. Chúng tôi ký hiệu embedding cho hàm 𝑓 là 𝐸f. Cuối cùng, chúng tôi xác định D là tập hợp tất cả các bộ tam giác đã tạo ra:
* (f, g +, g -)
* Hàm mục tiêu cho quá trình điều chỉnh jTrans của chúng tôi được huấn luyện bằng phương pháp học tương phản [25, 53] và được thực hiện như sau:



* Ở đây, 𝜃 đại diện cho các tham số của mô hình, và 𝜀 là một siêu tham số thường được chọn trong khoảng từ 0 đến 0.5 [48]. Sau khi hoàn thành giai đoạn điều chỉnh lại jTrans trên D, chúng ta có thể đo lường độ tương tự giữa hai hàm 𝑓1, 𝑓2 bằng cách tính toán độ tương tự Cosine của nhúng của chúng. Sau khi quá trình điều chỉnh lại hoàn tất, chúng ta có thể đo lường độ tương tự giữa hai hàm 𝑓1, 𝑓2 bằng cách tính toán độ tương tự Cosine của nhúng của chúng.

4.5 Large-Scale Dataset Construction:

* Chúng tôi xây dựng bộ dữ liệu cho việc phát hiện sự tương đồng trong mã nhị phân dựa trên kho lưu trữ chính thức của ArchLinux [2] và kho lưu trữ người dùng Arch User Repository [3]. ArchLinux là một bản phân phối Linux nổi tiếng với số lượng lớn gói phần mềm và cập nhật gói nhanh chóng. Kho lưu trữ chính thức của ArchLinux [2] chứa hàng ngàn gói phần mềm, bao gồm các phần mềm chỉnh sửa, phần mềm nhắn tin tức thời, máy chủ HTTP, trình duyệt web, trình biên dịch, thư viện đồ họa, thư viện mã hóa, v.v. Và kho lưu trữ người dùng Arch User Repository chứa hơn 77.000 gói phần mềm được tải lên và duy trì bởi người dùng, mở rộng đáng kể bộ dữ liệu của chúng tôi. Ngoài ra, ArchLinux cung cấp công cụ hữu ích makepkg cho các nhà phát triển để xây dựng gói phần mềm của họ từ mã nguồn. makepkg có thể biên dịch gói phần mềm cụ thể từ mã nguồn bằng cách phân tích tệp PKGBUILD, chứa các phụ thuộc cần thiết và các hàm hỗ trợ biên dịch. Nhiệm vụ phát hiện sự tương đồng mã nhị phân yêu cầu một số lượng lớn dữ liệu đã được gán nhãn, do đó chúng tôi sử dụng các cơ sở hạ tầng này để xây dựng bộ dữ liệu của chúng tôi.

4.5.1. Lọc Dự án.

* Vì lý do tương thích biên dịch, chúng tôi chọn các dự án C/C++ trong quá trình xây dựng bộ dữ liệu. Nếu hàm biên dịch trong tệp PKGBUILD chứa cuộc gọi của cmake, make, gcc và g++, thì rất có thể đó là một dự án C/C++. Ngược lại, nếu biến depend trong tệp PKGBUILD chứa rustc, go, jvm, thì không có khả năng đó là một dự án C/C++, chúng tôi có thể loại bỏ nó trước khi biên dịch.

4.5.2. Luồng biên dịch.

* Trong luồng công việc của chúng tôi, chúng tôi muốn tự động chỉ định mức tối ưu hóa mà chúng tôi muốn cho mỗi lần biên dịch. Một số dự án không sử dụng biến môi trường CFLAGS và CXXFLAGS, làm cho việc thay đổi mức tối ưu hóa trở nên khó khăn. Tuy nhiên, vì hầu hết các dự án gọi trình biên dịch bằng CC hoặc CXX, chúng tôi gán các biến môi trường một phiên bản được tự sửa đổi của gcc, g++, clang, clang++. Trình biên dịch đã sửa đổi thay đổi các tham số dòng lệnh liên quan đến mức tối ưu hóa thành các tham số biên dịch được mong đợi. Ngoài ra, nó bổ sung các đối số biên dịch mong đợi vào các tham số ban đầu. Chúng tôi sử dụng hai cách này để đảm bảo việc biên dịch được thực hiện với mức tối ưu hóa mong đợi.

4.5.3. Thu thập nhãn.

* Để thu thập nhãn, trước tiên chúng tôi cần lấy mã nhị phân không bị cắt và lấy địa chỉ của các hàm. Chúng tôi phát hiện rằng nhiều dự án trong thực tế sử dụng lệnh strip trong quá trình biên dịch, do đó chỉ định tham số trong PKGBUILD không giải quyết được vấn đề này. Chúng tôi đã thay thế lệnh strip bằng phiên bản đã sửa đổi của nó. Phiên bản này sẽ không cắt bảng ký tự bất kể tham số truyền vào.

5.1 Tập dữ liệu BinaryCorp

* Chúng tôi giới thiệu BinaryCorp, tập dữ liệu mà chúng tôi tạo ra để đánh giá việc phát hiện sự tương đồng giữa các tập tin nhị phân quy mô lớn. BinaryCorp bao gồm một số lượng lớn các tập tin nhị phân được tạo ra bằng cách sử dụng luồng biên dịch tự động, trong đó, dựa trên các gói ArchLinux chính thức và Arch User Repository, chúng tôi sử dụng gcc và g++ để biên dịch 48.130 chương trình nhị phân với các cấp độ tối ưu hóa khác nhau và tuân theo phương pháp được đề xuất trong SAFE [43] để loại bỏ các hàm trùng lặp.

A table with numbers and symbols

Description automatically generated

Bảng 1: Thống kê về số lượng dự án, tập tin nhị phân và hàm của các tập dữ liệu. Dự án đề cập đến các tập tin nhị phân được biên dịch từ cùng mã nguồn.

* Tập dữ liệu BinaryCorp được tạo bằng cách sử dụng g++ và gcc để biên dịch 48,130 chương trình nhị phân với các cấp độ tối ưu hóa khác nhau, dựa trên các gói ArchLinux chính thức và kho dữ liệu người dùng Arch User Repository, và tuân theo phương pháp được đề xuất trong SAFE[43] để lọc các hàm trùng lặp.
* Thống kê của các tập dữ liệu của chúng tôi được thể hiện trong Bảng 1. Trong khi nhiều công trình trước đây sử dụng Coreutils và GNUtils làm tập dữ liệu của họ, Bảng 1 rõ ràng cho thấy BinaryCorp-26M hoạt động ở một quy mô khác: tập dữ liệu mới được tạo của chúng tôi có khoảng 26 triệu hàm so với 161,202 của GNUtils và 76,174 của Coreutils. Do đó, BinaryCorp-26M lớn hơn hơn 160 lần so với GNUtils và lớn hơn 339 lần so với Coreutils.
* Kích thước của tập dữ liệu mới của chúng tôi không cho phép sử dụng một số phương pháp hiện có do tính không thể mở rộng đủ của chúng. Do đó, chúng tôi cũng cung cấp một tập dữ liệu nhỏ hơn, có tên là BinaryCorp-3M, chứa 10,265 chương trình nhị phân và khoảng 3.6 triệu hàm. Số lượng hàm trong tập dữ liệu nhỏ của chúng tôi là khoảng 22 lần số lượng của GNUtils và 47 lần số lượng của Coreutils.
* BinKit[36] là tập dữ liệu nhị phân lớn nhất, bao gồm 36,256,322 hàm. Tuy nhiên, BinKit sử dụng 1,352 tùy chọn biên dịch khác nhau để tạo ra 243k chương trình nhị phân từ chỉ 51 gói GNU, do đó có quá nhiều hàm tương tự trong tập dữ liệu. Chúng tôi, ngược lại, chỉ sử dụng 5 tùy chọn biên dịch khác nhau để biên dịch gần 10,000 dự án. Mặc dù số lượng chương trình nhị phân của chúng tôi nhỏ hơn, tập dữ liệu của chúng tôi phong phú hơn BinKit về các khía cạnh như các nhà phát triển, kích thước dự án, phong cách mã hóa và kịch bản ứng dụng. Chúng tôi cho rằng tập dữ liệu mới của chúng tôi cung cấp một cơ sở học tập và đánh giá phong phú và có tính thực tế hơn. Sử dụng các tập dữ liệu của chúng tôi, chúng tôi có thể đánh giá tính mở rộng và hiệu quả của jTrans (và các mô hình so sánh khác) trên quy mô mới và lớn hơn. Như đã đề cập trong phần đóng góp của chúng tôi ở phần 1, chúng tôi sẽ đưa tập dữ liệu và các mô hình đã được đào tạo của chúng tôi sẵn sàng cho cộng đồng.

5.2 Các mô hình so sánh:

* Chúng tôi so sánh jTrans với sáu mô hình so sánh hàng đầu như sau:
* Genius [21]: Baseline này là một phương pháp không sử dụng deep learning. Genius trích xuất các đặc trưng nguyên thô dưới dạng đồ thị kiểm soát có thuộc tính và sử dụng băm nhạy cảm địa phương (LSH) để tạo ra các vectơ số học để tìm kiếm lỗ hổng. Chúng tôi triển khai baseline này dựa trên mã nguồn chính thức của nó.
* Gemini [59]: Baseline này trích xuất các đặc trưng được tạo thủ công cho mỗi khối cơ bản và sử dụng mạng nơ-ron đồ thị (GNN) để học biểu diễn của CFG (đồ thị kiểm soát dòng lệnh) của hàm được phân tích. Chúng tôi đã triển khai phương pháp này dựa trên mã nguồn chính thức của Tensorflow và sử dụng cài đặt mặc định trong quá trình đánh giá.
* SAFE [43]: Baseline này sử dụng một kiến trúc RNN với cơ chế chú ý (attention mechanisms) để tạo ra một biểu diễn cho hàm được phân tích, nó nhận các hướng dẫn bằng mã hợp ngữ làm đầu vào. Chúng tôi đã triển khai phương pháp này dựa trên mã nguồn chính thức của Pytorch và cài đặt các thông số mặc định.
* Asm2Vec [14]: Phương pháp này sử dụng các bước đi ngẫu nhiên trên đồ thị kiểm soát dòng lệnh để lấy mẫu các dãy hướng dẫn, sau đó sử dụng mô hình PV-DM để cùng nhau học biểu diễn của hàm và các mã thông báo hướng dẫn. Phương pháp này không phải là mã nguồn mở, và do đó chúng tôi đã sử dụng một phiên bản thực hiện không chính thức. Chúng tôi đã sử dụng các thông số mặc định.
* GraphEmb [44]: Baseline này sử dụng word2vec [45] để học các biểu diễn của các mã thông báo hướng dẫn. Tiếp theo, nó sử dụng một mạng nơ-ron biến thể (RNN) để tạo ra các biểu diễn độc lập cho mỗi khối cơ bản, và cuối cùng sử dụng structure2vec [7] để kết hợp các biểu diễn và tạo ra biểu diễn cho hàm được phân tích. Để làm cho baseline này có khả năng mở rộng cho các bộ dữ liệu lớn như BinaryCorp-26M, chúng tôi tự triển khai lại mã nguồn Tensorflow ban đầu của tác giả bằng Pytorch.
* OrderMatters [62]: Phương pháp này kết hợp hai loại biểu diễn. Loại biểu diễn đầu tiên sử dụng BERT để tạo ra một biểu diễn cho mỗi khối cơ bản, và sau đó kết hợp tất cả các biểu diễn này bằng một mạng nơ-ron đồ thị (GNN) để tạo ra biểu diễn cuối cùng. Loại biểu diễn thứ hai được thu được bằng cách áp dụng một mạng nơ-ron tích chập (CNN) lên đồ thị kiểm soát dòng lệnh. Sau đó, hai loại biểu diễn này được ghép nối. Phương pháp này không phải là mã nguồn mở, và API online blackbox của nó không đáp ứng được nhu cầu của nghiên cứu này. Chúng tôi đã triển khai phương pháp này bằng cách sử dụng các thông số đã báo cáo.

5.3 Các chỉ số đánh giá:

* Cho tập hợp các hàm nhị phân F và tập hợp các hàm nhị phân G là các hàm nhị phân mẫu thực tế (ground truth).
* A close-up of a computer code

  Description automatically generated
* Chúng ta ký hiệu một hàm truy vấn 𝑓𝑖 ∈ F và hàm nhị phân tương ứng thực tế 𝑓𝑔𝑡𝑖 ∈ G. Trong nghiên cứu này, chúng tôi xử lý bài toán xác định sự tương đồng giữa các hàm nhị phân, và mục tiêu của chúng tôi là tìm các hàm trong tập hàm G có độ tương đồng cao nhất với 𝑓𝑖. Các hàm được trả về được xếp hạng bằng một điểm tương đồng 𝑅𝑎𝑛𝑘𝑓𝑔𝑡𝑖, biểu thị vị trí của chúng trong danh sách các hàm đã tìm thấy. Hàm chỉ số I được định nghĩa như sau.
* A close-up of words

  Description automatically generated
* Hiệu suất truy xuất có thể được đánh giá bằng hai chỉ số sau đây:
* A black and white text

  Description automatically generated

6. Đánh giá:

* Của chúng tôi nhằm trả lời các câu hỏi sau.

• RQ1: Độ chính xác của jTrans trong các nhiệm vụ BCSD so với các phương pháp cơ sở khác nhau là như thế nào? (§6.1)

• RQ2: Hiệu suất của jTrans và các phương pháp cơ sở khác nhau trong các nhiệm vụ BCSD với các kích thước hồ bơi hàm khác nhau là như thế nào? (§6.2)

• RQ3: Hiệu quả của jTrans trong việc phát hiện lỗ hổng đã biết là như thế nào? (§6.3)

• RQ4: Hiệu quả của thiết kế nhận biết các lệnh nhảy (jump-aware design) là như thế nào? (§6.4)

• RQ5: Hiệu quả của thiết kế tiền huấn luyện (pre-training design) là như thế nào? (§6.5)

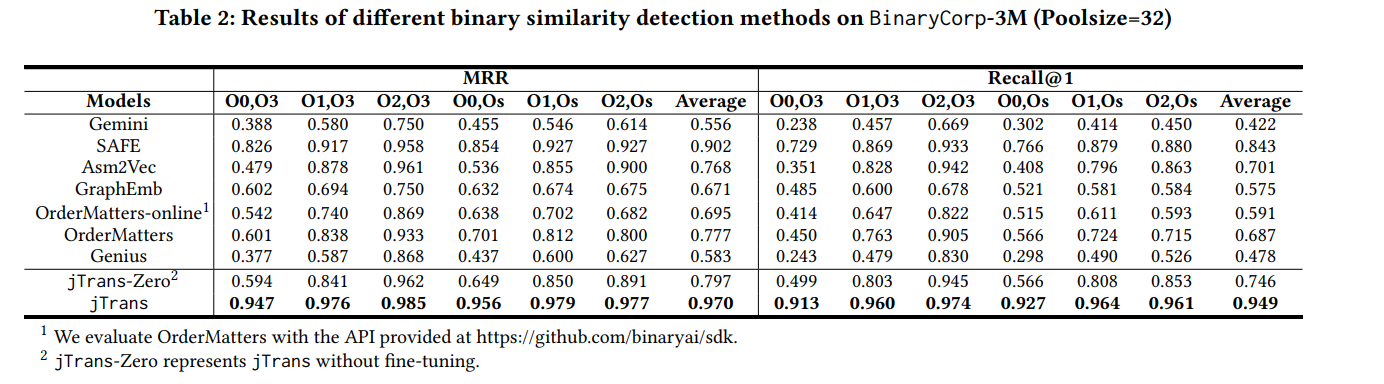
* Trong suốt các thí nghiệm, tất cả các tệp nhị phân đều bị loại bỏ thông tin để tránh rò rỉ thông tin. Chúng tôi sử dụng IDA Pro để disassemble và trích xuất các hàm từ mã nhị phân trong tất cả các thí nghiệm, đảm bảo môi trường công bằng. Đối với các phương pháp cơ sở không sử dụng IDA Pro, chúng tôi sử dụng các framework disassemble mặc định của chúng để tiền xử lý, sau khi đã trích xuất các hàm bằng IDA Pro. Tất cả quá trình huấn luyện và suy luận đều được thực hiện trên máy chủ Linux chạy Ubuntu 18.04 với CPU Intel Xeon 96 core 3.0GHz kèm hyperthreading, RAM 768GB và 8 GPU Nvidia A100.

6.1 Hiệu suất phát hiện tương đồng nhị phân

* Chúng tôi tiến hành đánh giá trên hai bộ dữ liệu của chúng tôi, BinaryCorp-3M và BinaryCorp-26M. Ngoài ra, chúng tôi sử dụng hai kích thước hồ bơi hàm - 32 và 10.000 - để jTrans và các phương pháp cơ sở có thể được đánh giá ở các mức độ khó khăn khác nhau. Cần lưu ý rằng chúng tôi gán ngẫu nhiên các dự án hoàn chỉnh vào tập huấn luyện hoặc tập kiểm tra của các thí nghiệm của chúng tôi, vì các nghiên cứu gần đây đã chỉ ra rằng việc phân chia ngẫu nhiên các nhị phân có thể dẫn đến rò rỉ thông tin.
* Kết quả của các thí nghiệm được trình bày trong Bảng 2-5. jTrans vượt trội hơn so với tất cả các phương pháp cơ sở một cách rõ rệt. Với kích thước hồ bơi hàm là 32 (Bảng 2) và 4, jTrans vượt trội so với phương pháp cạnh tranh gần nhất của nó với mức 0.07 cho độ đo MRR và hơn 10% cho độ đo recall@1. Sự khác biệt về hiệu suất trở nên rõ rệt hơn khi chúng tôi đánh giá các mô hình trên kích thước hồ bơi hàm lớn hơn là 10.000. Với cấu hình này, kết quả được trình bày trong Bảng 3 và 5, jTrans vượt trội hơn so với đối thủ cạnh tranh gần nhất của nó với mức 0.26 cho độ đo MRR và hơn 27% cho độ đo recall@1.
* Kết quả chứng minh ưu điểm của phương pháp được đề xuất của chúng tôi, sử dụng cả kiến trúc Transformer và một phương pháp mới cho biểu diễn và phân tích CFG. Chúng tôi có thể vượt trội đáng kể so với nhiều phương pháp SOTA như SAFE, sử dụng mạng RNN và thực hiện phân tích CFG không nghiêm ngặt, và OrderMatters, sử dụng Transformer nhưng phân tích mỗi khối độc lập nhau.
* Khía cạnh quan trọng khác của kết quả là sự suy giảm tương đối lớn trong hiệu suất của các phương pháp cơ sở với kích thước hồ bơi hàm lớn hơn. Trong phần tiếp theo, chúng tôi sẽ tìm hiểu sâu hơn về vấn đề này.

6.2 Tác động của kích thước hồ bơi hàm đến hiệu suất:

* Kết quả của phần trước nhấn mạnh tác động của biến số kích thước hồ bơi hàm đến hiệu suất các thuật toán phát hiện tương đồng nhị phân. Những kết quả này đặc biệt đáng chú ý vì các nghiên cứu trước đó sử dụng các kích thước hồ bơi hàm tương đối nhỏ giữa 10 và 200, với một số nghiên cứu [43, 59] sử dụng kích thước hồ bơi hàm là 2. Chúng tôi cho rằng các cài đặt như vậy có vấn đề, vì trong các ứng dụng thực tế như phát hiện nhân bản và tìm kiếm lỗ hổng, kích thước hồ bơi hàm thường lớn hơn nhiều lần. Do đó, chúng tôi hiện giờ trình bày một phân tích sâu hơn về tác động của kích thước hồ bơi hàm đến hiệu suất của các phương pháp SOTA trong phân tích nhị phân.



A table of data with numbers

Description automatically generated

A screenshot of a data analysis

Description automatically generated

* Kết quả được trình bày trong Hình 6. Chúng tôi đã tiến hành nhiều thí nghiệm với một loạt các kích thước hồ bơi hàm - 2, 10, 32, 128, 512, 1,000 và 10,000 - và vẽ biểu đồ kết quả cho các cặp tối ưu hóa khác nhau. Kết quả rõ ràng cho thấy hiệu suất tương đối của tất cả các phương pháp cơ sở tồi hơn so với jTrans khi kích thước hồ bơi hàm tăng lên. Hơn nữa, phương pháp của chúng tôi không hiển thị sự giảm sút đột ngột trong hiệu suất (lưu ý rằng trục X trong Hình 6 là logarit), trong khi hiệu suất của các phương pháp cơ sở thường giảm nhanh hơn sau khi đạt được kích thước hồ bơi hàm là 100. Điều này cho thấy phương pháp của chúng tôi không bị ảnh hưởng nhiều bởi kích thước hồ bơi hàm, mà hơn là do bài toán phân loại trở nên khó khăn hơn do có một số lượng lớn ứng viên.
* Cuối cùng, chúng tôi muốn chỉ ra rằng đối với kích thước hồ bơi hàm rất nhỏ (ví dụ, 2), hiệu suất của các phương pháp cơ sở SOTA như SAFE và Asm2Vec gần như tương đương với jTrans, với jTrans vượt trội khoảng 2%. Chúng tôi suy ra rằng đánh giá các công cụ phân tích nhị phân trên các kích thước hồ bơi nhỏ không cung cấp chỉ số có ý nghĩa về hiệu suất của chúng trong các tình huống thực tế.

A graph of different colored lines

Description automatically generated

6.3 Tìm kiếm lỗ hổng thực tế:

* Phát hiện lỗ hổng được coi là một trong những ứng dụng chính trong bảo mật máy tính. Chúng tôi muốn đánh giá hiệu suất của jTrans trong nhiệm vụ tìm kiếm lỗ hổng thực tế. Trong phần này, chúng tôi áp dụng jTrans vào một bộ dữ liệu lỗ hổng đã biết với nhiệm vụ tìm kiếm các hàm có lỗ hổng.
* Chúng tôi tiến hành đánh giá trên tám CVE (Common Vulnerabilities and Exposures) được trích xuất từ một bộ dữ liệu lỗ hổng đã biết [54]. Chúng tôi tạo ra 10 biến thể cho mỗi hàm bằng cách sử dụng các trình biên dịch khác nhau (gcc, clang) và các cấp độ tối ưu hóa khác nhau. Tiêu chí đánh giá của chúng tôi là độ nhớ tại 10 (recall@10).
* Để mô phỏng các tình huống thực tế, chúng tôi sử dụng tất cả các hàm trong dự án làm hồ bơi tìm kiếm. Số lượng hàm cho mỗi dự án dao động từ 3.038 đến 60.159, trong đó dự án có 60.159 hàm được coi là đòi hỏi mức độ thách thức cao.
* Hình 7 trình bày kết quả cho tiêu chí đánh giá recall@10 cho mỗi truy vấn của chúng tôi. Chúng tôi so sánh phương pháp của chúng tôi với hai phương pháp cơ sở hàng đầu từ các phần 6.1 và 6.2 - SAFE và Asm2Vec. Rõ ràng, đối với hầu hết các CVE, hiệu suất của jTrans cao hơn đáng kể so với hai phương pháp cơ sở. Ví dụ, trên CVE-2016-3183 từ dự án openjpeg chứa 3.038 hàm, phương pháp của chúng tôi đạt độ nhớ tại 10 là 100%, có nghĩa là nó đã thành công trong việc tìm lại tất cả 10 biến thể, trong khi Asm2Vec và SAFE đạt giá trị recall@10 là 36.9% và 28.6%, tương ứng. Kết quả của chúng tôi chứng tỏ rằng jTrans có thể được triển khai hiệu quả như một công cụ tìm kiếm lỗ hổng trong các tình huống thực tế, nhờ khả năng hoạt động tốt trên các kích thước hồ bơi lớn.

A graph of different colored bars

Description automatically generated

6.4 Tác động của Thiết kế Nhận biết Cấu trúc Nhảy của Chúng tôi:

* Trong phần này, chúng tôi kiểm tra giả thuyết của mình rằng thiết kế nhận biết cấu trúc nhảy của chúng tôi đóng góp đáng kể vào khả năng phân tích CFG của mã nhị phân của jTrans. Để làm điều này, chúng tôi huấn luyện một mô hình BERT tiêu chuẩn mà không sử dụng biểu diễn thông tin về nhảy của chúng tôi và so sánh nó với phương pháp của chúng tôi. Các siêu tham số được sử dụng bởi cả hai mô hình cho các nhiệm vụ tiền xử lý và điều chỉnh là giống nhau, với sự khác biệt duy nhất là đối với BERT, chúng tôi thay thế địa chỉ của mỗi token nhảy bằng một token cố định đại diện cho một địa chỉ tùy ý. Do đó, tương tự như SAFE, BERT tiêu chuẩn không nhận được thông tin về luồng điều khiển và chỉ có thể học từ thông tin chuỗi lệnh hợp ngữ.
* Chúng tôi đánh giá mô hình BERT tiêu chuẩn và jTrans trên BinaryCorp-3M bằng cách sử dụng Recall@1, với kích thước hồ bơi là 10.000. Kết quả của chúng tôi được trình bày trong Hình 8. Kết quả rõ ràng cho thấy rằng BERT tiêu chuẩn thực hiện kém hơn đáng kể so với jTrans, khi hiệu suất của nó thấp hơn trên mỗi cặp tối ưu hóa. Trung bình, jTrans vượt trội hơn BERT 7.3%. Kết quả này cho thấy rõ ràng rằng việc tích hợp thông tin luồng điều khiển vào việc mô hình hóa chuỗi ngôn ngữ hợp ngữ rất có lợi cho mô hình của chúng tôi.

A graph of bar graph with red and blue bars

Description automatically generated

A number of numbers in a box

Description automatically generated

* Để tiếp tục khám phá tính hiệu quả của thiết kế nhận biết cấu trúc nhảy của chúng tôi, chúng tôi phân tích khả năng của mô hình đã được tiền huấn luyện của chúng tôi trong việc dự đoán các địa chỉ nhảy bị ẩn trong mã nhị phân. Chúng tôi tiến hành thí nghiệm của mình trên BinaryCorp-3M. Đối với mỗi hàm trong tập dữ liệu đánh giá, chúng tôi ngẫu nhiên chọn một số vị trí nhảy với xác suất 15%, và thay thế chúng bằng <LOC> trong hàm. Sau đó, chúng tôi phân tích xác suất mà mô hình dự đoán đúng vị trí đích của mỗi nhảy bị ẩn. Kết quả của chúng tôi, được trình bày trong Bảng 6, cho thấy rằng jTrans có khả năng cao trong việc dự đoán các vị trí nhảy. Mô hình đã được tiền huấn luyện của chúng tôi có thể dự đoán đích của lệnh nhảy với độ chính xác top-1 là 92.9% và độ chính xác top-10 là 99.5%. Độ chính xác này khá cao, đặc biệt là top-1, khi có tổng cộng 512 vị trí nhảy có thể. Kết quả này cho thấy mô hình đã được tiền huấn luyện của chúng tôi đã thành công trong việc thu thập thông tin lệnh ngữ cảnh của mã nhị phân.

6.5 Đánh giá tính hiệu quả của tiền huấn luyện

* Tương tự như BERT gốc, tiền huấn luyện là một thành phần quan trọng của mô hình của chúng tôi. Lợi thế chính của tiền huấn luyện là nó có thể được thực hiện trên dữ liệu không được gán nhãn, điều này dễ dàng thu thập được số lượng lớn hơn. Để đánh giá hiệu quả của tiếp cận tiền huấn luyện (MLM và JTP), chúng tôi đã đánh giá một phiên bản của mô hình của chúng tôi mà không thực hiện bất kỳ điều chỉnh chi tiết nào. Chúng tôi tuân theo cùng một phương pháp như trong học không giám sát [58], trong đó chúng tôi sử dụng các mã nhị phân không có thông tin nhãn trong giai đoạn tiền huấn luyện. Sau đó, mà không cần điều chỉnh chi tiết tiền huấn luyện của mô hình, chúng tôi ngay lập tức áp dụng nó vào nhiệm vụ tìm kiếm tương tự nhị phân. Kết quả của mô hình này, được gọi là jTrans-zero, được trình bày trong các Bảng 2-5.
* Kết quả rõ ràng cho thấy tính hiệu quả của phương pháp tiền huấn luyện của chúng tôi. Ngay cả khi không thực hiện điều chỉnh chi tiết, jTrans-zero vẫn vượt trội hơn tất cả các cơ sở so sánh trong trường hợp poolsize=10000: trên BinaryCorp-26M, so với cơ sở so sánh gần nhất, jTrans-zero cải thiện 0.1 cho phương pháp MRR, và cải thiện 10.6% cho phương pháp recall@1. Tuy nhiên, điều quan trọng cần lưu ý là poolsize=32 ít có ý nghĩa hơn trong các kịch bản thực tế, và trong trường hợp poolsize=10000 khó khăn hơn, ngay cả phương pháp đã được huấn luyện một phần của chúng tôi cũng thể hiện hiệu suất tốt hơn đáng kể.

7. THẢO LUẬN

* Chúng tôi tập trung vào việc huấn luyện jTrans trên một kiến trúc (ví dụ x86) trong bài báo này, nhưng kỹ thuật mà chúng tôi đề xuất cũng có thể được áp dụng cho các kiến trúc khác. jTrans cung cấp một giải pháp mới cho các nhiệm vụ phát hiện tương đồng mã nhị phân, vượt trội hơn các giải pháp tối ưu hiện đại. Nó có thể được áp dụng cho nhiều ứng dụng, bao gồm phát hiện các lỗ hổng đã biết trong các tập tin nhị phân không xác định [11, 18, 49, 50], phát hiện mã độc [4] và phân cụm [30], phát hiện gian lận phần mềm [52], phân tích bản vá [32, 60], và phân tích chuỗi cung ứng phần mềm [27]. Ví dụ, do triển khai nhanh chóng của các thiết bị IoT, việc tái sử dụng mã rất phổ biến trong phát triển IoT. Các giải pháp BCSD như jTrans có thể giúp phát hiện xem liệu các thiết bị IoT có chứa lỗ hổng được tiết lộ trong các thư viện mã nguồn mở hay không. Trong kịch bản của các blockchain, một số lượng lớn các blockchain và hợp đồng thông minh được phát triển trong 5 năm qua, dựa trên nhiều mã nhân bản và nhánh. Tuy nhiên, các rủi ro về an ninh của blockchain và hợp đồng thông minh rất nghiêm trọng, và một phần lớn trong số chúng có lỗ hổng. Sự phụ thuộc mã giữa các blockchain và hợp đồng thông minh khác nhau làm cho vấn đề này trở nên tồi tệ hơn. Chúng tôi có thể sử dụng jTrans để phát hiện lỗ hổng một cách hiệu quả trong các blockchain và hợp đồng thông minh.
* Các công trình dựa trên học sâu hiện có, cũng như jTrans, nhúng các hàm nhị phân cá nhân thành các vector số và so sánh tương đồng giữa các vector đó. Kết quả là độ chính xác của chúng giảm khi kích thước pool tăng. Như đã thể hiện trong Hình 6, độ chính xác của hầu hết các giải pháp hiện có giảm xuống dưới 20% nếu poolsize là 10.000. Trong các kịch bản thực tế, poolsize sẽ lớn hơn nhiều. Một mô hình mà trực tiếp nhận hai hàm nhị phân làm đầu vào có thể giúp bắt kịp được các mối quan hệ giữa các hàm và cải thiện hiệu suất của BCSD, ngay cả trong một pool lớn. Tuy nhiên, huấn luyện một mô hình để trực tiếp so sánh hai hàm sẽ có độ phức tạp cao hơn. Chúng tôi để lại việc cân bằng độ chính xác và chi phí thời gian khi sử dụng jTrans trong các nhiệm vụ BCSD thực tế là một công việc trong tương lai.

8. KẾT LUẬN

* Trong công việc này, chúng tôi đề xuất jTrans, giải pháp đầu tiên để nhúng thông tin luồng điều khiển vào các mô hình ngôn ngữ dựa trên Transformer. Phương pháp của chúng tôi sử dụng một thiết kế kiến trúc thông minh nhận thức về các lệnh nhảy mà không dựa vào việc sử dụng GNN. Phân tích lý thuyết về tự chú ý cho thấy tính hợp lệ của thiết kế của chúng tôi. Kết quả thực nghiệm chứng minh rằng phương pháp của chúng tôi luôn vượt trội hơn các phương pháp tối ưu hiện đại một cách rõ rệt trong các nhiệm vụ BCSD. Thông qua đánh giá tập trung, chúng tôi cũng tiết lộ những điểm yếu trong việc đánh giá các phương pháp SOTA hiện tại. Ngoài ra, chúng tôi trình bày và công bố cho cộng đồng một tập dữ liệu mới được gọi là BinaryCorp. Tập dữ liệu của chúng tôi chứa một lượng lớn các tệp nhị phân đa dạng cho đến nay, và chúng tôi tin rằng nó có thể được sử dụng làm một bài đo kiểm đáng tin cậy cho các nghiên cứu trong lĩnh vực này trong tương lai.

Hết.