***Bindeep slide và thuyết trình.***

***BinDeep: A Deep Learning Approach to Binary Code Similarity Detection.***

***BinDeep: Một phương pháp Học Sâu cho Nhận Diện Sự Tương Tự trong Mã Nhị Phân.***

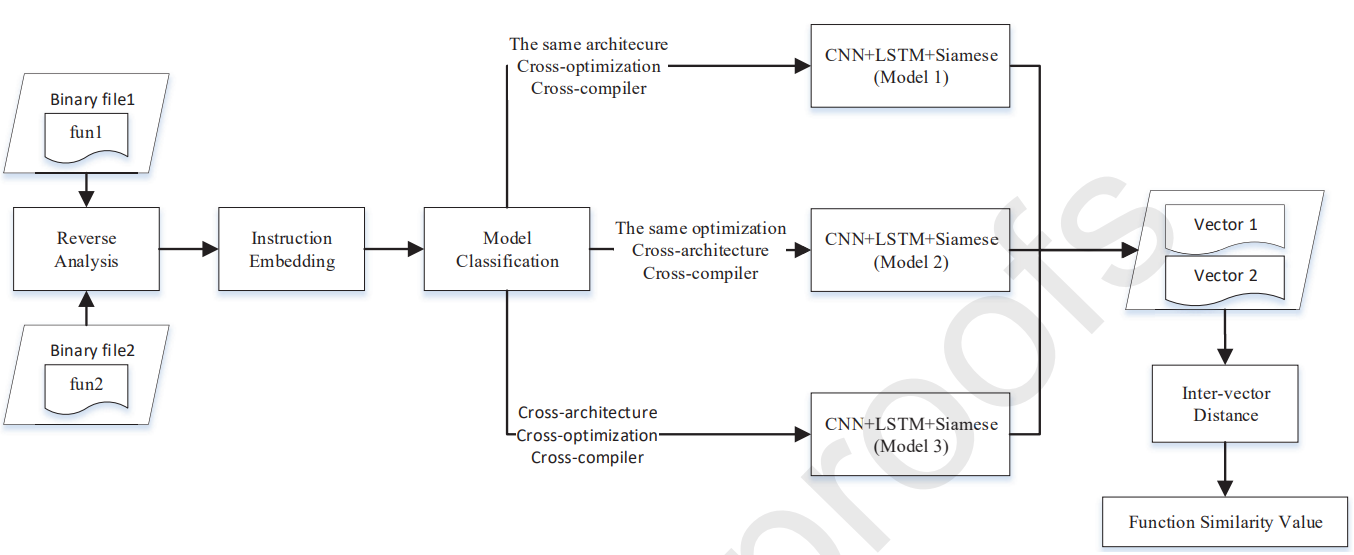
1. ***Giới thiệu:***

* Các giải pháp trước đây để giải quyết vấn đề binary code similarity có thể được chia thành hai loại: Phân tích tĩnh và phân tích động. Nhìn chung các giải pháp này gặp phải những hạn chế sau:
* Hầu hết các phương pháp hiện có chủ yếu dựa vào các đặc điểm cú pháp được thiết kế (Ví dụ: Biểu đồ luồng điều khiển) của mã nhị phân để so sánh độ tương tự.
* Một số phương pháp phát sinh chi phí hiệu suất đáng kể do để áp dụng các cơ chế tốn thời gian (Ví dụ: Khớp đồ thị và mô phỏng).
* Một số phương pháp không thể so sánh tốt độ tương tự của mã nhị phân giữa các kiến trúc CPU khác nhau và các phiên bản chương trình khác nhau.
* Dẫn đến việc phát triển bindeep để giải quyết các hạn chế trên.

***Các công việc em thực hiện ở đây:***

* Đề xuất một giải pháp dựa trên deep learning mới để phát hiện sự tương đồng mã nhị phân. Mô hình này sử dụng mạng hybrid siamese neural để đo độ tương đồng mã nhị phân.
* Sử dụng mô hình instruction embedding để vectơ hóa các hướng dẫn được trích xuất. Áp dụng mô hình phân loại deep learning để xác định các loại chức năng cần so sánh.
* Tiến hành các thử nghiệm rộng rãi để đánh giá cách tiếp cận được đề xuất. Kết quả thử nghiệm cho thấy BinDeep có thể đạt được độ chính xác trung bình, thu hồi và Điểm F1 lần lượt là 97,07%, 98,88% và 97,97% đối với BCSD.

1. ***Framework của BinDeep:***



***Hình 1: Framework Similarity Detection của BinDeep. Đầu vào của khung là hai hàm nhị phân. Đầu ra là giá trị tương tự của hai hàm này.***

* Như mô tả trong Hình 1, Framework của BinDeep có thể được chia thành ba giai đoạn. Ứng với các kỹ thuật được sử dụng như sau:
* ***Kỹ thuật 01:*** Khai thác IDA Pro để phân tích tĩnh mã nhị phân: Sau khi tháo rời các lệnh, chúng ta nhận được một chuỗi lệnh cho mỗi hàm trong mã nhị phân. Để thuận tiện cho việc sử dụng mô hình deep learning, chúng tôi sử dụng mô hình NLP để thực hiện nhúng hướng dẫn. Bằng cách đó, các chuỗi lệnh được chuyển đổi thành các vectơ.
* ***Kỹ thuật 02:*** Sử dụng mô hình deep learning để xác định kiến trúc CPU và mức độ tối ưu hóa của các hàm nhị phân mục tiêu: Theo các loại chức năng đã xác định, chúng tôi sẽ chọn một mô hình thích hợp để phát hiện sự tương tự mã nhị phân. Ưu điểm chính của giai đoạn này là chúng tôi có thể cải thiện khả năng phát hiện sự giống nhau một cách thích hợp. Nhờ giai đoạn này, các kịch bản so sánh khác nhau sẽ dẫn đến việc áp dụng các mô hình so sánh khác nhau trong giai đoạn tiếp theo.
* ***Kỹ thuật 03:*** Sử dụng mạng nơ-ron Siamese để phát hiện sự giống nhau về mã nhị phân: Có ba mô hình mạng Siamese và mỗi mô hình tương ứng với một kịch bản so sánh khác nhau. Cả ba mạng nơron Siamese này đều có cấu trúc giống nhau, nhưng các thông số mạng của chúng khác nhau. Khác với cấu trúc Siamese truyền thống, chúng tôi kết hợp mô hình CNN và LSTM để xây dựng mạng nơ-ron. Sau khi tất cả các mạng này được đào tạo tốt, chúng có thể chuyển đổi các hàm nhị phân tương tự (hoặc khác nhau) thành các vectơ tương tự (hoặc khác nhau). Bằng cách tính toán khoảng cách giữa hai hàm nhị phân, chúng ta có thể nhận được giá trị tương tự của chúng. Nếu giá trị nhỏ hơn ngưỡng được xác định trước, chúng tôi nghĩ rằng hai hàm nhị phân là tương tự. Nếu không, chúng khác nhau.
* ***Ứng với 3 kỹ thuật sẽ có 3 bước trong quá trình DCSD này như sau:***
* ***Bước 1:*** Khai thác và sử lý tính năng.
* ***Bước 2:*** Phân loại mô hình.
* ***Bước 3:*** So sánh sự tương đồng và đưa ra kết quả dự đoán.

1. ***Khai thác và sử lý tính năng:***

* Trong các cách tiếp cận trước đây, hầu hết chúng đều dựa vào kiến thức có sẵn để trích xuất đặc trưng.
* Sử dụng IDA Pro để phân tách mã nhị phân và sau đó nhận chuỗi hướng dẫn cho từng chức năng.
* Sử dụng mô hình NLP (Xử lý ngôn ngữ tự nhiên) để xây dựng cách nhúng hướng dẫn của mình.

***Phân tích cấu trúc của một lệnh:***

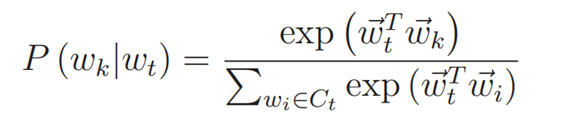
* Một lệnh có thể được chia thành hai phần: Một mã lệnh và một (hoặc nhiều) toán hạng.
* Số lượng loại opcode bị hạn chế trong khi biểu diễn của toán hạng thay đổi rất nhiều trong các bối cảnh điện toán khác nhau.
* Để giải quyết vấn đề này, một phương pháp đơn giản là chỉ sử dụng opcode lệnh làm mã thông báo để nhúng lệnh, bỏ qua các toán hạng lệnh.
* Tuy nhiên, làm như vậy sẽ dẫn đến mất thông tin. Trong thực tế, toán hạng hướng dẫn chứa thông tin ngữ nghĩa quan trọng để so sánh tương tự của các mã nhị phân.
* Để giữ thông tin toán hạng, quá trình chuẩn hóa là cần thiết. Cụ thể, chúng tôi phân loại các toán hạng phổ biến thành 8 loại khác nhau:

Ảnh có chứa bàn

Mô tả được tạo tự động

***Bảng 1: Chuẩn hóa hướng dẫn.***

* Bảng 1 cho thấy các ví dụ về chuẩn hóa lệnh cho kiến trúc x86.
* Sau khi các lệnh thô được chuẩn hóa, bước tiếp theo là thực hiện nhúng lệnh.
* Có hai phương pháp phổ biến để nhúng: One-hot encoding và word2vec.
* ***One-hot encoding:*** Rất đơn giản để biểu diễn một lệnh, nhưng nó không thể nắm bắt được sự liên quan của hai lệnh giống nhau.
* ***Word2vec:*** Có khả năng chuyển đổi các hướng dẫn tương tự thành các vectơ tương tự. Ví dụ: Bằng cách sử dụng word2vec, các lệnh thêm và phụ sẽ được chuyển đổi thành các vectơ tương tự. Word2vec chứa hai mô hình khác nhau: CBOW (Túi từ liên tục) và Skip-gram.
* So với mô hình CBOW, mô hình Skip-gram có thể đạt được hiệu suất tốt hơn trên một tập dữ liệu lớn. Do đó, chúng tôi sử dụng mô hình Skip-gram để xây dựng nhúng hướng dẫn của chúng tôi.
* ***Ý tưởng của mô hình Skip-gram:*** Là sử dụng thông tin ngữ cảnh để tìm hiểu cách nhúng từ trên luồng văn bản.
* Đối với mỗi từ, mô hình ban đầu sẽ thiết lập một vectơ mã hóa one-hot, sau đó nó sẽ được đào tạo khi đi qua từng sliding window.
* Điểm mấu chốt của mô hình là để tìm ra xác suất P của một từ tùy ý wk 1 trong sliding window Ct 2 được nhúng ~wt 3 của từ hiện tại wt. Với mục đích này, chức năng softmax được sử dụng như sau:



* ***Giải thích công thức:*** Trong đó ~wk và ~wi là phần nhúng của các từ wk và wi , ~w T t ~wi là phần giống nhau của hai từ wt và wi . Để đào tạo mô hình trên một chuỗi các từ T, chúng tôi sử dụng hệ số giảm dần độ dốc ngẫu nhiên để giảm thiểu hàm mục tiêu khả năng log J (w) được xác định như sau:

Ảnh có chứa văn bản, đồng hồ

Mô tả được tạo tự động

* Hình 2 minh họa các ví dụ về nhúng lệnh. Đầu vào của mô hình này là một lệnh, đầu ra là một vectơ 300 chiều. Nói cách khác, chuỗi lệnh sẽ được biểu diễn dưới dạng nhiều vectơ 300 chiều. Phần 3.4 cho thấy tác động của các kích thước nhúng khác nhau đối với các thử nghiệm.

A diagram of instructions

Description automatically generated

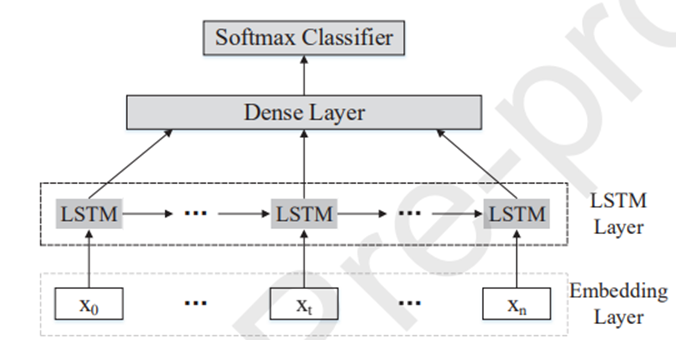
***Hình 2: Quá trình chuyển đổi hướng dẫn assembly thành các vectơ.***

1. ***Phân loại mô hình.***

* Sau khi chuỗi lệnh của các chức năng được vector hóa, bước tiếp theo là training mô hình so sánh dựa trên deep learning để có thể đo lường tốt sự giống nhau của hai chức năng.
* ***Vấn đề:*** Các phương pháp trước đây chỉ sử dụng một mô hình deep learning duy nhất để phát hiện sự tương đồng giữa cross-architecture, cross-compiler, cross-optimization và cross-version binary code. Xem xét các mục tiêu so sánh không đồng nhất, việc sử dụng một mô hình deep learning có thể không đạt được độ chính xác phát hiện tốt trong một số trường hợp.
* Để giải quyết vấn đề này, chúng tôi áp dụng các mô hình deep learning khác nhau cho các kịch bản so sánh khác nhau. Nói cách khác, chúng tôi sử dụng một mô hình phù hợp cho từng tình huống so sánh.
* ***Cụ thể, chúng tôi phân loại các kịch bản so sánh thành ba loại:***
* ***TH1:*** Các kiến trúc CPU giống nhau với cách biên dịch khác nhau.
* ***TH2:*** Cùng mức độ tối ưu hóa với các kiến trúc CPU khác nhau.
* ***TH3:*** Các kiến trúc CPU khác nhau với cách biên dịch và tối ưu hóa khác nhau.
* Theo đó, chúng tôi sử dụng ba mô hình mạng Siamese neural network khác nhau để so sánh sự giống nhau này. Như đã đề cập trước đây, các mô hình này có cấu trúc mạng giống nhau.

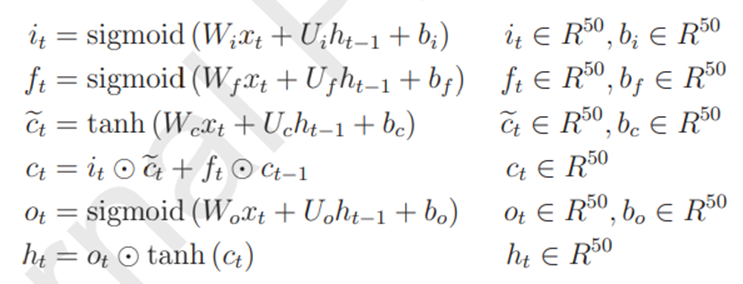
***Cách để chọn một mô hình neural network phù hợp:***

* Chúng ta cần xác định các loại cụ thể của hai chức năng được so sánh.
* Với mục đích này, chúng tôi sử dụng bộ phân loại dựa trên RNN. Bộ phân loại bao gồm hai mô hình: Một mô hình để xác định kiến trúc CPU (x86/x64/arm) và mô hình còn lại để xác định mức độ tối ưu hóa (O0/O1/O2/O3).
* Trong mỗi lớp ẩn của mô hình RNN, đầu ra của thời điểm tiếp theo được xác định bởi đầu ra của thời điểm hiện tại và đầu vào của thời điểm tiếp theo.
* Nhờ thiết kế tái chế này, mô hình RNN có thể xử lý tốt các thông tin tuần tự như câu trong văn bản. Vì các chuỗi hướng dẫn tương tự như các câu nên mô hình RNN phù hợp với nhiệm vụ phân loại để xác định kiến trúc CPU và mức độ tối ưu hóa.
* Mặc dù mô hình RNN mạnh mẽ để xử lý dữ liệu tuần tự, nhưng rất khó để huấn luyện mô hình này do khuếch tán gradient lan truyền ngược hoặc bùng nổ gradient. Để giải quyết vấn đề này, chúng tôi sử dụng mô hình LSTM.



***Hình 3: Kiến trúc Neural network để phân loại mô hình. Đầu vào của mạng là một hàm nhị phân. Đầu ra là loại nhận dạng của chức năng này.***

* Hình 3 minh họa kiến trúc của Neural network trên LSTM của chúng tôi để nhận dạng mức tối ưu hóa biên dịch. Neural network này bao gồm ba lớp.
* ***Lớp thứ nhất:*** Là lớp embedding. Bằng cách sử dụng lại các trọng số được đào tạo trước từ mô hình word2vec, lớp này có thể ánh xạ chuỗi lệnh thành các vectơ 300 chiều. Đầu vào của lớp này là các vectơ nhúng lệnh ~I = ~i1, ...~in,~ij ∈ R, 1 6 j 6 n, n = 1000 và đầu ra của lớp này là một lệnh nhúng khác vectơ ~l = ~l1, ..., ~lm , ~lj ∈ RL , 1 6 j 6 m, m = 1000, L = 300.
* ***Lớp thứ hai:*** Là lớp LSTM và kích thước đầu ra của nó là 50. Mỗi đơn vị LSTM chứa các phương trình sau:



* ***Giải thích các phương trình:*** Trong các phương trình trên, ct ,it , ft , ot biểu thị trạng thái bộ nhớ, cổng vào, quên, cổng ra tại thời điểm t ∈ {1, . . . Toán tử , T}, ⊙ là phép nhân theo điểm, W, U là ma trận trọng số và b là tham số sai lệch.
* ***Lớp cuối cùng:*** Là lớp Desend output dày đặc với bốn đơn vị neural và đầu ra của nó là kết quả phân loại. Để tạo điều kiện cho nhiều phân loại, chúng tôi sử dụng softmax làm chức năng kích hoạt. Để training mạng neural này, chúng tôi sử dụng crossentropy phân loại làm hàm mất mát và áp dụng adam làm trình tối ưu hóa. Để ngăn mạng neural được đào tạo trở nên quá mức, chúng tôi sử dụng dropout và đặt tốc độ của nó thành 0,5.

1. ***So sánh sự tương đồng và đưa ra kết quả dự đoán.***

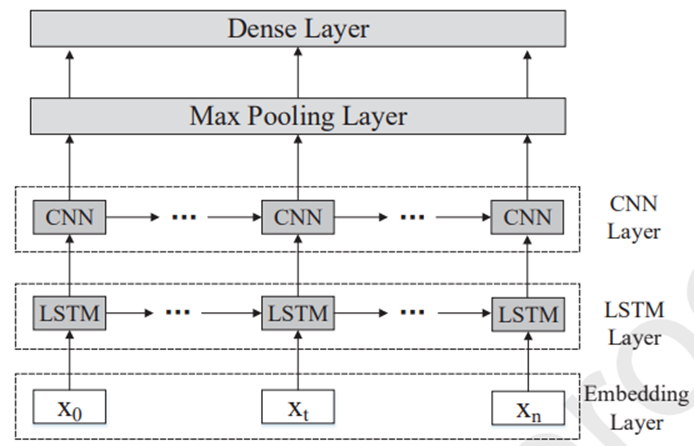
* Sau khi xác định được kiểu của hai hàm nhị phân, chúng ta chọn mô hình mạng neural tương ứng để tính toán độ tương tự.
* Sử dụng mạng Siamese để đo lường sự giống nhau của hai hàm nhị phân.
* ***Ý tưởng của mạng Siamese:*** Là sử dụng hai chức năng ánh xạ giống hệt nhau để chuyển đổi hai đầu vào thành hai vectơ chiều cố định. Bằng cách so sánh khoảng cách của hai vectơ này, chúng ta có thể tìm ra sự giống nhau của hai đầu vào ban đầu.
* Như được hiển thị trong Hình 4, kiến trúc mạng Siamese bao gồm hai mạng lưới neural nhúng giống hệt nhau. Khác với các phương pháp tiêu chuẩn, chúng tôi kết hợp các mô hình CNN và LSTM để xây dựng mạng lưới neural.

A diagram of a computer program

Description automatically generated

***Hình 4: Kiến trúc mạng Siamese để so sánh sự tương đồng. Đầu vào của mạng là hai hàm nhị phân. Đầu ra là kết quả so sánh tương tự.***

* ***Ưu điểm của CNN:*** CNN rất tốt trong việc học cấu trúc không gian.
* ***Ưu điểm của LSTM***: LSTM vượt trội trong việc học trình tự.
* Mô hình kết hợp của chúng tôi tận dụng cả hai ưu điểm của hai mô hình này để có thể cải thiện khả năng phát hiện sự tương đồng dựa trên deep learning. Để dễ trình bày, chúng tôi sử dụng thuật ngữ mô hình lai và mô hình CLSTM (tức là CNN+LSTM) thay thế cho nhau.



Hình 5: Kiến trúc mạng Hybrid neural.

* Hình 5 minh họa cấu trúc của mạng nơ-ron lai. Nó bao gồm 5 lớp.
* ***Lớp embedding:*** Được sử dụng để ánh xạ chuỗi lệnh thành một vectơ có chiều cố định. Lớp này có thể được đại diện bởi một hàm ánh xạ f : X → X~ , trong đó X là vectơ chỉ dẫn và X~ là vectơ ánh xạ. X = (x1, ..., x1000), xj ∈ R, 1 6 j 6 1000; X~ = (~x1, ..., ~x1000), ~xj ∈ RL , 1 6 j 6 1000, L = 300. Lớp này sử dụng lại các trọng số được đào tạo trước từ mô hình word2vec.
* ***Lớp LSTM***: Kích thước đầu ra của nó là 20. Đầu ra của lớp này có thể được biểu thị như sau:

Ảnh có chứa văn bản

Mô tả được tạo tự động

* ***Giải thích công thức:*** Trong đó σ là hàm ReLU, Conv đại diện cho hoạt động tích chập, W2 và b2 là các tham số trọng số và độ lệch của các bộ lọc tích chập. A = (a1, ..., am), ai ∈ RL , 1 6 i 6 m, L = 996, m = 300.
* ***Lớp thứ tư là Max pooling layer.*** Nó được sử dụng để đơn giản hóa các tính năng trích xuất. Kích thước của các cửa sổ gộp tối đa được đặt thành 2 và sải chân cũng được đặt thành 2. Để đơn giản, thao tác gộp tối đa có thể được biểu diễn dưới dạng biểu thức: Ae = M ax(A). Ae = (ae1, ..., afm), aei ∈ RL , 1 6 i 6 m, L = 498, m = 300.
* ***Lớp cuối cùng là dense layer***, có thể giúp kết nối các đặc trưng cục bộ. Đầu ra của nó có thể được biểu diễn như sau:

Ảnh có chứa văn bản, đồng hồ, thiết bị đo

Mô tả được tạo tự động

* ***Giải thích công thức:*** Trong đó · đại diện cho hoạt động sản phẩm chấm, W3 và b3 là các tham số trọng số và độ lệch của lớp này. O = (o1, ..., om), oi ∈ R, 1 6 i 6 m, m = 300.

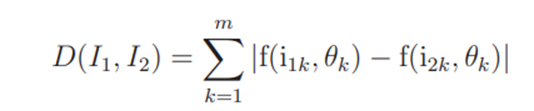
***Khái quát lại:***

* Đầu vào của Siamese network là hai hàm nhị phân, đó là I1 và I2.
* Hai chức năng này có thể được biên dịch từ các kiến trúc CPU, trình biên dịch, mức độ tối ưu hóa và phiên bản chương trình khác nhau.
* Độ dài đầu vào được đặt thành 1000. Nếu hàm chứa ít hơn 1000 lệnh, chúng tôi sẽ sử dụng lệnh nop làm phần đệm. Ngược lại, nếu hàm chứa hơn 1000 lệnh, chúng ta sẽ cắt bớt các lệnh đuôi.
* Đầu ra của các lớp embedding của Siamese network là hai vectơ nhúng, cụ thể là f(I1, θ) và f(I2, θ).
* Ngoài ra, có một đầu vào chỉ báo y cho Siamese network, cho biết liệu hai đầu vào có giống nhau hay không. Cụ thể, nếu y bằng 1 chứng tỏ hai hàm nhị phân đồng dạng, nếu y bằng 0 chứng tỏ hai hàm nhị phân không đồng dạng.

***Xác định hàm mất mát của mạng:***

* Để xác định hàm mất mát của mạng, chúng tôi tận dụng hàm mất mát tương phản. Ý tưởng cơ bản của hàm mất mát này là tối đa hóa khoảng cách giữa hai đầu vào khác nhau, nhưng để giảm thiểu khoảng cách giữa hai đầu vào tương tự. Với mục đích này, hàm mất mát được định nghĩa như sau:





* Trong đó D(I1, I2) biểu thị khoảng cách Manhattan giữa hai hàm nhị phân.Training Siamese network là tìm các tham số θ để cực tiểu hóa hàm mất mát. Để đạt được điều này, chúng tôi sử dụng Adam với thuật toán lan truyền ngược tiêu chuẩn.
* Sau khi Siamese network được đào tạo tốt, chúng ta có thể suy ra hai tuyên bố. Khi hai hàm nhị phân tương tự nhau (Nghĩa là y=1), khoảng cách Manhattan của chúng phải gần bằng 0 để giá trị tổn thất sẽ nhỏ nhất. Khi hai hàm nhị phân không giống nhau (Nghĩa là y=0), khoảng cách Manhattan của chúng phải gần bằng one và giá trị hàm mất mát vẫn sẽ nhỏ nhất.

1. ***Cấu hình máy thực hiện:***

* Các thử nghiệm được thực hiện trên Máy chủ Dell T360 được trang bị hai CPU Intel Xeon E5-2603 V4, bộ nhớ 16GB, ổ cứng 2TB và một card GPU NVIDIA Tesla P100 12GB.
* Sử dụng công cụ IDA Pro 7.0 để trích xuất một chuỗi lệnh từ mỗi hàm nhị phân.
* Các mô hình mạng này được triển khai trong TensorFlow-1.8 và Keras-2.2.

1. ***Tập dữ liệu:***

* Để chuẩn bị tập dữ liệu, chúng tôi chọn 5 gói Linux phổ biến, bao gồm coreutils, findutils, diffutils, sg3utils và util-linux,…
* Sau khi nhận được mã nguồn của gói, chúng tôi sử dụng ba kiến trúc CPU (x86, x86-64 và ARM) và hai trình biên dịch (gcc và clang) với bốn mức tối ưu hóa (O0, O1, O2 và O3) để biên dịch từng chương trình.

Ảnh có chứa bàn

Mô tả được tạo tự động

***Bảng 2: Mô tả các loại samples.***

* Tổng cộng, chúng tôi thu được 4729140 mẫu.
* Như thể hiện trong Bảng 2, các mẫu này có thể được chia thành 5 loại: Cross-compiler, cross-optimization, cross-version, cross-architecture, và mixed function pairs.
* Để đánh giá tính hiệu quả của phương pháp của chúng tôi đối với mã nhị phân không nhìn thấy, toàn bộ tập dữ liệu được chia thành ba tập hợp con riêng biệt để đào tạo, xác thực và thử nghiệm. Chúng tôi đặt tỷ lệ của ba tập hợp con này thành 4:1:1. Tất cả thông tin về biểu tượng gỡ lỗi đều bị loại bỏ trong các mẫu này.

1. ***Các chỉ số đánh giá:***

* Để đánh giá hiệu suất của phương pháp, chúng tôi sử dụng các chỉ số tiêu chuẩn: độ chính xác, độ chính xác, thu hồi, F1 và TPR, được định nghĩa như sau:

Ảnh có chứa văn bản

Mô tả được tạo tự động

* The True positive (TP) biểu thị số cặp hàm khớp được xác định chính xác.
* Kết quả Dương tính Sai (FP) đề cập đến số cặp chức năng được xác định sai khi mô hình deep learning xác định các cặp chức năng chưa khớp là khớp.
* True Negative (TN) đại diện cho số lượng các cặp chức năng chưa khớp được xác định chính xác.
* Phủ định sai (FN) đề cập đến số cặp chức năng chưa khớp được xác định sai.
* Độ chính xác đề cập đến tỷ lệ phần trăm của các cặp chức năng được xác định chính xác. Độ chính xác đo tỷ lệ phần trăm của các cặp chức năng phù hợp được dán nhãn chính xác.
* Recall thể hiện khả năng xác định chính xác các cặp chức năng phù hợp.
* FPR đo tỷ lệ phần trăm của các cặp chức năng chưa khớp được gắn nhãn không chính xác là các cặp chức năng đã khớp.
* Điểm F1 đề cập đến giá trị trung bình hài hòa của Độ chính xác và Độ thu hồi.

1. ***Kết quả thử nghiệm:***
2. ***Hiệu quả của việc xử lý tính năng.***

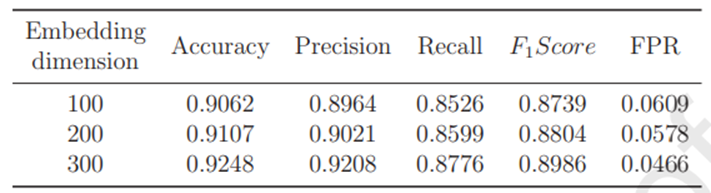
Ảnh có chứa văn bản, bàn

Mô tả được tạo tự động

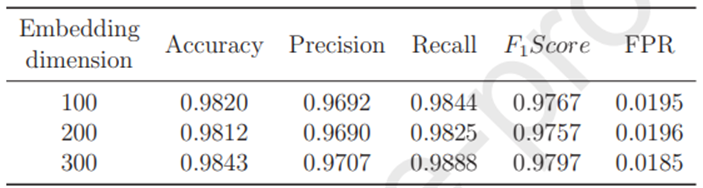
***Bảng 3: Kết quả theo các tính năng hướng dẫn khác nhau.***

* Như được hiển thị trong Bảng 3, phương pháp của chúng tôi tốt hơn các phương pháp chỉ sử dụng opcode làm tính năng trong các chỉ số đánh giá khác nhau, bao gồm độ chính xác, độ chính xác, khả năng thu hồi, điểm F1 và FPR.
* Lý do chính là toàn bộ hướng dẫn chứa nhiều thông tin hơn opcode.

1. ***Hiệu quả của embedding dimension.***



***Bảng 4: Kết quả theo các embedding dimension khác nhau trong mô hình LSTM.***



***Bảng 5: Kết quả theo các embedding dimension khác nhau trong mô hình CLSTM.***

* Trong phần này, chúng tôi khám phá tác động của embedding dimension với kết quả nhận dạng.
* Bảng 4 cho thấy các số liệu nhận dạng của mô hình LSTM. Khi embedding dimension tăng lên, kết quả nhận dạng sẽ tốt hơn.
* Đối với mô hình CLSTM được trình bày trong Bảng 5, khi embedding dimension được thay đổi từ 100 thành 300, kết quả nhận dạng tương đối ổn định.

1. ***Hiệu quả của number of hidden unites.***

A close-up of a graph

Description automatically generated

***Hình 6: Kết quả dưới các đơn vị ẩn khác nhau trong mô hình LSTM và CLSTM.***

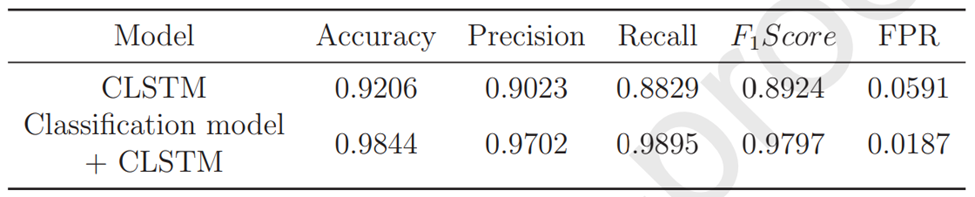
* Để kiểm tra xem số lượng liên kết ẩn có ảnh hưởng đến kết quả nhận dạng hay không, chúng tôi thực hiện các thử nghiệm với số lượng liên kết ẩn khác nhau trong mạng neural từ 2 đến 20.
* Hình 6a và 6b hiển thị kết quả về độ chính xác, khả năng thu hồi, điểm F1 và FPR khi sử dụng mô hình mạng neural LSTM và CLSTM.
* Nói chung, khi số lượng liên kết ẩn tăng lên, độ chính xác và khả năng thu hồi của các mô hình này tăng lên, FPR giảm xuống.
* Trong mô hình LSTM, các số liệu nhận dạng tương tự khi đặt số liên kết ẩn thành 16, 18 và 20. Xem xét càng nhiều liên kết ẩn sẽ dẫn đến chi phí tính toán cao hơn, chúng tôi nghĩ rằng việc đặt số liên kết thành 16 là một sự thỏa hiệp tốt giữa thực dụng và hiệu quả.
* Trong mô hình CLSTM, khi số lượng liên kết ẩn là 18, các chỉ số đánh giá khác nhau là tối ưu.

1. ***Tác dụng của việc thêm mô hình phân loại.***

Ảnh có chứa bàn

Mô tả được tạo tự động

***Bảng 6: Kết quả LSTM và mô hình phân loại + LSTM.***



***Bảng 7: Kết quả của CLSTM và mô hình phân loại + CLSTM.***

* Bảng 6 và Bảng 7 cho thấy kết quả so sánh của mô hình neural network LSTM và CLSTM khi bật/tắt mô hình phân loại.
* Với sự trợ giúp của mô hình phân loại, độ chính xác nhận dạng, độ chính xác, thu hồi và điểm F1 đều được cải thiện và FPR giảm trong các mẫu LSTM và CLSTM.
* Trong đó, độ chính xác, độ chính xác, độ thu hồi và điểm F1 của mẫu CLSTM lần lượt tăng lên lần lượt là 6,38%, 6,79%, 10,66% và 8,73%, FPR của mẫu CLSTM giảm 4,04%.
* Lý do chính cho hiệu quả của việc thêm mô hình phân loại là chúng ta có thể sử dụng mô hình neural network được nhắm mục tiêu nhiều hơn để phát hiện sự tương đồng.

1. ***Hiệu quả của cấu trúc mạng neural network.***

Ảnh có chứa bàn

Mô tả được tạo tự động

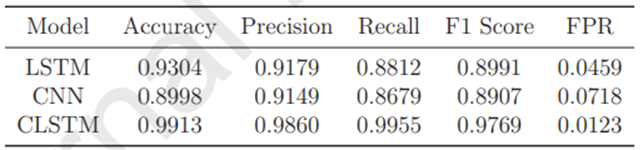
***Bảng 8: Kết quả so sánh các mô hình LSTM, CNN và CLSTM trên bộ dữ liệu hỗn hợp.***

* Để khám phá tác động của cấu trúc mạng đối với phép đo độ tương tự, chúng tôi thực hiện một bộ thử nghiệm trong các tình huống khác nhau.
* Bảng 8 cho thấy kết quả đánh giá của các mô hình neural network CNN, LSTM và CLSTM.
* Trong các mô hình này, embedding dimension được đặt thành 300 và số lượng liên kết ẩn được đặt thành 18.
* Từ bảng này, chúng ta có thể thấy mô hình CLSTM rõ ràng có hiệu suất tốt hơn so với các mô hình CNN và LSTM.

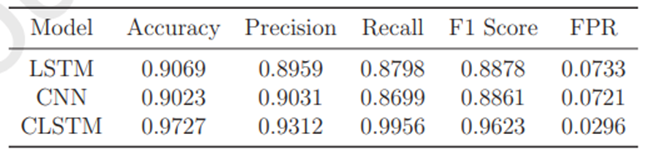
Ảnh có chứa văn bản

Mô tả được tạo tự động

***Bảng 9: Kết quả so sánh các mô hình LSTM, CNN và CLSTM trên bộ dữ liệu kiến trúc chéo.***



***Bảng 10: Kết quả so sánh các mô hình LSTM, CNN và CLSTM trên bộ dữ liệu tối ưu hóa chéo.***



**Bảng 11: Kết quả so sánh các mô hình LSTM, CNN và CLSTM trên bộ dữ liệu trình biên dịch chéo.**

* Bảng 9, bảng 10 và bảng 11 hiển thị kết quả hoạt động trên bộ dữ liệu kiến trúc chéo, trình biên dịch chéo và bộ dữ liệu tối ưu hóa chéo tương ứng. Tương tự, mô hình CLSTM có hiệu suất tốt hơn trong các thử nghiệm này. Đối với hiệu suất trên bộ dữ liệu phiên bản chéo.

Ảnh có chứa bàn

Mô tả được tạo tự động

***Bảng 12: Kết quả so sánh LSTM, CNN và CLSTM trên bộ dữ liệu phiên bản chéo.***

* Bảng 12 minh họa kết quả đánh giá. Trong đánh giá này, tập dữ liệu bao gồm 6 phiên bản của GNU Core Utilities, bao gồm cả phiên bản 8.31. Đánh giá này cũng cho thấy mô hình CLSTM vượt trội hơn so với mô hình CNN và LSTM.

1. ***Hạn chế.***

* Trước khi áp dụng bindeep, chúng ta cần phải thực hiện quy trình giải mã mã nguồn để trích xuất logic bên trong mã nguồn.