***BinShot slide – thuyết trình.***

1. ***Giới thiệu chung:***

***Nội dung Slide:***

* BinShot, một kiến trúc học tương đồng dựa trên BERT có tính chuyển giao cao để thực hiện BCSD hiệu quả.
* Đề xuất Binshot.
* Thiết kế và triển khai Binshot.
* Thực hiện đánh giá.

***Nội dung thuyết trình:***

* Ở phần nội dung này em sẽ thực hiện các công việc sau:
* 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.
* Thiết kế và triển khai nguyên mẫu của BinShot.
* 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.

1. ***Kiến thức Nền tảng: (Bao gồm kiến thức về BERT và Mạng nơ-ron Siamese và Few-shot Learning).***

***BERT:***

* BERT là một kiến trúc máy học dựa trên transformer xuất phát từ lĩnh vực xử lý ngôn ngữ tự nhiên (NLP).

1. ***Kiến trúc Transformer:***

* Áp dụng cơ chế tự chú ý nhằm suy luận thông tin ngữ cảnh cho mỗi từ trong câu đầu vào.
* 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).

1. ***Kiến trúc BERT:***

* 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 (Tiền đào tạo và điều chỉnh lại).

***Tiền đào tạo:***

* 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.
* 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.
* ***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.

***Điều chỉnh lại:***

* 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.

***Mạng nơ-ron Siamese:***

* Mạng nơ-ron Siamese gồm nhiều mạng con đồng nhất trích xuất các đặc trưng từ nhiều đầu vào khác nhau trong quá trình đào tạo.
* Nó tính toán một hàm khoảng cách thể hiện cách hai vectơ đặc trưng gần nhau.
* Nghĩa 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.
* 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:



***Few-shot Learning:***

* Học ít dữ liệu khác biệt với học có giám sát 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).
* 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

1. ***Kiến trúc của Binshot:***

***Hình ảnh luồng hoạt động của Binshot:***

A diagram of a computer program

Description automatically generated with medium confidence

***Hình 2: Luồng công việc tổng quan của BinShot bao gồm bốn thành phần: ○1 một bộ tiền xử lý (§4.2) tạo ra một bộ dữ liệu để chuẩn bị cho việc huấn luyện, ○2 một bộ tiền huấn luyện (§4.3) xây dựng một mô hình biểu diễn hợp ngữ (tức là BERT được tiền huấn luyện), ○3 một bộ điều chỉnh tốt hơn (§4.4) xây dựng một mô hình BCSD (tức là BERT được điều chỉnh tốt hơn), và ○4 một bộ dự đoán (§4.5) đánh giá sự tương đồng của một hàm mục tiêu.***

* ***Gồm bốn thành phần như sau:***
* Bộ tiền sử lý.
* Bộ tiền đào tạo.
* Bộ điều chỉnh.
* Trình dự đoán.

1. ***Tiền sử 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.

A table with numbers and a few black text

Description automatically generated

***Bảng 1: Ngữ liệu nhị phân của chúng tôi. Chúng tôi xây dựng 1,4 nghìn tệp nhị phân chứa 1,77 triệu hàm, thu được 18 nghìn mã thông báo duy nhất sau quá trình chuẩn hóa hướng dẫn cân đối tốt.***

* ***Phân mảnh (Disassembly Process).*** Đầu tiên, thực hiện việc phân mảnh mọi lệnh trong một vùng mã của mỗi tệp thực thi bằng công cụ IDA Pro [35]).
* ***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. Để giải quyết vấn đề này, chúng tôi áp dụng chiến lược chuẩn hóa lệnh cân đối được giới thiệu bởi DeepSemantic, 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***.

1. ***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 (Bước 2 trong Hình 2) có nhiệm vụ huấn luyện một mô hình chung cho mã hợp ngữ.
* Nhiệm vụ MLM (Masked Language Modeling). Sử dụng chiến lược giống như BERT, thay thế 15% các biểu thức đầu vào (lệnh) bằng biểu tượng 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

* ***Giải thích công thức:*** 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.
* 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.
* Binshot 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.

1. ***Fine tuner: Mô hình cho Sự tương đồng mã:***

* Ở bước 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.

A diagram of a network

Description automatically generated with medium confidence

***Hình 3: Mạng neural Siamese để xây dựng một mô hình BCSD. Mô hình của chúng tôi học một vector khoảng cách có trọng số từ một bộ dữ liệu đã được gán nhãn (tức là một tập hợp hai hàm và một nhãn). X, W, L, F, FC và p đại diện cho một đầu vào, trọng số, hàm mất mát, vector khoảng cách, một lớp kết nối đầy đủ và xác suất của sự tương đồng mã.***

* 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

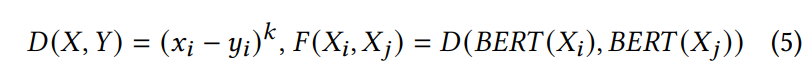
* Sau đó, bộ phân loại nhị phân của chúng tôi học một vectơ khoảng cách có trọng số bằng ***hàm mất mát*** nhị phân cross entropy sau đây:



* ***Giải thích công thức:*** 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)).
* ***Giải thích về hàm mất mát:*** 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.

1. ***Dự đoán cho BCSD:***

* Với mô hình đã điều chỉnh tốt cho BCSD, một bộ dự đoán (Bước 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.
* Khi bộ dự đoán thu thập hai vectơ chức năng với 𝐵𝐸𝑅𝑇 (𝑋𝑖 – Chức năng mục tiêu so sánh)) và 𝐵𝐸𝑅𝑇 (𝑋𝑗 – chức năng nhúng trong cơ sở dữ liệu)), 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:***

* ***Gồm hai bước: Phân tích tĩnh và thực hiện Binshot.***
* ***Phân Tích Tĩnh Binary:*** Sử dụng IDA Pro 7.6 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.
* ***Thực Hiện BinShot:*** Tại trái tim của BinShot, chúng tôi thực hiện nguyên mẫu sử dụng BERT và một mạng neural Siamese.

1. ***Môi trường và thông số đánh giá:***

* ***Môi trường.*** 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.
* ***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

1. ***Tập dữ liệu sử dụng:***

***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.
* 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), Libgmp (v6.2.1), ImageMagick (v7.0.10), Libcurl (v7.78.0), LibTomCrypt (v1.18.2), OpenSSL (v1.1.1f), SQLite (v3.30.1), zlib (v1.2.8), PuTTYgen (v0.76), Nginx (v1.16.1), và vsftpd (v3.0.3). 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.

A table with numbers and text

Description automatically generated

***Bảng 1: Ngữ liệu nhị phân của chúng tôi. Chúng tôi xây dựng 1.4K tệp nhị phân chứa 1.77 triệu hàm, thu được 18K mã thông báo duy nhất sau quá trình chuẩn hóa hướng dẫn cân bằng tốt.***

***Cải tiến Tập dữ liệu. (Thực hiện ở bước 3 trong hình số 4).***

A diagram of a data processing process

Description automatically generated

***Hình 4: Tạo dữ liệu huấn luyện cho việc học sự tương đồng. Tất cả các hàm đã được chuẩn hóa từ ngữ liệu nhị phân gốc (○1) tạo thành toàn bộ bộ dữ liệu (○2) để tiền huấn luyện. Chúng tôi tinh chỉnh một bộ dữ liệu (○3) bằng cách loại bỏ các hàm quá nhỏ/lớn, tiếp theo tạo cặp hàm (○4), chia chúng thành tập huấn luyện, tập xác thực và tập kiểm tra (○5).***

* Chúng tôi cải tiến tập dữ liệu của mình bằng cách loại bỏ một số hàm cụ thể tùy theo kích thước của một hàm (tức là số lượng lệnh bên trong).

***Tạo dữ liệu huấn luyện.***

* Hình 4 mô tả toàn bộ quy trình tạo tập dữ liệu (Bước 2) từ tập dữ liệu nhị phân (Bước 1), được sử dụng trong kiến trúc BERT.
* Mượn ý tưởng chiến lược chuẩn hóa hợp lý của các lệnh, tổng cộng 117.655.422 lệnh (từ 1.770.675 hàm) dẫn đến 18.449 từ vựng.
* 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.
* 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. (Tạo mô hình cho sự tương đồng mã).
* 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.
* Trong thí nghiệm của chúng tôi, chúng tôi tạo tổng cộng 5.259.310 cặp hàm (Bước 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.

1. ***Tính toán hiệu xuất và so sánh với các mô hình trước đây:***

* 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.

A table with text on it

Description automatically generated

***Bảng 2: Tóm tắt về hàm khoảng cách, hàm mất mát và kiến trúc cho các mô hình cơ sở, BinShot và biến thể của nó (\*). GNN và PV-DM đại diện cho Mạng Nơ-ron Đồ Thị và Mô Hình Bộ Nhớ Phân Phối của Véc-tơ Đoạn Văn.***

* ***Mô hình cơ sở.*** Như một mô hình cơ sở, chúng tôi đánh giá các mô hình BCSD tiên tiến hiện nay, bao gồm Gemini, Asm2Vec, PalmTree và DeepSemantic.
* ***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, Asm2Vec, PalmTree và BinShot-CTR.

***Đánh giá tập thử nghiệm:***

* Hình 5 thể hiện tính hiệu quả của BinShot, so sánh với bốn mô hình cơ sở (Gemini, Asm2Vec, PalmTree, DeepSemantic) và biến thể của BinShot (BinShot-CTR).

A screenshot of a graph

Description automatically generated

***Hình 5: Kết quả so sánh BCSD với tập kiểm tra. BinShot vượt trội hơn các mô hình khác với sự biến đổi thấp nhất về độ chính xác và điểm F1.***

* 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.
* 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.

1. ***Khả năng chuyển giao:***

* Ở phần này chúng tôi đặt ra các câu hỏi nghiên cứu sau:
* ***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?***
* ***Mô hình nào hoạt động tốt nhất?***
* ***Lưu ý:*** 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).
* Tương tự, sử dụng Gemini, Asm2Vec, PlamTree, DeepSemantic. 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 chưa từng thấy trước đây.

***Kết quả:***

A graph of different sizes and shapes

Description automatically generated with medium confidence

***Hình 6: Kết quả của các thử nghiệm về khả năng chuyển giao. Như dự kiến, những phỏng đoán đối với cùng bộ dữ liệu (ví dụ: SPEC2006 → SPEC2006, SPEC2017 → SPEC2017) có giá trị cao nhất trên tất cả các chỉ số. Thực nghiệm cho thấy BinShot thể hiện khả năng tổng quát tốt nhất khi áp dụng mô hình của nó vào các bộ dữ liệu khác. Chúng tôi sẽ giải thích tại sao BinShot cho thấy hiệu suất kém trên các tiện ích GNU ở mục 6.3.***

* 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.
* 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.

***Tổng quát lại:***

* Mô hình được điều chỉnh lại với BinShot có thể áp dụng cho các tập dữ liệu khác chưa từng thấy trước đây.
* Tập huấn luyện và kiểm tra được phân phối đều dựa trên hình 6a và hình 6b.
* 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.

1. ***Thử nghiệm 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 O0,01,02,O3).

***Mô phỏng 3 trường hợp giả định sau:***

* Cơ sở dữ liệu của các hàm có lỗ hỗng (được biên dịch bởi GCC).
* Tệp truy vấn đã bị cắt bỏ, và được biên dịch bằng Clang.
* một người cố gắng tìm một hàm có lỗ hỗng trong 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 O0,01,02,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.
* Kết quả cho thấy rằng tất cả các mô hình ngoại trừ DeepSemantic [51] đều phát hiện ra tất cả các hàm có lỗ hổng 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%).