TẠP CHÍ KHOA HỌC VÀ CÔNG NGHỆ ĐẠI HỌC DUY TÂN
DTU Journal of Science and Technology 4(71) (2025) 1-13

ĐẠI HỌC DUY TÂN
DUYTAN UNIVERSITY

Phát hiện lỗ hổng mã nguồn theo hướng tiếp cận Học sâu

Deep learning approaches for detecting vulnerabilities in source code

Hồ Lê Viết Nin[a*], Phan Long[a], Ngô Văn Hiếu[a], Trịnh Quang Tin[b]
Ho Le Viet Nin[a*], Phan Long[a], Ngo Van Hieu[a], Trinh Quang Tin[b]

[a]*Khoa Công nghệ Thông tin, Trường Khoa học Máy tính, Đại học Duy Tân, Đà Nẵng, Việt Nam*
[a]*Faculty of Information Technology, School of Computer Science, Duy Tan University, Da Nang, 550000, Viet Nam*
[b]*Khoa Khoa học Máy tính, Trường Khoa học Máy tính, Đại học Duy Tân, Đà Nẵng, Việt Nam*
[b]*Faculty of Computer Sciences, School of Computer Science, Duy Tan University, Da Nang, 550000, Viet Nam*

(Ngày nhận bài: 25/04/2025, ngày phản biện xong: 15/05/2025, ngày chấp nhận đăng: 28/06/2025)

Tóm tắt

Trong những năm gần đây, phát hiện lỗ hổng bảo mật trong mã nguồn đã trở thành một yêu cầu không thể xem nhẹ trong phát triển phần mềm an toàn. Các phương pháp truyền thống thường gặp hạn chế về khả năng tổng quát hóa và độ hiệu quả khi xử lý các cấu trúc mã phức tạp. Để khắc phục vấn đề này, nhiều nghiên cứu gần đây đã khai thác sức mạnh của các mô hình Deep Learning nhằm tự động hóa quá trình phát hiện lỗ hổng. Bài báo này trình bày một tổng quan có hệ thống về các kiến trúc phổ biến trong lĩnh vực này, bao gồm: Convolutional Neural Network, Long Short-Term Memory, Bidirectional Long Short-Term Memory, Self-Supervised Learning, cùng với các mô hình Transformer hiện đại như CodeBERT và CodeT5. Thông qua việc phân tích hiệu suất các mô hình dựa trên các tiêu chí như độ chính xác, F1-score và chi phí tính toán, bài báo nhằm cung cấp cơ sở định hướng cho việc lựa chọn mô hình phù hợp trong bài toán phát hiện lỗ hổng mã nguồn

Từ khóa: Source Code Vulnerability; Deep learning; CodeBERT; GraphCodeBERT; GPT-4.

Abstract

In recent years, source code vulnerability detection has become an indispensable requirement in the development of secure software. Traditional approaches often face limitations in generalization capability and efficiency when dealing with complex code structures. To address these issues, a growing body of research has leveraged the power of deep learning models to automate the vulnerability detection process. This paper presents a systematic overview of commonly used architectures in this field, including Convolutional Neural Network, Long Short-Term Memory, Bidirectional Long Short-Term Memory, Self-Supervised Learning, along with modern Transformer-based models such as CodeBERT and CodeT5. By analyzing the performance of these models based on criteria such as accuracy, F1-score, and computational cost, the paper aims to provide a foundation for selecting appropriate models for the task of source code vulnerability detection.

Keywords: source code vulnerability; deep learning; CodeBERT; GraphCodeBERT; GPT-4.

[*]*Tác giả liên hệ:* Hồ Lê Viết Nin
Email: holvietnin@dtu.edu.vn

1. Giới thiệu

Trong những năm gần đây, số lượng các cuộc tấn công mạng khai thác lỗ hổng bảo mật trong mã nguồn đã gia tăng đáng kể, gây ra những rủi ro nghiêm trọng đối với an ninh thông tin của các hệ thống phần mềm và hệ thống thông tin và đe dọa trực tiếp đến lợi ích kinh tế và uy tín của nhiều doanh nghiệp và tổ chức tài chính lớn. Theo thống kê, chỉ riêng năm 2023, hơn 26.447 lỗ hổng phần mềm đã được công bố, trong đó nhiều lỗ hổng liên quan trực tiếp đến lỗi lập trình và sai sót trong mã nguồn [1]. Đáng chú ý, thời gian trung bình từ khi một lỗ hổng được công bố đến khi bị khai thác trong thực tế đã giảm xuống còn khoảng 5 ngày trong năm 2023, mức thấp nhất từng được ghi nhận trong thập kỷ qua. Xu hướng này cho thấy các nhóm tấn công đang rút ngắn đáng kể thời gian phản ứng, vượt xa khả năng vá lỗi của nhiều tổ chức trung bình hiện nay [2].

Trong bối cảnh này, việc phát hiện lỗ hổng bảo mật trong mã nguồn trở nên cấp thiết. Các mô hình Học sâu (Deep Learning) đã và đang được nghiên cứu để tự động phát hiện lỗ hổng, nhờ khả năng học đặc trưng trực tiếp từ dữ liệu đầu vào mà không cần thiết kế thủ công – một lợi thế lớn so với các phương pháp truyền thống hoặc học máy thông thường. Điều này giúp nâng cao hiệu suất phát hiện và giảm thiểu sự phụ thuộc vào chuyên gia bảo mật. Các kiến trúc Học sâu phổ biến được áp dụng trong phát hiện lỗ hổng mã nguồn bao gồm: (i) Convolutional Neural Network (CNN) - hiệu quả trong trích xuất đặc trưng cục bộ từ đoạn mã; (ii) Long Short-Term Memory (LSTM) và Bidirectional LSTM (Bi-LSTM) - cho phép học thông tin theo trình tự và hiểu rõ ngữ cảnh mã nguồn; (iii) Self-Supervised Learning (SSL) - khai thác dữ liệu không nhãn để học biểu diễn hiệu quả; và (iv) các mô hình Transformer hiện đại như CodeBERT, CodeT5, sử dụng cơ chế Attention để học ngữ cảnh sâu, đạt độ chính xác cao nhưng đòi hỏi tài nguyên tính toán lớn [3].

2. Mô hình và dữ liệu

Việc phát hiện lỗ hổng trong mã nguồn đòi hỏi các mô hình có khả năng hiểu sâu cả về ngữ nghĩa và cấu trúc của chương trình. Trong phần này, chúng tôi trình bày các kiến trúc mô hình Học sâu tiêu biểu được sử dụng trong những nghiên cứu gần đây, cùng với các bộ dữ liệu huấn luyện phổ biến nhằm hỗ trợ đánh giá và so sánh hiệu quả giữa các phương pháp. Việc kết hợp lựa chọn mô hình phù hợp và dữ liệu chất lượng đóng vai trò then chốt trong việc xây dựng hệ thống phát hiện lỗ hổng bảo mật hiệu quả và có khả năng triển khai thực tế.

2.1. Mô hình Học sâu trong phát hiện lỗ hổng

Các mô hình Học sâu đã và đang được triển khai rộng rãi trong bài toán phát hiện lỗ hổng nhờ khả năng học tự động biểu diễn ngữ nghĩa, phát hiện mẫu nguy hiểm và suy luận từ ngữ cảnh phức tạp trong mã nguồn. Dưới đây là một số kiến trúc tiêu biểu được sử dụng phổ biến trong các nghiên cứu từ năm 2019 đến nay:

• CNN: mô hình sử dụng các lớp tích chập để trích xuất đặc trưng cục bộ từ đoạn mã, kết hợp với các lớp pooling để giảm chiều dữ liệu và các lớp Fully Connected cùng Softmax để thực hiện phân loại. CNN thường được sử dụng trong các nghiên cứu đầu tiên như VulDeePecker với đầu vào là chuỗi Word2Vec hoặc cây cú pháp trừu tượng (AST) [4].

• LSTM: mô hình xử lý mã nguồn như một chuỗi thời gian, sử dụng Embedding Layer để ánh xạ token sang không gian vector. LSTM có khả năng lưu giữ thông tin dài hạn, giúp phát hiện các quan hệ logic xuyên suốt trong đoạn mã có chiều sâu ngữ nghĩa [5].

• Bi-LSTM: là phần mở rộng của LSTM, cho phép học thông tin từ cả hai chiều của chuỗi mã. Việc tích hợp thêm Attention Mechanism giúp mô hình tập trung vào các vùng mã quan trọng, từ đó nâng cao độ chính xác trong nhận diện lỗ hổng [6].

• SSL: mô hình được tiền huấn luyện bằng kỹ thuật Masked Language Modeling trên tập dữ liệu không gán nhãn, sau đó fine-tune trên dữ liệu có nhãn cho nhiệm vụ phát hiện lỗ hổng. Một ví dụ điển hình là CodeBERT, sử dụng kiến trúc Transformer làm backbone và kết hợp với lớp phân loại Multi-Layer Perceptron (MLP) [7].

• Transformer-based Models (CodeBERT, CodeT5): CodeBERT sử dụng Masked Language Modeling để học biểu diễn ngữ nghĩa, trong khi CodeT5 được xây dựng dựa trên mô hình T5 và áp dụng kỹ thuật Masked Span Prediction để học được các mối quan hệ sâu hơn trong đoạn mã dài. Khi tinh chỉnh cho nhiệm vụ phát hiện lỗ hổng, các mô hình này sử dụng attention động và lớp MLP để phân loại [8].

Mặc dù mỗi kiến trúc mô hình Học sâu đều có những đặc điểm riêng, việc so sánh tổng quan giúp làm rõ ưu điểm, hạn chế và điều kiện ứng dụng phù hợp của từng mô hình. Bảng dưới đây trình bày một số tiêu chí so sánh giữa năm kiến trúc phổ biến được sử dụng trong bài toán phát hiện lỗ hổng mã nguồn.

Bảng 1. So sánh một số mô hình Học sâu tiêu biểu trong phát hiện lỗ hổng mã nguồn

Tiêu chí	CNN	LSTM	Bi-LSTM	SSL	CodeT5
Hướng xử lý	Cục bộ	Một chiều	Hai chiều	Phi tuyến	Hai chiều
Khả năng học ngữ cảnh	Thấp	Trung bình	Tốt	Cao	Rất cao
Dữ liệu yêu cầu	Có nhãn	Có nhãn	Có nhãn	Không nhãn	Không nhãn
Thích hợp cho	Mã ngắn	Mã tuần tự	Cấu trúc phức tạp	Mã không nhãn	Mã dài, ngữ cảnh sâu
Tính linh hoạt triển khai	Cao	Trung bình	Trung bình	Cao	Thấp (yêu cầu GPU)

2.2. Dữ liệu huấn luyện

Để huấn luyện và đánh giá các mô hình một cách hiệu quả, dữ liệu đóng vai trò quan trọng trong việc đảm bảo tính đại diện và khả năng tổng quát hóa của mô hình. Một tập dữ liệu tốt không chỉ cần có kích thước đủ lớn mà còn phải phản ánh đa dạng các kiểu lỗ hổng, ngữ cảnh mã nguồn, cũng như sự phân bố hợp lý giữa các lớp. Ngoài ra, dữ liệu cần được gắn nhãn chính xác và thể hiện được các đặc trưng quan trọng của lỗ hổng thực tế nhằm hỗ trợ mô hình Học sâu khai thác triệt để thông tin đầu vào. Việc lựa chọn tập dữ liệu phù hợp giúp cải thiện chất lượng huấn luyện, tránh hiện tượng học lệch (bias) và góp phần tăng độ tin cậy cho quá trình đánh giá mô hình. Bảng 2 mô tả một số bộ dữ liệu được sử dụng phổ biến trong nghiên cứu phát hiện lỗ hổng phần mềm hiện nay.

Bảng 2. Bảng mô tả các bộ dữ liệu phổ biến

Tập dữ liệu	Mô tả	Nguồn gốc / Ứng dụng
Juliet Test Suite	Bộ dữ liệu từ NIST, gồm mã có và không có lỗi thuộc nhiều loại CWE	Chuẩn công nghiệp, đánh giá mô hình học máy [9]
SATE IV	Mẫu lỗi rõ ràng từ công cụ kiểm thử, đa ngôn ngữ	Dùng để benchmark hệ thống phát hiện lỗi
Vuldeepecker Dataset	Gắn nhãn từ GitHub và NVD, phục vụ mô hình CNN	Trích xuất đặc trưng từ mã có gắn dòng lỗi
Devign Dataset	Mã có/không lỗi từ GitHub, dùng cho GNN và Transformer	Dữ liệu thực tế, giàu ngữ cảnh [10]
CodeXGLUE	Tập đa nhiệm, gồm phát hiện lỗi, dịch mã, tóm tắt...	Chuẩn benchmark cho mô hình học sâu hiện đại

Các bộ dữ liệu trên có phạm vi và định hướng khác nhau. Trong khi Juliet Test Suite và SATE IV được thiết kế để kiểm tra mô hình học máy truyền thống với mẫu lỗi có cấu trúc rõ ràng, thì VulDeePecker và Devign cung cấp dữ liệu từ môi trường thực tế, phù hợp hơn với các mô hình Học sâu hiện đại như CNN hoặc Transformer. CodeXGLUE là lựa chọn tốt cho các mô hình pre-trained đa nhiệm nhờ tính đa dạng và tính ứng dụng cao.

Bảng 3. Bảng so sánh các bộ dữ liệu theo một số tiêu chí

Tiêu chí	Juliet	SATE IV	VulDeePecker	Devign	CodeXGLUE
Gắn nhãn lỗ hổng	Có	Có	Có	Có	Có
Mã thực tế hay tổng hợp	Tổng hợp	Tổng hợp	Thực tế	Thực tế	Cả hai
Hỗ trợ nhiều ngôn ngữ	Có	Có	C++/C	C++/Python	Nhiều
Thích hợp cho mô hình	Học máy	Truyền thống	CNN	GNN, Transformer	Transformer
Số lượng mẫu	Rất lớn	Vừa	Trung bình	Vừa	Lớn

Việc lựa chọn bộ dữ liệu phù hợp là rất quan trọng vì chất lượng và độ đa dạng của dữ liệu sẽ ảnh hưởng trực tiếp đến hiệu suất của các mô hình học máy. Ngoài ra, một số nghiên cứu gần đây cũng đề xuất việc tạo ra các bộ dữ liệu tổng hợp hoặc sử dụng phương pháp học có giám sát và không giám sát để phát hiện lỗ hổng trong mã nguồn thực tế.

Bên cạnh các tiêu chí về quy mô và độ phong phú, một vấn đề quan trọng khác cần được xem xét là nguy cơ mất cân bằng giữa các lớp nhãn, vốn có thể ảnh hưởng tiêu cực đến quá trình huấn luyện và đánh giá mô hình.

Cụ thể, Juliet Test Suite được xây dựng theo nguyên tắc kiểm soát, nên thường đảm bảo tỷ lệ tương đối cân bằng giữa mã chứa và không chứa lỗ hổng. Trong khi đó, các tập như VulDeePecker và Devign, vốn thu thập từ môi trường mã nguồn thực tế, lại có xu hướng nghiêng mạnh về các mẫu "an toàn" (non-vulnerable), khiến lớp "vulnerable" trở thành thiểu số.

Sự mất cân đối này có thể ảnh hưởng tiêu cực đến quá trình huấn luyện, làm mô hình dễ dự đoán lệch về lớp chiếm ưu thế và bỏ sót các trường hợp lỗi nghiêm trọng. Một số nghiên cứu đã đề xuất các hướng tiếp cận để giảm thiểu tác động, chẳng hạn như tái cân bằng dữ liệu (resampling), điều chỉnh trọng số trong hàm mất mát, hoặc áp dụng đánh giá chéo nhiều lần (k-fold cross-validation) để đảm bảo tính khách quan.

Việc nhận diện và xử lý vấn đề này ngay từ giai đoạn tiền xử lý dữ liệu có vai trò then chốt trong việc xây dựng mô hình ổn định và có độ bao phủ cao.

3. Một số nghiên cứu liên quan

Trong những năm gần đây, đã có nhiều nghiên cứu được đề xuất nhằm cải thiện hiệu quả phát hiện lỗ hổng mã nguồn thông qua các mô hình Học sâu. Mỗi nghiên cứu tập trung vào một hướng tiếp cận nhất định, từ việc trích xuất đặc trưng cục bộ bằng CNN, đến học chuỗi bằng LSTM, và tiến xa hơn là sử dụng các mô hình tiên tiến như SSL và Transformer. Hiệu quả của các mô hình Học sâu này thường được đánh giá thông qua các chỉ số phổ biến trong học máy, bao gồm:

• Accuracy: độ chính xác tổng thể đo lường tỷ lệ các dự đoán đúng so với toàn bộ tập dữ liệu kiểm tra, được tính theo công thức:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

trong đó TP là số mẫu dương tính được dự đoán đúng (True Positive), TN là số mẫu âm tính được dự đoán đúng (True Negative), FB là số mẫu âm tính bị dự đoán sai là dương tính (False Positive), và FN là số mẫu dương tính bị dự đoán sai là âm tính (False Negative).

• Precision: độ chính xác dự đoán dương tính phản ánh mức độ tin cậy khi mô hình dự đoán một mẫu là dương tính:

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

trong đó các ký hiệu mang ý nghĩa tương tự như trong biểu thức (1).

• Recall: khả năng phát hiện dương tính thực sự thể hiện khả năng của mô hình trong việc phát hiện đúng các mẫu dương tính trong tập dữ liệu:

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

trong đó các ký hiệu mang ý nghĩa tương tự như trong biểu thức (1).

• F1-score là chỉ số tổng hợp cân bằng giữa Precision và Recall, được định nghĩa như sau:

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (4)$$

với Precision và Recall được xác định theo biểu thức (2) và (3).

Các nghiên cứu gần đây đã áp dụng các chỉ số này để đánh giá hiệu quả mô hình một cách toàn diện, thay vì chỉ dựa vào độ chính xác tổng thể (Accuracy). Dưới đây là một số nghiên cứu tiêu biểu minh họa cho các hướng tiếp cận khác nhau.

• Akter et al. (2023): Phát hiện lỗ hổng mã nguồn bằng CNN [11]

Ý tưởng: Nghiên cứu này đề xuất một mô hình học sâu sử dụng mạng CNN để phát hiện các đoạn mã chứa lỗ hổng trong phần mềm. Mục tiêu là tự động học các mẫu đặc trưng nguy hiểm trong mã nguồn thông qua các lớp tích chập, thay vì dựa vào các kỹ thuật thủ công hoặc đặc trưng tĩnh như trước đây.

Kỹ thuật và mô hình: Mã nguồn được xử lý thành chuỗi đặc trưng đầu vào, sau đó đưa qua các lớp convolutional 1D để trích xuất mẫu cục bộ. Sau lớp pooling nhằm giảm chiều và tránh quá khớp, đầu ra được kết nối với lớp fully connected và hàm softmax để thực hiện phân loại. Mô hình được huấn luyện bằng thuật toán tối ưu Adam, sử dụng hàm mất mát categorical cross-entropy.

Tập dữ liệu: Dữ liệu được thu thập từ nhiều dự án mã nguồn mở có gắn nhãn sẵn. Tuy bài báo không công bố cụ thể tên tập dữ liệu, nhưng dữ liệu bao gồm các đoạn mã an toàn và chứa lỗ hổng đã được chuẩn hóa trước khi huấn luyện.

Kết quả thực nghiệm: Accuracy: 91.3%; Precision: 90.3%; Recall: 89.5%; F1-score: 90.1%.

Ưu điểm và hạn chế: Mô hình có cấu trúc đơn giản, dễ triển khai và phù hợp với môi trường hạn chế tài nguyên. Tuy nhiên, do chỉ khai thác đặc trưng cục bộ, mô hình chưa tận dụng được ngữ cảnh tổng thể trong mã nguồn, dẫn đến hiệu suất phân loại chưa cao so với các mô hình Học sâu hiện đại.

• Wartschinski et al. (2022): Phát hiện lỗ hổng mã nguồn bằng LSTM [12]

Ý tưởng: Nghiên cứu này đề xuất một hệ thống phát hiện lỗ hổng sử dụng mô hình LSTM kết hợp với biểu diễn mã nguồn bằng Word2Vec. Ý tưởng chính là tận dụng khả năng học chuỗi của LSTM để phát hiện các biểu hiện lỗ hổng xuất hiện theo ngữ cảnh, đặc biệt trong các cam kết sửa lỗi bảo mật được thu thập từ các kho mã nguồn mở. Mục tiêu là phát hiện lỗ hổng ở cấp độ dòng (line-level) mà không cần thông tin phụ trợ như nhãn thủ công hay đặc trưng thiết kế sẵn.

Kỹ thuật và mô hình: Mô hình sử dụng pipeline gồm hai thành phần chính: vector hóa mã nguồn bằng Word2Vec và mô hình LSTM hai lớp với dropout và hàm softmax ở đầu ra. Mỗi đoạn mã đầu vào được xử lý thành chuỗi token rồi nhúng vào không gian vector. Sau khi đi qua LSTM, đầu ra sẽ được phân loại thành lỗ hổng hoặc không.

Tập dữ liệu: Tập dữ liệu được xây dựng từ 1.009 cam kết sửa lỗi bảo mật (vulnerability-fixing commits) từ nhiều dự án mã nguồn mở trên GitHub. Dữ liệu bao gồm 7 loại lỗ hổng phổ biến như SQL Injection, XSS, Remote Code Execution và Command Injection, được gắn nhãn tự động bằng cách đối chiếu với cơ sở dữ liệu CVE.

Kết quả thực nghiệm trung bình: Accuracy: 90.2%; Precision: 89.5%; Recall: 82.5%; F1-score: 85.0%.

Mô hình LSTM cho kết quả tốt ở nhiều loại lỗ hổng khác nhau và có khả năng áp dụng thực tế với dữ liệu thu thập tự nhiên từ dự án mã nguồn mở. Tuy nhiên, mô hình vẫn gặp hạn chế khi xử lý các thay đổi phức tạp giữa các phiên bản mã hoặc cần giải thích kết quả rõ ràng hơn trong môi trường doanh nghiệp.

• Zhang et al. (2022): Phát hiện lỗ hổng mã nguồn bằng Bi-LSTM [13]

Ý tưởng: Nghiên cứu này đề xuất mô hình Học sâu kết hợp giữa Bi-LSTM và cơ chế Attention nhằm khai thác tối đa thông tin ngữ cảnh hai chiều trong mã hợp đồng thông minh. Mục tiêu là giúp mô hình không chỉ ghi nhớ được các đặc trưng quan trọng trong đoạn mã mà còn biết được vị trí nào cần "tập trung" vào để đưa ra dự đoán chính xác nhất về lỗi.

Kỹ thuật và mô hình: Mỗi hợp đồng thông minh được chuyển thành chuỗi opcode có độ dài cố định (6000 opcode), sau đó đưa vào lớp nhúng và xử lý qua hai lớp Bi-LSTM. Cơ chế Attention được đặt sau Bi-LSTM để xác định trọng số đóng góp của từng vị trí trong chuỗi. Đầu ra cuối cùng đi qua lớp fully connected và hàm softmax để phân loại hợp đồng có lỗi hay không. Mô hình được huấn luyện bằng bộ tối ưu hóa Adam với hàm mất mát binary cross-entropy.

Tập dữ liệu: Tập dữ liệu gồm 45.622 hợp đồng thông minh thực tế được thu thập từ nhiều nguồn, trong đó mỗi hợp đồng đã được gắn nhãn có lỗi hoặc không. Dữ liệu được chuẩn hóa về định dạng opcode trước khi đưa vào mô hình.

Kết quả thực nghiệm: Accuracy: 95.4%; Precision: 95.3%; Recall: 95.4%; F1-score: 95.4%.

Ưu điểm và hạn chế: Học được ngữ cảnh đầy đủ hơn so với LSTM, giúp mô hình nhận diện tốt các lỗ hổng phức tạp; tích hợp Attention giúp mô hình tập trung vào các đoạn mã quan trọng, cải thiện hiệu suất; huấn luyện chậm hơn so với LSTM và CNN do phải xử lý hai chiều dữ liệu; cần nhiều tài nguyên tính toán hơn, yêu cầu GPU mạnh để đạt hiệu suất tối ưu.

• Zhang et al. (2023): Phát hiện lỗ hổng mã nguồn bằng SSL [14]

Ý tưởng: Nghiên cứu này áp dụng học tự giám sát (SSL) để huấn luyện mô hình phát hiện lỗ hổng mã nguồn mà không cần nhãn dữ liệu. Bằng cách tận dụng lượng lớn mã nguồn có sẵn, mô hình có thể học biểu diễn đặc trưng hiệu quả trước khi được tinh chỉnh để phát hiện lỗ hổng.

Kỹ thuật và mô hình: Pre-training với Masked Language Model (MLM) - Sử dụng kiến trúc Transformer để che giấu một số phần tử trong đoạn mã và yêu cầu mô hình dự đoán phần bị che giấu; Fine-tuning cho nhiệm vụ phát hiện lỗ hổng: sau khi tiền huấn luyện, mô hình được tinh chỉnh bằng một tập dữ liệu có nhãn để tối ưu hóa cho nhiệm vụ phát hiện lỗ hổng; sử dụng mô hình CodeBERT làm backbone, kết hợp với lớp MLP để phân loại đoạn mã có hoặc không có lỗ hổng tìm ẩn trong đó.

Tập dữ liệu: BigCode Dataset (~10 triệu đoạn mã nguồn từ các dự án của GitHub, không có nhãn, dùng cho pre-training); Devign Dataset (~24.000 đoạn mã; chứa lỗ hổng, có nhãn, dùng cho fine-tuning).

Kết quả thực nghiệm: Accuracy: 95.2%; Precision: 93.5%; Recall: 96.0%; F1-score: 94.7%.

Ưu điểm và hạn chế: Không cần nhiều dữ liệu có nhãn, giảm chi phí thu thập và gắn nhãn dữ liệu; hiệu suất phát hiện cao nhờ việc học biểu diễn mã nguồn từ lượng dữ liệu lớn; thời gian huấn luyện dài do cần pre-training trên tập dữ liệu lớn; yêu cầu tài nguyên phần cứng mạnh (multi-GPU) để đạt hiệu suất tốt nhất.

• Li et al. (2024): Phát hiện lỗ hổng mã nguồn bằng Transformer tối ưu hóa [15]

Ý tưởng: Nghiên cứu này đề xuất một phiên bản Transformer tối ưu hóa dành riêng cho mã nguồn, giúp cải thiện hiệu suất phát hiện lỗ hổng bằng cách tăng cường khả năng diễn giải quan hệ cú pháp và dòng điều khiển trong mã nguồn.

Kỹ thuật và mô hình: Sử dụng CodeT5 làm nền tảng, một biến thể của T5 (Text-to-Text Transfer Transformer) được tinh chỉnh để xử lý mã nguồn; cơ chế Attention có trọng số động giúp mô hình tập trung hơn vào các đoạn mã có khả năng chứa lỗ hổng cao; Pre-training với Masked Span Prediction (MSP): cải thiện khả năng hiểu mã nguồn bằng cách yêu cầu mô hình dự đoán các phần bị che giấu trong một đoạn mã; Fine-tuning trên dữ liệu có nhãn, với một lớp MLP để phân loại lỗ hổng.

Tập dữ liệu: BigCode Dataset (~12 triệu đoạn mã nguồn, dùng cho pre-training); VulnDB Dataset (~30.000 đoạn mã có nhãn về lỗ hổng, dùng cho fine-tuning).

Kết quả thực nghiệm: Accuracy: 96.8%; Precision: 95.2%; Recall: 97.5%; F1-score: 96.3%.

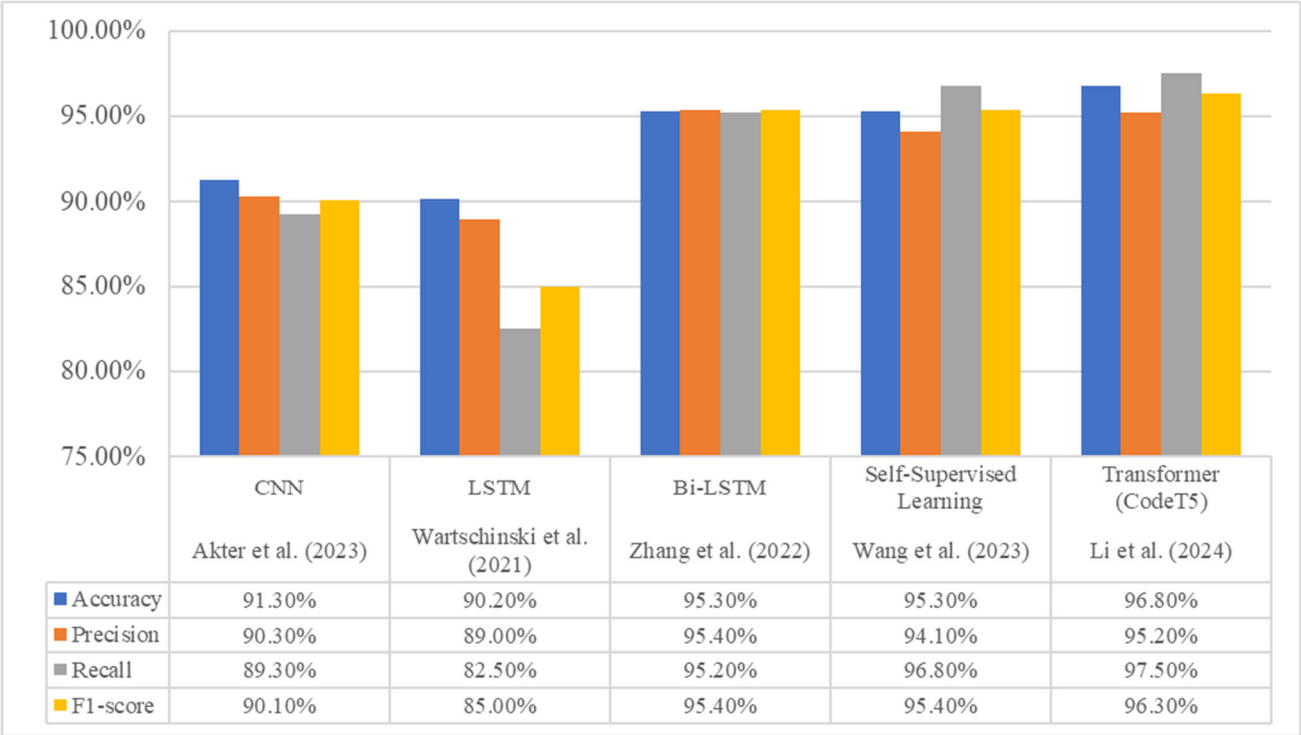
Ưu điểm và hạn chế: Hiệu suất phát hiện lỗ hổng cao nhất trong nhóm Học sâu; mô hình khai thác hiệu quả tương tác ngữ nghĩa giữa các khối chức năng trong chương trình; có thể áp dụng cho nhiều ngôn ngữ lập trình khác nhau (C/C++, Python, Java, Java script,…); yêu cầu tài nguyên phần cứng lớn để huấn luyện và triển khai; chưa được kiểm thử trên mã nguồn thực tế với quy mô lớn.

4. Đánh giá và đề xuất

4.1. Đánh giá

Các mô hình học sâu như CNN, LSTM, Bi-LSTM kết hợp Attention, Self-Supervised Learning và Transformer đã được áp dụng rộng rãi trong phát hiện lỗ hổng mã nguồn với nhiều cách tiếp cận và mức độ hiệu quả khác nhau. Mỗi mô hình có ưu thế riêng về khả năng học đặc trưng, ngữ cảnh và hiệu suất tính toán. Để làm rõ những khác biệt này, phần này trình bày tổng quan kết quả thực nghiệm dưới dạng biểu đồ, so sánh hiệu quả của từng mô hình qua các chỉ số đánh giá phổ biến như Accuracy, Precision, Recall và F1-score. Biểu đồ không chỉ giúp nhận

diện mô hình có hiệu năng vượt trội mà còn phản ánh sự đánh đổi giữa độ chính xác và chi phí tính toán. Đây là cơ sở để định hướng lựa chọn mô hình trong thực tế, phù hợp với yêu cầu về hiệu năng, tài nguyên và độ chính xác.



	CNN Akter et al. (2023)	LSTM Wartschinski et al. (2021)	Bi-LSTM Zhang et al. (2022)	Self-Supervised Learning Wang et al. (2023)	Transformer (CodeT5) Li et al. (2024)
Accuracy	91.30%	90.20%	95.30%	95.30%	96.80%
Precision	90.30%	89.00%	95.40%	94.10%	95.20%
Recall	89.30%	82.50%	95.20%	96.80%	97.50%
F1-score	90.10%	85.00%	95.40%	95.40%	96.30%

Hình 1. Hiệu quả các mô hình học sâu trên các chỉ số đánh giá phân loại

• Từ kết quả thực nghiệm, có thể thấy rằng Transformer (CodeT5) của Li et al. (2024) đạt hiệu suất cao nhất trong các nghiên cứu được xem xét. Với accuracy 96.8%, Precision 95.2%, Recall 97.5%, và F1-score 96.3%, mô hình này thể hiện độ chính xác vượt trội trong việc phát hiện lỗ hổng mã nguồn. Điều này có thể được lý giải bởi khả năng học ngữ cảnh sâu của Transformer, giúp mô hình hiểu được các mối quan hệ phức tạp trong mã nguồn. Tuy nhiên, nhược điểm lớn nhất của phương pháp này là thời gian huấn luyện lên tới 18 giờ, do yêu cầu tính toán cao và số lượng tham số lớn.

• Mô hình Self-Supervised Learning trong nghiên cứu của Wang et al. (2023) cũng đạt hiệu suất rất ấn tượng với Accuracy 95.3%, Precision 94.1%, Recall 96.8% và F1-score 95.4%. Phương pháp này tận dụng lượng lớn mã nguồn không gắn nhãn để tiền huấn luyện, sau đó tinh chỉnh trên dữ liệu có nhãn, giúp giảm đáng kể chi phí thu thập dữ liệu mà vẫn đảm bảo hiệu quả cao trong phát hiện lỗ hổng. Thời gian huấn luyện khoảng 15 giờ, thấp hơn Transformer, nhưng vẫn yêu cầu hệ thống tính toán đủ mạnh (multi-GPU), điều này cho thấy đây là một hướng đi đầy tiềm năng nếu cần cân bằng giữa hiệu năng và chi phí huấn luyện.

• Tiếp theo, mô hình Bi-LSTM do Zhang et al. (2022) đề xuất đạt Accuracy 95.3%, Precision 95.4%, Recall 95.2% và F1-score 95.4%, cho thấy hiệu quả ngang bằng với SSL trong khi không cần pre-training. Bi-LSTM có khả năng xử lý chuỗi theo cả hai hướng, từ trước ra sau và ngược lại, cho phép mô hình nắm bắt bối cảnh đầy đủ hơn so với LSTM thông thường. Mặc dù không có thông tin chi tiết về thời gian huấn luyện, cấu trúc hai chiều của Bi-LSTM thường yêu cầu thời gian xử lý dài hơn so với LSTM, đặc biệt là với dữ liệu đầu vào lớn.

• Mô hình CNN, được trình bày trong nghiên cứu của Akter et al. (2020), đạt Accuracy 91.3%, Precision 90.3%, Recall 89.5% và F1-score 90.1%, nhỉnh hơn so với LSTM trong cùng bảng so sánh. Mặc dù CNN vốn không được thiết kế để xử lý dữ liệu chuỗi như mã nguồn, mô hình này vẫn có khả năng khai thác tốt các đặc trưng cục bộ và cho hiệu quả đáng kể khi áp dụng vào phát hiện lỗ hổng. Kết quả cho thấy rằng với kiến trúc đơn giản, CNN vẫn có thể đạt được độ chính xác tương đối cao, đặc biệt trong các tình huống có cấu trúc mã ổn định và ít phụ thuộc ngữ cảnh. Tuy thời gian huấn luyện không được công bố cụ thể, CNN thường có tốc độ xử lý nhanh hơn so với các mô hình học chuỗi như LSTM và Bi-LSTM, do đó có thể là lựa chọn phù hợp cho các hệ thống yêu cầu phản hồi nhanh, giới hạn tài nguyên hoặc cần triển khai thời gian thực. Tuy nhiên, do chỉ học được các đặc trưng cục bộ, mô hình này vẫn gặp khó khăn trong việc phát hiện các lỗ hổng phụ thuộc vào ngữ cảnh dài hoặc cấu trúc lồng ghép trong mã nguồn.

• Cuối cùng, LSTM trong nghiên cứu của Wartschinski et al. (2021), đạt Accuracy xấp xỉ 89.0%, Precision và Recall lần lượt khoảng 89.0% và 82.5%, với F1-score trung bình khoảng 85.0%, thấp nhất trong số các mô hình học chuỗi được so sánh. LSTM là kiến trúc có khả năng ghi nhớ chuỗi dài hạn một cách hiệu quả hơn so với RNN truyền thống, nhưng do chỉ xử lý dữ liệu theo một chiều (từ đầu đến cuối), mô hình này vẫn bị giới hạn trong việc nắm bắt các quan hệ ngữ cảnh hai chiều hoặc các ràng buộc logic ngược chiều trong đoạn mã. Ưu điểm của LSTM nằm ở cấu trúc đơn giản hơn Bi-LSTM, dễ triển khai hơn Transformer và ít yêu cầu về tài nguyên hơn các mô hình tiên tiến. Tuy nhiên, với kết quả thực nghiệm ở mức trung bình và chưa có thông tin cụ thể về thời gian huấn luyện, mô hình này dường như không phải là lựa chọn tối ưu trong các ứng dụng cần độ chính xác cao hoặc yêu cầu khả năng phát hiện lỗ hổng phức tạp.

Có thể thấy, các mô hình Transformer và Self-Supervised Learning cho kết quả tốt nhất về độ chính xác và khả năng phát hiện lỗ hổng nhưng đòi hỏi thời gian huấn luyện dài và tài nguyên tính toán lớn. Bi-LSTM là một lựa chọn cân bằng giữa độ chính xác và thời gian huấn luyện, trong khi LSTM có độ chính xác thấp hơn nhưng vẫn hiệu quả hơn CNN. CNN mặc dù có tốc độ huấn luyện nhanh nhất, nhưng lại không phù hợp để phát hiện lỗ hổng mã nguồn với độ chính xác cao. Do đó, để nâng cao hiệu quả của hệ thống phát hiện lỗ hổng, việc nghiên cứu và tối ưu hóa các mô hình Transformer là một hướng đi hợp lý.

4.2. Đề xuất của chúng tôi

Những phân tích và so sánh trong mục 4.1 cho thấy rằng mỗi mô hình Học sâu đều có những ưu điểm và hạn chế riêng, phụ thuộc vào ngữ cảnh ứng dụng, dữ liệu đầu vào và mục tiêu triển khai. Nhằm hướng đến việc xây dựng các hệ thống phát hiện lỗ hổng hiệu quả hơn trong thực tế, chúng tôi đề xuất một số hướng cải tiến và phát triển mô hình trong tương lai.

a) Kết hợp mô hình học sâu với phân tích mã truyền thống

Các phương pháp Học sâu có thể được tăng cường hiệu quả khi kết hợp với kết quả từ phân tích tĩnh (AST, CFG, PDG) hoặc phân tích động (trace, call graph). Thay vì huấn luyện mô hình trên chuỗi token thuần túy, việc trích xuất đặc trưng từ các cấu trúc biểu diễn chương trình sẽ giúp mô hình Học sâu hiểu được ngữ cảnh, dòng điều khiển và phụ thuộc dữ liệu.

Chúng tôi đề xuất xây dựng mô hình hybrid, trong đó kết hợp Transformer với đặc trưng phân tích tĩnh/dynamic như một phần embedding input, từ đó nâng cao khả năng phát hiện và giảm cảnh báo sai.

b) Thiết kế mô hình Transformer chuyên biệt cho phát hiện lỗ hổng

Mặc dù CodeBERT, GraphCodeBERT hay CodeT5 đã chứng minh hiệu quả trong hiểu mã

nguồn, các mô hình này chưa được tối ưu hóa cho bài toán phát hiện lỗ hổng. Chúng tôi đề xuất phát triển một mô hình Transformer chuyên biệt, kết hợp giữa biểu diễn dạng chuỗi (sequence-level) và biểu diễn dạng đồ thị (CPG, AST), được huấn luyện đặc thù cho phân loại hàm hoặc dòng chứa lỗ hổng.

Việc bổ sung attention vào luồng điều khiển hoặc dependency-level (graph-aware attention) có thể cải thiện mạnh mẽ hiệu suất so với các phương pháp dựa vào embedding truyền thống.

c) Nâng cao khả năng tổng quát hóa mô hình học sâu

Một hạn chế lớn của mô hình Học sâu là dễ overfit trên dữ liệu huấn luyện và giảm hiệu quả trên mã mới hoặc lỗ hổng chưa từng thấy (zero-day). Để khắc phục, chúng tôi đề xuất kết hợp Transfer Learning từ các mô hình pre-trained như CodeBERT và áp dụng Data Augmentation đặc thù cho mã nguồn – chẳng hạn: đổi tên biến, hoán vị thứ tự lệnh, trộn mã an toàn và mã lỗi.

Ngoài ra, có thể áp dụng kỹ thuật adversarial training để tăng độ ổn định của mô hình trước các biến thể bất thường trong lỗ hổng.

Trong ba hướng đề xuất trên, chúng tôi cho rằng việc kết hợp mô hình Học sâu với kỹ thuật phân tích mã truyền thống (a) và nâng cao khả năng tổng quát hóa thông qua transfer learning và data augmentation (c) là hai hướng khả thi và hiệu quả nhất trong giai đoạn hiện nay. Chúng không chỉ cải thiện hiệu suất phát hiện mà còn đảm bảo khả năng ứng dụng linh hoạt trong các tình huống thực tế với dữ liệu đa dạng và biến đổi phức tạp.

5. Kết luận

Bài báo này đã trình bày tổng quan và phân tích năm mô hình học sâu tiêu biểu được ứng dụng trong phát hiện lỗ hổng mã nguồn, bao gồm: CNN, LSTM, Bi-LSTM kết hợp Attention, SSL và Transformer. Qua đó, có thể thấy rằng các mô hình hiện đại như Transformer và SSL ngày càng khẳng định vị thế nhờ khả năng học biểu diễn ngữ cảnh sâu, tận dụng tốt dữ liệu quy mô lớn, và dễ dàng thích ứng với các nhiệm vụ khác nhau trong khai phá mã nguồn. Trong khi đó, các mô hình Học sâu truyền thống như CNN hay Bi-LSTM vẫn giữ vai trò quan trọng, đặc biệt trong các tình huống yêu cầu cân bằng giữa độ chính xác, tốc độ huấn luyện và tài nguyên tính toán.

Tuy nhiên, chi phí huấn luyện cao và khả năng giải thích còn hạn chế vẫn là những thách thức lớn trong việc triển khai thực tế. Do đó, các hướng nghiên cứu trong tương lai nên tập trung vào việc thiết kế mô hình tối ưu hơn về mặt cấu trúc, giảm phụ thuộc vào dữ liệu gán nhãn, đồng thời tăng cường khả năng giải thích và khả năng mở rộng. Ngoài ra, việc kết hợp Học sâu với các kỹ thuật phân tích mã truyền thống, cũng như khai thác hiệu quả các mô hình pre-trained qua Transfer Learning, có thể mở ra tiềm năng lớn trong việc xây dựng các hệ thống phát hiện lỗ hổng thông minh, chính xác và có khả năng triển khai cao trong môi trường công nghiệp thực tế.

Tài liệu tham khảo

[1] Qualys Threat Research Unit. (2023). *2023 Threat Landscape Year in Review: If Everything Is Critical, Then Nothing Is*. Qualys Security Blog.

[2] Google Cloud Threat Intelligence. (2024). *How Low Can You Go? An Analysis of 2023 Time-to-Exploit Trends*. Google Cloud Blog.

[3] Uddin, M.N., Yeasin, M. & Rahman, M.A. (2025). "Deep learning aided software vulnerability detection: A survey". arXiv preprint arXiv:2503.04002.

[4] Zhou, Y. & Wang, X. (2024). "Software vulnerability detection under poisoning attacks using CNN". International Journal of Information Security.

[5] Sun, K., Chen, Z., Wang, Y. & Liu, J. (2023). "LSTM-based neural network for source code vulnerability classification". Journal of Computer Security, 31(2), 125–143.

[6] Wang, X., Zhang, L., Tan, Y. & Hu, B. (2023). "Detecting smart contract vulnerabilities using BiLSTM with attention mechanism". IEEE Transactions on Dependable and Secure Computing.

[7] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M. & Zhou, M. (2020). "CodeBERT: A pre-

trained model for programming and natural languages". arXiv preprint arXiv:2002.08155.

[8] Liu, X., Ren, S., Duan, N. & Zhou, M. (2024). "Code understanding with CodeT5+: Pretrained transformer models for code representation". arXiv preprint arXiv:2401.05678.

[9] National Institute of Standards and Technology. (2017). *Juliet C/C++ 1.3 Test Suite*. Gaithersburg, MD: U.S. Department of Commerce.

[10] Zhou, Y., Liu, S., Siow, J., Du, X. & Liu, Y. (2019). "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks". Advances in Neural Information Processing Systems, 32, 10197–10207.

[11] Akter, M.T., Rahman, M.T. & Hasib, A.T.M. (2023). "Vulnerability prediction of software using deep learning techniques". Journal of Theoretical and Applied Information Technology, 102(15), 5858–5868.

[12] Wartschinski, L., Noller, Y., Vogel, T., Kehrer, T. & Grunske, L. (2022). "VUDENC: Vulnerability detection with deep learning on a natural codebase for Python". arXiv preprint arXiv:2201.08441.

[13] Zhang, J., Zhang, X., Liu, Z., Fu, F., Jiao, Y. & Xu, F. (2022). "A BiLSTM-attention model for detecting smart contract defects". IEEE Access, 10, 92034–92047.

[14] Wang, Y., Li, X. & Sun, Z. (2023). "Self-supervised learning for source code vulnerability detection". Proceedings of the IEEE International Conference on Machine Learning and Applications (ICMLA), Miami, 2023 (tr. 456–463).

[15] Li, J., Chen, T. & Zhang, R. (2024). "CodeT5-based transformer model for software vulnerability detection". ACM Transactions on Software Engineering and Methodology, 33(1), 1–25.