

Phòng thí nghiệm An toàn thông tin

BÁO CÁO CUỐI KỲ ĐỒ ÁN CHUYÊN NGÀNH

Research on an Effective Method for Detecting Security Vulnerabilities in Python Programs

GVHD: TS. Phan Thế Duy



Nhóm báo cáo: Lê Hồng Hiển - Nguyễn Nhật Quang

Phòng thí nghiệm An toàn thông tin (InSecLab)
Trường ĐH Công nghệ Thông Tin, ĐHQG Tp. HCM





Tài liệu liên quan

- L. Wartschinski, Y. Noller, T. Vogel, T. Kehrer, and L. Grunske, “VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python,” in Information and Software Technology, vol. 144, p. 106809, Apr. 2022, doi: 10.1016/j.infsof.2021.106809.
- M. Ehrenberg, S. Sarkani, and T. Mazzuchi, “Python source code vulnerability detection with named entity recognition,” in Computers & Security, vol. 140, p. 103802, Mar. 2024, doi: 10.1016/j.cose.2024.103802.
- H.-C. Tran, A.-D. Tran, and K.-H. Le, “DetectVul: A statement-level code vulnerability detection for Python,” in Future Generation Computer Systems, vol. 163, p. 107504, Feb. 2025, doi: 10.1016/j.future.2024.107504.
- A. Mechri, M. A. Ferrag, and M. Debbah, “SecureQwen: Leveraging LLMs for vulnerability detection in python codebases,” in Computers & Security, vol. 148, p. 104151, Jan. 2025, doi: 10.1016/j.cose.2024.104151.





Nội dung báo cáo

- **Phần I: Ngữ cảnh vấn đề**
- **Phần II: Kiến thức nền tảng**
- **Phần III: Phương pháp triển khai**
- **Phần IV: Thực nghiệm và đánh giá**
- **Phần V: Kết luận và hướng phát triển**



Bối cảnh và vấn đề

- Trong những năm gần đây, Python đã trở thành một trong những ngôn ngữ lập trình phổ biến nhất trên toàn cầu. Tuy nhiên, sự phổ biến này cũng đi kèm với rủi ro bảo mật, đặc biệt khi sử dụng các thư viện bên thứ ba hoặc mã nguồn mở, vốn có thể chứa lỗ hổng.
- Các phương pháp truyền thống thường gặp hạn chế như sau:

Phương pháp	Ưu điểm	Hạn chế chính
Static Analysis	Nhanh, dễ tích hợp CI/CD pipeline	Dễ bỏ sót lỗi phức tạp, tạo dương tính giả, thiếu ngữ cảnh phân tích
Dynamic Analysis	Phát hiện lỗi khi chạy thực tế	Tốn thời gian, cần môi trường test đầy đủ, khó tái hiện hết trường hợp
Manual Code Review	Chính xác cao khi do chuyên gia thực hiện	Tốn nhân lực, không mở rộng theo dự án lớn, phụ thuộc kinh nghiệm
Machine Learning	Tự động hóa, khả năng học mẫu lỗi	Cần dữ liệu gán nhãn lớn, dễ gây dương tính giả, đòi hỏi tài nguyên tính toán cao



Đề xuất giải pháp

- Ứng dụng các mô hình Học sâu (Deep Learning) và Mô hình Ngôn ngữ lớn (Large Language Models - LLMs) trong việc phát hiện lỗ hổng bảo mật trong mã nguồn Python.
- Phương pháp tiếp cận được chia làm hai hướng chính: dựa trên ngữ nghĩa (semantic) của mã nguồn và dựa trên cấu trúc đồ thị (graph-based), cụ thể là đồ thị phụ thuộc chương trình (Program Dependence Graph - PDG). Việc kết hợp hai hướng tiếp cận này nhằm khai thác tối đa cả thông tin ngữ nghĩa lẫn cấu trúc, từ đó cải thiện hiệu quả phát hiện lỗ hổng phần mềm.





Nội dung báo cáo

- Phần I: Ngữ cảnh vấn đề
- **Phần II: Kiến thức nền tảng**
- Phần III: Phương pháp triển khai
- Phần IV: Thực nghiệm và đánh giá
- Phần V: Kết luận và hướng phát triển



Kiến thức nền tảng

- Học sâu (Deep Learning) là một nhánh của học máy (Machine Learning) tập trung vào việc xây dựng và huấn luyện các mạng nơ-ron nhân tạo sâu (deep neural networks) để tự động học và hiểu dữ liệu.
- Mô hình học sâu được sử dụng ở đây là CodeBERT. Là một mô hình pre-trained language model được thiết kế đặc biệt cho ngôn ngữ lập trình và ngôn ngữ tự nhiên.
- Mô hình ngôn ngữ lớn, còn gọi là LLM, là các mô hình học sâu rất lớn, được đào tạo trước dựa trên một lượng dữ liệu khổng lồ.
- Mô hình LLM được sử dụng ở đây là Qwen2.5-Coder. Là dòng mô hình ngôn ngữ lớn (LLM) chuyên dụng cho lập trình, phát triển bởi nhóm Qwen thuộc Alibaba Cloud, dựa trên kiến trúc Qwen2.5
- Kỹ thuật LoRA (Low-Rank Adaptation) là một phương pháp fine-tuning hiệu quả và tiết kiệm tài nguyên dành cho các mô hình ngôn ngữ lớn





Kiến thức nền tảng

- Đồ thị phụ thuộc chương trình (Program Dependence Graph - PDG) là một đồ thị có hướng mô tả các phụ thuộc dữ liệu và điều khiển giữa các câu lệnh trong một chương trình máy tính
- Mô hình RGCN (Relational Graph Convolutional Network) là một mở rộng của Graph Convolutional Network (GCN), được thiết kế để xử lý các đồ thị có nhiều loại quan hệ (relational graphs)
- Fusion trong máy học là quá trình kết hợp thông tin từ hai hay nhiều nguồn hoặc nhiều mô hình khác nhau nhằm cải thiện hiệu suất, độ ổn định và khả năng tổng quát hóa của hệ thống.
- Các phương pháp fusion trong học máy: Early Fusion (Feature-level Fusion), Late Fusion (Decision-level Fusion), Hybrid Fusion, Ensemble Learning (Fusion Giữa Các Mô Hình),...



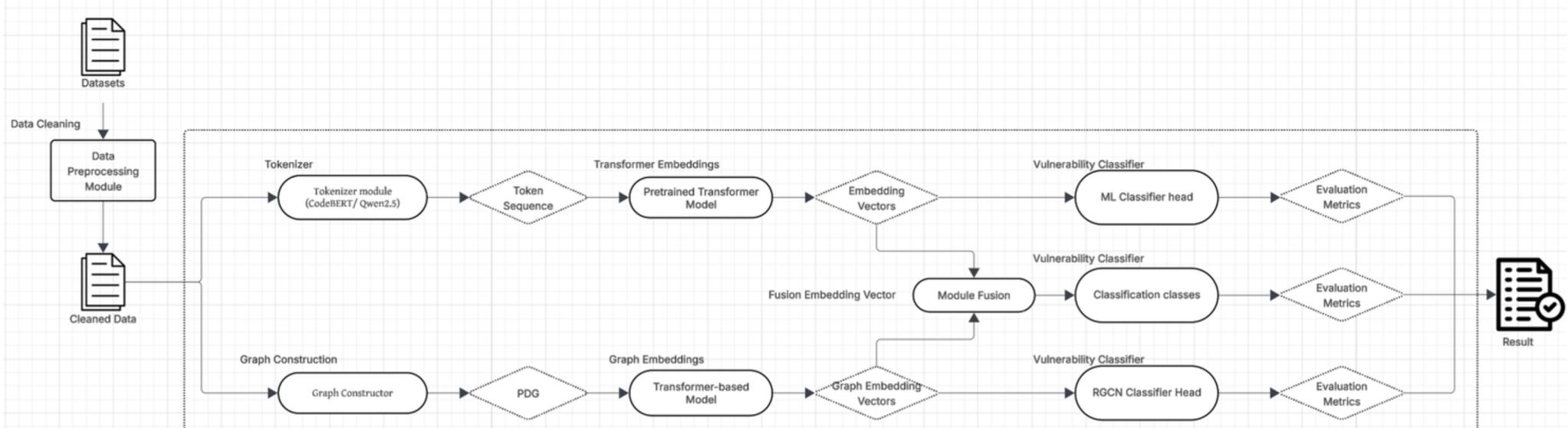


Nội dung báo cáo

- Phần I: Ngữ cảnh vấn đề
- Phần II: Kiến thức nền tảng
- **Phần III: Phương pháp triển khai**
- Phần IV: Thực nghiệm và đánh giá
- Phần V: Kết luận và hướng phát triển



Kiến trúc hệ thống





Luồng hoạt động

1. Tiền xử lí dữ liệu:

- Làm sạch và lọc mã nguồn Python hợp lệ
- Chuẩn hóa định dạng trước khi đưa vào mô hình

2. Hướng tiếp cận ngữ nghĩa (Semantic-based)

- Mã nguồn → Token hóa → Embedding vector (CodeBERT / Qwen2.5)
- Đưa embedding vào lớp phân loại → Dự đoán lỗ hổng
- Train/Validation/Test → Đánh giá bằng các metric (Accuracy, Precision, Recall, F1)
- So sánh giữa model gốc (không fine-tune) và model đã fine-tuned

3. Hướng tiếp cận cấu trúc đồ thị (Graph-based)

- Mã nguồn → Tạo đồ thị PDG (Program Dependency Graph)
- PDG → Node embedding (qua CodeBERT)
- Đưa vào mô hình RGCN để phân loại
- Đánh giá hiệu quả qua các chỉ số metric (Accuracy, Precision, Recall, F1, AUC-ROC)





Luồng hoạt động

4. Kết hợp hai hướng tiếp cận (Fusion)

- Kết hợp embedding từ semantic & graph → Tạo embedding vector hợp nhất
- Đưa vector này vào classifier để dự đoán lỗ hổng
- Thực hiện train/test/validation như các hướng trên
- So sánh hiệu quả tổng thể qua các chỉ số đánh giá





Nội dung báo cáo

- Phần I: Ngữ cảnh vấn đề
- Phần II: Kiến thức nền tảng
- Phần III: Phương pháp triển khai
- Phần IV: Thực nghiệm và đánh giá
- Phần V: Kết luận và hướng phát triển



- Đánh giá hiệu quả của các mô hình trong việc phát hiện lỗ hổng bảo mật trong mã nguồn Python.
- So sánh hiệu suất giữa các hướng tiếp cận:
 - Ngữ nghĩa (Semantic-based): dựa trên biểu diễn embedding của mã nguồn thông qua mô hình DL/LLM như CodeBERT, Qwen2.5.
 - Đồ thị chương trình (Graph-based): dựa trên biểu diễn cấu trúc từ Program Dependence Graph (PDG) kết hợp với RGCN.
 - Kết hợp (Fusion-based): tích hợp ngữ nghĩa và cấu trúc để nâng cao chất lượng phân loại.
- Sử dụng các chỉ số đánh giá chuẩn như Accuracy, Precision, Recall, F1-Score và AUC để đo hiệu quả mô hình.
- Xác định mô hình tối ưu nhất phục vụ cho quy trình phát hiện và cảnh báo lỗ hổng phần mềm tự động.





Môi trường và các công cụ

- Các thí nghiệm được thực hiện trên nền tảng Kaggle, với giao diện chính là Jupyter Notebook, môi trường python 3.11.
- Framework chính được sử dụng: Pytorch
- Thư viện hỗ trợ: scikit-learn, torch, tensorflow, transformers, datasets, pandas, numpy,...
- Phần cứng: Intel(R) Xeon(R) CPU @ 2.20GHz 2 lõi 4 luồng , 30GB RAM, GPU P100 16GB VRAM
- Công cụ gán nhãn bandit





DATASETS

- Nguồn: Hugging Face - codeparrot/github-code-clean
- Ngôn ngữ: Python
- Số lượng mẫu: ≈ 128.000 đoạn mẫu ($\approx 1/5$ tổng tập dữ liệu gốc)
- Tiền xử lý:
 - Xóa comment, docstring, dòng code rỗng
 - Gán nhãn label = 0/1 theo tính chất có lỗ hổng hoặc không
- Chia theo tỉ lệ:
 - Train: 70%
 - Validation: 15%
 - Test: 15%





Kết quả thực nghiệm

	Accuracy	Precision	Recall	F1
CodeBERT (not-finetune)	0,4645	0,2438	0,8094	0,3747
CodeBERT (finetuned)	0,8876	0,7181	0,7124	0,7152
Qwen2.5-Coder (not-finetune)	0,5725	0,2113	0,4232	0,2818
Qwen2.5-Coder (finetuned)	0,9683	0,9468	0,89	0,9175
Graph	0,8255	0,6191	0,7209	0,6602
CodeBERT + Graph	0,8107	0,7407	0,7872	0,7568
Qwen2.5-Coder + Graph	0,8744	0,8384	0,7867	0,8078





Tham số huấn luyện

	CodeBERT	Qwen2.5-Coder	Graph	CodeBERT + Graph	Qwen2.5-Coder + Graph
Learning rate	“1e-4”	“1e-4”	“3e-5”	“3e-5”	“3e-5”
Batch size	16	16	32	16	16
Epochs	3	3	100	3	3
Mixed Precision	fp16	fp16		fp16	fp16
Logging steps	500	500		100	100





So sánh hiệu suất

Hiệu năng giữa các mô hình semantic-only

- Sau khi fine-tuning, cả CodeBERT và Qwen2.5-Coder đều thể hiện khả năng học tập mạnh mẽ từ dữ liệu code/text, nhưng với mức độ khác biệt rõ rệt.
- CodeBERT đạt Accuracy $\approx 89\%$ và F1 ≈ 0.72 , cho thấy nó đã cải thiện rất nhiều so với trạng thái not-fine-tuned nhưng vẫn gặp giới hạn khi phải nắm bắt ngữ cảnh phức tạp.
- Trong khi đó, Qwen2.5-Coder thể hiện ưu thế vượt trội với Accuracy $\approx 97\%$ và F1 ≈ 0.92 , cho thấy mô hình này không chỉ hiểu sâu hơn ngữ nghĩa mà còn sinh code – lý giải code một cách chính xác hơn.



So sánh hiệu suất

Hiệu năng giữa mô hình semantic so với mô hình graph

- Chỉ sử dụng embeddings từ cấu trúc đồ thị (PDG), mô hình Graph-only đạt Accuracy $\approx 84.6\%$ và F1 ≈ 0.71 .
- Điều này cho thấy thông tin cấu trúc ngữ pháp và luồng dữ liệu trong code tự thân đã rất mạnh, đủ để thực hiện phân loại một cách hiệu quả mà không cần textual encoder.
- Graph-only thậm chí gần bằng CodeBERT (fine-tuned) về F1, khẳng định giá trị của các đặc trưng cấu trúc.



So sánh hiệu suất

Hiệu năng sau khi kết hợp semantic + graph

- CodeBERT + Graph: Accuracy giảm nhẹ xuống $\approx 81.1\%$ nhưng F1 tăng lên ≈ 0.76 , tức cải thiện ≈ 4 điểm so với semantic-only, cho thấy Graph đã bù đắp các thiếu sót về ngữ nghĩa của CodeBERT, cân bằng tốt Precision và Recall.
- Qwen2.5-Coder + Graph: Accuracy và F1 đều giảm (Accuracy $\approx 87.4\%$, F1 ≈ 0.81) so với Qwen2.5-Coder thuần túy, chủ yếu do ngữ cảnh token bị cắt ngắn, khiến graph features mất đi nhiều thông tin quan trọng. Tuy nhiên, kết quả này vẫn vượt trội so với Graph-only.





Nội dung báo cáo

- Phần I: Ngữ cảnh vấn đề
- Phần II: Kiến thức nền tảng
- Phần III: Phương pháp triển khai
- Phần IV: Thực nghiệm và đánh giá
- Phần V: Kết luận và hướng phát triển



- Phương pháp tiếp cận ngữ nghĩa (semantic-based): Sử dụng các mô hình ngôn ngữ học sâu như CodeBERT và Qwen2.5-Coder nhằm học biểu diễn ngữ cảnh sâu từ mã nguồn. Kết quả cho thấy phương pháp này mang lại hiệu suất vượt trội, đặc biệt khi được fine-tune, nhờ khả năng hiểu cú pháp, logic và hành vi chương trình từ biểu diễn chuỗi mã.
- Phương pháp tiếp cận cấu trúc đồ thị, sử dụng biểu diễn đồ thị PDG (Program Dependence Graph) nhằm mô hình hóa quan hệ phụ thuộc dữ liệu và điều khiển trong chương trình. Kết quả thực nghiệm cho thấy biểu diễn PDG cung cấp thông tin cấu trúc quan trọng, giúp mô hình đạt độ chính xác cao ngay cả khi không dựa vào biểu diễn ngôn ngữ tự nhiên.
- Phương pháp kết hợp đa biểu diễn (semantic + PDG): Khi tích hợp đồng thời biểu diễn ngữ nghĩa từ các mô hình học sâu và đặc trưng cấu trúc từ PDG, mô hình có khả năng học được cả logic nội tại lẫn ngữ cảnh bao quanh. Cách tiếp cận này giúp cải thiện rõ rệt các chỉ số đánh giá như Accuracy và F1-score so với từng phương pháp đơn lẻ, đồng thời nâng cao tính ổn định và tổng quát của hệ thống phát hiện lỗ hổng.



- Hạn chế về tài nguyên phần cứng (CPU, RAM, GPU) trên các nền tảng thực nghiệm miễn phí như kaggle, colab,... Điều này đặc biệt ảnh hưởng khi làm việc với các mô hình LLM nhiều tham số, hoặc khi cần ngữ cảnh dài và huấn luyện kết hợp đa biểu diễn (semantic + PDG).
- Hạn chế về thời gian thực nghiệm trên các nền tảng khi huấn luyện các mô hình LLM hay huấn luyện kết hợp tốn khá nhiều thời gian trong khi phiên làm việc tối đa chỉ ở mức 12 tiếng.

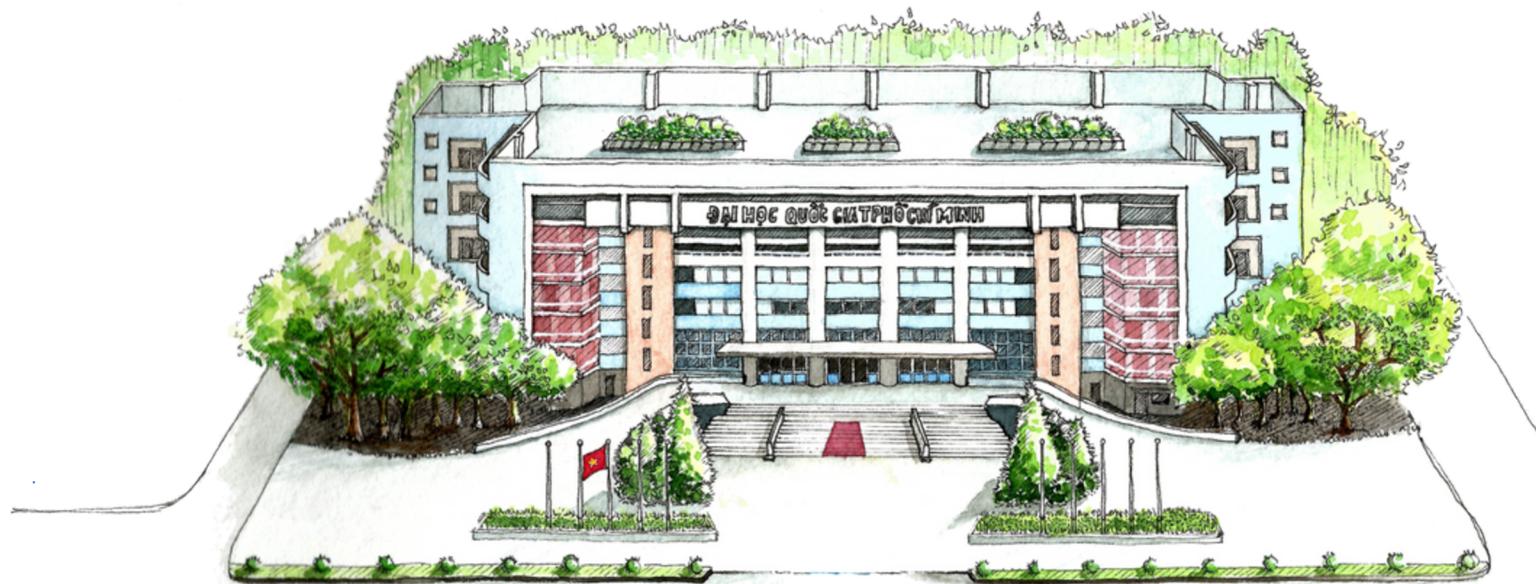




Hướng phát triển

- Mở rộng mô hình với các LLM mới hơn: Thủ nghiệm với các mô hình như CodeT5+, StarCoder, hoặc Claude để cải thiện độ chính xác.
- Tăng kích thước và sự đa dạng của dataset: Sử dụng nhiều nguồn dữ liệu thực tế (ví dụ: CVE, GitHub, CodeQL) để tăng độ bao phủ.
- Mở rộng sang đa ngôn ngữ lập trình: thử nghiệm và đánh giá phương pháp trên các ngôn ngữ khác như JavaScript, C/C++, Java nhằm kiểm tra khả năng khái quát và tái sử dụng mô hình.
- Xây dựng công cụ kiểm thử thực tế: phát triển một hệ thống phát hiện lỗ hổng dưới dạng plugin hoặc web app, hỗ trợ lập trình viên kiểm tra mã nguồn tự động trong môi trường phát triển phần mềm.





**Chúng em xin chân thành cảm ơn Thầy và
các bạn đã lắng nghe !**



Nhóm nghiên cứu InSecLab
**Phòng Thí nghiệm
An toàn thông tin**

Email: inseclab@uit.edu.vn
Website: <https://inseclab.uit.edu.vn/>
Fanpage: <https://www.facebook.com/inseclab>