ĐẠI HỌC BÁCH KHOA HÀ NỘI

BÁO CÁO GIỮA KỲ

Sử dụng thuật toán Học máy để phát hiện rủi ro bảo mật trong DevOps pipelines

VŨ TRUNG THÀNH
thanh.vt220066@sis.hust.edu.vn

Ngành: Khoa học dữ liệu

Giảng viên hướng dẫn: TS. Vũ Thị Hương Giang _____

Chữ kí GVHD

Khoa: Khoa học máy tính

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 04/2025

TÓM TẮT NỘI DUNG ĐỒ ÁN

Trong bối cảnh DevOps đang trở thành tiêu chuẩn trong quy trình phát triển phần mềm hiện đại, vấn đề đảm bảo an toàn bảo mật trong các pipeline CI/CD là một thách thức lớn. Các rủi ro bảo mật có thể phát sinh từ việc rò rỉ thông tin nhạy cảm, thay đổi cấu hình môi trường, hoặc hành vi bất thường trong quá trình tự động hóa. Tuy nhiên, các công cụ kiểm tra hiện tại thường chỉ dựa trên các quy tắc định sẵn, không có khả năng phát hiện sớm những mối đe dọa tiềm ẩn, hoặc những mối đe dọa mới.

Đồ án này đề xuất phương pháp sử dụng học máy, kết hợp giữa mô hình LSTM và các lớp Bayesian, nhằm phát hiện rủi ro bảo mật trong quy trình DevOps thông qua dữ liệu hành vi theo thời gian. Đồ án hướng tới xây dựng một pipeline xử lý dữ liệu, chia chuỗi đặc trưng theo repo và thời gian, sau đó huấn luyện mô hình phát hiện các hành vi bất thường. LSTM được sử dụng để học quan hệ tuần tự, trong khi các lớp Bayesian giúp mô hình đánh giá được mức độ bất định trong dự đoán.

Kết quả thực nghiệm cho thấy mô hình chưa đạt độ chính xác mong muốn, một phần do dữ liệu hạn chế, mất cân bằng giữa các lớp, và sự khác biệt hành vi giữa các repository. Ngoài ra, việc sử dụng lớp Bayesian trong bối cảnh dữ liệu chưa đủ mạnh có thể gây nhiễu trong quá trình huấn luyện. Để cải thiện, đồ án đề xuất mở rộng dữ liệu, áp dụng kỹ thuật cân bằng lớp và thử nghiệm các hướng tiếp cận bán giám sát trong tương lai.

Sinh viên thực hiện

(Ký và ghi rõ họ tên)

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

Thuật ngữ	**Ý nghĩa**
BNN	Mạng nơ-ron Bayes (Bayesian Neural Network)
CI/CD	Tích hợp liên tục và Triển khai liên tục (Continuous Integration / Continuous Deployment)
CVE	Danh sách lỗ hổng và điểm yếu thường gặp (Common Vulnerabilities and Exposures)
LSTM	Bộ nhớ ngắn hạn dài hạn (Long Short-Term Memory)
Metadata	Siêu dữ liệu – dữ liệu mô tả thông tin về dữ liệu khác (ví dụ: ngày tạo, nguồn, loại dữ liệu)
RNN	Mạng nơ-ron hồi tiếp (Recurrent Neural Network)
SAST	Phân tích bảo mật tĩnh (Static Application Security Testing)
YAML	Ngôn ngữ đánh dấu có thể mở rộng (YAML Ain't Markup Language)
Zero-day	Lỗ hổng chưa được công bố hoặc chưa có bản vá tại thời điểm bị khai thác

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong những năm gần đây, DevOps đã trở thành mô hình tiêu chuẩn trong phát triển phần mềm hiện đại nhờ khả năng tích hợp và triển khai liên tục (CI/CD), giúp rút ngắn chu kỳ phát hành và nâng cao chất lượng sản phẩm. Tuy nhiên, chính đặc điểm tự động hóa mạnh mẽ và tốc độ thay đổi cao trong DevOps đã tạo ra một mặt trái: các lỗ hổng bảo mật có thể bị bỏ sót hoặc khai thác mà không bị phát hiện kịp thời.

Một số rủi ro điển hình trong DevOps pipelines bao gồm rò rỉ thông tin nhạy cảm (secrets, credentials), cấu hình sai lệch, sử dụng thư viện độc hại, hoặc hành vi bất thường trong quá trình build/test/deploy. Các công cụ kiểm tra hiện tại phần lớn vẫn dựa trên rule-based scanning (ví dụ: static analysis, SAST) và không phát hiện được các mối nguy tiềm ẩn phát sinh theo thời gian.

Điều này đặt ra nhu cầu cấp thiết cho việc áp dụng các phương pháp học máy, có khả năng phát hiện tự động các dấu hiệu bất thường, thay vì chỉ dựa trên quy tắc định sẵn.

1.2 Các giải pháp hiện tại và hạn chế

Trong bối cảnh bảo mật DevOps, nhiều nghiên cứu và công cụ hiện tại đã được phát triển nhằm phát hiện rủi ro bảo mật trong quá trình CI/CD. Các phương pháp phổ biến có thể chia thành ba nhóm chính:

- **Phân tích tĩnh mã nguồn (SAST)**: Các công cụ như SonarQube, Snyk, hoặc Checkmarx kiểm tra mã nguồn để tìm kiếm các lỗ hổng bảo mật dựa trên các luật định sẵn. Phương pháp này có ưu điểm là nhanh, không phụ thuộc ngữ cảnh runtime, nhưng dễ bỏ sót các rủi ro liên quan đến hành vi động hoặc cấu hình môi trường.
- **Quét bảo mật môi trường và container**: Các công cụ như Trivy, Clair, Anchore, hoặc Docker scan tập trung vào việc phát hiện các dependency chứa lỗ hổng bảo mật (CVE) hoặc cấu hình sai. Các giải pháp này thường được triển khai dưới dạng plugin trong pipeline hoặc hook trong giai đoạn build/deploy. Tuy nhiên, chúng chủ yếu hoạt động theo kiểu dò mẫu và phụ thuộc vào cơ sở dữ liệu CVE, nên khó phát hiện các mối đe dọa chưa biết (zero-day) [1].
- **Phân tích log và hành vi bất thường**: Một số nghiên cứu đề xuất sử dụng mô hình học máy để phân tích log pipeline hoặc hành vi của developer nhằm phát hiện các dấu hiệu bất thường [2]–[4]. Tuy nhiên, các phương pháp này thường

không xử lý tốt dữ liệu theo repo, không tích hợp thông tin thời gian một cách chặt chẽ, và thiếu khả năng đo độ bất định trong dự đoán.

1.3 Mục tiêu và định hướng giải pháp

Mục tiêu chính của đồ án là xây dựng một mô hình học máy có khả năng phát hiện sớm các rủi ro bảo mật trong quy trình DevOps dựa trên dữ liệu thời gian thực từ các pipeline CI/CD. Cụ thể, đồ án tập trung vào các mục tiêu sau:

- Xây dựng pipeline thu thập và tiền xử lý dữ liệu từ nhiều repository DevOps khác nhau.
- Trích xuất đặc trưng từ dữ liệu theo cấu trúc chuỗi thời gian và gán nhãn theo mức độ rủi ro bảo mật.
- Thiết kế và huấn luyện mô hình học máy kết hợp giữa LSTM và lớp Bayesian để học hành vi và phản ánh độ bất định trong dự đoán.
- Đánh giá hiệu quả mô hình trên tập dữ liệu thực tế với nhiều kịch bản phân chia (theo thời gian, theo repository).

1.4 Đóng góp của đồ án

Đề tài có những đóng góp chính sau:

- Xây dựng một phương pháp biểu diễn dữ liệu bảo mật DevOps dưới dạng chuỗi thời gian, giúp làm giàu đặc trưng đầu vào cho các mô hình học máy.
- Đề xuất kiến trúc mô hình kết hợp LSTM và Bayesian layers, có khả năng vừa học được hành vi theo thời gian, vừa phản ánh mức độ bất định trong dự đoán – một yếu tố quan trọng trong hệ thống cảnh báo.
- Thiết kế quy trình đánh giá thực nghiệm theo từng repository độc lập, giúp phân tích khả năng tổng quát hóa và độ ổn định của mô hình trên các môi trường DevOps khác nhau.
- Cung cấp một bộ dữ liệu đã xử lý, có thể được sử dụng cho các nghiên cứu trong lĩnh vực bảo mật DevOps.

1.5 Bố cục đồ án

Phần còn lại của báo cáo đồ án này được tổ chức như sau:

- Chương 2: Cơ sở lý thuyết và các nghiên cứu liên quan – Tổng hợp kiến thức về DevOps, bảo mật trong DevOps và lý thuyết liên quan đến học máy.
- Chương 3: Phương pháp đề xuất – Mô tả pipeline xử lý dữ liệu, kiến trúc mô hình và các bước huấn luyện.
- Chương 4: Thực nghiệm và đánh giá – Trình bày quy trình đánh giá, kết quả

mô hình và phân tích.

- Chương 5: Kết luận và hướng phát triển – Tóm tắt đóng góp, nêu ra hạn chế và đề xuất hướng cải tiến trong tương lai.

CHƯƠNG 2. NỀN TẢNG LÝ THUYẾT

2.1 Tổng quan về DevOps và bảo mật trong DevOps (DevSecOps)

DevOps là một phương pháp kết hợp giữa phát triển phần mềm (Development) và vận hành hệ thống (Operations) nhằm tạo ra quy trình làm việc linh hoạt, tự động hóa và hiệu quả. Mô hình này giúp rút ngắn thời gian triển khai phần mềm, nâng cao chất lượng sản phẩm và giảm thiểu rủi ro trong quá trình phát triển. DevOps hoạt động theo một vòng lặp liên tục, trong đó mỗi giai đoạn đóng vai trò quan trọng trong việc đảm bảo sự ổn định và cải tiến không ngừng của phần mềm.

Trong DevOps pipelines, việc tích hợp bảo mật vào trong quy trình là điều cần thiết để giảm thiểu rủi ro và đảm bảo an toàn cho sản phẩm phần mềm. Do đặc điểm tự động hóa cao và tốc độ triển khai nhanh, DevOps tiềm ẩn nhiều lỗ hổng bảo mật nghiêm trọng nếu không được kiểm soát chặt chẽ.

Một số điểm dễ bị tấn công trong DevOps bao gồm:

- Thông tin nhạy cảm bị lộ trong code, biến môi trường, hoặc log (secrets leakage).
- Tấn công vào CI runner/agent, cài mã độc vào quá trình build.
- Sử dụng thư viện chứa mã độc hoặc có lỗ hổng.
- Cấu hình sai trong file YAML, Dockerfile, hoặc policy.

Trong mô hình phát triển phần mềm truyền thống, bảo mật thường chỉ được xem xét ở cuối quy trình, khi sản phẩm đã gần hoàn thiện. Nếu phát hiện lỗ hổng bảo mật ở giai đoạn này, việc sửa chữa sẽ rất tốn kém và có thể làm chậm tiến độ triển khai. DevSecOps ra đời nhằm tích hợp bảo mật vào từng giai đoạn của quy trình phát triển, giúp phát hiện và xử lý các rủi ro trước khi phần mềm được đưa vào vận hành chính thức [5].

Cốt lõi của DevSecOps là mô hình “shift left", đề cập đến việc di chuyển các hoạt động bảo mật về sớm hơn trong vòng đời phát triển phần mềm - điều này có nghĩa là các vấn đề bảo mật sẽ được phát hiện và khắc phục ngay từ khi phần mềm còn đang được thiết kế, lập trình và kiểm thử.

2.2 Đặc điểm về dữ liệu DevOps

Đặc điểm nổi bật của dữ liệu DevOps có thể kể đến như sau:

- **Đa nguồn**: dữ liệu đến từ nhiều thành phần khác nhau trong pipeline như Git repo, CI log, scan tools, Dockerfile, deployment events...

- **Không đồng đều**: các pipeline khác nhau có tần suất chạy và lịch trình khác nhau (có thể theo giờ, theo lần push, hoặc tùy chỉnh).
- **Gắn với repository**: mỗi repo có cấu trúc, ngôn ngữ, và quy trình CI/CD riêng biệt, dẫn đến các hành vi của hệ thống từ đó khác nhau.
- **Mang yếu tố tuần tự**: các sự kiện diễn ra theo thời gian và có mối liên hệ tuần tự.

Đồ án này tập trung khai thác vào yếu tố tuần tự của dữ liệu. Quy trình DevOps tạo ra một lượng lớn dữ liệu mang tính tuần tự theo thời gian. Các dữ liệu này bao gồm log từ quá trình build/test/deploy, thông tin về các bản quét bảo mật định kỳ, số liệu về số lượng thay đổi mã nguồn, hành vi commit của lập trình viên, v.v.

Một ví dụ điển hình là kết quả quét bảo mật định kỳ cho các repository. Các công cụ như Trivy hoặc Snyk được cấu hình để chạy theo lịch (hằng ngày hoặc mỗi lần build) và sinh ra kết quả quét. Khi ghi nhận các kết quả này trong nhiều ngày liên tiếp, ta thu được một chuỗi dữ liệu có tính liên kết theo thời gian, phản ánh diễn biến an toàn bảo mật của một hệ thống.

Chính những đặc điểm này khiến dữ liệu DevOps trở nên phù hợp cho các mô hình học chuỗi thời gian như RNN, LSTM, hay Transformer. Việc xử lý và biểu diễn đúng dữ liệu này đóng vai trò quan trọng trong việc phát hiện các rủi ro bất thường mang tính chất "ẩn" trong dòng thời gian mà các phương pháp kiểm tra tĩnh hoặc quét mẫu (signature-based) không thể phát hiện được.

2.3 Bayesian Neural Network và LSTM trong bài toán

Các phương pháp học máy (ML) đang dần trở nên phổ biến để thay thế các phương pháp truyền thống trong việc phát hiện rủi ro bảo mật trong DevOps pipelines. Các phương pháp ML vượt trội trong việc thích nghi với môi trường DevOps thay đổi liên tục, nơi mà các quy tắc bảo mật cứng nhắc có thể không hiệu quả. Bên cạnh đó, chúng có thể phát hiện tự động các hành vi bất thường, nằm ngoài danh sách/quy tắc có sẵn.

BNN được mở rộng từ mạng nơ-ron truyền thống, trong đó trọng số được mô hình hóa như các biến ngẫu nhiên thay vì giá trị cố định [6]. Mục tiêu của BNN không chỉ là đưa ra một kết quả dự đoán duy nhất mà còn ước lượng độ không chắc chắn (uncertainty) của dự đoán đó thông qua phân phối hậu nghiệm.

Ưu điểm nổi bật của BNN là khả năng phản ánh mức độ tin cậy của mô hình, giúp xác định những trường hợp mà mô hình "không chắc chắn", từ đó hỗ trợ việc đưa ra cảnh báo trong các tình huống nhạy cảm về bảo mật. Ngoài ra, BNN còn có khả năng tích hợp tri thức tiên nghiệm (prior knowledge) [7], điều này đặc biệt hữu

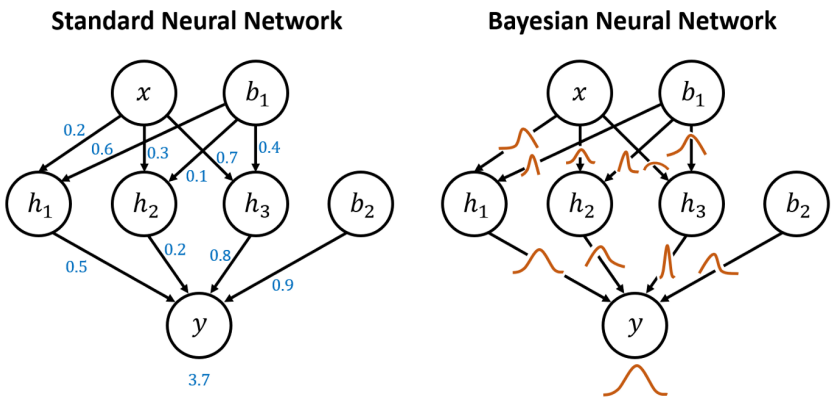

Figure 2.1: So sánh mạng nơ-ron truyền thống và BNN

ích khi làm việc với dữ liệu hạn chế hoặc thiên lệch (biased).

LSTM là một kiến trúc mạng nơ-ron hồi quy (RNN) được thiết kế để xử lý dữ liệu chuỗi thời gian bằng cách ghi nhớ thông tin dài hạn và ngắn hạn thông qua các cổng kiểm soát. Khác với RNN truyền thống dễ bị "quên" thông tin khi chuỗi quá dài, LSTM sử dụng các cơ chế như forget gate và cell state để giữ lại thông tin quan trọng trong quá trình học.

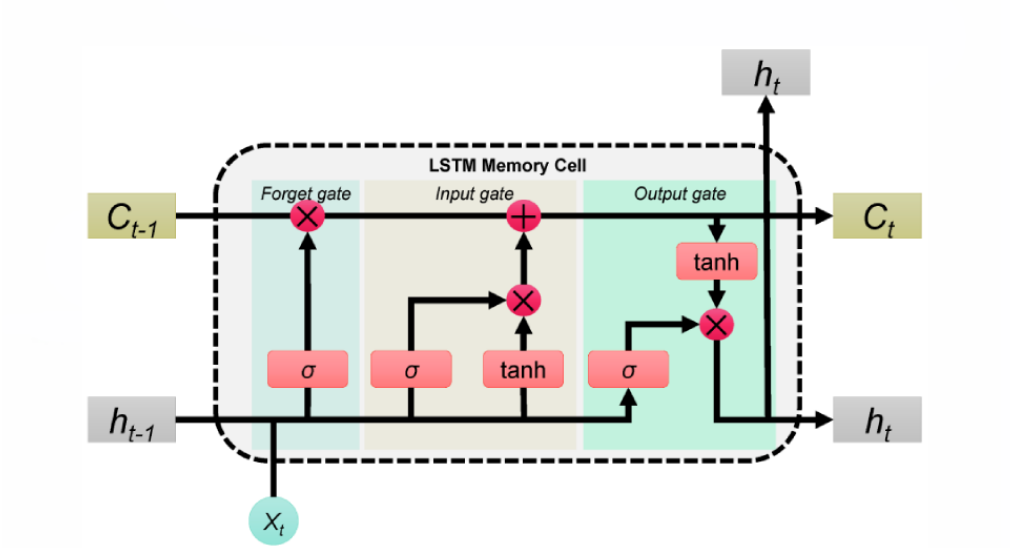

Figure 2.2: Cấu trúc một cell LSTM

RNN nói chung và LSTM nói riêng đặc biệt phù hợp với dữ liệu DevOps dạng chuỗi, như lịch sử quét bảo mật hoặc log hành vi, vì nó có khả năng học được mối quan hệ tuần tự và xu hướng thay đổi theo thời gian, từ đó phát hiện những bất thường xuất hiện ngầm trong quá trình phát triển phần mềm.

CHƯƠNG 3. PHƯƠNG PHÁP ĐỀ XUẤT

3.1 Tổng quan giải pháp

Giải pháp đề xuất trong đồ án hướng tới việc phát hiện sớm các rủi ro bảo mật trong quy trình DevOps bằng cách mô hình hóa hành vi của hệ thống dưới dạng chuỗi thời gian và sử dụng mô hình học máy kết hợp giữa LSTM và Bayesian Neural Networks (BNN) để đưa ra dự đoán.

Quá trình này bao gồm bốn giai đoạn chính:

- Thu thập và tiền xử lý dữ liệu từ các DevOps pipelines thực tế.
- Biểu diễn dữ liệu dưới dạng chuỗi thời gian có cấu trúc.
- Huấn luyện mô hình học máy để phát hiện rủi ro.
- Đánh giá khả năng dự đoán và độ tin cậy của mô hình.

3.2 Tiền xử lý và biểu diễn dữ liệu

Dữ liệu ban đầu thu thập từ các DevOps pipelines bao gồm nhiều cột đặc trưng khác nhau liên quan đến bảo mật, lịch sử thay đổi mã nguồn, môi trường triển khai, và hành vi cập nhật của từng repository. Tuy nhiên, do giới hạn trong khả năng thu thập đồng nhất và đầy đủ các nguồn dữ liệu, chỉ có thể sử dụng 9 đặc trưng đầu vào ổn định và sẵn có nhất cho việc huấn luyện mô hình.

Table 3.1: Các đặc trưng được lựa chọn

Tên đặc trưng	Loại dữ liệu
Số lượng dòng code thay đổi	Số
Số lượng các module dễ bị tấn công	Số
Số người tham gia phát triển	Số
Thời gian hoàn thiện mỗi phiên bản	Rời rạc
Tần suất commit	Rời rạc
Loại môi trường	Rời rạc
Số lượng thư viện có lỗ hổng	Số
Số điểm yếu tiềm năng	Số
Số lượng lỗ hổng cấu hình môi trường	Số

Bên cạnh đó, dữ liệu còn có nhãn là mức độ rủi ro, được chia làm 5 mức từ 0 đến 4, đồng thời có các metadata bao gồm tên nguồn repo thu thập và thời gian thu thập dữ liệu. Dữ liệu được thu thập bằng cách checkout lại các commit cũ theo lịch sử phiên bản.

Để chuẩn bị dữ liệu cho mô hình học máy theo chuỗi thời gian, các bước tiền xử

lý đã được thực hiện, cụ thể như sau:

- **Chuẩn hóa thời gian và phân tách theo repository**: Tập dữ liệu được sắp xếp theo cột thời gian quét bảo mật (Scan date), sau đó được chia thành các nhóm riêng biệt theo từng repository. Mỗi repository được coi như một chuỗi thời gian độc lập.
- **Tổng hợp dữ liệu theo ngày**: Trong mỗi repository, có thể tồn tại nhiều bản ghi trùng ngày (tương đương với nhiều commit được thực hiện trong một ngày). Để tránh dư thừa và đảm bảo tính nhất quán, dữ liệu được gộp theo từng ngày. Cụ thể, các cột dạng định lượng như "số dòng code thay đổi", "số module dễ bị tấn công", "số điểm yếu tiềm năng",... được tính trung bình hoặc tổng sao cho phù hợp với từng loại. Các đặc trưng rời rạc như "loại môi trường", "tần suất commit", "thời gian hoàn thiện phiên bản" được giữ giá trị đầu tiên trong ngày.
- **Bổ sung ngày thiếu**: Với mỗi repository, một chuỗi ngày liên tục được tạo ra từ ngày đầu tiên đến ngày cuối cùng có dữ liệu. Các ngày không có bản ghi thực tế (không có commit) được bổ sung và điền giá trị bằng phương pháp forward-fill, tức là sao chép lại giá trị gần nhất trước đó. Riêng đối với đặc trưng "số dòng code thay đổi", giá trị được gán bằng 0 để phản ánh việc không có thay đổi thực sự trong mã nguồn.
- **Chuẩn hóa và mã hóa dữ liệu**: Sau khi chuẩn hóa cấu trúc thời gian, dữ liệu được mã hóa và biến đổi theo đúng định dạng đầu vào cho mô hình học máy. Các đặc trưng rời rạc như "loại môi trường", "tần suất commit" được mã hóa thành chỉ số nguyên (label encoding). Các đặc trưng liên tục được chuẩn hóa về phân phối chuẩn (zero mean, unit variance).

Sau khi xử lý, dữ liệu được tách thành ba phần:

- **Đầu vào**: bao gồm 9 đặc trưng đã chuẩn hóa, loại bỏ các cột ngày quét và tên repository.
- **Nhãn**: mức độ rủi ro bảo mật tương ứng với từng bản ghi.
- **Thông tin bổ sung (metadata)**: bao gồm ngày quét và repository, phục vụ cho quá trình tạo chuỗi thời gian và đánh giá mô hình theo repo.

Phần dữ liệu đầu ra sau tiền xử lý được đảm bảo có định dạng đồng nhất, phù hợp với mô hình LSTM dưới dạng chuỗi thời gian, đồng thời bảo toàn được thông tin lịch sử của từng repository, giúp mô hình học được xu hướng và phát hiện rủi ro một cách hiệu quả.

3.3 Kiến trúc mô hình

Mô hình được thiết kế để học hành vi bảo mật theo chuỗi thời gian từ các DevOps pipelines, với mục tiêu phát hiện các rủi ro xuất hiện ngầm theo thời gian. Kiến trúc mô hình là sự kết hợp giữa mạng LSTM hai tầng để học các mối quan hệ tuần tự trong dữ liệu, và các lớp Bayesian Dense nhằm phản ánh độ bất định trong từng dự đoán.

Table 3.2: Cấu trúc mô hình phát hiện rủi ro bảo mật

Thành phần	**Kích thước đầu ra**
Đầu vào ban đầu	(batch_size, T, 9)
BayesianDense 1	(batch_size, T, 32)
LSTM (2 layers)	(batch_size, T, 64)
BayesianDense 2	(batch_size, T, 64)
BayesianDense 3	(batch_size, T, 48)
BayesianDense 4	(batch_size, T, 48)
BayesianDense 5	(batch_size, T, 32)
Dropout (p=0.1)	(batch_size, T, 32)
Final BayesianDense	(batch_size, T, 5)

Dữ liệu đầu vào là một chuỗi có định dạng $(\text{batch\_size}, T, F)$, trong đó:

- T: độ dài chuỗi thời gian
- $F = 9$: số đặc trưng đầu vào (9 đặc trưng như đã trình bày tại Mục 3.2)

Mỗi bước thời gian trong chuỗi thể hiện trạng thái hệ thống DevOps tại một thời điểm nhất định (theo ngày), bao gồm thông tin về thay đổi mã nguồn, số lượng lỗ hổng, loại môi trường triển khai, v.v.

Thành phần mô hình:

- Lớp Dense đầu vào (BayesianDense) biến đổi đặc trưng đầu vào từ 9 chiều lên 32 chiều. Khác với Dense truyền thống, lớp này học phân phối cho các trọng số, cho phép mô hình phản ánh bất định đầu tiên từ dữ liệu.
- Khối LSTM (2 tầng) biến đổi đầu vào sang không gian 64 chiều, đồng thời ghi nhớ xu hướng và quan hệ thời gian trong chuỗi. Việc sử dụng 2 tầng giúp mô hình học được biểu diễn sâu hơn về hành vi hệ thống qua thời gian.
- Chuỗi các lớp Bayesian Dense kế tiếp giảm dần chiều không gian từ 64 về 32, kèm theo hàm kích hoạt Swish. Những lớp này đóng vai trò tinh chỉnh biểu diễn thời gian trước khi đưa ra dự đoán.
- Lớp Dropout được áp dụng để hạn chế hiện tượng overfitting.

- Lớp đầu ra tạo ra phân phối xác suất trên các lớp đầu ra tương ứng với các mức độ rủi ro (0–4). Việc dùng Bayesian Dense ở đây giúp định lượng độ tin cậy của dự đoán.

Mô hình sinh ra một tensor có kích thước $(\text{batch\_size}, T, \text{num\_classes})$, trong đó mỗi bước thời gian trong chuỗi đầu vào tương ứng với một dự đoán mức rủi ro tại thời điểm đó. Đầu ra là một phân phối xác suất trên 5 lớp (tương ứng với các mức độ rủi ro từ thấp đến cao). Việc dự đoán theo chuỗi giúp mô hình không chỉ phát hiện rủi ro tại một thời điểm cụ thể, mà còn theo dõi xu hướng nguy cơ qua thời gian.

3.4 Hàm mất mát và huấn luyện

Do đầu ra của mô hình tại mỗi bước thời gian là một phân phối xác suất trên các class rủi ro, bài toán được mô hình hóa dưới dạng phân loại đa lớp (multi-class classification) theo chuỗi. Do đó, hàm mất mát được sử dụng là Negative Log-Likelihood (NLL), tương đương với hàm Cross-Entropy trong các bài toán phân loại rời rạc.

Cụ thể, tại mỗi bước thời gian t trong chuỗi, mô hình sinh ra một phân phối p_t trên tập nhãn $\mathcal{Y}$, và tính log-likelihood đối với nhãn thực tế y_t:

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T} \log p_t(y_t)$$

Hàm mất mát này được tính trung bình trên toàn bộ chuỗi thời gian và trên toàn bộ batch trong quá trình huấn luyện. Trong trường hợp sử dụng lớp BayesianDense, xác suất $p_t(y_t)$ được sinh ra thông qua quá trình lấy mẫu (sampling) từ các trọng số phân phối.

Mô hình được huấn luyện theo cơ chế lan truyền ngược (backpropagation), sử dụng một bộ tối ưu hóa gradient dựa trên hàm mất mát ở trên. Dữ liệu đầu vào được tổ chức thành các chuỗi thời gian có chiều dài cố định, và quá trình huấn luyện diễn ra theo mini-batch. Việc huấn luyện mô hình với các lớp Bayesian đòi hỏi kỹ thuật sampling ngẫu nhiên trong quá trình forward, làm cho mỗi lần dự đoán đều có thể cho kết quả khác nhau. Điều này giúp tăng cường khả năng mô hình hóa bất định, nhưng cũng yêu cầu một số lượng epoch đủ lớn để mô hình hội tụ.

CHƯƠNG 4. ĐÁNH GIÁ THỰC NGHIỆM

4.1 Thiết lập thí nghiệm

Hiện tại, tập dữ liệu được xây dựng từ lịch sử quét bảo mật của ba repository private, kéo dài liên tục từ tháng 01 đến tháng 04 năm 2025. Mỗi repository được biểu diễn dưới dạng chuỗi thời gian với 9 đặc trưng đầu vào. Dữ liệu được làm đầy bằng kỹ thuật forward-fill để đảm bảo liên tục về mặt thời gian. Sau khi tiền xử lý, dữ liệu thu được gồm 272 dòng và 9 cột.

Hai chiến lược chia tập train/test được triển khai và so sánh:

- Chia theo repository: chọn hai repository để huấn luyện, và repository còn lại dùng để kiểm tra.
- Chia theo thời gian: với mỗi repository, sử dụng 80% khoảng thời gian đầu để huấn luyện và 20% thời gian gần nhất để kiểm tra.

Table 4.1: Chiến lược chia tập train/test

Chiến lược	Train set	Test set
Chia theo repository	Repo 0 + Repo 1	Repo 2
Chia theo thời gian	80% chuỗi mỗi repo	20% chuỗi mỗi repo

Table 4.2: Tham số huấn luyện

Tham số	Giá trị
Batch size	32
Epochs	200
Optimizer	Adam
Learning rate	0.001
Loss function	CrossEntropyLoss
Dropout	0.1
LSTM hidden units	64
LSTM layers	2
Activation	Swish

4.2 Kết quả thực nghiệm

4.2.1 Kết quả chia theo repository

Khi mô hình được huấn luyện với hai repository và kiểm tra trên repository còn lại, độ chính xác ghi nhận được chỉ đạt khoảng 20%, xấp xỉ với mức dự đoán ngẫu nhiên trong bài toán phân loại 5 lớp. Điều này cho thấy mô hình gặp khó khăn khi tổng quát hóa sang một repository hoàn toàn mới, có thể do sự khác biệt đáng kể về cấu trúc pipeline, loại lỗ hổng, hoặc hành vi cập nhật mã nguồn giữa các dự án.

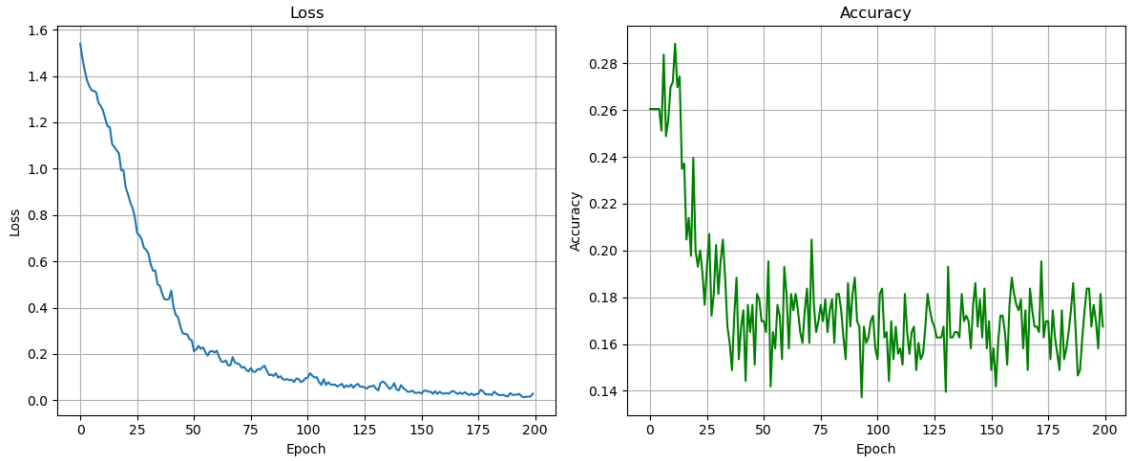

Figure 4.1: Plot loss và validation accuracy của cách chia theo repository

4.2.2 Kết quả chia theo thời gian

Với chiến lược chia theo thời gian trong từng repository, mô hình đạt được độ chính xác lên đến 85% trên tập kiểm tra, cho thấy khả năng học được xu hướng trong từng hệ thống. Mô hình tận dụng hiệu quả tính liên tục và lặp lại trong hành vi bảo mật, từ đó dự đoán chính xác các rủi ro sắp xảy ra dựa trên các biểu hiện trong quá khứ.

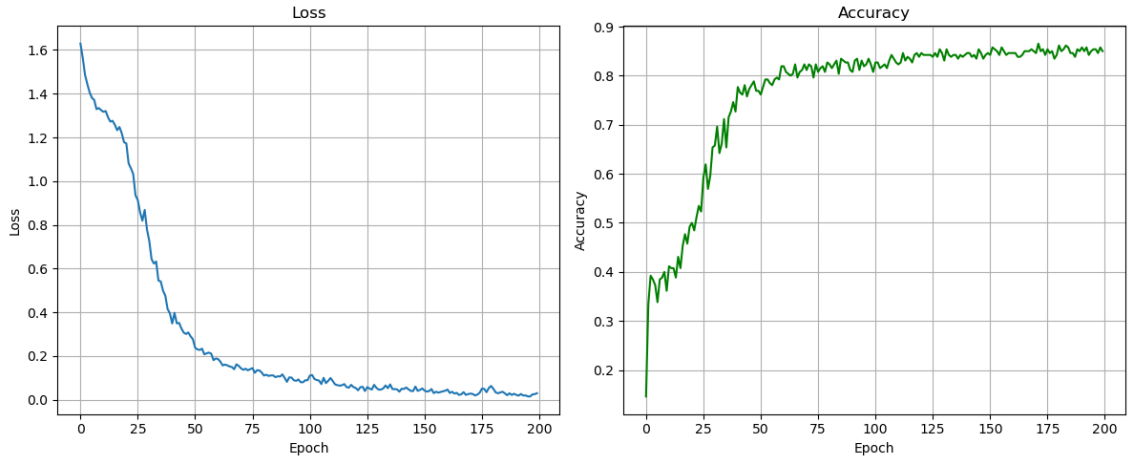

Figure 4.2: Plot loss và validation accuracy của cách chia theo thời gian

4.3 Phân tích kết quả

Kết quả thí nghiệm cho thấy chiến lược chia theo thời gian trong từng repository phản ánh tốt hơn cách sử dụng mô hình trong thực tế: huấn luyện từ lịch sử và dự đoán cho tương lai. Mô hình LSTM có thể khai thác mối quan hệ tuần tự trong chuỗi pipeline bảo mật, trong khi các lớp Bayesian giúp tăng khả năng biểu diễn và kiểm soát độ tin cậy.

Ở chiều ngược lại, việc chia theo repository làm tập kiểm tra quá khác biệt so

với dữ liệu huấn luyện, dẫn đến kết quả gần như ngẫu nhiên. Điều này phản ánh rằng các pipeline DevOps mang tính cá biệt hóa cao, và cần thêm các đặc trưng cụ thể hơn hoặc/và huấn luyện nhiều hơn để đạt được tổng quát hóa thực sự cho mô hình.

CHƯƠNG 5. KẾT LUẬN

5.1 Kết luận

Đồ án đã xây dựng thành công một mô hình học máy kết hợp LSTM và Bayesian Neural Network nhằm phát hiện rủi ro bảo mật theo chuỗi thời gian trong các pipeline DevOps. Mô hình được huấn luyện và đánh giá trên dữ liệu thực tế từ ba repository mã nguồn mở, cho kết quả khả quan khi chia tập theo thời gian với độ chính xác đạt khoảng 85%. Mô hình không chỉ đưa ra dự đoán mà còn phản ánh được mức độ tin cậy của từng cảnh báo.

Tuy nhiên, mô hình vẫn gặp hạn chế khi áp dụng sang repository mới do đặc thù khác biệt giữa các dự án. Ngoài ra, số lượng đặc trưng và nguồn dữ liệu đầu vào còn hạn chế, chưa phản ánh đầy đủ các yếu tố ảnh hưởng đến bảo mật trong thực tế.

5.2 Hướng phát triển trong tương lai

Trong tương lai, mô hình có thể được cải tiến theo các hướng:

- Mở rộng dữ liệu đầu vào, kết hợp thêm log hệ thống, hành vi CI/CD hoặc dữ liệu runtime.
- Tăng số lượng repository để cải thiện khả năng tổng quát hóa.
- Đánh giá độ bất định của dự đoán từ mô hình.
- Khảo sát các mô hình tiên tiến như Transformer cho chuỗi thời gian.

TÀI LIỆU THAM KHẢO

[1] O. Tunde-Onadele, J. He, T. Dai, and X. Gu, "A study on container vulnerability exploit detection," Jun. 2019, pp. 121–127. DOI: 10.1109/IC2E.2019.00026.

[2] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, and C. Talhi, "An anomaly detection system based on variable n-gram features and one-class svm," *Information and Software Technology*, vol. 91, pp. 186–197, 2017, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2017.07.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584917304548.

[3] H. Studiawan, F. Sohel, and C. Payne, "Anomaly detection in operating system logs with deep learning-based sentiment analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2136–2148, 2021. DOI: 10.1109/TDSC.2020.3037903.

[4] X. Wu, H. Li, and F. Khomh, "On the effectiveness of log representation for log-based anomaly detection," *Empirical Software Engineering*, vol. 28, no. 6, p. 137, 2023, ISSN: 1573-7616. DOI: 10.1007/s10664-023-10364-1. [Online]. Available: https://doi.org/10.1007/s10664-023-10364-1.

[5] O. Abiona, O. Oladapo, O. Modupe, O. Oyeniran, A. Adewusi, and A. Komolafe, "The emergence and importance of devsecops: Integrating and reviewing security practices within the devops pipeline," *World Journal of Advanced Engineering Technology and Sciences*, vol. 11, pp. 127–133, Mar. 2024. DOI: 10.30574/wjaets.2024.11.2.0093.

[6] E. Goan and C. Fookes, "Bayesian neural networks: An introduction and survey," in *Case Studies in Applied Bayesian Data Science*. Springer International Publishing, 2020, 45–87, ISBN: 9783030425531. DOI: 10.1007/978-3-030-42553-1_3. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-42553-1_3.

[7] D. Sam, R. Pukdee, D. P. Jeong, Y. Byun, and J. Z. Kolter, *Bayesian neural networks with domain knowledge priors*, 2024. arXiv: 2402.13410 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2402.13410.