

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

HÀ THỊ MINH PHƯƠNG

DỰ ĐOÁN LỖI PHẦN MỀM DỰA TRÊN
MÃ NGUỒN

LUẬN ÁN TIẾN SĨ KỸ THUẬT

Đà Nẵng, 2026

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

HÀ THỊ MINH PHƯƠNG

DỰ ĐOÁN LỖI PHẦN MỀM DỰA TRÊN
MÃ NGUỒN

LUẬN ÁN TIẾN SĨ KỸ THUẬT

Giảng viên hướng dẫn
PGS. TS. Nguyễn Thanh Bình
TS. Lê Thị Mỹ Hạnh

Đà Nẵng, 2026

Lời cam đoan

Tôi xin cam đoan đây là công trình nghiên cứu do tôi thực hiện, dưới sự hướng dẫn khoa học của PGS. TS. Nguyễn Thanh Bình (Khoa Khoa học máy tính, Trường Đại học Công nghệ Thông tin và Truyền thông Việt Hàn, Đại học Đà Nẵng) và TS. Lê Thị Mỹ Hạnh (Khoa Công nghệ Thông tin, Trường Đại học Bách khoa, Đại học Đà Nẵng).

Tôi cam đoan các kết quả nghiên cứu được trình bày trong luận án là trung thực và không sao chép từ bất kỳ luận án nào khác. Một số kết quả nghiên cứu là thành quả tập thể và đã được các đồng tác giả đồng ý cho sử dụng. Mọi trích dẫn đều có ghi nguồn gốc xuất xứ rõ ràng.

Tác giả

NCS. Hà Thị Minh Phương

Lời cảm ơn

Trước tiên, tác giả xin gửi lời cảm ơn đến Trường Đại học Bách khoa - Đại Học Đà Nẵng đã tạo mọi điều kiện thuận lợi cho tác giả trong thời gian nghiên cứu và hoàn thành Luận án.

Với lòng kính trọng và biết ơn sâu sắc, tác giả xin gửi lời cảm ơn tới thầy giáo hướng dẫn PGS. TS. Nguyễn Thanh Bình và cô TS. Lê Thị Mỹ Hạnh đã tận tình giúp đỡ tác giả từ những bước đi đầu tiên xây dựng ý tưởng nghiên cứu, cũng như trong suốt quá trình nghiên cứu và hoàn thiện Luận án. Hai thầy cô đã luôn ủng hộ, động viên và hỗ trợ những điều kiện tốt nhất để tác giả hoàn thành luận án.

Tác giả xin gửi lời cảm ơn chân thành tới các thầy, cô giáo và các bạn đồng nghiệp của Trường Đại học Công nghệ Thông tin và Truyền thông Việt - Hàn - Đại học Đà Nẵng đã tạo mọi điều kiện thuận lợi và giúp đỡ tác giả trong thời gian học tập và nghiên cứu.

Cuối cùng, với tình yêu từ đáy lòng, tác giả xin gửi lời cảm ơn tới bố, mẹ, anh, chị, em cùng chồng và con của tác giả, những người thân yêu trong gia đình đã luôn ở bên cạnh tác giả, động viên tác giả về vật chất và tinh thần để tác giả vững tâm hoàn thành luận án của mình.

NCS. Hà Thị Minh Phương

Mục lục

Lời cam đoan	i
	iii
Danh sách hình ảnh	v
Danh sách bảng	vi
Mở đầu	1
1 Tổng quan dự đoán lỗi phần mềm	15
1.1 Dự đoán lỗi phần mềm	15
1.1.1 Khái niệm	15
1.1.2 Quy trình dự đoán lỗi phần mềm	17
1.2 Các phương pháp dự đoán lỗi phần mềm	18
1.2.1 Dự đoán lỗi phần mềm dựa trên đặc trưng độ đo mã nguồn	18
1.2.2 Dự đoán lỗi phần mềm dựa trên đặc trưng cú pháp và ngữ nghĩa mã nguồn	30
1.3 Các thách thức	32
1.4 Các mô hình học máy	35
1.4.1 Random Forest	35
1.4.2 Extra Trees	36
1.4.3 AdaBoost	37
1.4.4 Gradient Boosting	38
1.4.5 HistGradient Boosting	38
1.4.6 eXtreme Gradient Boosting	39
1.4.7 Multilayer Perceptron	41
1.5 Các độ đo đánh giá	41
1.6 Kết chương	43

2	Nâng cao khả năng dự đoán lỗi phần mềm bằng các phương pháp lựa chọn đặc trưng	45
2.1	Giới thiệu	45
2.2	Các nghiên cứu liên quan	47
2.2.1	Lựa chọn đặc trưng	47
2.2.2	Vấn đề nghiên cứu	48
2.3	Phương pháp đề xuất	49
2.3.1	Tập dữ liệu	50
2.3.2	Các phương pháp lựa chọn đặc trưng metaheuristic	51
2.3.3	Kết quả thử nghiệm	68
2.4	Kết chương	75
3	Kết hợp các kỹ thuật lựa chọn đặc trưng và lấy mẫu dữ liệu trong dự đoán lỗi phần mềm	77
3.1	Giới thiệu	77
3.2	Cơ sở lý thuyết	79
3.2.1	Các nghiên cứu liên quan	79
3.2.2	Đánh giá hiệu quả của các kỹ thuật lấy mẫu dữ liệu	81
3.3	Các mô hình GAN lấy mẫu dữ liệu trong dự đoán lỗi phần mềm	83
3.3.1	Mô hình GAN	83
3.3.2	Mô hình GAN trong dự đoán lỗi phần mềm	83
3.3.3	Vấn đề nghiên cứu	88
3.4	Phương pháp đề xuất	89
3.5	Kết quả thử nghiệm và phân tích kết quả	92
3.5.1	Trả lời cho Câu hỏi nghiên cứu 1: Các biến thể của GAN có hiệu quả như thế nào trong việc lấy mẫu dữ liệu để xử lý mất cân bằng lớp trong dự đoán lỗi phần mềm?	93
3.5.2	Trả lời cho Câu hỏi nghiên cứu 2: Trong số các kỹ thuật tiên tiến nhất hiện nay, biến thể nào của GAN vượt trội hơn trong việc xử lý mất cân bằng dữ liệu trong dự đoán lỗi phần mềm?	104
3.5.3	Kết quả kiểm định thống kê	106
3.6	Kết chương	110
4	Dự đoán lỗi phần mềm dựa trên học sâu	113
4.1	Giới thiệu	113
4.2	Các nghiên cứu liên quan	114
4.3	Vấn đề nghiên cứu	117
4.4	Phương pháp đề xuất	118

4.4.1	Biểu diễn mã nguồn và trích xuất các đặc trưng	120
4.4.2	Áp dụng FastText cho Token Embedding	124
4.4.3	Transformer Encoder	128
4.5	Triển khai thử nghiệm, kết quả thử nghiệm và phân tích kết quả	129
4.5.1	Tập dữ liệu thử nghiệm	130
4.5.2	Các mô hình so sánh	132
4.5.3	Tham số huấn luyện	133
4.5.4	Trả lời cho Câu hỏi nghiên cứu 1: Mô hình FastText-Transformer được đề xuất có vượt trội hơn so với các mô hình dự đoán lỗi tiên tiên hiện nay đối với dự đoán lỗi trong cùng một dự án không? . .	135
4.5.5	Trả lời cho Câu hỏi nghiên cứu 2: Mô hình FastText-Transformer được đề xuất có vượt trội hơn so với các mô hình dự đoán lỗi tiên tiên hiện nay đối với dự đoán lỗi trong liên dự án không?	138
4.5.6	Kết quả kiểm định thống kê	140
4.5.7	So sánh chi phí thời gian huấn luyện và thời gian kiểm thử mô hình giữa FastText-LSTM và FastText-Transformer	141
4.5.8	Các mối đe dọa đến tính tin cậy và giá trị của nghiên cứu	143
4.6	Kết chương	144
Kết luận		147
Các công trình khoa học đã công bố		150
Tài liệu tham khảo		170

Danh sách hình vẽ

0.1	Cấu trúc của luận án.	12
1.1	Quy trình dự đoán lỗi phần mềm	17
1.2	Các thách thức tồn tại trong dự đoán lỗi phần mềm	33
1.3	Mô hình AdaBoost	37
2.1	Quy trình đánh giá các phương pháp metaheuristic	50
2.2	Biểu diễn giải pháp dưới dạng vectơ nhị phân	65
2.3	So sánh các giá trị AUC của các phương pháp lựa chọn đặc trưng metaheuristic trên các bộ dữ liệu khác nhau	73
3.1	Phương pháp đề xuất	90
3.2	Giá trị AUC của các phương pháp lấy mẫu khác nhau trên các tập dữ liệu khác nhau (3a) CM1, (3b) PC1, (3c) KC1, (3d) KC2, (3e) JM1	107
4.1	Kiến trúc mô hình đề xuất	120
4.2	Kiến trúc Transformer Encoder sử dụng	120
4.3	Một ví dụ AST cho một chương trình Java	121
4.4	Ví dụ về quá trình phân tách mã thông qua cây cú pháp trừu tượng cho một chương trình Java	123
4.5	Kiến trúc tổng thể của mô hình FastText	126
4.6	So sánh kết quả F1-score giữa FastText-Transformer và DP-Transformer đối với dự đoán lỗi trong cùng một dự án	138

Danh sách bảng

1.1	Độ đo McCabe, Halstead trong dự đoán lỗi phần mềm	20
1.2	Độ đo hướng đối tượng trong dự đoán lỗi phần mềm	21
1.1	Ma trận nhầm lẫn	41
2.1	Tập dữ liệu sử dụng thử nghiệm	51
2.2	Mô tả chi tiết của tập dữ liệu từ kho PROMISE	52
2.3	Hiệu suất của các phương pháp lựa chọn đặc trưng metaheuristic khi áp dụng với Random Forest	69
2.4	Hiệu suất của các phương pháp lựa chọn đặc trưng metaheuristic khi áp dụng với AdaBoost	70
2.5	Hiệu suất của các phương pháp lựa chọn đặc trưng metaheuristic khi áp dụng với Extra Trees	71
2.6	So sánh độ quan trọng của các đặc trưng	74
2.7	Kết quả kiểm định thống kê so sánh giữa các phương pháp chọn đặc trưng metaheuristic và phương pháp cơ sở đối với các mô hình Random Forest, Extra Trees và AdaBoost	75
3.1	Một số nghiên cứu liên quan đến giải quyết vấn đề mất cân bằng lớp trong dự đoán lỗi phần mềm	80
3.2	Một số nghiên cứu liên quan đến giải quyết vấn đề mất cân bằng lớp trong dự đoán lỗi phần mềm (tt)	81
3.1	Chi tiết tập dữ liệu thử nghiệm trước và sau khi cân bằng	91
3.2	Cấu hình các tham số huấn luyện cho VanillaGAN	92
3.3	Cấu hình các tham số huấn luyện cho CTGAN	92
3.4	Cấu hình các tham số huấn luyện cho WGANGP	92
3.5	Cấu hình các tham số huấn luyện cho CopulaGAN	93
3.6	Cấu hình các tham số huấn luyện cho TGAN	93
3.1	Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng Random Forest	94
3.2	Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng XGBoost	95

3.3	Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng AdaBoost	96
3.4	Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng HGBost	97
3.5	Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng MLP	98
3.6	Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng Extra Trees	99
3.7	Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy Random Forest	101
3.8	Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy XGboost	102
3.9	Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy AdaBoost	103
3.10	Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy HGBost	104
3.11	Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy MLP	105
3.12	Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy ExTra Trees	106
3.13	Kết quả kiểm định thống kê cho so sánh giữa VanillaGAN và các phương pháp lấy mẫu tăng	108
3.15	Kết quả kiểm định thống kê cho so sánh giữa WGANP và các phương pháp lấy mẫu tăng	109
3.16	Kết quả kiểm định thống kê cho so sánh giữa CopulaGAN và các phương pháp lấy mẫu tăng	109
3.14	Kết quả kiểm định thống kê cho so sánh giữa CTGAN và các phương pháp lấy mẫu tăng	109
3.17	Kết quả kiểm định thống kê cho so sánh giữa TGAN và các phương pháp lấy mẫu tăng	110
4.1	Nút AST được chọn	123
4.2	Phân tích so sánh các mô hình dựa trên Transformer với những chiến lược embedding khác nhau	125
4.1	Tập dữ liệu Apache được sử dụng	131
4.2	Một số mẫu trong tập dữ liệu PROMISE được sử dụng	131
4.3	Tham số huấn luyện cho mô hình Transformer	134
4.4	Tham số huấn luyện cho mô hình LSTM	135
4.5	Thông số hệ thống chi tiết	136
4.6	Kết quả so sánh hiệu suất của DTL-DP, Seml, DBN-CP, FastText-LSTM và FastText-Transformer đối với dự đoán lỗi trong cùng một dự án	137

4.7	So sánh giá trị F1-score DTL-DP, DBN-CP, TCA+, FastText-LSTM và FastText-Transformer đối với dự đoán lỗi trong liên dự án	139
4.8	Kết quả kiểm định thống kê so sánh giữa FastText-Transformer và các mô hình đối sánh đối với dự đoán lỗi trong cùng một dự án	141
4.9	Kết quả kiểm định thống kê cho so sánh giữa FastText-Transformer và các mô hình đối sánh đối với dự đoán lỗi trong liên dự án	142
4.10	So sánh chi phí thời gian huấn luyện và thử nghiệm mô hình giữa FastText-LSTM và FastText-Transformer	142

Danh mục từ viết tắt

ABO Artificial Butterfly Optimization.

AdaGrad Adaptive Gradient Algorithm.

ASO Atom Search Optimization.

AST Abstract Syntax Tree.

AUC Area Under the Curve.

BCE Binary-Cross Entropy.

CPDP Cross-Project Defect Prediction.

CTGAN Conditional GAN.

DBN Deep Belief Network.

DDG Data Dependence Graph.

EO Equilibrium Optimizer.

ET Extra Trees.

GA Genetic algorithm.

GAN Generative Adversarial Networks.

GCNN Graph Convolutional Neural Network.

GH-LSTMs Gated Hierarchical Long Short-Term Memory.

GND0 Generalized Normal Distribution Optimization.

GNN Graph Neural Network.

GRUs Gated Recurrent Units.

HGBoost Histogram-based Gradient Boosting.

HHO Harris Hawks Optimization.

LSTM Long Short-Term Memory.

LSTM-BT Long Short-Term Memory network based on bidirectional and tree structure.

MCC Matthews Correlation Coefficient.

MRFO Manta Ray Foraging Optimization.

PFA Pathfinder Algorithm.

PRO Poor And Rich Optimization.

ReLU Rectified Linear Unit.

RFE Recursive Feature Elimination.

RNN Recurrent Neural Network.

SMA Slime Mold Algorithm.

TGAN Tabular Generative Adversarial Networks.

VAE Variational Autoencoder.

WGANGP Wasserstein GAN With Gradient Penalty.

WPDP Within-Project Defect Prediction.

XGBoost eXtreme Gradient Boosting.

Mở đầu

1. Đặt vấn đề

Trong thời đại công nghệ số, phần mềm đã trở thành một thành phần thiết yếu trong hầu hết các lĩnh vực của khoa học kỹ thuật và đời sống xã hội. Từ các hệ thống điều khiển công nghiệp, hệ thống ngân hàng, đến các ứng dụng phần mềm doanh nghiệp và tiêu dùng, tất cả đều đòi hỏi độ tin cậy và chất lượng cao nhằm đảm bảo an toàn, hiệu quả và trải nghiệm người dùng. Tuy nhiên, việc đảm bảo chất lượng phần mềm là một thách thức lớn, đòi hỏi nguồn lực đáng kể cả về nhân lực và chi phí. Theo Arar và cộng sự [1] và Ainapure và cộng sự [2], các hoạt động kiểm thử phần mềm và đảm bảo chất lượng chiếm khoảng 23% tổng chi phí trong các dự án phát triển phần mềm. Theo nghiên cứu của Rana và cộng sự [3], hoạt động đảm bảo chất lượng là một trong những công đoạn tốn kém nhất trong quá trình phát triển phần mềm, khi các kỹ sư phải dành phần lớn thời gian để kiểm tra và xác minh tính đúng đắn của mã nguồn thay vì phát triển các tính năng mới. Theo báo cáo từ Consortium for IT Software Quality (CISQ) năm 2022 [4], 2410 tỷ USD đã được chi để chi cho chi phí chất lượng phần mềm tại Hoa Kỳ. Với tốc độ tăng trưởng kho mã nguồn khoảng 7% mỗi năm, và mức lương trong lĩnh vực CNTT tăng khoảng 3% mỗi năm, thì chi phí dành cho việc tìm và sửa lỗi phần mềm vào năm 2020 ước tính đã lên tới khoảng 607 tỷ USD. Điều này dẫn đến hàng tỷ USD thiệt hại hàng năm cho các doanh nghiệp. Ngoài ra, nghiên cứu của Đại học Cambridge [5] cũng cho thấy khoảng 30–50% thời gian làm việc của lập trình viên là dành cho việc tìm kiếm và khắc phục lỗi. Điều này cho thấy rõ ràng rằng lỗi phần mềm là một vấn đề không chỉ về kỹ thuật mà còn về kinh tế và chiến lược quản trị.

Kiểm thử phần mềm là hoạt động quan trọng được thực hiện trong quy trình phát triển phần mềm nhằm phát hiện và loại bỏ lỗi trước khi phần mềm được đưa vào vận hành trong thực tế. Tuy nhiên, đây cũng là một trong những giai đoạn tiêu tốn nhiều thời gian và chi phí nhất trong quy trình phát triển phần mềm. Việc kiểm thử toàn bộ các mô-đun với cùng một mức độ ưu tiên không chỉ làm gia tăng chi phí mà còn chưa thực sự hiệu quả đối với các dự án phần mềm quy mô lớn. Các phương pháp truyền thống như kiểm thử thủ công,

kiểm thử tự động và phân tích tĩnh tuy vẫn hữu ích nhưng gặp nhiều hạn chế: tiêu tốn tài nguyên, phụ thuộc vào kỹ năng con người, và không tận dụng được dữ liệu lịch sử để đưa ra quyết định thông minh hơn. Vì vậy, xu hướng hiện nay là chuyển từ kiểm thử toàn diện sang kiểm thử có trọng tâm, trong đó các mô-đun có nguy cơ phát sinh lỗi cao được ưu tiên kiểm thử trước. Trong bối cảnh đó, việc phát hiện lỗi sớm trong vòng đời phát triển phần mềm trở nên hết sức cấp thiết. Để thực hiện được điều đó, cần có các mô hình dự đoán lỗi phần mềm đủ chính xác nhằm hỗ trợ nhận diện sớm các thành phần tiềm ẩn rủi ro, từ đó giúp tối ưu hóa nguồn lực kiểm thử và nâng cao chất lượng phần mềm. Do đó, bài toán dự đoán lỗi phần mềm, đặc biệt là sử dụng các mô hình trí tuệ nhân tạo, đang trở thành một hướng đi chiến lược nhằm nâng cao chất lượng phần mềm một cách chủ động, hiệu quả và quy mô lớn.

Một trong những hướng tiếp cận nổi bật của dự đoán lỗi phần mềm là sử dụng các kỹ thuật học máy và khai phá dữ liệu để xác định các mô-đun trong hệ thống có xác suất chứa lỗi cao. Các phương pháp này cho phép xây dựng các mô hình dự đoán dựa trên lỗi trong quá khứ, đặc trưng mã nguồn hoặc các độ đo phần mềm, từ đó xếp hạng mức độ rủi ro và ưu tiên kiểm thử tương ứng [6]. Bằng cách tập trung nguồn lực kiểm thử vào các khu vực có khả năng xảy ra lỗi cao nhất, các phương pháp này không chỉ giúp giảm thiểu chi phí và thời gian kiểm thử, mà còn góp phần nâng cao độ tin cậy tổng thể của hệ thống và hiệu quả vận hành của doanh nghiệp [7]. Mục tiêu của các mô hình dự đoán lỗi phần mềm là xác định các mô-đun có khả năng dễ xảy ra lỗi, chẳng hạn như tệp, lớp, gói, phương thức và hàm trong hệ thống phần mềm nhất định [8]. Trong những năm qua, các kỹ thuật học máy đã được áp dụng rộng rãi để phát triển các mô hình dự đoán lỗi phần mềm. Các kỹ thuật này sử dụng dữ liệu lỗi phần mềm thu thập từ các dự án phần mềm trước đó để huấn luyện mô hình dự đoán và xác định các mô-đun có lỗi [9]. Dữ liệu lỗi này cho phép kiểm tra xem một mô-đun có thể được gán nhãn là “*có lỗi*” hay “*không lỗi*”. Việc xây dựng các độ đo đặc trưng mã nguồn là rất quan trọng trong mô hình dự đoán lỗi dựa trên các mô hình học máy. Các nghiên cứu trước đây [10] chủ yếu tập trung vào việc thiết kế các đặc trưng khác nhau một cách thủ công để phân biệt giữa các mô-đun lỗi và mô-đun không lỗi. Các đặc trưng truyền thống được xây dựng thủ công này bao gồm: độ đo Halstead [11] dựa trên số lượng toán tử và toán hạng trong mã nguồn; độ đo McCabe [12] dựa trên mức độ phụ thuộc; độ đo CK (Chidamber và Kemerer 1994) [13] dựa trên kế thừa; số lượng hàm và số dòng của mã nguồn; độ đo MOOD (Metrics for Object-Oriented Design) [14] dựa trên các yếu tố như đa hình và liên kết. Trong khi đó, nhiều mô hình học máy sử dụng trên các tập dữ liệu lịch sử bao gồm các đặc trưng mã nguồn trên và nhãn tương ứng (lỗi hoặc không lỗi) để thực hiện huấn luyện các mô hình dự đoán lỗi và dự đoán các mô-đun lỗi, chẳng hạn như Naive Bayes [15], Support Vector Machine [16], Decision Tree [17] và Random Forest [18],... Trong quá trình huấn luyện, mô hình dự đoán thu nhận kiến thức về

các đặc trưng của dự án phần mềm và sử dụng kiến thức này để dự đoán liệu một mô-đun mới có bị lỗi hay không [19]. Tuy nhiên, có nhiều yếu tố ảnh hưởng đến hiệu suất của các mô hình dự đoán lỗi, bao gồm tập dữ liệu, độ đo mã nguồn và các vấn đề liên quan đến tập dữ liệu [20]. Qua khảo sát các nghiên cứu trong và ngoài nước, có thể nhận thấy rằng hiệu quả của mô hình dự đoán lỗi phụ thuộc chủ yếu vào hai yếu tố cốt lõi, bao gồm (i) chất lượng dữ liệu đầu vào và (ii) khả năng biểu diễn thông tin của mã nguồn. Hai yếu tố này quyết định trực tiếp khả năng học, khả năng tổng quát hóa và độ chính xác của mô hình dự đoán. Vì vậy, để nâng cao hiệu quả dự đoán lỗi phần mềm cần có một chiến lược nghiên cứu xuyên suốt, bắt đầu từ việc tối ưu hóa dữ liệu đầu vào nhằm cải tiến hiệu suất của các mô hình dự đoán lỗi dựa trên đặc trưng độ đo và đồng thời cũng xây dựng các mô hình khả năng học biểu diễn mã nguồn với các phương pháp học sâu tốt hơn. Một số thách thức đáng kể trong dự đoán lỗi phần mềm được trình bày trong dưới đây.

Thứ nhất, đối với các mô hình dự đoán lỗi truyền thống dựa trên đặc trưng độ đo mã nguồn, chất lượng dữ liệu đầu vào là yếu tố quyết định hiệu suất của mô hình. Một thách thức đáng kể trong quá trình xây dựng mô hình là vấn đề dữ liệu có số chiều lớn của tập dữ liệu, nghĩa là sự tồn tại của quá nhiều đặc trưng, bao gồm cả đặc trưng không liên quan hoặc dư thừa. Các đặc trưng không liên quan là những đặc trưng không ảnh hưởng đến lỗi phần mềm; và các biến dư thừa là những đặc trưng (hoặc biến độc lập) có mối tương quan mạnh với nhau, nghĩa là chúng cung cấp thông tin gần như giống hệt nhau. Điều này gây ra sự dư thừa trong dữ liệu và làm giảm hiệu quả của mô hình học máy. Nhiều nghiên cứu đã chỉ ra rằng điều này không chỉ làm gia tăng chi phí tính toán mà còn ảnh hưởng tiêu cực đến hiệu suất của một số mô hình dự đoán lỗi nhất định [21]. Bên cạnh đó, khảo sát tài liệu cho thấy rằng lựa chọn đặc trưng giúp cải thiện hiệu suất của mô hình phân loại [22]. Các kỹ thuật lựa chọn đặc trưng được áp dụng trong giai đoạn tiền xử lý dữ liệu nhằm nâng cao hiệu quả của các mô hình dự đoán lỗi phần mềm. Trong những năm qua, đã có nhiều nghiên cứu liên tục nhằm cải thiện độ chính xác thống kê của mô hình. Tuy nhiên, với các tập dữ liệu lớn, quá trình huấn luyện mô hình dự đoán lỗi đòi hỏi nhiều thời gian hơn và tiêu tốn nhiều tài nguyên tính toán hơn, điều này cũng có thể ảnh hưởng đến độ chính xác của dự đoán. Do đó, việc lựa chọn đặc trưng là cần thiết [21]. Các phương pháp lựa chọn đặc trưng được sử dụng trong học máy và phân tích dữ liệu nhằm lựa chọn một tập con các đặc trưng hoặc biến có liên quan từ một tập đặc trưng ban đầu lớn hơn. Mục tiêu của các phương pháp này là nâng cao hiệu suất của mô hình, rút ngắn thời gian huấn luyện bằng cách chọn ra những đặc trưng quan trọng và mang nhiều thông tin nhất [23] và giảm hiện tượng quá khớp, một hiện tượng mà mô hình dự đoán hoạt động tốt trên tập dữ liệu huấn luyện nhưng sẽ cho ra kết quả không tốt đối với tập dữ liệu mới. Nhiều phương pháp lựa chọn đặc trưng đã được các nhà nghiên cứu đề xuất nhằm cải thiện hiệu quả của mô hình đang được xem xét. Các kỹ thuật lựa chọn đặc trưng có thể được phân

loại rộng rãi thành ba nhóm chính: filter, wrapper và embedded. Phương pháp filter áp dụng các độ đo thống kê cho từng đặc trưng và lựa chọn các đặc trưng có điểm số cao nhất. Phương pháp wrapper sử dụng thuật toán học máy để đánh giá hiệu suất của từng tập con đặc trưng. Thuật toán sẽ được huấn luyện với nhiều tập con khác nhau, và tập con mang lại hiệu suất tốt nhất sẽ được chọn. Trong khi đó, phương pháp embedded sẽ kết hợp quá trình lựa chọn đặc trưng ngay trong thuật toán học máy. Ví dụ, một số thuật toán học máy có cơ chế lựa chọn đặc trưng tích hợp sẵn, như Lasso regression, Decision Tree, và Random Forest. Theo Savina Colaco và cộng sự [24], phương pháp dựa trên wrapper là kỹ thuật được sử dụng phổ biến nhất trong số các phương pháp lựa chọn đặc trưng. Hiện nay, lựa chọn đặc trưng vẫn là một hướng nghiên cứu tiếp tục phát triển trong dự đoán lỗi phần mềm. Mặc dù nhiều phương pháp truyền thống dựa trên filter, wrapper và embedded đã được áp dụng rộng rãi, nhưng các thách thức trong việc xử lý dữ liệu phức tạp, có số chiều cao và phân bố không đồng đều tiếp tục đặt ra yêu cầu cải tiến. Do đó, bên cạnh việc xem xét các phương pháp truyền thống, luận án này cũng đề xuất và áp dụng một số kỹ thuật lựa chọn đặc trưng mới nhằm đánh giá khả năng cải thiện hiệu suất của mô hình dự đoán lỗi phần mềm. Qua đó, luận án góp phần xác định được những kỹ thuật lựa chọn đặc trưng hiệu quả hơn, đồng thời mở rộng nghiên cứu và ứng dụng thực tiễn trong lĩnh vực dự đoán lỗi.

Thứ hai, sau khi loại bỏ các đặc trưng dư thừa, hiệu quả của mô hình vẫn chịu ảnh hưởng đáng kể bởi hiện tượng mất cân bằng dữ liệu. Các tập dữ liệu đầu vào dự đoán lỗi phần mềm có sự phân bố không đồng đều giữa các mẫu, bao gồm cả mẫu đa số và mẫu thiểu số, do đó ảnh hưởng đến hiệu suất của các mô hình dự đoán lỗi. Bản chất của tập dữ liệu huấn luyện có vai trò quan trọng trong việc quyết định hiệu suất của mô hình phân loại [25]. Nhìn chung, trong các độ đo của dự án phần mềm, số lượng mẫu không lỗi thường lớn hơn đáng kể so với số mẫu có lỗi, dẫn đến thách thức mất cân bằng lớp trong các mô hình dự đoán lỗi. Khi được huấn luyện trên các tập dữ liệu mất cân bằng lớp, các mô hình dự đoán lỗi có xu hướng thiên vị các mẫu không lỗi, trong khi dễ dàng phân loại sai các trường hợp có lỗi [26]. Tuy nhiên, khi các tập dữ liệu huấn luyện được cân bằng lại, các mô hình dự đoán lỗi có thể đạt được hiệu suất dự đoán tốt hơn [27]. Do đó, mất cân bằng là một trong những yếu tố chính ảnh hưởng đến hiệu suất và có thể dẫn đến các mô hình dự đoán không đáng tin cậy, và ngày càng có nhiều nghiên cứu tập trung giải quyết vấn đề này. Các kỹ thuật lấy mẫu dữ liệu như lấy mẫu tăng (oversampling) và lấy mẫu giảm (undersampling) được sử dụng rộng rãi để xử lý vấn đề mất cân bằng lớp trong dự đoán lỗi [28]. Kỹ thuật lấy mẫu tăng tạo ra các mô-đun mới thuộc lớp thiểu số bằng cách tổng hợp dữ liệu nhằm tăng số lượng mẫu của lớp này. Ngược lại, lấy mẫu giảm loại bỏ một số mô-đun thuộc lớp đa số để cân bằng tỷ lệ giữa hai lớp [29]. Cả hai nhóm kỹ thuật này đều hướng đến mục tiêu cân bằng sự phân bố giữa các lớp, từ đó cải thiện hiệu suất

của mô hình dự đoán. Trong bài toán dự đoán lỗi phần mềm, kỹ thuật lấy mẫu tăng thường được ưu tiên hơn so với kỹ thuật lấy mẫu giảm, bởi mục tiêu chính là bổ sung và mở rộng dữ liệu của lớp thiểu số thay vì loại bỏ dữ liệu của lớp đa số, qua đó giữ được nhiều thông tin huấn luyện hơn và cải thiện khả năng học của mô hình. Các phương pháp lấy mẫu tăng phổ biến bao gồm Synthetic Minority Over-sampling Technique (SMOTE) và các biến thể của nó như Adaptive Synthetic Sampling (ADASYN), Borderline-SMOTE, cùng với Random Oversampling. Trong đó, Random Oversampling chỉ đơn giản là sao chép các mô-đun thuộc lớp thiểu số để cân bằng tập dữ liệu, mà không bổ sung thông tin mới [30]. Cách tiếp cận này có thể khiến mô hình bị *quá khớp* và hoạt động kém trên các mô-đun kiểm thử chưa từng gặp. Ngược lại, các kỹ thuật dựa trên SMOTE tạo ra các mô-đun tổng hợp thuộc lớp thiểu số bằng cách nội suy giữa các mô-đun lân cận (dựa trên khoảng cách) [31, 32]. Tuy nhiên, do các mô-đun tổng hợp thường có đặc điểm tương tự nhau, phương pháp này có thể thiếu sự đa dạng, dẫn đến quá khớp và giảm hiệu quả dự đoán trên tập kiểm thử. Trước những hạn chế đó, mạng sinh đối kháng (GAN - Generative Adversarial Network) nổi lên như một hướng tiếp cận tiềm năng nhờ khả năng học và sinh dữ liệu mới một cách chân thực hơn. Khác với các kỹ thuật nội suy truyền thống, GAN học phân phối dữ liệu của lớp thiểu số thông qua quá trình huấn luyện đối kháng giữa hai mạng bao gồm bộ sinh và bộ phân biệt, từ đó tạo ra các mẫu mới có tính đa dạng cao và phản ánh tốt hơn đặc điểm của dữ liệu gốc. Việc ứng dụng GAN trong xử lý mất cân bằng dữ liệu không chỉ giúp giảm thiểu quá khớp mà còn có thể nâng cao đáng kể hiệu suất dự đoán lỗi phần mềm. Do đó, trong luận án này, chúng tôi hướng đến việc khai thác tiềm năng của mạng GAN như một giải pháp hiệu quả cho các phương pháp lấy mẫu tăng truyền thống.

Từ việc nghiên cứu hai thách thức đặt ra ở trên cho thấy việc nâng cao hiệu quả dự đoán lỗi cần được tiếp cận theo hướng tối ưu hóa đồng thời nhiều khía cạnh của dữ liệu đầu vào. Trong đó, lựa chọn đặc trưng giúp xây dựng một không gian đặc trưng tối ưu hơn, loại bỏ nhiễu và giữ lại các thông tin có giá trị nhất đối với quá trình phân loại; trong khi các kỹ thuật xử lý mất cân bằng dữ liệu giúp cải thiện sự phân bố giữa các lớp, tạo điều kiện để mô hình học được đầy đủ đặc trưng của lớp thiểu số. Hai nhóm kỹ thuật này không thay thế mà bổ sung cho nhau: lựa chọn đặc trưng làm tăng chất lượng biểu diễn của dữ liệu, còn cân bằng dữ liệu làm tăng chất lượng phân bố của dữ liệu huấn luyện. Việc kết hợp đồng thời hai kỹ thuật được kỳ vọng sẽ tạo ra tập dữ liệu vừa có tính đại diện cao, vừa có phân bố hợp lý, từ đó cải thiện khả năng học, nâng cao độ chính xác, độ ổn định và khả năng tổng quát hóa của các mô hình dự đoán lỗi phần mềm.

Xuất phát từ nhận định đó, luận án không xem lựa chọn đặc trưng và xử lý mất cân bằng dữ liệu là hai hướng nghiên cứu độc lập mà là hai bước kế tiếp trong quá trình tối ưu hóa dữ liệu đầu vào. Trên cơ sở xác định được tập đặc trưng tối ưu, luận án tiếp tục nghiên

cứu kết hợp các phương pháp lựa chọn đặc trưng với mô hình sinh dữ liệu dựa trên mạng đối kháng sinh (GAN) nhằm xây dựng tập dữ liệu huấn luyện có chất lượng cao hơn. Sự kết hợp này không chỉ khắc phục đồng thời hiện tượng dư thừa đặc trưng và mất cân bằng dữ liệu mà còn góp phần nâng cao hiệu suất và độ chính xác của các mô hình dự đoán lỗi, tạo tiền đề cho việc nghiên cứu các phương pháp biểu diễn mã nguồn tiên tiến ở giai đoạn tiếp theo của luận án.

Thứ ba, mặc dù việc tối ưu hóa dữ liệu đầu vào có thể nâng cao đáng kể hiệu suất của các mô hình dự đoán truyền thống, hướng tiếp cận này vẫn tồn tại một số hạn chế. Trước hết, các mô hình dự đoán yêu cầu trích xuất đặc trưng thủ công, và hiệu quả của mô hình phụ thuộc nhiều vào chất lượng và mức độ phù hợp của các độ đo được chọn [33]. Hơn nữa, các tập đặc trưng truyền thống thường không phản ánh được thông tin ngữ nghĩa của mã nguồn, một trong những yếu tố quan trọng trong dự đoán lỗi phần mềm, do đó làm giảm khả năng của mô hình trong việc hiểu cấu trúc và hành vi sâu hơn của phần mềm. Mặc dù hai đoạn mã nguồn thể hiện các đặc điểm lập trình tương đồng về độ dài mã, chức năng, và chuỗi ký hiệu thô của mã nguồn, chúng vẫn có thể được biểu diễn dưới dạng vec-tơ đặc trưng thông qua các đặc trưng truyền thống. Tuy nhiên, thông tin ngữ nghĩa và cú pháp giữa hai đoạn mã lại có sự khác biệt đáng kể. Do đó, để xây dựng các mô hình dự đoán lỗi phần mềm một cách chính xác hơn, cần có các đặc trưng có khả năng phân biệt được những khác biệt ngữ nghĩa này. Để biểu diễn các đặc trưng ngữ nghĩa và cú pháp, một số phương pháp thường được hay sử dụng như cây cú pháp trừu tượng (AST - Abstract Syntax Tree), đồ thị luồng điều khiển (CFG - Control Flow Graph) và đồ thị luồng dữ liệu (DFG - Data Flow Graph), mô hình ngôn ngữ dành cho mã nguồn. Các đặc trưng trích xuất từ những phương pháp này sẽ được đưa vào học bởi mô hình học sâu nhằm nâng cao hiệu quả của các mô hình dự đoán lỗi phần mềm. Lin và cộng sự [34] sử dụng (Bi-LSTM - Bidirectional Long Short-Term Memory) để học biểu diễn của AST trong việc phát hiện các hàm dễ bị lỗi. Gupta và cộng sự [35] phát triển một mạng nơ-ron tuần tự nhiều lớp kết hợp với tập trung chú ý (*attention*), có tên là DeepFix, nhằm dự đoán vị trí của các lỗi trong chương trình. Zhou và cộng sự [36] xây dựng mô hình (LSTM - Long Short-Term Memory) dựa trên cấu trúc hai chiều và cây nhằm nâng cao khả năng học đặc trưng và hiệu quả dự đoán. Họ sử dụng một lớp biểu diễn không gian vec-tơ để trích xuất đặc trưng đầu vào cho LSTM-BT và thực hiện dự đoán. Tuy nhiên, các mạng tuần tự không thể biểu diễn thông tin cú pháp và ngữ nghĩa được trích xuất từ cây cú pháp trừu tượng và chúng gặp khó khăn trong việc nắm bắt các mối quan hệ phụ thuộc phức tạp trong mã nguồn [36]. Do đặc điểm xử lý tuần tự của các mô hình học sâu như LSTM và RNN gặp khó khăn khi mô hình hóa các quan hệ phụ thuộc xa trong mã nguồn. Khi chiều dài chương trình tăng lên, thông tin từ các câu lệnh ở đầu chuỗi có xu hướng bị suy giảm hoặc mất dần trong quá trình truyền qua nhiều bước thời gian, làm hạn chế khả năng nắm

bắt các mối quan hệ giữa các thành phần nằm cách xa nhau trong chương trình. Đồng thời, cơ chế xử lý tuần tự cũng làm giảm khả năng tính toán song song, dẫn đến thời gian huấn luyện dài hơn và khó mở rộng đối với các dự án phần mềm có quy mô lớn. Trong khi đó, kiến trúc Transformer với cơ chế self-attention cho phép mô hình đánh giá trực tiếp mức độ liên quan giữa mọi cặp token trong mã nguồn mà không phụ thuộc vào khoảng cách vị trí của chúng trong chuỗi. Nhờ đó, mô hình có thể đồng thời học được các quan hệ phụ thuộc ngắn và dài, chẳng hạn như mối liên hệ giữa khai báo và sử dụng biến, giữa lời gọi hàm và định nghĩa hàm, hoặc giữa các cấu trúc điều khiển nằm ở những vị trí cách xa nhau trong chương trình. Bên cạnh đó, cơ chế self-attention còn hỗ trợ tính toán song song, giúp nâng cao hiệu quả huấn luyện và cải thiện khả năng mở rộng của mô hình đối với các tập dữ liệu lớn. Xuất phát từ những ưu điểm đó, luận án lựa chọn nghiên cứu và phát triển mô hình học sâu dựa trên kiến trúc Transformer nhằm nâng cao hiệu quả dự đoán lỗi phần mềm. Cụ thể, luận án khai thác biểu diễn mã nguồn thông qua kỹ thuật nhúng FastText kết hợp với cơ chế self-attention của Transformer để học trực tiếp các đặc trưng cú pháp và ngữ nghĩa của chương trình. Hướng tiếp cận này được kỳ vọng sẽ khắc phục những hạn chế của các mô hình dựa trên đặc trưng được thiết kế thủ công cũng như các mạng học sâu tuần tự.

Từ những phân tích về các vấn đề đặt ra trong nghiên cứu dự đoán lỗi phần mềm ở trên có thể thấy rằng, việc lựa chọn hướng tiếp cận phù hợp đóng vai trò quan trọng trong việc nâng cao hiệu quả của các mô hình dự đoán. Các nghiên cứu hiện nay chủ yếu phát triển theo hai hướng tiếp cận chính, bao gồm dự đoán lỗi phần mềm dựa trên các độ đo mã nguồn và dựa trên các đặc trưng ngữ nghĩa của mã nguồn. Mỗi hướng tiếp cận khai thác những loại thông tin khác nhau từ mã nguồn, có những ưu điểm, hạn chế và phạm vi áp dụng riêng. Việc phân tích, đối chiếu hai hướng tiếp cận này không chỉ giúp làm rõ xu hướng phát triển của lĩnh vực mà còn là cơ sở để xác định định hướng nghiên cứu của luận án và làm nổi bật tính mới cũng như các đóng góp khoa học đạt được. Hướng tiếp cận dựa trên độ đo mã nguồn khai thác các đặc trưng định lượng như độ phức tạp, kích thước và lịch sử thay đổi của mã nguồn. Ưu điểm của hướng tiếp cận này là chi phí trích xuất đặc trưng thấp, mô hình có khả năng diễn giải tốt, dễ triển khai trên các dự án thực tế và hoạt động hiệu quả ngay cả khi dữ liệu huấn luyện không lớn. Tuy nhiên, các độ đo mã nguồn chủ yếu phản ánh các đặc điểm bề mặt của phần mềm nên chưa biểu diễn đầy đủ ngữ cảnh lập trình, cấu trúc cú pháp và ý nghĩa ngữ nghĩa của mã nguồn. Do đó, khả năng phát hiện các mẫu lỗi phức tạp hoặc các quan hệ phụ thuộc dài hạn còn hạn chế.

Ngược lại, hướng tiếp cận dựa trên đặc trưng ngữ nghĩa mã nguồn sử dụng các biểu diễn như mã thông báo (token), cây cú pháp trừu tượng (AST), đồ thị luồng điều khiển (CFG), đồ thị luồng dữ liệu (DFG) hoặc các biểu diễn được học tự động thông qua các mô hình học sâu như CNN, RNN, GNN, Transformer và các mô hình ngôn ngữ mã nguồn

(CodeBERT, GraphCodeBERT,...). Hướng tiếp cận này có khả năng khai thác sâu các quan hệ cú pháp và ngữ nghĩa, học được các đặc trưng biểu diễn có tính khái quát cao và nâng cao độ chính xác của dự đoán lỗi, đặc biệt đối với các hệ thống phần mềm lớn và phức tạp. Tuy nhiên, phương pháp này đòi hỏi chi phí tính toán cao, yêu cầu tập dữ liệu lớn, chất lượng gán nhãn tốt và quá trình huấn luyện, tinh chỉnh mô hình phức tạp hơn so với các phương pháp dựa trên độ đo mã nguồn.

Từ góc độ phạm vi áp dụng, các phương pháp dựa trên độ đo mã nguồn phù hợp với các bài toán yêu cầu khả năng triển khai nhanh, tài nguyên tính toán hạn chế hoặc cần tính giải thích cao. Trong khi đó, các phương pháp dựa trên đặc trưng ngữ nghĩa thích hợp với các hệ thống có quy mô lớn, dữ liệu phong phú và yêu cầu độ chính xác dự đoán cao. Hai hướng tiếp cận không mang tính thay thế hoàn toàn mà có tính bổ sung cho nhau; việc kết hợp các đặc trưng định lượng với các biểu diễn ngữ nghĩa đang được xem là xu hướng nghiên cứu đầy tiềm năng nhằm khai thác đồng thời thông tin thống kê, cấu trúc và ngữ nghĩa của mã nguồn.

Trong bối cảnh đó, luận án **“Dự đoán lỗi phần mềm dựa trên mã nguồn”** được chọn làm nội dung của luận án Tiến sĩ kỹ thuật nhằm đóng góp cho sự phát triển và đưa vào ứng dụng thực tế trong ngành công nghệ phần mềm. Luận án hướng tới việc phát triển các phương pháp dự đoán lỗi phần mềm có khả năng hỗ trợ hiệu quả cho hoạt động kiểm thử, giúp rút ngắn thời gian tìm kiếm lỗi, tối ưu hóa quy trình kiểm thử và giảm thiểu chi phí liên quan. Với mục tiêu nghiên cứu và phát triển các phương pháp nâng cao hiệu quả dự đoán lỗi phần mềm theo hai hướng tiếp cận bổ trợ. Hướng thứ nhất tập trung nâng cao chất lượng các mô hình dự đoán dựa trên đặc trưng độ đo mã nguồn thông qua lựa chọn đặc trưng và xử lý mất cân bằng dữ liệu bằng các mô hình GAN. Hướng thứ hai tập trung khai thác trực tiếp thông tin cú pháp và ngữ nghĩa của mã nguồn bằng mô hình FastText-Transformer nhằm khắc phục những hạn chế của các đặc trưng được thiết kế thủ công. Các kết quả nghiên cứu của luận án tạo thành một lộ trình hoàn chỉnh, có tính kế thừa từ cải thiện chất lượng dữ liệu, cải thiện quá trình học của mô hình đến nâng cao khả năng biểu diễn mã nguồn, góp phần nâng cao độ chính xác và khả năng ứng dụng của các mô hình dự đoán lỗi phần mềm trong thực tiễn.

2. Mục tiêu, đối tượng và phạm vi nghiên cứu

Mục tiêu của luận án là nghiên cứu về bài toán dự đoán lỗi phần mềm, tập trung vào việc phân tích các vấn đề, hạn chế và thách thức còn tồn tại trong các phương pháp dự đoán hiện nay. Trên cơ sở đó, luận án nghiên cứu, đề xuất và phát triển các kỹ thuật học máy và học sâu nhằm nâng cao hiệu suất của mô hình dự đoán lỗi phần mềm. Luận án tập trung

vào mục tiêu cụ thể sau:

- Thứ nhất, luận án nghiên cứu tổng quan về dự đoán lỗi phần mềm và các kỹ thuật được nghiên cứu trong dự đoán lỗi phần mềm; tình hình nghiên cứu và ứng dụng của dự đoán lỗi phần mềm trong nước và trên thế giới.
- Thứ hai, luận án tập trung vào việc nghiên cứu và đề xuất các phương pháp lựa chọn đặc trưng nhằm loại bỏ những đặc trưng dư thừa hoặc không liên quan trong tập dữ liệu dự đoán lỗi phần mềm, giúp giảm nhiễu và độ phức tạp của mô hình, đồng thời nâng cao chất lượng của dữ liệu đầu vào và độ chính xác của quá trình dự đoán lỗi. Bên cạnh đó, luận án phân tích và đánh giá hiệu quả của các kỹ thuật lựa chọn đặc trưng, đồng thời đề xuất tích hợp các phương pháp này với kỹ thuật xử lý mất cân bằng dữ liệu để cải thiện chất lượng dự đoán trong các hệ thống có phân bố lỗi không đồng đều.
- Thứ ba, tiếp theo, trên cơ sở tập đặc trưng đã được tối ưu, luận án tập trung vào việc giải quyết vấn đề mất cân bằng lớp trong các mô hình dự đoán lỗi phần mềm nhằm nâng cao hiệu suất và độ chính xác của mô hình. Luận án hướng đến việc phân tích ảnh hưởng của sự mất cân bằng dữ liệu đến hiệu năng mô hình dự đoán lỗi, đồng thời đề xuất áp dụng các kỹ thuật lấy mẫu dữ liệu để cân bằng lại tập huấn luyện kết hợp với áp dụng các phương pháp chọn đặc trưng; đặc biệt luận án đề xuất sử dụng mạng đối kháng (GAN) như một giải pháp tạo dữ liệu tổng hợp có tính chân thực cao hơn, phản ánh tốt hơn phân phối toàn cục và mối quan hệ giữa các đặc trưng. Việc kết hợp hai nhóm kỹ thuật lựa chọn đặc trưng và cân bằng dữ liệu hướng tới mục tiêu tối ưu hóa toàn diện dữ liệu đầu vào, qua đó nâng cao hiệu suất và độ chính xác của các mô hình dự đoán lỗi phần mềm.
- Thứ tư, trên cơ sở những giới hạn của các mô hình dự đoán dựa trên đặc trưng được thiết kế thủ công, luận án nghiên cứu hướng tiếp cận dựa trên học sâu nhằm khai thác trực tiếp thông tin cú pháp và ngữ nghĩa của mã nguồn. Cụ thể, luận án đề xuất giải pháp khai thác thông tin cú pháp và ngữ nghĩa của chương trình để mô hình có thể học được các đặc trưng ngữ nghĩa quan trọng, qua đó nâng cao độ chính xác và khả năng khái quát hóa trong dự đoán lỗi phần mềm. Từ đó, luận án đề xuất mô hình dự đoán lỗi phần mềm dựa trên kiến trúc Transformer, kết hợp với embedding bằng Gensim FastText được trích xuất từ cây AST của mã nguồn.

Dựa vào mục tiêu nghiên cứu nói trên, luận án tập trung vào đối tượng nghiên cứu của luận án bao gồm các phương pháp và kỹ thuật trong dự đoán lỗi phần mềm, các kỹ thuật lựa chọn đặc trưng, xử lý mất cân bằng dữ liệu nhằm tối ưu tập dữ liệu đầu vào, trích xuất đặc trưng ngữ nghĩa và cú pháp của mã nguồn, mô hình word-embedding để học các

véc-tơ biểu diễn cho chuỗi token trong mã nguồn, cùng với các mô hình học máy và học sâu được sử dụng để xây dựng hệ thống dự đoán lỗi phần mềm có chất lượng. Đối tượng nghiên cứu này cung cấp cơ sở lý thuyết và thực tiễn cho việc nghiên cứu, phát triển và cải tiến chất lượng của các mô hình dự đoán lỗi phần mềm.

Phạm vi nghiên cứu của luận án này tập trung nghiên cứu bài toán dự đoán lỗi phần mềm trên các dự án được phát triển bằng ngôn ngữ lập trình Java. Việc lựa chọn Java xuất phát từ tính phổ biến của ngôn ngữ này trong phát triển phần mềm, đồng thời các bộ dữ liệu chuẩn được sử dụng rộng rãi trong nghiên cứu dự đoán lỗi phần mềm, như NASA, PROMISE và Apache, chủ yếu được xây dựng từ các dự án Java. Điều này tạo điều kiện thuận lợi cho việc xây dựng, đánh giá và so sánh các phương pháp đề xuất với các nghiên cứu hiện có.

3. Nhiệm vụ nghiên cứu và kết quả đạt được

Xác định mục tiêu và đối tượng nghiên cứu như trên, luận án tập trung vào thực hiện các nhiệm vụ chính sau đây:

1. Nghiên cứu tổng quan về dự đoán lỗi phần mềm, bao gồm các khái niệm, vai trò, quy trình dự đoán lỗi và các mô hình học máy, học sâu đã được áp dụng trong dự đoán lỗi phần mềm. Trên cơ sở phân tích, đánh giá các công trình nghiên cứu trong và ngoài nước, xác định những hạn chế và thách thức nghiên cứu, làm cơ sở cho việc đề xuất các phương pháp và mô hình dự đoán lỗi phần mềm.
2. Nghiên cứu các phương pháp nâng cao chất lượng dữ liệu đầu vào cho mô hình dự đoán lỗi phần mềm thông qua lựa chọn đặc trưng. Nhiệm vụ này tập trung vào việc phân tích, đánh giá và đề xuất các phương pháp lựa chọn đặc trưng nhằm loại bỏ các đặc trưng dư thừa hoặc không liên quan đến lỗi, giảm nhiễu, giảm số chiều dữ liệu và xây dựng tập đặc trưng tối ưu, qua đó nâng cao hiệu quả học và độ chính xác của các mô hình dự đoán lỗi.
3. Trên cơ sở tập đặc trưng đã được tối ưu, nghiên cứu các phương pháp xử lý mất cân bằng dữ liệu nhằm nâng cao chất lượng tập dữ liệu huấn luyện. Luận án tập trung phân tích ảnh hưởng của hiện tượng mất cân bằng dữ liệu đến hiệu quả dự đoán lỗi, đồng thời nghiên cứu và đề xuất kết hợp các phương pháp lựa chọn đặc trưng với các kỹ thuật lấy mẫu, đặc biệt là mạng đối kháng sinh (GAN). Việc kết hợp lựa chọn đặc trưng và xử lý mất cân bằng dữ liệu hướng tới mục tiêu tối ưu hóa toàn diện dữ liệu đầu vào, góp phần nâng cao hiệu suất, độ chính xác và khả năng tổng quát hóa của các mô hình dự đoán lỗi phần mềm.

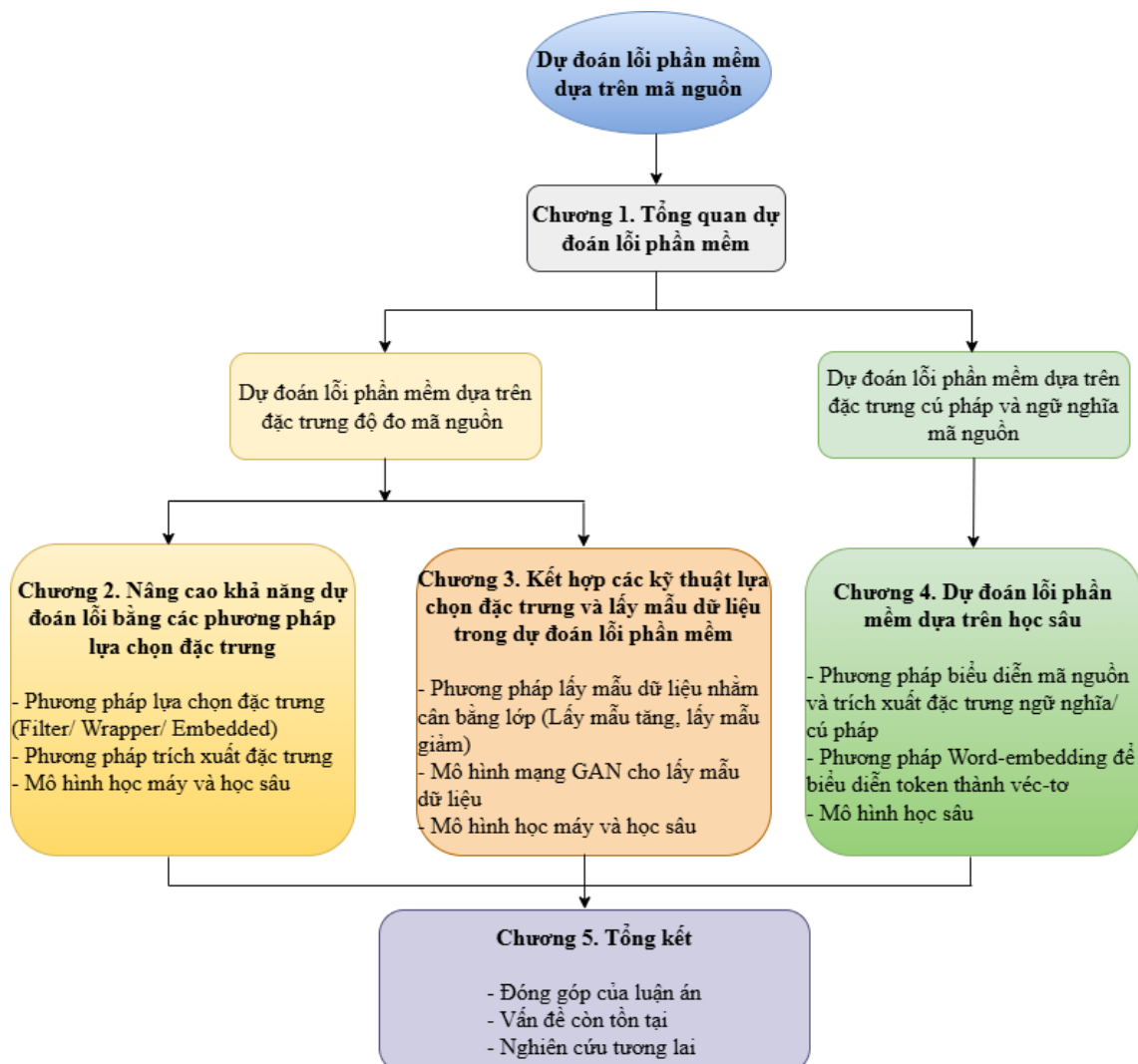
-
4. Sau khi tối ưu hóa dữ liệu đầu vào cho các mô hình dự đoán dựa trên các đặc trưng được thiết kế thủ công truyền thống, nghiên cứu hướng tiếp cận dựa trên học sâu nhằm nâng cao khả năng biểu diễn của mã nguồn. Cụ thể, luận án nghiên cứu các phương pháp trích xuất thông tin cú pháp và ngữ nghĩa từ mã nguồn nhằm biểu diễn chương trình dưới dạng đặc trưng có ý nghĩa phục vụ cho bài toán dự đoán lỗi phần mềm. Đồng thời, nghiên cứu kiến trúc Transformer trong các mô hình học sâu và xử lý ngôn ngữ tự nhiên để tận dụng khả năng học biểu diễn ngữ cảnh và quan hệ phụ thuộc giữa các thành phần trong mã nguồn, qua đó nâng cao hiệu suất của mô hình dự đoán.

Các kết quả chính của luận án bao gồm:

1. Luận án đã hệ thống hóa cơ sở lý thuyết, quy trình và các vấn đề ảnh hưởng đến chất lượng của mô hình dự đoán lỗi phần mềm, qua đó hình thành nghiên cứu tổng thể kết hợp giữa tiền xử lý dữ liệu, lựa chọn đặc trưng, cân bằng dữ liệu và trích xuất đặc trưng ngữ nghĩa mã nguồn.
2. Luận án đã đề xuất giải pháp phân tích, đánh giá hiệu quả vai trò của các phương pháp lựa chọn đặc trưng trong việc nâng cao chất lượng của mô hình dự đoán lỗi. Đồng thời, đề xuất kết hợp các kỹ thuật xử lý mất cân bằng dữ liệu với các phương pháp chọn đặc trưng để cải thiện hơn nữa chất lượng dự đoán các mô hình.
3. Luận án đã đề xuất giải pháp tối ưu và cân bằng dữ liệu đầu vào bằng cách kết hợp các phương pháp lựa chọn đặc trưng (Recursive Feature Elimination và Autoencoders) với các biến thể GAN (bao gồm CTGAN, WGAN-GP, VanillaGAN, CopulaGAN và TGAN) để sinh thêm mẫu cho lớp thiểu số. Kết quả thực nghiệm trên năm bộ dữ liệu lỗi phần mềm cho thấy CTGAN, WGAN-GP và VanillaGAN cho hiệu suất vượt trội so với các phương pháp truyền thống như SMOTE hay ADASYN, đồng thời khẳng định rằng việc kết hợp chọn đặc trưng và sinh dữ liệu bằng GAN giúp nâng cao độ hiệu suất và khả năng khái quát của mô hình dự đoán lỗi phần mềm.
4. Luận án đã đề xuất, xây dựng và thử nghiệm thành công mô hình dự đoán lỗi phần mềm FastText-Transformer, kết hợp giữa mô hình nhúng từ Gensim FastText và kiến trúc học sâu Transformer nhằm khai thác hiệu quả thông tin ngữ nghĩa và cú pháp tiềm ẩn trong mã nguồn thông qua cấu trúc cây cú pháp trừu tượng. Từ đó tạo tiền đề cho việc áp dụng mô hình này trong các công cụ hỗ trợ kiểm thử phần mềm và đánh giá chất lượng mã nguồn tự động.

4. Cấu trúc của luận án

Hình 0.1 trình bày cấu trúc tổng thể của luận án, được tổ chức theo hai hướng nghiên cứu chính. Hướng thứ nhất tập trung vào dự đoán lỗi phần mềm dựa trên đặc trưng độ đo mã nguồn, bao gồm các kỹ thuật lựa chọn đặc trưng, trích xuất đặc trưng và mở rộng dữ liệu nhằm nâng cao khả năng phân loại. Hướng thứ hai mở rộng sang dự đoán lỗi dựa trên cú pháp và ngữ nghĩa của mã nguồn, nơi các mô hình học sâu và các phương pháp biểu diễn mã nguồn được khai thác để cải thiện hiệu quả dự đoán. Trên cơ sở các nội dung nghiên cứu để đạt mục tiêu đề ra và đảm bảo tính logic và tính thống nhất, luận án được cấu trúc gồm các chương như sau:



Hình 0.1: Cấu trúc của luận án.

Chương 1: Tổng quan dự đoán lỗi phần mềm. Chương này trình bày các khái niệm cơ bản về dự đoán lỗi phần mềm, đồng thời giới thiệu các kỹ thuật, phương pháp và mô

hình trong lĩnh vực này, qua đó cung cấp cái nhìn tổng quan về tình hình nghiên cứu và xu hướng phát triển của bài toán dự đoán lỗi phần mềm trong những năm gần đây.

Chương 2: Nâng cao khả năng dự đoán lỗi phần mềm bằng các phương pháp lựa chọn đặc trưng. Chương này giới thiệu tổng quan về phương pháp lựa chọn đặc trưng trong dự đoán lỗi phần mềm. Trên cơ sở của các nghiên cứu lựa chọn đặc trưng, đề xuất nghiên cứu và đánh giá hiệu quả của các phương pháp lựa chọn đặc trưng.

Chương 3. Kết hợp các kỹ thuật lựa chọn đặc trưng và lấy mẫu dữ liệu trong dự đoán lỗi phần mềm. Để cải tiến chất lượng của các mô hình dự đoán lỗi phần mềm dựa trên đặc trưng mã nguồn, đã có nhiều kỹ thuật cải tiến được nghiên cứu, trong đó kết hợp các kỹ thuật lựa chọn đặc trưng và lấy mẫu dữ liệu được đề xuất.

Chương 4. Dự đoán lỗi phần mềm dựa trên học sâu. Chương này trình bày đề xuất mô hình dự đoán lỗi phần mềm dựa có khả năng học các đặc trưng ngữ nghĩa quan trọng của chương trình phần mềm nhằm cải thiện chất lượng dự đoán của mô hình.

5. Đóng góp chính của Luận án

- Nâng cao hiệu quả dự đoán lỗi phần mềm thông qua việc đánh giá và so sánh có hệ thống các phương pháp lựa chọn đặc trưng filter và wrapper nhằm xác định tập đặc trưng tối ưu, góp phần giảm chiều dữ liệu, loại bỏ các đặc trưng dư thừa và không liên quan, từ đó cải thiện độ chính xác và cung cấp cơ sở thử nghiệm giúp lựa chọn phương pháp phù hợp cho bài toán dự đoán lỗi phần mềm.
- Đề xuất và đánh giá thử nghiệm việc ứng dụng các mô hình sinh dữ liệu đối kháng GAN và các biến thể của chúng nhằm giải quyết vấn đề mất cân bằng dữ liệu trong các tập dữ liệu lỗi phần mềm dạng bảng. Áp dụng các mô hình GAN nhằm tạo dữ liệu tổng hợp chất lượng cao cho lớp thiểu số, từ đó cân bằng phân phối lớp, cải thiện độ chính xác và tăng khả năng khái quát hóa của các mô hình dự đoán lỗi phần mềm. Kết quả nghiên cứu cung cấp cơ sở thử nghiệm quan trọng giúp các nhà nghiên cứu lựa chọn phương pháp sinh dữ liệu phù hợp cho các bài toán học máy có dữ liệu mất cân bằng trong lĩnh vực công nghệ phần mềm. Trên cơ sở tập đặc trưng được tối ưu, luận án đề xuất kết hợp các phương pháp lựa chọn đặc trưng Recursive Feature Elimination (RFE) và Autoencoder với các mô hình sinh dữ liệu dựa trên mạng đối kháng GAN nhằm giải quyết bài toán mất cân bằng dữ liệu trong dự đoán lỗi phần mềm. Kết quả thực nghiệm cho thấy việc kết hợp RFE và Autoencoder với các biến thể GAN, đặc biệt là CTGAN, WGAN GP và VanillaGAN, tạo ra tập dữ liệu huấn luyện có chất lượng cao hơn, giúp các mô hình dự đoán đạt hiệu suất vượt

trội so với các kỹ thuật lấy mẫu truyền thống như SMOTE và ADASYN. Kết quả này khẳng định rằng việc tối ưu hóa đồng thời tập đặc trưng và cân bằng dữ liệu là hướng tiếp cận hiệu quả, góp phần nâng cao độ chính xác, khả năng tổng quát hóa và tính ổn định của các mô hình dự đoán lỗi phần mềm.

- Đề xuất mô hình học sâu dựa trên kiến trúc Transformer, kết hợp với biểu diễn ngữ nghĩa mã nguồn bằng kỹ thuật FastText embedding được trích xuất từ cây cú pháp trừu tượng. Mô hình được thiết kế nhằm khai thác và học tự động các đặc trưng cú pháp và ngữ nghĩa tiềm ẩn trong mã nguồn, từ đó nâng cao khả năng nhận diện và dự đoán các mô-đun dễ phát sinh lỗi. Khác với các phương pháp truyền thống dựa trên đặc trưng thủ công, mô hình được đề xuất có khả năng trích chọn biểu diễn véc-tơ ngữ nghĩa có ý nghĩa từ cấu trúc AST, đồng thời khắc phục hạn chế của các mạng tuần tự trong việc mô hình hóa các phụ thuộc phức tạp trong mã nguồn. Bằng cách sử dụng cơ chế (*self-attention*) của Transformer, mô hình có thể nắm bắt quan hệ ngữ cảnh toàn cục, góp phần cải thiện độ chính xác và tính khái quát hóa trong dự đoán lỗi phần mềm.

Chương 1

Tổng quan dự đoán lỗi phần mềm

Chương này trình bày tổng quan về lĩnh vực dự đoán lỗi, các kỹ thuật nhằm cải tiến hiệu suất các mô hình dự đoán lỗi phần mềm dựa trên mã nguồn và các ứng dụng trên cơ sở tổng hợp và phân tích các công trình nghiên cứu liên quan.

1.1 Dự đoán lỗi phần mềm

1.1.1 Khái niệm

Nhiều ứng dụng của lĩnh vực công nghệ phần mềm đã được triển khai phổ biến trên nhiều hệ thống, chẳng hạn như hệ thống phòng thủ an ninh quốc gia, hệ thống kiểm soát không lưu và hệ thống mạng hoặc lưới điện cho các thiết bị tiêu dùng [37]. Tuy nhiên, trên thực tế, không thể tạo ra một phần mềm hoàn toàn không có lỗi do các giới hạn về nhân lực, ngân sách và thời gian. Do đó, cần có một chiến lược phù hợp để phát hiện lỗi phần mềm thông qua quy trình lập kế hoạch hợp lý. Lỗi phần mềm là một lỗi ẩn trong lập trình, và một số nguyên nhân phổ biến dẫn đến lỗi phần mềm có thể bao gồm: mục tiêu dự án phi thực tế hoặc ước tính tài nguyên không chính xác do yêu cầu hệ thống không phù hợp [38]. Việc loại bỏ các lỗi này giúp cải thiện chất lượng phần mềm và kiểm soát chi phí. Hơn nữa, các lỗi tồn tại trong hệ thống phần mềm ảnh hưởng đến chất lượng và độ tin cậy của phần mềm. Ngay cả những lỗi nhỏ cũng có thể dẫn đến đầu ra sai và hành vi không mong muốn của chương trình, từ đó gây ra thất bại cho dự án. Để phù hợp với mục tiêu chính của kỹ sư phần mềm là cung cấp và đảm bảo phần mềm không có lỗi cho người dùng cuối, phần mềm phải được kiểm thử kỹ lưỡng [10].

Dự đoán lỗi phần mềm [39] tập trung vào việc thiết kế và phát triển các mô hình dự

đoán dựa trên dữ liệu phần mềm (ví dụ: độ phức tạp của mã nguồn, lịch sử thay đổi, v.v.) để dự đoán lỗi trong các phần mã mới (tức là tệp, lớp, phương thức, v.v.). Các mô hình dự đoán này giúp các nhà phát triển phần mềm tập trung vào các mô-đun dễ xảy ra lỗi và tối ưu hóa nỗ lực cũng như tài nguyên của họ [40]. Trong nhiều năm qua, các kỹ thuật học máy đã được áp dụng để phát triển các mô hình dự đoán lỗi phần mềm. Những kỹ thuật này sử dụng dữ liệu lỗi phần mềm lịch sử (thu thập từ các dự án phần mềm trước đó) để huấn luyện mô hình dự đoán và xác định các mô-đun có lỗi. Ban đầu, các phương pháp xây dựng mô hình dự đoán dựa trên các kỹ thuật học máy truyền thống như Naive Bayes (NB) [15], Support Vector Machine (SVM) [16], Decision Tree (DT) [17], v.v. Những mô hình này được huấn luyện trên dữ liệu lịch sử và sử dụng một tập hợp các độ đo để tối ưu hóa quá trình học. Để xác định một tệp mã nguồn có chứa lỗi hay không, một tác vụ dự đoán được ký hiệu là $predict(f)$, trong đó tệp f là đầu vào và hàm trả về giá trị 0 nếu không có lỗi và 1 nếu có lỗi. Hàm phân loại này $predict(f)$ được xấp xỉ thông qua quá trình học từ một tập mẫu được cung cấp trong bộ dữ liệu huấn luyện.

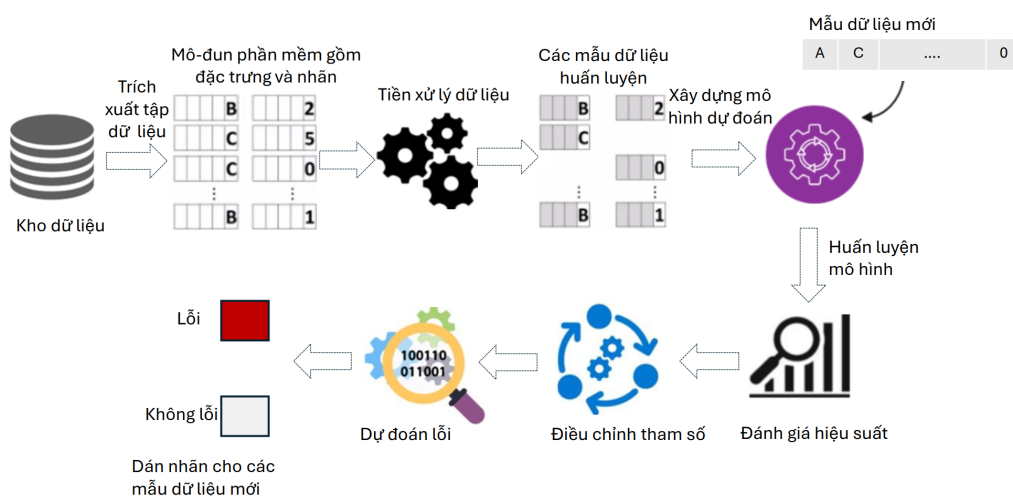
$$predict(f) = \begin{cases} 1 & \text{Nếu tệp } f \text{ có chứa lỗi} \\ 0 & \text{Ngược lại} \end{cases} \quad (1.1)$$

Tuy nhiên, các mô hình học máy thường đối mặt với một số hạn chế nhất định. Thứ nhất, đối với tập dữ liệu lỗi, không phải tất cả các đặc trưng đều có giá trị phân loại; nhiều đặc trưng có thể dư thừa hoặc nhiễu. Vì vậy, dựa vào áp dụng phương pháp lựa chọn đặc trưng, các dữ liệu không liên quan được loại bỏ, từ đó cải thiện tốc độ học và hiệu quả dự đoán của mô hình. Thứ hai, trong các tập dữ liệu dự đoán lỗi phần mềm, số lượng mô-đun chứa lỗi thường rất nhỏ so với số lượng mô-đun không có lỗi. Tỷ lệ mất cân bằng này dẫn đến việc các mô hình học máy có xu hướng dự đoán thiên về lớp “không có lỗi”. Vấn đề này đặt ra nhu cầu cấp thiết phải nghiên cứu các kỹ thuật cân bằng dữ liệu. Thứ ba, đối với các mô hình dự đoán lỗi truyền thống, cần phải trích xuất đặc trưng một cách thủ công để xây dựng các mô hình này, trong khi chất lượng của các độ đo đặc trưng mã nguồn ảnh hưởng trực tiếp đến hiệu suất của mô hình đó [17]. Hơn nữa, các đặc trưng truyền thống không xem xét đến đặc trưng ngữ nghĩa và cú pháp (một đặc trưng quan trọng của dự đoán lỗi phần mềm) của mã nguồn. Do đó, các mô hình gần đây đã áp dụng các thuật toán học sâu [41, 42] như Deep Belief Network (DBN), Convolutional Neural Network (CNN) và LSTM để khai thác ngữ nghĩa nổi bật và thông tin ngữ cảnh khi dự đoán lỗi phần mềm. Các mô hình này trích xuất các nút cụ thể từ cây cú pháp trừu tượng được trích xuất từ mã nguồn và chuyển các nút đó thành chuỗi để đưa vào mô hình học sâu đề xuất. Mô hình này tạo ra các véc-tơ để biểu diễn các đặc trưng ngữ nghĩa của chương trình. Mô hình sử dụng cả nhãn lỗi và các đặc trưng được tạo ra đã đạt hiệu suất đáng kể trong dự đoán lỗi cả trong

cùng dự án và giữa các dự án so với các mô hình truyền thống dựa trên đặc trưng. Vì vậy trong luận án này, luận án xem xét nghiên cứu đến cả nội dung lựa chọn đặc trưng và cân bằng dữ liệu đối với các mô hình đoán lỗi phần mềm dựa trên độ đo đặc trưng mã nguồn nhằm cải thiện hiệu suất mô hình dự đoán lỗi và mô hình dựa trên phân tích cú pháp, ngữ nghĩa.

1.1.2 Quy trình dự đoán lỗi phần mềm

Hình 1.1 trình bày quy trình dự đoán lỗi phần mềm [43].



Hình 1.1: Quy trình dự đoán lỗi phần mềm

1. Thu thập dữ liệu: Tập dữ liệu lỗi có thể được tạo ra hoặc thu thập từ các kho dữ liệu công khai khác nhau như PROMISE, NASA và Apache.
2. Tiền xử lý dữ liệu: Tập dữ liệu dùng cho dự đoán lỗi phần mềm thường tiềm ẩn nhiều hạn chế, chẳng hạn như nhiễu trong dữ liệu, sự tồn tại của các đặc trưng dư thừa hoặc không mang ý nghĩa, sự xuất hiện của các giá trị ngoại lai, cũng như vấn đề mất cân bằng giữa các lớp. Do đó, các kỹ thuật tiền xử lý dữ liệu khác nhau được áp dụng, chẳng hạn như chuẩn hóa dữ liệu, lựa chọn đặc trưng, trích xuất đặc trưng và lấy mẫu dữ liệu, nhằm phục vụ việc nâng cao chất lượng dự đoán lỗi.
3. Trích xuất đặc trưng và tạo tập dữ liệu huấn luyện: Trong bước này, các thuộc tính đặc trưng được tạo ra dựa trên thông tin thu thập từ mã nguồn và dữ liệu nhật ký phát triển của dự án phần mềm. Các đặc trưng đã trích xuất được kết hợp với thông tin lỗi để tạo thành tập dữ liệu huấn luyện, đóng vai trò là đầu vào cho mô hình học máy nhằm dự đoán lỗi.

-
4. Xây dựng mô hình dự đoán lỗi được thực hiện bằng cách áp dụng các phương pháp thống kê và thuật toán học máy trên tập dữ liệu huấn luyện. Sau quá trình huấn luyện, mô hình có khả năng nhận diện các mô-đun phần mềm tiềm ẩn nguy cơ phát sinh lỗi.
 5. Đánh giá hiệu suất: Sau các bước huấn luyện, một tập dữ liệu kiểm thử được sử dụng để đánh giá mô hình dự đoán lỗi. Hiệu suất của mô hình có thể được đo lường bằng cách so sánh giá trị dự đoán của các mô-đun dễ mắc lỗi do mô hình tạo ra với giá trị thực tế của các mô-đun này, sử dụng các độ đo chuẩn như accuracy (Acc), precision (P), recall (R), F1-score (F1), Area Under the Curve (AUC) và Matthews Correlation Coefficient (MCC).
 6. Điều chỉnh tham số: Các tham số của mô hình được điều chỉnh nhằm đạt độ chính xác cao và cải thiện hiệu suất.
 7. Dự đoán lỗi: Mô hình dự đoán lỗi được áp dụng để xác định các mô-đun phần mềm có khả năng chứa lỗi trong các dự án thực tế.

1.2 Các phương pháp dự đoán lỗi phần mềm

Các kỹ thuật dự đoán lỗi phần mềm truyền thống chủ yếu dựa trên đặc trưng độ đo mã nguồn, bao gồm các độ đo độ phức tạp, độ kết nối, độ gắn kết, kích thước mã, hoặc các độ đo hướng đối tượng. Các độ đo này phản ánh đặc tính cấu trúc và chất lượng của mã nguồn, từ đó hỗ trợ các mô hình học máy như Logistic Regression, Naïve Bayes, SVM, Random Forest hay các thuật toán metaheuristic trong việc phân loại mô-đun lỗi và không lỗi. Tuy nhiên, các độ đo trên chỉ mô tả mã nguồn ở mức định lượng, chưa phản ánh đầy đủ ngữ cảnh cú pháp và mối quan hệ ngữ nghĩa giữa các thực thể trong chương trình. Do đó, các phương pháp mới hiện nay chuyển sang khai thác đặc trưng cú pháp và ngữ nghĩa của mã nguồn để tăng khả năng hiệu suất của các mô hình dự đoán. Các mục tiếp theo sẽ trình bày chi tiết hai nhóm phương pháp dự đoán lỗi phần mềm, bao gồm phương pháp dự đoán lỗi phần mềm dựa trên đặc trưng độ đo mã nguồn và phương pháp dựa trên dự đoán lỗi phần mềm đặc trưng cú pháp và ngữ nghĩa, nhằm làm rõ cơ sở lý thuyết và cách tiếp cận của từng hướng nghiên cứu.

1.2.1 Dự đoán lỗi phần mềm dựa trên đặc trưng độ đo mã nguồn

Để đảm bảo quy trình đảm bảo chất lượng phần mềm đạt hiệu quả và hiệu suất cao, các nhà phát triển thường cần ước lượng chất lượng của các thành phần phần mềm đang được xây dựng. Vì mục đích này, các độ đo mã nguồn đã được giới thiệu. Thông qua các độ

đo, dự án phần mềm có thể được phân tích một cách định lượng và chất lượng của nó có thể được đánh giá một cách hệ thống. Thông thường, mỗi độ đo mã nguồn phản ánh một số đặc tính chức năng nhất định của dự án, chẳng hạn như độ gắng kết, kế thừa, mức độ thay đổi mã,... Các độ đo này được sử dụng để suy luận về các thuộc tính chất lượng bên ngoài như độ tin cậy, khả năng kiểm thử hoặc mức độ dễ phát sinh lỗi. Các mô hình dự đoán lỗi thường được xây dựng dựa trên các độ đo phần mềm được trích xuất trực tiếp từ mã nguồn. Các kỹ thuật học máy đã được áp dụng trong nhiều năm để phát triển các mô hình dự đoán lỗi phần mềm. Những kỹ thuật này sử dụng dữ liệu lỗi phần mềm trong quá khứ (thu thập từ các dự án phần mềm trước đó) để huấn luyện mô hình dự đoán và xác định các mô-đun có lỗi [9]. Trong quá trình huấn luyện, mô hình dự đoán học được đặc điểm của dự án phần mềm và sử dụng kiến thức này để xác định liệu một mô-đun chưa thấy trước đó có lỗi hay không [19]. Tuy nhiên, trước khi xây dựng mô hình dự đoán, việc tiền xử lý dữ liệu là bước bắt buộc, bởi hầu hết các bộ dữ liệu này đều gặp phải hai vấn đề chính: số chiều lớn, và mất cân bằng lớp. Việc lựa chọn đúng các tập con độ đo phần mềm có ảnh hưởng trực tiếp đến hiệu năng tổng thể của mô hình dự đoán [44]. Hiệu quả của các mô hình dự đoán lỗi phụ thuộc đáng kể vào các độ đo này; vì vậy, việc lựa chọn tập đặc trưng phù hợp là điều thiết yếu nhằm tối ưu hóa hiệu năng của các mô hình được triển khai trong việc phát hiện các lớp lỗi. Ngoài ra, rong các bộ độ đo của dự án phần mềm, số lượng trường hợp không lỗi luôn vượt trội so với số trường hợp có lỗi, dẫn đến thách thức mất cân bằng lớp trong các mô hình dự đoán lỗi. Khi được huấn luyện trên các bộ dữ liệu mất cân bằng này, mô hình dự đoán lỗi có xu hướng thiên vị các ví dụ không lỗi và dễ phân loại sai các trường hợp lỗi. Khi các bộ dữ liệu huấn luyện được cân bằng lại, các mô hình dự đoán có thể đạt được hiệu năng dự đoán tốt hơn; vì vậy, các kỹ thuật lấy mẫu dữ liệu là phương pháp quan trọng xử lý vấn đề mất cân bằng lớp trong các bộ dữ liệu dự đoán lỗi phần mềm nhằm nâng cao hiệu suất của mô hình dự đoán lỗi. Các tiêu mục tiếp theo sẽ trình bày chi tiết về đặc trưng độ đo mã nguồn, các phương pháp lựa chọn đặc trưng và trích xuất đặc trưng, cũng như các kỹ thuật lấy mẫu dữ liệu được sử dụng trong dự đoán lỗi phần mềm.

1. Đặc trưng độ đo mã nguồn

Một yếu tố then chốt của bất kỳ quy trình kỹ thuật nào chính là đo lường. Các phép đo được sử dụng nhằm hiểu rõ hơn các thuộc tính của mô hình mà chúng ta xây dựng. Quan trọng hơn, chúng được dùng để đánh giá chất lượng của sản phẩm kỹ thuật hoặc quy trình tạo ra sản phẩm đó. Khác với các ngành kỹ thuật khác, công nghệ phần mềm không dựa trên các định luật định lượng cơ bản của vật lý. Những phép đo tuyệt đối như điện áp, khối lượng, vận tốc hay nhiệt độ hiếm khi xuất hiện trong lĩnh vực phần mềm. Thay vào đó, chúng ta tìm cách xây dựng một tập hợp các phép đo gián tiếp để hình thành nên các độ đo (metrics) nhằm phản ánh chất lượng

của một dạng biểu diễn phần mềm nào đó. Nhận thức được tầm quan trọng của các độ đo phần mềm, đã có nhiều độ đo được đề xuất để đo lường. Những độ đo này cố gắng nắm bắt các khía cạnh khác nhau của sản phẩm phần mềm cũng như quy trình phát triển phần mềm.

Một số độ đo được sử dụng để đo lường cùng một khía cạnh của phần mềm; ví dụ, có nhiều độ đo được đề xuất để đo mức độ liên kết (coupling) giữa các lớp khác nhau. Do đó, các nhà phát triển phần mềm cần chỉ rõ mối quan hệ giữa các độ đo khác nhau cùng đo lường một khía cạnh phần mềm. Chẳng hạn, nếu chúng ta muốn biết kích thước của một chiếc bàn, có thể có nhiều độ đo liên quan như chiều dài, chiều rộng, diện tích, hay đường chéo của bàn. Tuy nhiên, các phép đo chiều dài và chiều rộng đã đủ, bởi các độ đo khác có thể được suy ra từ chúng khi cần thiết. Tương tự trong lĩnh vực phần mềm, cần xác định những độ đo thực sự cần thiết và cung cấp thông tin hữu ích; nếu không, nhà quản lý sẽ bị rối bởi quá nhiều con số và mục tiêu sử dụng độ đo sẽ bị sai lệch. Khi đối chiếu số lượng độ đo hiện có với số lượng đặc trưng cốt lõi mà chúng thực sự phản ánh, có thể nhận thấy một mức độ dư thừa đáng kể. Bảng 1.1 và Bảng 1.2 trình bày một số độ đo McCabe, Halstead [11, 12] và hướng đối tượng được sử dụng trong dự đoán lỗi phần mềm.

Bảng 1.1: Độ đo McCabe, Halstead trong dự đoán lỗi phần mềm

TT	Độ đo	Ký hiệu	Nguồn	Mô tả
1	Cyclomatic Complexity	v(G)	McCabe	Các đường đi độc lập về mặt cấu trúc điều khiển
2	Essential Complexity	ev(G)	McCabe	Biểu thị mức độ mà một đồ thị luồng điều khiển (flowgraph) có thể được rút gọn bằng cách phân tách tất cả các đồ thị con (subflowgraphs)
3	Design Complexity	iv(G)	McCabe	Độ phức tạp chu trình (cyclomatic complexity) của đồ thị luồng điều khiển đã được rút gọn (reduced flowgraph) của một mô-đun.
4	Lines of code	LOC	McCabe	Số dòng mã được xác định dựa trên các quy ước đếm dòng do McCabe đề xuất.
5	n	n	Halstead	Tổng số toán tử và toán hạng
6	Volume	v	Halstead	Đo kích thước thông tin của chương trình
7	Program length	l	Halstead	Đo độ dài logic của chương trình
8	Difficulty	d	Halstead	Phản ánh mức độ khó của chương trình
9	Intelligence	i	Halstead	Ước lượng thông tin thực sự hữu ích so với độ khó
10	Effort	e	Halstead	Tổng nỗ lực cần thiết để cài đặt một chương trình.
11	Halstead	b	Halstead	Độ đo Halstead
12	Time Estimator	t	Halstead	Ước lượng thời gian
13	IOComment	IOComment	Halstead	Số dòng chú thích
14	IOCode	IOCode	Halstead	Số dòng được đếm
15	IOBlank	IOBlank	Halstead	Số dòng trống được đếm
16	uniq_Op	n1	Lines of code measure	Số lượng toán tử duy nhất
17	uniq_Opnd	n2	Lines of code measure	Số lượng toán hạng duy nhất
18	total_Op	N1	Lines of code measure	Tổng số toán tử
19	total_Opnd	N2	Lines of code measure	Tổng số toán hạng
20	Flow graph	branchCount	Branch-count	Biểu thị số lượng nhánh trong đồ thị dòng điều khiển
21	IOCodeAndComment	IOCodeAndComment	Lines of code measure	Biểu thị tổng số dòng mã và dòng chú thích
22	Defects	Defects	Goal field	{false, true}: biểu thị việc mô-đun có hoặc không có ít nhất một lỗi đã được báo cáo.

Bảng 1.2: Độ đo hướng đối tượng trong dự đoán lỗi phần mềm

TT	Độ đo	Hướng đối tượng	Source
1	Response for a Class (RFC)	Class	[Chidamber94] [13]
2	Number of Attributes per Class (NOA)	Class	[Henderson96] [45]
3	Number of Methods per Class (NOM)	Class	[Henderson96] [45]
4	Weighted Methods per Class (WMC)	Class	[Chidamber94] [13]
5	Coupling between Objects (CBO)	Coupling	[Chidamber94] [13]
6	Data Abstraction Coupling (DAC)	Coupling	[Henderson96] [45]
7	Message Passing Coupling (MPC)	Coupling	[Henderson96] [45]
8	Coupling Factor (CF)	Coupling	[Harrison98] [14]
9	Lack of Cohesion (LCOM)	Cohesion	[Chidamber94] [13]
10	Tight Class Cohesion (TCC)	Cohesion	[Braind99] [46]
11	Loose Class Cohesion (LCC)	Cohesion	[Braind99] [46]
12	Information-based Cohesion (ICH)	Cohesion	[Lee95] [47]
13	Method Hiding Factor (MHF)	Information Hiding	[Harrison98] [14]
14	Attribute Hiding Factor (AHF)	Information Hiding	[Harrison98] [14]
15	Number of Children (NOC)	Inheritance	[Chidamber94] [13]
16	Depth of Inheritance (DIT)	Inheritance	[Chidamber94] [13]
17	Method Inheritance Factor (MIF)	Inheritance	[Harrison98] [14]
18	Attribute Inheritance Factor (AIF)	Inheritance	[Harrison98] [14]
19	Number of Methods Overridden by a Subclass (NMO)	Polymorphism	[Henderson96] [45]
20	Polymorphism Factor (PF)	Polymorphism	[Harrison98] [14]
21	Reuse ratio	Reuse	[Henderson96] [45]
22	Specialization ratio	Reuse	[Henderson96] [45]

- Cyclomatic Complexity $v(G)$ đo lường số lượng các đường đi độc lập tuyến tính trong đồ thị luồng điều khiển của chương trình.

$$v(G) = e - n + 2 \quad (1.2)$$

Trong đó e là số lượng cạnh và n là số lượng nút trong đồ thị luồng điều khiển. $v(G)$ phản ánh mức độ phức tạp của luồng điều khiển trong chương trình; giá trị càng lớn cho thấy cấu trúc điều khiển càng phức tạp. Theo McCabe, các mô-đun có $v(G) > 10$ được xem là có khả năng cao phát sinh lỗi.

- Essential Complexity $ev(G)$ phản ánh mức độ mà một đồ thị luồng điều khiển có thể được giảm thiểu thông qua việc phân rã toàn bộ các đồ thị con của G .

$$ev(G) = v(G) - m \quad (1.3)$$

Với m là số lượng các đồ thị con.

-
- Design Complexity, ký hiệu là $iv(G)$, được định nghĩa là độ phức tạp chu trình của đồ thị luồng điều khiển đã được rút gọn của một mô-đun. Đồ thị luồng điều khiển G của mô-đun được rút gọn nhằm loại bỏ các thành phần phức tạp không ảnh hưởng đến mối quan hệ tương tác giữa các mô-đun trong thiết kế.
 - Volume v đo kích thước thông tin của chương trình.

$$v = N \times \log_2(n) \quad (1.4)$$

Trong đó, $N = N_1 + N_2$ là tổng số toán tử và toán hạng, $n = n_1 + n_2$ là tổng số toán tử và toán hạng duy nhất.

- Program length l đo độ dài logic của chương trình, với

$$l = V^*/N \quad (1.5)$$

Trong đó, $N = N_1 + N_2$ là tổng số toán tử và toán hạng của chương trình và V^* là kích thước của chương trình trong trường hợp cài đặt tối thiểu.

$$V^* = (2 + \mu_{2'}) * \log_2(2 + \mu_{2'}) \quad (1.6)$$

Với $\mu_{2'}$ biểu thị số lượng toán hạng duy nhất trong cài đặt tối thiểu.

- Difficulty d phản ánh mức độ khó của chương trình.

$$d = 1/l \quad (1.7)$$

- Intelligence i ước lượng thông tin thực sự hữu ích so với độ khó.

$$i = L' * V' \quad (1.8)$$

Với $L' = 1/d$

- Effort e là tổng nỗ lực cần thiết để cài đặt một chương trình.

$$e = v/l \quad (1.9)$$

- Time Estimator t ước lượng thời gian để viết chương trình.

$$t = e/18 \quad (1.10)$$

-
- Response for a Class (RFC) được định nghĩa là tập hợp các phương thức có thể được thực thi để đáp lại một thông điệp được gửi tới một đối tượng của lớp đó. Độ đo RFC được xác định bởi:

$$RFC = |RS| \quad (1.11)$$

trong đó RS là tập phản hồi của lớp, được xác định như sau:

$$RS = M_i \cup_{all\ j} R_{ij} \quad (1.12)$$

Trong đó, M_i là tập hợp tất cả các phương thức được định nghĩa trong lớp (tổng số là n), và $R_i = \{R_{ij}\}$ là tập hợp các phương thức được gọi bởi các phương thức thuộc M_i .

- Number of Attributes per Class (NOA) đo lường tổng số các thuộc tính được định nghĩa trong một lớp.
- Number Of Methods per Class (NOM) đo lường tổng số các phương thức được định nghĩa trong một lớp.
- Weighted Methods per Class (WMC) biểu thị tổng độ phức tạp của tất cả các phương thức trong một lớp. Để tính toán độ phức tạp của một lớp, độ đo phức tạp cụ thể được lựa chọn (ví dụ: độ phức tạp chu trình – cyclomatic complexity) cần được chuẩn hóa, sao cho độ phức tạp danh nghĩa của mỗi phương thức có giá trị bằng 1.0.

Xét một lớp K_i gồm các phương thức $M_1, \dots, M_n$ được định nghĩa trong lớp đó. Gọi $C_1, \dots, C_n$ lần lượt là độ phức tạp của các phương thức tương ứng. Khi đó, độ đo WMC được xác định như sau:

$$WMC = \sum_{i=1}^n C_i \quad (1.13)$$

- Coupling between Objects (CBO) biểu thị số lượng các lớp khác mà lớp đó có mối quan hệ liên kết. Hai lớp được xem là có liên kết khi các phương thức được khai báo trong một lớp sử dụng các phương thức được định nghĩa trong lớp còn lại.
- Message Passing Coupling (MPC) số lượng các câu lệnh truyền thông điệp (message-passing statements) được định nghĩa trong một lớp. Theo đó, nếu

hai phương thức khác nhau trong lớp A cùng truy cập (gọi) một phương thức thuộc lớp B, thì giá trị MPC sẽ bằng 2.

- Coupling Factor (CF) đo lường liên kết (coupling) giữa các lớp có thể phát sinh do truyền thông điệp hoặc do các mối liên kết ngữ nghĩa tĩnh giữa các thể hiện của lớp. Trong thiết kế hướng đối tượng, người ta cho rằng các lớp nên giao tiếp với càng ít lớp khác càng tốt, và ngay cả khi có giao tiếp, lượng thông tin trao đổi cũng nên được giảm thiểu ở mức thấp nhất. Độ đo Coupling Factor được định nghĩa như sau:

$$CF = \frac{\sum_{i=1}^{TC} \sum_{j=1}^{TC} [is_client(C_i, C_j)]}{TC^2 - TC} \quad (1.14)$$

trong đó TC là tổng số lớp trong hệ thống. Hàm $is_client(C_c, C_s)$ được xác định như sau:

$$is_client(C_c, C_s) = \begin{cases} 1, & \text{nếu } C_c \Rightarrow C_s \text{ và } C_c \neq C_s \\ 0, & \text{trong các trường hợp còn lại} \end{cases} \quad (1.15)$$

Các mối liên kết phát sinh từ quan hệ kế thừa không được tính trong chỉ số CF, do trong quan hệ kế thừa, một lớp vốn đã có mức độ phụ thuộc cao vào các lớp cha của nó. Nếu không có lớp nào liên kết với lớp khác thì $CF = 0\%$. Ngược lại, nếu tất cả các lớp đều liên kết với mọi lớp còn lại thì $CF = 100\%$.

- Lack of Cohesion (LCOM) đo lường mức độ thiếu gắn kết giữa các phương thức trong một lớp thông qua việc xem xét các biến thể hiện hay các thuộc tính được sử dụng bởi các phương thức đó. Xét một lớp C_1 gồm n phương thức $M_1, M_2, \dots, M_n$. Gọi I_j là tập hợp tất cả các biến thể hiện được sử dụng bởi phương thức M_j . Khi đó, tồn tại n tập hợp tương ứng $\{I_1, I_2, \dots, I_n\}$.

Định nghĩa hai tập P và Q như sau:

$$P = \{(I_i, I_j) \mid I_i \cap I_j = \emptyset\}, \quad (1.16)$$

$$Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \emptyset\}. \quad (1.17)$$

Nếu tất cả n tập $\{I_1, I_2, \dots, I_n\}$ đều là $\emptyset$ thì $P = 0$.

$$LCOM = \begin{cases} |P| - |Q|, & \text{nếu } |P| > |Q| \\ 0, & \text{ngược lại} \end{cases} \quad (1.18)$$

Một giá trị LCOM dương và lớn cho thấy mức độ gắn kết của các lớp là thấp. Vì vậy, một giá trị LCOM nhỏ là điều mong muốn trong thiết kế hướng đối tượng.

- Tight Class Cohesion (TCC) là tỷ lệ phần trăm các cặp phương thức công khai của một lớp có sử dụng chung ít nhất một thuộc tính.
- Loose Class Cohesion (LCC) đo lường mức độ liên kết giữa các phương thức công khai trong một lớp, bao gồm cả các mối liên kết trực tiếp và gián tiếp thông qua lời gọi phương thức. LCC được tính bằng tỷ lệ phần trăm các cặp phương thức có liên kết với nhau. So với TCC, LCC mở rộng phạm vi đánh giá bằng cách xét thêm các quan hệ gián tiếp.
- Information-based Cohesion (ICH) của một lớp được định nghĩa là số lần một phương thức trong lớp gọi các phương thức khác của chính lớp đó, trong đó mỗi lời gọi được tính trọng số dựa trên số lượng tham số của phương thức được gọi.
- Method Hiding Factor (MHF) là chỉ số dùng để đánh giá mức độ đóng gói (encapsulation) của các phương thức trong một hệ thống hướng đối tượng. Chỉ số này phản ánh tỷ lệ các phương thức không được truy cập từ các lớp khác. Cụ thể, với mỗi phương thức được khai báo trong một lớp, nếu phương thức đó không thể được gọi từ bất kỳ lớp nào khác, nó được tính giá trị là 1; ngược lại, nếu phương thức có thể được sử dụng bởi ít nhất một lớp khác thì được tính là 0. Giá trị MHF của toàn hệ thống được tính bằng cách lấy tổng các giá trị này chia cho tổng số phương thức được định nghĩa trong tất cả các lớp.
- Attribute Hiding Factor (AHF) là chỉ số dùng để đo lường mức độ đóng gói (encapsulation) của các thuộc tính trong một hệ thống hướng đối tượng. Chỉ số này phản ánh tỷ lệ các thuộc tính không thể được truy cập từ các lớp khác. AHF được định nghĩa như sau:

$$AHF = \frac{\sum_{i=1}^{TC} \sum_{a=1}^{A_d(C_i)} (1 - V(A_{ai}))}{\sum_{i=1}^{TC} A_d(C_i)} \quad (1.19)$$

Trong đó: TC là tổng số lớp trong hệ thống; $A_d(C_i)$ là số thuộc tính được khai báo trong lớp C_i ; $V(A_{ai})$ là mức độ hiển thị của thuộc tính A_{ai} .

Giá trị $V(A_{ai})$ được xác định như sau:

$$V(A_{ai}) = \sum_{j=1}^{TC} is_visible(A_{ai}, C_j) \quad (1.20)$$

Hàm $is_visible(A_{ai}, C_j)$ được định nghĩa bởi:

$$is_visible(A_{ai}, C_j) = \begin{cases} 1, & \text{nếu } j \neq i \text{ và lớp } C_j \text{ có thể truy cập thuộc tính } A_{ai} \\ 0, & \text{ngược lại} \end{cases} \quad (1.21)$$

Như vậy, một thuộc tính được tính giá trị là 0 nếu nó có thể được sử dụng bởi ít nhất một lớp khác, và được tính giá trị là 1 nếu nó không thể được truy cập từ bất kỳ lớp nào khác. Giá trị AHF của toàn hệ thống được xác định bằng tỷ lệ giữa tổng số thuộc tính được che giấu và tổng số thuộc tính được khai báo trong hệ thống.

- Number of Children (NOC) là chỉ số đo lường mức độ kế thừa của một lớp, được xác định bằng số lượng các lớp con trực tiếp kế thừa từ lớp đó trong cây phân cấp kế thừa. Chỉ số này phản ánh vai trò của lớp trong cấu trúc hệ thống và mức độ tái sử dụng thông qua cơ chế kế thừa.
- Depth of Inheritance (DIT) là chỉ số biểu thị độ sâu của một lớp trong cây phân cấp kế thừa, được xác định bằng số lượng các lớp tổ tiên (ancestor) từ lớp đó đến lớp gốc của cây. Trong trường hợp tồn tại đa kế thừa, giá trị DIT được lấy bằng độ dài lớn nhất của đường đi từ lớp đang xét đến lớp gốc.
- Method Inheritance Factor (MIF) là chỉ số ở mức toàn hệ thống, dùng để đo lường mức độ sử dụng cơ chế kế thừa phương thức trong thiết kế hướng đối tượng. Chỉ số này phản ánh tỷ lệ các phương thức được kế thừa so với tổng số phương thức hiện có trong toàn bộ hệ thống. Về mặt hình thức, MIF được định nghĩa như sau:

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} \quad (1.22)$$

trong đó: TC là tổng số lớp trong hệ thống; $M_d(C_i)$ là số phương thức được khai báo trong lớp C_i ; $M_i(C_i)$ là số phương thức được kế thừa trong lớp C_i ;

$M_a(C_i)$ là tổng số phương thức sẵn có của lớp C_i , được xác định bởi:

$$M_a(C_i) = M_d(C_i) + M_i(C_i) \quad (1.23)$$

Chỉ số MIF cho biết mức độ mà các lớp trong hệ thống tận dụng các phương thức được kế thừa từ các lớp cha. Giá trị MIF càng lớn thì mức độ sử dụng kế thừa trong hệ thống càng cao, phản ánh khả năng tái sử dụng và cấu trúc kế thừa của thiết kế phần mềm.

- Number of Methods Overridden by a subclass (NMO) đo lường số phương thức của lớp cha bị ghi đè bởi phương thức của lớp con.
- Polymorphism Factor (PF) là chỉ số dùng để đo lường mức độ đa hình (polymorphism) trong thiết kế hướng đối tượng. Chỉ số này phản ánh mức độ sử dụng cơ chế ghi đè phương thức (method overriding) trong cây phân cấp kế thừa của hệ thống. PF được định nghĩa như sau:

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]} \quad (1.24)$$

Trong đó: TC là tổng số lớp trong hệ thống; $M_n(C_i)$ là số phương thức mới được khai báo trong lớp C_i ; $M_o(C_i)$ là số phương thức ghi đè trong lớp C_i ; $DC(C_i)$ là số lượng lớp con (descendants) của lớp C_i trong cây kế thừa.

Chỉ số PF biểu thị tỷ lệ giữa số phương thức thực sự được ghi đè và số phương thức *có khả năng* bị ghi đè trong toàn bộ hệ thống. Giá trị PF càng lớn cho thấy mức độ sử dụng đa hình càng cao, phản ánh tính linh hoạt và khả năng mở rộng của thiết kế hướng đối tượng.

- Reuse Ratio (U) đánh giá mức độ tái sử dụng có thể đạt được giữa các lớp trong hệ thống. Reuse Ratio được định nghĩa như sau:

$$U = \frac{\text{Number of Superclasses}}{\text{Total Number of Classes}} \quad (1.25)$$

Trong đó: Total Number of Classes là tổng số lớp trong hệ thống; Number of Superclasses là số lớp đóng vai trò là lớp cha (superclass) trong quan hệ kế thừa. Chỉ số U phản ánh mức độ tái sử dụng thông qua cơ chế kế thừa trong thiết kế hướng đối tượng. Giá trị U càng lớn cho thấy mức độ tái sử dụng giữa các lớp càng cao.

- Specialization Ratio (S) là chỉ số dùng để đo lường mức độ *chuyên biệt hóa* (specialization) trong thiết kế hướng đối tượng. Chỉ số này phản ánh mối quan hệ giữa các lớp cha và các lớp con trong cây phân cấp kế thừa của hệ thống. Specialization Ratio được định nghĩa như sau:

$$S = \frac{\text{Number of Superclasses}}{\text{Number of Subclasses}} \quad (1.26)$$

Trong đó, Number of Superclasses là số lớp đóng vai trò là lớp cha trong hệ thống; Number of Subclasses là số lớp con tham gia vào quan hệ kế thừa. Chỉ số S cho biết mức độ mà các lớp trong hệ thống được chuyên biệt hóa thông qua cơ chế kế thừa. Giá trị S càng nhỏ cho thấy hệ thống có nhiều lớp con hơn, phản ánh mức độ chuyên biệt hóa cao; ngược lại, giá trị S lớn cho thấy thiết kế ít phân nhánh và mức độ chuyên biệt hóa thấp hơn.

2. Lựa chọn đặc trưng

Các nhà nghiên cứu [48, 49] đã xác định được các độ đo mã nguồn có mối tương quan mạnh với lỗi trong các tập dữ liệu lỗi phần mềm, dựa trên các yếu tố như độ phức tạp của mã nguồn và phân tích quy trình phát triển. Các độ đo này cũng được gọi là thuộc tính hoặc đặc trưng. Tuy nhiên, không phải tất cả các thuộc tính đều phù hợp để xây dựng mô hình dự đoán lỗi. Thời gian cần thiết để huấn luyện mô hình và chất lượng dự đoán của nó có thể bị ảnh hưởng tiêu cực bởi các thuộc tính dư thừa và không liên quan, thường được gọi là chiều dữ liệu cao. Các nghiên cứu trước đây [50, 51] đã chứng minh rằng các kỹ thuật lựa chọn đặc trưng và trích xuất đặc trưng giúp giảm thiểu vấn đề chiều dữ liệu lớn trong các tập dữ liệu lỗi phần mềm. Các phương pháp lựa chọn đặc trưng ngày càng được chú trọng hơn nhờ khả năng cải thiện độ chính xác của dự đoán và rút ngắn thời gian xây dựng mô hình [49]. Có ba loại phương pháp lựa chọn đặc trưng chính: filter [52], wrapper [53] và embedded [54].

- Phương pháp filter là một trong những phương pháp lựa chọn đặc trưng ra đời sớm nhất. Nó đánh giá các đặc trưng dựa trên các thuộc tính nội tại trước khi thực hiện các tác vụ học máy. Phương pháp này dựa trên bốn tiêu chí đo lường chính: thông tin (information), phụ thuộc (dependency), tính nhất quán (consistency), và khoảng cách (distance). Nó thực hiện lựa chọn đặc trưng một cách độc lập với thuật toán học máy và sử dụng các tiêu chuẩn thống kê để xếp hạng các tập hợp đặc trưng [55]. Một số phương pháp tiêu biểu bao gồm Information Gain, Chi-square test, và Correlation Coefficient, v.v.

- Phương pháp wrapper bao gồm quá trình lựa chọn đặc trưng sử dụng các thuật

toán và sử dụng độ chính xác hoặc tỷ lệ lỗi của quá trình phân loại để đánh giá các đặc trưng. Mục tiêu của phương pháp này là chọn ra tập hợp đặc trưng có tính phân biệt cao nhất bằng cách giảm thiểu lỗi ước lượng của một mô hình dự đoán cụ thể. Phương pháp wrapper thực hiện lựa chọn đặc trưng dựa trên hiệu suất của thuật toán học máy, từ đó lựa chọn các đặc trưng tối ưu nhất cho quá trình dự đoán. Do tính chất đa biến, hầu hết các phương pháp wrapper đòi hỏi thời gian tính toán lớn để hội tụ, khiến chúng trở nên không thực tế đối với các tập dữ liệu lớn. Một số phương pháp tiêu biểu bao gồm Forward Selection, Backward Elimination, và Recursive Elimination [56].

- Phương pháp embedded là một cơ chế lựa chọn đặc trưng tích hợp sẵn, trong đó quá trình lựa chọn đặc trưng được tích hợp trực tiếp vào thuật toán, tận dụng các đặc tính của thuật toán để hướng dẫn việc đánh giá đặc trưng. So với phương pháp wrapper, phương pháp embedded có hiệu suất tính toán cao hơn trong khi vẫn duy trì mức hiệu quả tương đương [57]. Bằng cách kết hợp các ưu điểm của các phương pháp filter và embedded, phương pháp embedded thực hiện lựa chọn đặc trưng ngay trong quá trình triển khai thuật toán, giúp giảm độ phức tạp tính toán [23]. Điều này đạt được nhờ việc tránh lặp lại quá trình thực thi mô hình dự đoán và kiểm tra từng tập hợp đặc trưng. Một số phương pháp tiêu biểu bao gồm LASSO và các phương pháp mô hình cây quyết định có áp dụng kỹ thuật chuẩn hóa.

- Ngoài ra, thuật toán metaheuristic đã được ứng dụng rộng rãi trong nhiều bài toán tối ưu trong những năm gần đây và lựa chọn đặc trưng là một trong những lĩnh vực quan trọng mà các phương pháp này được nghiên cứu. Tuy nhiên, do tính chất ngẫu nhiên của các thuật toán metaheuristic, không có đảm bảo rằng quá trình lựa chọn đặc trưng sẽ tìm được tổ hợp thuộc tính tối ưu nhất. Nhiều thuật toán metaheuristic đã được áp dụng trong dự đoán lỗi phần mềm bao gồm Genetic Algorithms (GA), Particle Swarm Optimization (PSO) [58], Artificial Bee Colony (ABC) [22] v.v.

3. Trích xuất đặc trưng

Trích xuất đặc trưng [59] là một kỹ thuật nhằm trích xuất các đặc trưng không dư thừa và có liên quan từ tập hợp đặc trưng đầu vào bằng cách sử dụng một số phép biến đổi để giảm độ phức tạp và cung cấp một cách biểu diễn dễ hiểu cho từng biến trong không gian đặc trưng dưới dạng một tổ hợp tuyến tính của các biến đầu vào. Đây là một kỹ thuật tổng quát hơn so với kỹ thuật lựa chọn đặc trưng. Một số phương pháp trích xuất đặc trưng bao gồm Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) và Kernel-based Principal Component Analysis (KPCA).

4. Lấy mẫu dữ liệu

Trong quá trình xây dựng mô hình dự đoán lỗi phần mềm, đã tồn tại một thách thức là hầu hết các tập dữ liệu lỗi lịch sử chứa nhiều mô-đun không lỗi hơn (tức là lớp chiếm đa số) so với các mô-đun có lỗi (lớp chiếm thiểu số). Đây được gọi là vấn đề mất cân bằng lớp trong dự đoán lỗi phần mềm. Các kỹ thuật lấy mẫu dữ liệu như lấy mẫu tăng và lấy mẫu giảm được sử dụng rộng rãi để giải quyết vấn đề mất cân bằng lớp trong dự đoán lỗi phần mềm [60]. Kỹ thuật lấy mẫu tăng tạo ra các mô-đun tổng hợp mới thuộc lớp thiểu số nhằm tăng số lượng mẫu của lớp này. Ngược lại, kỹ thuật lấy mẫu giảm loại bỏ một số mô-đun thuộc lớp đa số để cân bằng tỷ lệ giữa hai lớp [29]. Cả hai nhóm kỹ thuật này đều nhằm mục đích cân bằng sự phân bố giữa các mô-đun thuộc lớp thiểu số và lớp đa số nhằm cải thiện hiệu suất của mô hình dự đoán. Trong dự đoán lỗi phần mềm, kỹ thuật lấy mẫu tăng thường được ưu tiên hơn so với lấy mẫu giảm, vì phương pháp lấy mẫu giảm loại bỏ ngẫu nhiên các mô-đun khỏi tập dữ liệu, có thể dẫn đến mất mát thông tin quan trọng [48].

1.2.2 Dự đoán lỗi phần mềm dựa trên đặc trưng cú pháp và ngữ nghĩa mã nguồn

Học sâu dựa trên mạng nơ-ron nhân tạo (ANN - Artificial Neural Network), vốn thực hiện dự đoán cho dữ liệu đầu vào thông qua quá trình huấn luyện có giám sát hoặc không giám sát [61]. Gần đây, nhiều mô hình học sâu đã được ứng dụng rộng rãi trong lĩnh vực dự đoán lỗi phần mềm. Qiao và Wang [62] đã khai thác học sâu để tạo ra các đặc trưng tối ưu cho việc dự đoán lỗi tức thời (just-in-time - JIT) có xem xét đến nỗ lực. Kết quả thử nghiệm cho thấy mô hình được đề xuất đã đạt được cải thiện đáng kể về độ đo recall trung bình. Cụ thể, recall trung bình tăng 15,6% và popt tăng 8,1% so với các phương pháp hiện có. Ma và cộng sự [63] đã đề xuất một kỹ thuật gỡ lỗi mô hình (MODE) để tiến hành phân tích trạng thái mô hình nhằm xác định các nơ-ron bị lỗi và thực hiện lựa chọn dữ liệu huấn luyện đầu vào. Kết quả đánh giá của họ dựa trên 29 mô hình cho 6 ứng dụng cho thấy kỹ thuật được đề xuất có thể cải thiện độ chính xác kiểm thử so với các mô hình hiện có. Hellendoorn và cộng sự [64] đã đề xuất một mô hình học sâu có tên là DEEPTYPE để học các chú thích kiểu từ mã nguồn cơ sở và sau đó có thể được áp dụng để dự đoán chú thích kiểu của các biến và hàm.

Lin và cộng sự [34] đã sử dụng Bi-LSTM để học biểu diễn của cây AST của chương trình nhằm phát hiện các hàm dễ bị lỗi, và thử nghiệm đã cho thấy hiệu năng tốt hơn trên cả các dự đoán lỗi trong cùng dự án và dự đoán lỗi giữa các dự án. Zhou và cộng sự [36] đã xây dựng LSTM dựa trên LSTM hai chiều và cấu trúc cây để cải thiện khả năng học đặc trưng và hiệu năng dự đoán. Cheng và cộng sự [65] đã đề xuất một mạng tích chập đồ thị có tên là GCN2defect để nắm bắt các phụ thuộc giữa các nút trong mạng phụ

thuộc lớp (Class Dependency Network) của một chương trình và sử dụng kỹ thuật lấy mẫu SMOTETomek trong việc xử lý vấn đề mất cân bằng lớp trong lĩnh vực dự đoán lỗi phần mềm. Yan và cộng sự [66] đã khai thác mạng nơ-ron tích chập đồ thị sâu để nhúng thông tin cấu trúc vốn có trong đồ thị luồng điều khiển cho việc phân loại phần mềm độc hại. Kết quả thử nghiệm cho thấy hệ thống được đề xuất của họ có thể phân loại phần mềm độc hại của chương trình với hiệu năng vượt trội so với các phương pháp hiện có. Khleel và Nehéz [67] đã trình bày một sự kết hợp giữa BiLSTM và các kỹ thuật lấy mẫu tăng cường để giải quyết vấn đề dữ liệu mất cân bằng. Kết quả thử nghiệm đã chứng minh những cải thiện đáng kể về hiệu năng của mô hình được đề xuất khi áp dụng cho các bộ dữ liệu đã được cân bằng so với các bộ dữ liệu gốc. Cụ thể, độ chính xác trung bình của mô hình tăng 6% và độ đo F1-score cải thiện 43%. Pornprasit Cheng và cộng sự [68] đã trình bày mô hình Deeplinedp, với mục tiêu xác định các tệp tin bị lỗi và các dòng mã bị lỗi trong mã nguồn phần mềm bằng cách tự động học các thuộc tính ngữ nghĩa của các token và các dòng mã xung quanh. Lucija và cộng sự [69] đã đề xuất một mô hình dự đoán lỗi dựa trên Graph Convolutional Network (GCN) có khả năng nắm bắt toàn bộ thông tin của các cạnh và nút được trích xuất từ một cây cú pháp trừu tượng. Họ kết luận rằng mô hình được đề xuất của họ đạt hiệu năng tốt hơn so với các mô hình hiện có trong việc phân loại các thành phần bị lỗi bằng cách sử dụng cây cú pháp trừu tượng được tạo ra từ mã nguồn.

Trong phương pháp được mô tả bởi Wang và cộng sự [70], Gated Hierarchical Long Short-Term Memory Networks (GH-LSTMs) đã được sử dụng để nắm bắt cả thông tin ngữ nghĩa được trích xuất từ cây cú pháp trừu tượng và các độ đo phần mềm truyền thống được cung cấp bởi tập dữ liệu PROMISE. Batool và cộng sự [71] đã sử dụng ba mô hình học sâu, bao gồm LTSM, BiLSTM và Radial Basis Function Network (RBFN) để thực hiện dự đoán lỗi phần mềm bằng cách sử dụng các bộ dữ liệu dựa trên độ đo CK. Họ kết luận rằng LSTM và BiLSTM đạt được các giá trị độ chính xác cao nhất lần lượt là 93,53% và 93,75%, trong khi RBFN được coi là nhanh hơn trong việc tạo ra các kết quả mong muốn. Trong [72], Ahmed Abdu và cộng sự đã đề xuất một mô hình Deep Hierarchical Convolutional Neural Network (DH-CNN) để tạo ra thông tin ngữ nghĩa từ đồ thị luồng điều khiển và đồ thị phụ thuộc dữ liệu (DDG - Data Dependence Graph) và đồ thị cú pháp từ cây cú pháp trừu tượng. Kết quả thử nghiệm trên 9 dự án PROMISE cho thấy DH-CNN đạt hiệu năng tốt hơn so với các phương pháp hiện có về F1-score và AUC cho cả trong cùng dự án và giữa các dự án.

Zhou và cộng sự [73] đã giới thiệu một phương pháp tích hợp thể mạnh của CNN và CGCN để trích xuất cả thông tin ngữ nghĩa và cấu trúc của mã nguồn. Các thử nghiệm được thực hiện trên 21 dự án mã nguồn mở đã chứng minh rằng mô hình được đề xuất của họ vượt trội hơn các mô hình hiện có, mang lại kết quả ổn định hơn và thể hiện khả năng biểu diễn đặc trưng mạnh mẽ hơn. Rafi và cộng sự [74] đã đề xuất DepGraph, GNN

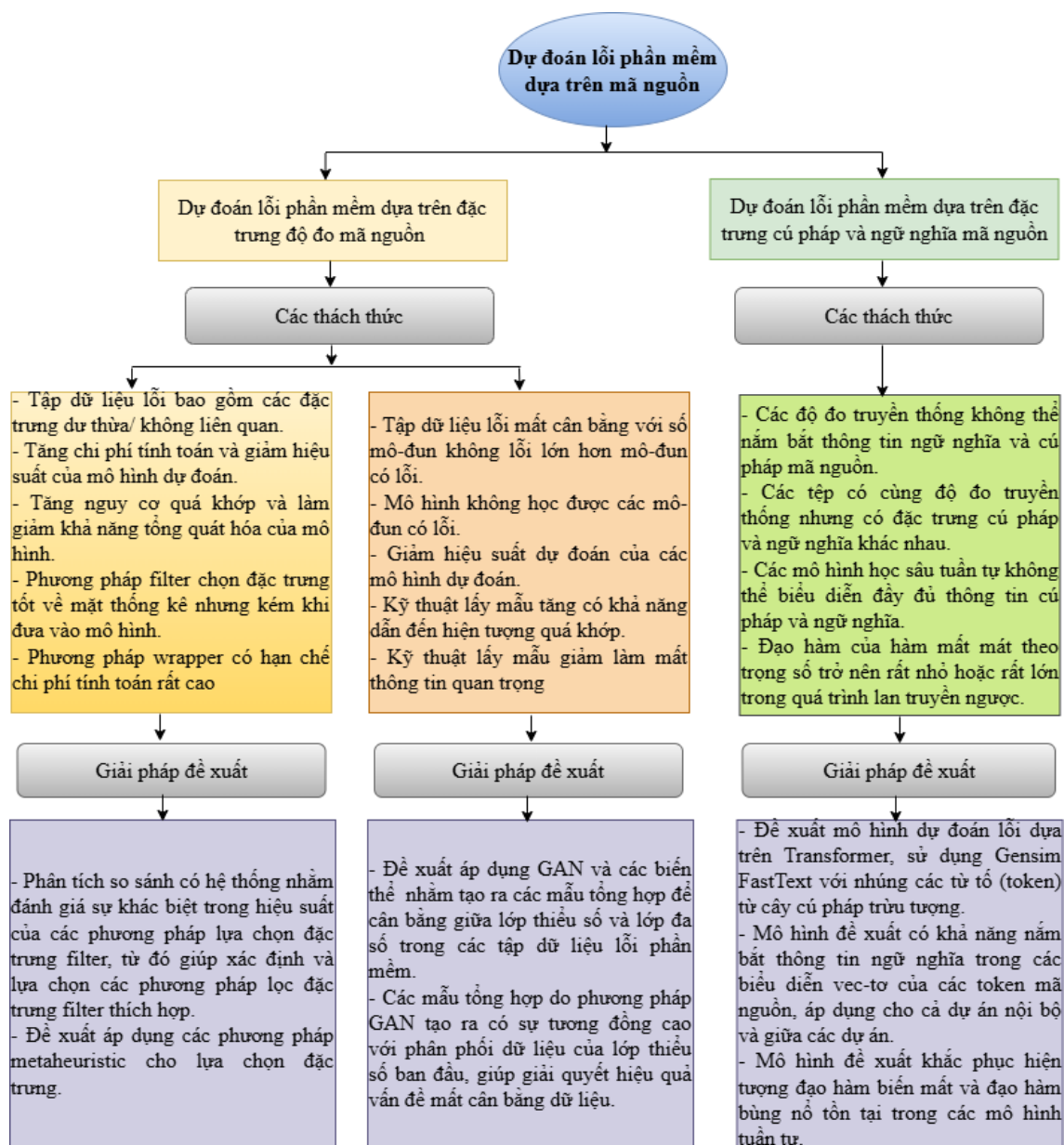
mới cho việc định vị lỗi. Họ đã kết hợp các đặc trưng bổ sung, chẳng hạn như thông tin thay đổi mã nguồn, vào đồ thị dưới dạng thuộc tính, cho phép mô hình sử dụng hiệu quả dữ liệu lịch sử phong phú của dự án. Đánh giá của họ trên bộ dữ liệu Defects4j đã chứng minh rằng DepGraph đạt được hiệu năng vượt trội so với các phương pháp hiện có.

1.3 Các thách thức

Có nhiều yếu tố ảnh hưởng đến hiệu suất của các mô hình dự đoán lỗi bao gồm chất lượng tập dữ liệu, các độ đo đặc trưng phần mềm, kỹ thuật học máy và học sâu được áp dụng để xây dựng mô hình dự đoán lỗi, các độ đo đánh giá hiệu suất và các vấn đề liên quan đến tập dữ liệu [20]. Trong đó, chất lượng tập dữ liệu là một trong những yếu tố quan trọng nhất trong xây dựng mô hình dự đoán lỗi phần mềm. Hình 1.2 minh họa toàn cảnh các thách thức chính trong dự đoán lỗi phần mềm dựa trên mã nguồn, đồng thời tóm lược các hướng giải pháp tương ứng cho từng nhóm thách thức.

Thứ nhất, thách thức đối với quy trình xây dựng mô hình dự đoán lỗi là các tập dữ liệu lỗi có quá nhiều đặc trưng, bao gồm cả những đặc trưng không liên quan đến lỗi phần mềm và các đặc trưng dư thừa. Các nghiên cứu hiện có đã chỉ ra rằng vấn đề này có thể dẫn đến chi phí tính toán lớn và làm suy giảm hiệu suất của một số mô hình nhất định [75]. Tốc độ và độ chính xác của các mô hình dự đoán đã được huấn luyện có thể bị ảnh hưởng bởi các đặc trưng không liên quan hoặc dư thừa. Ngoài ra, khảo sát tài liệu cho thấy rằng việc lựa chọn đặc trưng giúp cải thiện hiệu suất của mô hình phân loại [44]. Các kỹ thuật lựa chọn đặc trưng được áp dụng trong giai đoạn tiền xử lý dữ liệu nhằm tăng cường hiệu suất của các mô hình dự đoán lỗi. Trong nhiều năm qua, đã có những nỗ lực nghiên cứu không ngừng để nâng cao độ chính xác dự đoán lỗi của các mô hình. Các tập dữ liệu lớn hơn sẽ yêu cầu nhiều thời gian hơn và tăng số vòng tính toán để huấn luyện mô hình dự đoán lỗi, điều này cũng có thể ảnh hưởng đến độ chính xác của dự đoán. Vì lý do này, việc lựa chọn các đặc trưng phù hợp và giảm kích thước tập đặc trưng là hai giai đoạn không thể thiếu để đạt được kết quả tối ưu hơn. Về cơ bản, quá trình giảm số lượng đặc trưng bao gồm lựa chọn đặc trưng và trích xuất đặc trưng. Lựa chọn đặc trưng liên quan đến việc chọn ra các đặc trưng quan trọng nhất cần thiết cho quá trình phân tích, trong khi trích xuất đặc trưng bao gồm việc kết hợp các yếu tố tương đồng thành một đặc trưng mới có kích thước nhỏ hơn [20].

Thứ hai, thách thức lớn khác ảnh hưởng đến hiệu suất của mô hình dự đoán lỗi đó chính là vấn đề mất cân bằng dữ liệu. Hầu hết các tập dữ liệu lỗi lịch sử đều chứa nhiều mô-đun không có lỗi (tức là tập đa số) hơn so với mô-đun có lỗi (tập thiểu số). Đây được gọi là vấn đề mất cân bằng lớp trong dự đoán lỗi phần mềm. Nhiều kỹ thuật học máy giả



Hình 1.2: Các thách thức tồn tại trong dự đoán lỗi phần mềm

định rằng tỷ lệ giữa lớp thiếu số và lớp đa số là xấp xỉ bằng nhau [76]. Tuy nhiên, giả định này không đúng với các tập dữ liệu lỗi, vốn thường có sự mất cân bằng từ mức trung bình đến nghiêm trọng. Do đó, các mô hình dự đoán được xây dựng trên các tập dữ liệu phần mềm mất cân bằng có thể không học được các mẫu của mô-đun có lỗi và do đó có xu hướng dự đoán sai nhiều mô-đun lỗi [28]. Điều này có thể dẫn đến các mô hình dự đoán không đáng tin cậy và không thể áp dụng trong thực tế. Vì vậy, việc đề xuất các kỹ thuật/phương pháp để giải quyết vấn đề mất cân bằng lớp trong tập dữ liệu lỗi là một vấn đề quan trọng được quan tâm rộng rãi trong dự đoán lỗi phần mềm. Để giải quyết vấn đề mất cân bằng dữ liệu, có hai phương pháp chính trong lấy mẫu dữ liệu là lấy mẫu tăng và lấy mẫu giảm. Các kỹ thuật lấy mẫu tăng cố gắng tạo ra các mẫu mới cho lớp tối thiểu nhằm tăng số lượng mẫu và giúp cân bằng phân phối lớp. SMOTE và các biến thể của nó, như Borderline SMOTE và ADASYN là những kỹ thuật lấy mẫu tăng phổ biến nhất trong tài liệu nghiên cứu nhằm xử lý mất cân bằng dữ liệu trong học sâu. Tuy nhiên, các phương pháp này thường dẫn đến vấn đề quá khớp trong các mô hình dự đoán do các mẫu tổng hợp có tính tương đồng cao với các mẫu hiện có trong lớp tối thiểu. Ngược lại, các kỹ thuật lấy mẫu giảm giảm số lượng lớp tối thiểu bằng cách chọn ngẫu nhiên hoặc có chủ đích một tập con các mẫu không lỗi. Phương pháp lấy mẫu giảm phổ biến nhất là Random Under-Sampling, trong đó một số mẫu thuộc lớp tối thiểu bị loại bỏ một cách ngẫu nhiên. Tuy nhiên, do việc loại bỏ các mẫu lớp đa số, một phần thông tin quan trọng có thể bị mất, dẫn đến sự suy giảm hiệu suất của mô hình dự đoán [77]. Ngoài ra, các phương pháp thay thế như học kết hợp và các kỹ thuật học máy nhạy cảm với chi phí cũng đã được khai thác để xử lý vấn đề mất cân bằng dữ liệu.

Thứ ba, các phương pháp truyền thống với các độ đo đặc trưng được thiết kế thủ công đã đạt được những tiến bộ trong dự đoán lỗi cho cả trường hợp trong cùng dự án và chéo giữa các dự án. Tuy nhiên, các độ đo truyền thống này không thể nắm bắt được thông tin cú pháp và ngữ nghĩa của các tệp mã nguồn. Các tệp mã nguồn có cùng độ đo đặc trưng truyền thống có thể có đặc trưng ngữ nghĩa khác nhau, điều này có thể làm giảm hiệu suất dự đoán lỗi. Với sự phát triển nhanh chóng của các mô hình học sâu, các nghiên cứu đã tận dụng các mô hình học sâu để xây dựng các mô hình dự đoán lỗi có khả năng tự động học các đặc trưng ngữ nghĩa từ mã nguồn. Nhiều nghiên cứu [78] đã trích xuất thông tin cú pháp từ mã nguồn bằng cách sử dụng cây cú pháp trừu tượng. Sau đó, các cây cú pháp trừu tượng được chuyển đổi thành các véc-tơ số và đưa vào các mô hình học sâu để trích xuất đặc trưng cú pháp của mã nguồn và dự đoán liệu mã nguồn đó có lỗi hay không. Một số nghiên cứu khác [79] đã sử dụng các đồ thị để biểu diễn mã nguồn, chẳng hạn như đồ thị luồng điều khiển. Các mô hình học sâu là một hướng tiếp cận đầy hứa hẹn trong việc học các đặc trưng ngữ nghĩa ẩn nhằm nâng cao hiệu suất của các mô hình dự đoán lỗi. Tuy nhiên, các mạng tuần tự không thể biểu diễn đầy đủ thông tin cú pháp và ngữ nghĩa được

trích xuất từ cây cú pháp trừu tượng, đồng thời gặp khó khăn trong việc nắm bắt các phụ thuộc phức tạp trong mã nguồn [36].

1.4 Các mô hình học máy

Dữ liệu lịch sử về các phiên bản phần mềm có thể được sử dụng để huấn luyện mô hình học máy nhằm dự đoán khả năng xuất hiện lỗi trong các lớp đã được chỉnh sửa, dựa trên việc liệu các lớp đó từng có lỗi được ghi nhận ở các phiên bản trước hay không. Học máy, một trong những phương pháp cốt lõi của trí tuệ nhân tạo, đóng vai trò quan trọng trong nhiệm vụ này. Bằng cách khai thác tập dữ liệu huấn luyện, các mô hình học máy xây dựng mô hình dự đoán có khả năng suy luận và dự đoán khả năng chứa lỗi cho những mô-đun mã nguồn mới. Trong luận án này, luận án xem xét và lựa chọn các mô hình tiêu biểu và đạt hiệu suất cao để đánh giá hiệu suất của các mô hình dự đoán lỗi, cụ thể bao gồm: Random Forest, Extra Trees, AdaBoost, Gradient Boosting, HistGradient Boosting, eXtreme Gradient Boosting và Multilayer Perceptron.

1.4.1 Random Forest

Random Forests (RF) [80] là một mô hình học kết hợp dựa trên kỹ thuật bagging, được xây dựng từ nhiều cây quyết định nhỏ, trong đó mỗi cây được huấn luyện trên một tập con ngẫu nhiên của dữ liệu. Để tăng tính đa dạng giữa các cây, tại mỗi nút, mô hình chọn ngẫu nhiên một tập con các đặc trưng nhằm tìm ra phép tách tối ưu. Việc đưa yếu tố ngẫu nhiên vào cả dữ liệu lẫn đặc trưng giúp giảm thiểu nguy cơ quá khớp. Trong các bài toán phân lớp, dự đoán cuối cùng được xác định dựa trên nguyên tắc biểu quyết đa số giữa các cây trong rừng. Random Forest tập hợp các bộ phân loại có cấu trúc dạng cây $\{h(x, \Theta_k)\}$, $k = 1, 2, \dots$, trong đó $\{\Theta_k\}$ là các vectơ ngẫu nhiên độc lập và phân phối giống nhau (i.i.d.), và mỗi cây sẽ bỏ một phiếu bầu cho lớp phổ biến nhất tại đầu vào x . Trong quá trình xây dựng rừng, khi tập mẫu bootstrap được tạo ra bằng cách lấy mẫu có hoàn lại cho mỗi cây, khoảng 1/3 số mẫu ban đầu sẽ không được chọn. Tập các mẫu này được gọi là dữ liệu OOB (Out-of-Bag). Mỗi cây có một tập dữ liệu OOB riêng, được sử dụng để ước lượng sai số của từng cây trong rừng, gọi là ước lượng sai số OOB.

Minh họa độ chính xác của Random Forest:

Sai số tổng quát hóa (Generalization error, PE^*) của Random Forest được xác định như sau:

$$PE^* = P_{X,Y}(mg(X, Y) < 0) \quad (1.27)$$

Trong đó $mg(X, Y)$ là hàm biên (Margin function). Hàm biên đo lường mức độ mà số phiếu bầu trung bình cho lớp đúng tại (X, Y) vượt quá số phiếu bầu trung bình cho bất kỳ lớp nào khác. Ở đây, X là vectơ đặc trưng và Y là nhãn phân lớp.

Hàm biên được xác định như sau:

$$mg(X, Y) = \text{av}_k I(h_k(X) = Y) - \max_{j \neq Y} \text{av}_k I(h_k(X) = j) \quad (1.28)$$

Trong đó $I(\cdot)$ là hàm chỉ thị (Indicator function).

Giá trị Margin tỉ lệ thuận với mức độ tin cậy của mô hình trong quá trình phân loại. Độ mạnh (Strength) của Random Forest được xác định thông qua giá trị kỳ vọng của hàm biên như sau:

$$S = \mathbb{E}_{X, Y}(mg(X, Y)) \quad (1.29)$$

Nếu ρ là giá trị trung bình của hệ số tương quan, thì cận trên của sai số tổng quát hóa được xác định như sau:

$$PE^* \leq \frac{\rho(1 - s^2)}{s^2} \quad (1.30)$$

1.4.2 Extra Trees

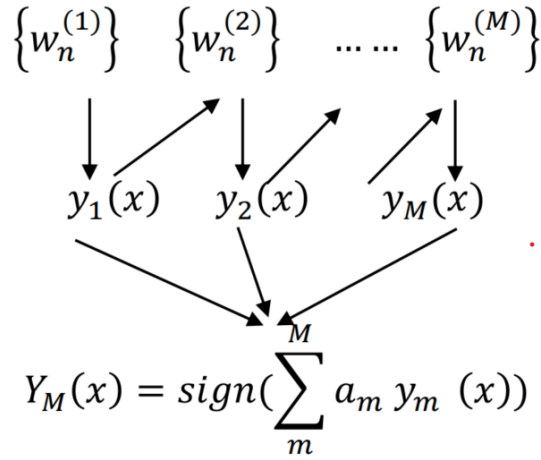
Extra Trees (ET) [81] hay còn được gọi là extremely randomized trees, có cơ chế vận hành tương tự như Random Forest nhưng tích hợp mức độ ngẫu nhiên cao hơn. ET khác biệt so với RF ở hai điểm chính: (1) mỗi cây quyết định được xây dựng dựa trên toàn bộ tập dữ liệu thay vì các mẫu bootstrap, và (2) tại mỗi nút, các phép tách được lựa chọn hoàn toàn ngẫu nhiên thay vì tìm kiếm phép tách tối ưu. Thuật toán Extra-Trees sử dụng cơ chế phân chia ngẫu nhiên đối với các thuộc tính số, được điều khiển bởi các tham số K (số lượng thuộc tính được chọn ngẫu nhiên tại mỗi nút) và n_{min} (số lượng mẫu tối thiểu để thực hiện phân chia nút). Phương pháp này khai thác toàn bộ tập dữ liệu huấn luyện ban đầu nhiều lần nhằm xây dựng một mô hình tổ hợp (ensemble) gồm M cây quyết định.

Đối với bài toán phân loại và hồi quy, dự đoán cuối cùng được tổng hợp lần lượt thông qua cơ chế bỏ phiếu đa số (majority voting) hoặc trung bình số học. Cách tiếp cận này hướng đến việc giảm phương sai thông qua quá trình ngẫu nhiên hóa rõ ràng các điểm cắt và lựa chọn thuộc tính, qua đó đạt hiệu quả cao hơn so với nhiều phương pháp khác. Việc sử dụng toàn bộ tập dữ liệu huấn luyện góp phần giảm thiểu độ chệch (bias) của mô hình.

Mặc dù độ phức tạp tính toán đạt mức $O(N \log N)$, sự đơn giản trong quy trình phân chia nút giúp nâng cao hiệu quả tính toán tổng thể. Các tham số K , n_{min} và M lần lượt ảnh hưởng đến quá trình lựa chọn thuộc tính, khả năng làm mượt nhiễu và mức độ giảm phương sai của mô hình. Mặc dù có khả năng điều chỉnh linh hoạt, các thiết lập mặc định thường được ưu tiên sử dụng nhằm đảm bảo lợi thế tính toán và tính tự động của phương pháp.

1.4.3 AdaBoost

AdaBoost (Ada) [82], hay Adaptive Boosting, là một thuật toán boosting phổ biến do Freund và Schapire đề xuất. Thuật toán huấn luyện một chuỗi các bộ phân lớp cơ sở và cập nhật trọng số của các mẫu bằng cách tăng trọng số đối với các mẫu bị phân lớp sai, sau đó huấn luyện bộ phân lớp tiếp theo dựa trên các trọng số đã cập nhật. Dự đoán cuối cùng được tạo ra bằng cách kết hợp đầu ra của tất cả các bộ phân lớp cơ sở thông qua kỹ thuật bỏ phiếu đa số có trọng số, trong đó mỗi bộ phân lớp được gán trọng số tương ứng với hiệu năng của nó.



Hình 1.3: Mô hình AdaBoost

Trong hình 1.3, các đại lượng $y_1(x), y_2(x), \dots, y_m(x)$ biểu diễn các đặc trưng và x là tín hiệu đầu vào. Các trọng số $w_n^{(1)}, w_n^{(2)}, \dots, w_n^{(m)}$ tương ứng với trọng số được gán cho từng thuộc tính tại mỗi vòng lặp.

Đầu ra của mô hình học mạnh tại vòng lặp thứ m được xác định bởi:

$$Y_m(x) = \text{sign}\left(\sum_{n=1}^M \alpha_n y_m(x)\right) \quad (1.31)$$

trong đó α_m là hệ số trọng số của mô hình học yếu tại vòng lặp thứ m . Việc điều chỉnh trọng số sau mỗi vòng lặp nhằm cải thiện dần hiệu năng phân loại là đặc trưng cốt lõi của thuật toán AdaBoost. AdaBoost cung cấp một khuôn khổ tổng quát cho các mô hình học yếu, do đó có khả năng thích ứng với nhiều bộ phân loại khác nhau. Bên cạnh đó, thuật toán có thể xử lý hiệu quả từng thuộc tính mà không yêu cầu giảm chiều dữ liệu. Cuối cùng, AdaBoost góp phần làm giảm nguy cơ quá khớp một cách hiệu quả.

1.4.4 Gradient Boosting

Gradient Boosting (GB) [83] là mở rộng của AdaBoost, cho phép sử dụng nhiều loại hàm mất mát khác nhau. Trong khi AdaBoost dựa vào trọng số của các mẫu bị phân loại sai để huấn luyện bộ phân lớp tiếp theo, Gradient Boosting thay thế cơ chế này bằng việc sử dụng đạo hàm của hàm mất mát. Cách tiếp cận này giúp nâng cao chất lượng huấn luyện của các bộ phân lớp cơ sở; tuy nhiên, GB có nhược điểm là tiêu tốn nhiều bộ nhớ và thời gian xử lý. GB hoạt động bằng cách xác định những hạn chế của các mô hình yếu thông qua đạo hàm của hàm mất mát. Quá trình này được thực hiện theo cơ chế lặp, trong đó các bộ học cơ sở được kết hợp tuần tự nhằm giảm thiểu sai số dự báo. Cụ thể, các cây quyết định được tích hợp theo mô hình cộng đồng thời tối thiểu hóa hàm mất mát bằng phương pháp hạ dốc theo đạo hàm.

Mô hình GB $F_n(x_t)$ được định nghĩa là tổng của n cây hồi quy: [83]

$$F_n(x_t) = \sum_{i=1}^n f_i(x_t) \quad (1.32)$$

Trong đó mỗi $f_i(x_t)$ là một cây quyết định (cây hồi quy). Tập hợp các cây được xây dựng tuần tự bằng cách ước lượng cây quyết định mới $f_{n+1}(x_t)$ theo bài toán tối ưu sau:

$$\arg \min_t \sum L(y_t, F_n(x_t) + f_{n+1}(x_t)) \quad (1.33)$$

trong đó $L(\cdot)$ là hàm mất mát khả vi.

1.4.5 HistGradient Boosting

HistGradient Boosting (HGB) [84], hay histogram-based gradient boosting, là một mô hình boosting sử dụng các histogram đặc trưng nhằm tăng tốc và nâng cao độ chính xác trong việc lựa chọn điểm tách tối ưu. Phương pháp này hiệu quả hơn Gradient Boosting truyền thống về tốc độ xử lý cũng như mức tiêu thụ bộ nhớ.

Mô hình HistGradient có thể được mô tả về mặt toán học như sau:

1. Khởi tạo dự đoán của mô hình:

$$F_0(x) = \arg \min_c \sum_i L(y_i, c), \quad (1.34)$$

trong đó L là hàm mất mát.

2. Với mỗi vòng lặp tăng cường $t = 1$ đến T :

- *Tính gradient âm của hàm mất mát:*

$$g_{it} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x_i)=F_{t-1}(x_i)}, \quad (1.35)$$

- *Tính xấp xỉ đạo hàm bậc hai của hàm mất mát:*

$$h_{it} = \left[\frac{\partial^2 L(y_i, F(x_i))}{\partial F(x_i)^2} \right]_{F(x_i)=F_{t-1}(x_i)}, \quad (1.36)$$

- Xây dựng histogram của các giá trị đặc trưng cùng với các gradient và đạo hàm bậc hai tương ứng.
- Xác định điểm chia tối ưu trong histogram bằng thuật toán tham lam (greedy algorithm).
- Tính toán giá trị tại các nút lá dựa trên các bin của histogram.
- Cập nhật dự đoán của mô hình:

$$F_t(x) = F_{t-1}(x) + \eta h_t(x), \quad (1.37)$$

trong đó η là tốc độ học (learning rate).

3. Xuất mô hình tăng cường cuối cùng:

$$H(x) = F_T(x). \quad (1.38)$$

1.4.6 eXtreme Gradient Boosting

eXtreme Gradient Boosting (XGB) [85], hay Extreme Gradient Boosting, là một biến thể của Gradient Boosting, trong đó thuật toán sử dụng đạo hàm bậc hai của hàm mất mát để xác định bộ phân lớp cơ sở tối ưu một cách chính xác và hiệu quả hơn. Nếu như Gradient

Boosting chỉ sử dụng đạo hàm bậc nhất để cập nhật mô hình, XGBoost khai thác đạo hàm bậc hai nhằm cải thiện đáng kể hiệu suất và tốc độ hội tụ. XGBoost là một kỹ thuật phổ biến được ghi nhận với hiệu năng vượt trội trong nhiều bài toán khoa học dữ liệu, đảm bảo cả độ chính xác cao và tốc độ xử lý nhanh. Quá trình huấn luyện của XGBoost được thực hiện theo cơ chế lặp, trong đó các cây quyết định mới được bổ sung tuần tự nhằm dự đoán sai số của các cây trước đó. Các cây mới này sau đó được kết hợp với các cây hiện có để tạo ra dự đoán cuối cùng của mô hình.

Các bước của thuật toán XGBoost [85]

Bước 1: Tính phần dư (Hàm mục tiêu mới). Tính phần dư R_i cho tất cả các mẫu trong biến mục tiêu y :

$$av = \frac{1}{N} \sum_{i=1}^N y_i \quad (1.39)$$

$$R_i = y_i - av \quad (1.40)$$

Trong đó, av là giá trị trung bình của các mẫu và R_i biểu thị phần dư của mẫu thứ i so với giá trị trung bình của biến mục tiêu.

Bước 2: Xây dựng cây quyết định mới. Huấn luyện một cây quyết định tối ưu từ tập đặc trưng f dựa trên các giá trị phần dư R_i .

Bước 3: Cập nhật phần dư mới. Tính toán phần dư mới dựa trên sai số của cây quyết định vừa được xây dựng DT_i . Giá trị phần dư được cập nhật dựa trên phần dư trước đó và tốc độ học α :

$$R_i^{(t+1)} = R_i^{(t)} + \alpha R_i^{(t)} \quad (1.41)$$

Trong đó, $\alpha \in [0.01, 0.1, 0.001, 0.3]$ là tốc độ học (learning rate), kiểm soát mức độ đóng góp của mỗi cây mới vào mô hình tổng thể.

Bước 4: Tăng cường (Boosting). Lặp lại Bước 2 và Bước 3 cho đến khi toàn bộ các cây quyết định được huấn luyện. Đối với dữ liệu mới, XGBoost thực hiện dự đoán bằng cách tổng hợp tuần tự đầu ra của tất cả các cây quyết định DT_i .

1.4.7 Multilayer Perceptron

Multilayer Perceptron (MLP) thuộc nhóm mạng nơ-ron nhân tạo, được cấu trúc từ nhiều lớp nơ-ron sắp xếp tuần tự. Kiến trúc này được xây dựng với mục tiêu mô phỏng cơ chế xử lý thông tin của não bộ con người. Một MLP điển hình bao gồm lớp đầu vào, một hoặc nhiều lớp ẩn, và lớp đầu ra. Trong đó, các lớp ẩn vốn không thể quan sát trực tiếp từ bên ngoài mạng [86] đóng vai trò then chốt trong việc trích xuất và học các đặc trưng phức tạp ẩn trong dữ liệu. Một MLP điển hình bao gồm ba loại tầng riêng biệt, bao gồm tầng đầu vào, tầng ẩn và tầng đầu ra. Ngoại trừ các nút ở tầng đầu vào, mỗi nút còn lại đều hoạt động như một nơ-ron (hoặc phần tử xử lý) với hàm kích hoạt phi tuyến, cho phép mạng học và biểu diễn các mối quan hệ phi tuyến phức tạp trong dữ liệu.

Với dữ liệu đầu vào x_i ($i = 1, 2, \dots, N$), đầu ra của mô hình mạng nơ-ron y được xác định theo phương trình sau:

$$y = f(\mathbf{W}^T \mathbf{x}) = f\left(\sum_{i=1}^N W_i x_i + b\right) \quad (1.42)$$

trong đó $f(\cdot)$ là hàm kích hoạt, N biểu thị số lượng nơ-ron, $\mathbf{W}$ là vector trọng số của mô hình mạng nơ-ron nhân tạo (ANN), và b là hệ số chệch (bias).

Đối với bài toán phân loại nhị phân, đầu ra của MLP là một giá trị nằm trong khoảng $[0, 1]$, có thể được diễn giải như xác suất thuộc về lớp mục tiêu dương. Quá trình lựa chọn mô hình được thực hiện thông qua việc tối ưu hóa số lượng tầng ẩn cũng như số lượng nơ-ron trong mỗi tầng ẩn của kiến trúc MLP.

1.5 Các độ đo đánh giá

Đánh giá hiệu suất của mô hình dự đoán lỗi là một giai đoạn quan trọng trong quá trình phát triển mô hình. Quá trình này giúp đánh giá mức độ hoạt động của mô hình và xác định hiệu quả trong việc dự đoán lỗi một cách chính xác. Các độ đo được sử dụng để đánh giá mô hình dự đoán lỗi được trình bày như dưới đây.

Bảng 1.1: Ma trận nhầm lẫn

	Dự đoán lỗi - Predicted defective	Dự đoán không lỗi - Predicted non-defective
Thực tế có lỗi	True Positive	False Negative
Thực tế không có lỗi	False Positive	True Negative

-
- **True Positive (TP):** Các mẫu thực tế có lỗi và được mô hình dự đoán đúng là có lỗi.
 - **True Negative (TN):** Các mẫu không có lỗi mà mô hình dự đoán chính xác là không lỗi.
 - **False Positive (FP):** Các mẫu thực tế không có lỗi nhưng bị mô hình dự đoán sai là có lỗi.
 - **False Negative (FN):** Các mẫu thực tế có lỗi nhưng bị mô hình dự đoán sai là không có lỗi.
 - **Accuracy:** Tỷ lệ tổng số mẫu được mô hình dự đoán đúng trên toàn bộ dữ liệu.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1.43)$$

- **Precision:** Tỷ lệ giữa số lượng và tổng số mẫu được mô hình dự đoán là có lỗi.

$$Precision = \frac{TP}{TP + FP} \quad (1.44)$$

- **Recall:** Tỷ lệ giữa số lượng mẫu có lỗi được dự đoán đúng so với tổng số mẫu lỗi thực tế.

$$Recall = \frac{TP}{TP + FN} \quad (1.45)$$

- **F1-score:** Sự cân bằng giữa giá trị Precision và Recall.

$$F1-score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (1.46)$$

- **ROC Curve:** là đồ thị biểu diễn mối quan hệ giữa TP và FP khi thay đổi ngưỡng phân loại. Một mô hình tốt sẽ có đường ROC nằm gần góc trên bên trái, thể hiện phân loại chính xác cao.
- **Area Under Curve (AUC):** là diện tích dưới đường cong ROC. Giá trị cực đại của AUC bằng 1.0. Trường hợp AUC đạt 0.5 tương ứng với đường cong ROC trùng với đường chéo, cho thấy mô hình không tốt hơn so với việc dự đoán ngẫu nhiên ở mọi ngưỡng. Ngược lại, giá trị AUC tiến gần 1.0 phản ánh mô hình hoàn hảo.
- **Matthews Correlation Coefficient (MCC):** còn được gọi là hệ số tương quan Matthews. MCC không chỉ xem xét TP và TN, mà còn tính đến cả FP và FN, nhờ

đó cung cấp cái nhìn cân bằng hơn về chất lượng mô hình.

$$MCC = \frac{TP.TN - FP.FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (1.47)$$

1.6 Kết chương

Trong chương này, luận án đã tập trung trình bày một cách toàn diện cơ sở lý luận và thực tiễn của bài toán dự đoán lỗi phần mềm, qua đó làm rõ tầm quan trọng, mục tiêu và những thách thức cốt lõi của lĩnh vực này trong bối cảnh phát triển phần mềm hiện đại. Trên cơ sở tổng hợp các công trình nghiên cứu trong và ngoài nước, chương này đã khái quát quá trình hình thành và phát triển của dự đoán lỗi phần mềm, từ các phương pháp thống kê truyền thống đến những kỹ thuật tiên tiến dựa trên học máy và học sâu. Các kết quả tổng quan cho thấy rằng dự đoán lỗi phần mềm không chỉ mang ý nghĩa kỹ thuật trong việc hỗ trợ phát hiện sớm và giảm thiểu rủi ro trong quá trình phát triển phần mềm, mà còn mang ý nghĩa chiến lược đối với việc tối ưu hóa chi phí, nâng cao độ tin cậy và chất lượng sản phẩm phần mềm.

Cụ thể, chương này đã hệ thống hóa quy trình dự đoán lỗi phần mềm gồm các bước thu thập dữ liệu, tiền xử lý, lựa chọn và trích xuất đặc trưng, huấn luyện mô hình và đánh giá hiệu suất. Trong đó, hai yếu tố then chốt quyết định đến độ chính xác của mô hình là chất lượng của đặc trưng đầu vào và đặc điểm phân bố của tập dữ liệu. Luận án đã phân tích sâu vai trò của các đặc trưng mã nguồn và các kỹ thuật lựa chọn đặc trưng, bao gồm nhóm phương pháp filter, wrapper và embedded. Các phương pháp này được xem là bước quan trọng trong việc giảm chiều dữ liệu, loại bỏ nhiễu, và tăng tính khái quát hóa của mô hình. Đồng thời, chương này cũng chỉ ra vấn đề phổ biến của các tập dữ liệu lỗi là sự mất cân bằng giữa lớp lỗi và không lỗi, khiến mô hình có xu hướng dự đoán sai lệch về lớp chiếm đa số. Các kỹ thuật lấy mẫu dữ liệu như lấy mẫu tăng, lấy mẫu giảm, cũng như các phương pháp tiên tiến như GAN đã được thảo luận như những hướng tiếp cận hiệu quả nhằm cải thiện phân bố dữ liệu và nâng cao hiệu năng của mô hình.

Bên cạnh đó, luận án đã chỉ ra hạn chế của các phương pháp truyền thống khi chỉ dựa trên đặc trưng thủ công, vốn không phản ánh được thông tin cú pháp và ngữ nghĩa sâu của mã nguồn. Các nghiên cứu gần đây đã chứng minh rằng việc khai thác thông tin ngữ nghĩa từ cây cú pháp trừu tượng, đồ thị luồng điều khiển, hoặc đồ thị phụ thuộc dữ liệu, kết hợp với các mô hình học sâu như Bi-LSTM, CNN, GCN, và đặc biệt là Transformer, có thể mang lại hiệu quả vượt trội trong việc dự đoán lỗi phần mềm nhờ khả năng học biểu diễn ngữ nghĩa và mối quan hệ ngữ cảnh giữa các thành phần trong chương trình.

Từ những phân tích trên, chương này kết luận rằng ba yếu tố đóng vai trò quyết định đối với hiệu quả của mô hình dự đoán lỗi phần mềm bao gồm: (1) lựa chọn đặc trưng đầu vào có tính phân biệt cao và phù hợp với bản chất của dữ liệu lỗi; (2) xử lý hiệu quả vấn đề mất cân bằng dữ liệu để đảm bảo khả năng học công bằng giữa các lớp; và (3) khai thác các mô hình học sâu có khả năng biểu diễn ngữ nghĩa phong phú nhằm tăng cường tính chính xác và khả năng khái quát hóa. Đây cũng chính là ba hướng nghiên cứu trọng tâm được luận án kế thừa và phát triển trong các chương tiếp theo.

Chương 2

Nâng cao khả năng dự đoán lỗi phần mềm bằng các phương pháp lựa chọn đặc trưng

Chương này tập trung trình bày một cách có hệ thống vai trò của các phương pháp lựa chọn đặc trưng nhằm loại bỏ các thuộc tính dư thừa, không liên quan hoặc nhiễu. Trong chương này, ba nhóm phương pháp lựa chọn đặc trưng filter, wrapper và embedded được nghiên cứu đồng thời nhấn mạnh tiềm năng của các thuật toán metaheuristic trong việc tối ưu hóa tập đặc trưng.

2.1 Giới thiệu

Lựa chọn đặc trưng là một bước quan trọng trong quá trình phân tích dữ liệu và áp dụng mô hình học máy. Mục tiêu của các phương pháp này là xác định tập các đặc trưng hoặc biến có mức độ liên quan cao nhất đến biến mục tiêu. Các mô hình phân loại được huấn luyện trên không gian đặc trưng đã được rút gọn thường có tính ổn định và khả năng tái lập cao hơn so với các mô hình được xây dựng trên toàn bộ không gian đặc trưng ban đầu có kích thước lớn. Kỹ thuật lựa chọn đặc trưng giúp cải thiện hiệu suất của mô hình bằng cách giảm độ phức tạp tính toán và tăng khả năng diễn giải của kết quả mô hình. Các phương pháp lựa chọn đặc trưng đóng vai trò then chốt trong quá trình xây dựng mô hình học máy. Trong quá trình lựa chọn đặc trưng, mục tiêu chính là tìm kiếm các đặc trưng có ý nghĩa hoặc các đặc trưng có tương quan cao. Các đặc trưng không cung cấp thông tin hữu ích được gọi là đặc trưng không liên quan trong khi các đặc trưng không mang thêm thông tin mới so với các đặc trưng đã được chọn được gọi là đặc trưng dư thừa. Bên cạnh đó, những đặc trưng không có mối quan hệ hoặc tương quan với biến mục tiêu được xem

như là nhiễu. Sự tồn tại của nhiễu sẽ làm sai lệch quá trình dự đoán, từ đó làm giảm hiệu suất phân loại. Do đó, việc xử lý nhiễu là cần thiết để cải thiện hiệu quả dự đoán, và điều này có thể đạt được thông qua giảm chiều dữ liệu [55]. Một thuật toán lựa chọn đặc trưng tốt hơn nên mang lại nhiều lợi ích như cung cấp cái nhìn sâu sắc về dữ liệu, xây dựng mô hình phân loại hiệu quả hơn, tăng cường khả năng khái quát hóa và nhận diện các đặc trưng không liên quan. Bên cạnh đó, nó cũng giúp hiểu rõ hơn mối quan hệ giữa các đặc trưng và biến mục tiêu, giảm yêu cầu tính toán trong quá trình giải quyết một bài toán cụ thể. Trong các tập dữ liệu có số chiều lớn, nơi số thuộc tính vượt trội so với số lượng mẫu, việc lựa chọn đặc trưng thích hợp giúp giảm chiều dữ liệu, nâng cao hiệu suất mô hình, và tiết kiệm chi phí cũng như thời gian tính toán. Hơn nữa, quá trình lựa chọn đặc trưng còn đóng vai trò quan trọng trong khám phá tri thức, khi các đặc trưng được phát hiện có thể được sử dụng trực tiếp trong các nghiên cứu tiếp theo. Điều này không chỉ giúp tăng độ tin cậy của mô hình mà còn góp phần tạo nền tảng cho các công trình khoa học sau này trong lĩnh vực học máy và khai phá dữ liệu. Các phương pháp lựa chọn đặc trưng được chia thành 3 nhóm cơ bản bao gồm filter, wrapper và embedded. Đối với các phương pháp filter, các đặc trưng quan trọng của dữ liệu được sử dụng để đánh giá mức độ ảnh hưởng của từng đặc trưng trước khi được bổ sung vào tập đặc trưng được chọn [87]. Phương pháp lựa chọn đặc trưng wrapper tích hợp thuật toán học có giám sát vào quá trình lựa chọn đặc trưng, nhằm đánh giá và xếp hạng các đặc trưng dựa trên phương pháp đánh giá tập con. Hạn chế chính của phương pháp wrapper là chi phí tính toán rất cao, do phải tìm kiếm tập đặc trưng tối ưu trong một không gian có số chiều lớn. Quá trình này đòi hỏi lặp lại nhiều lần việc huấn luyện và đánh giá mô hình, dẫn đến mức tiêu tốn tài nguyên đáng kể cả về thời gian lẫn năng lực xử lý. Bên cạnh đó, phương pháp wrapper có nguy cơ cao dẫn đến hiện tượng quá khớp, do mô hình có thể học quá chi tiết các mẫu trong dữ liệu huấn luyện, làm giảm khả năng khái quát hóa đối với dữ liệu mới. Đối với phương pháp embedded, tập đặc trưng tối ưu được tìm kiếm đồng thời với quá trình xây dựng bộ phân loại. Phương pháp lựa chọn tập đặc trưng tối ưu này phụ thuộc vào từng thuật toán phân loại cụ thể, tức là mỗi mô hình học máy có thể áp dụng một cơ chế chọn đặc trưng khác nhau nhằm tối ưu hóa hiệu suất dự đoán [55]. Ngoài ra, các nghiên cứu trước đây đã chỉ ra rằng các phương pháp wrapper thường đạt hiệu quả cao hơn so với các phương pháp filter. Thêm vào đó, vẫn còn nhiều biến thể của thuật toán metaheuristic chưa được nghiên cứu đầy đủ trong bối cảnh lựa chọn đặc trưng. Xuất phát từ những vấn đề trên, chương này được thực hiện với mục tiêu xây dựng và triển khai các thử nghiệm có hệ thống nhằm đánh giá hiệu suất của các nhóm phương pháp lựa chọn đặc trưng trên các bộ dữ liệu khác nhau. Kết quả đánh giá không chỉ giúp xác định các phương pháp lựa chọn đặc trưng hiệu quả mà còn góp phần nâng cao chất lượng giai đoạn tiền xử lý dữ liệu thông qua việc loại bỏ các đặc trưng dư thừa, không liên quan hoặc gây nhiễu, từ đó tạo ra tập dữ liệu tối ưu

hơn cho quá trình huấn luyện mô hình. Đây là một bước quan trọng trong việc cải thiện chất lượng dữ liệu đầu vào, góp phần nâng cao hiệu quả và độ tin cậy của các mô hình dự đoán lỗi phần mềm.

2.2 Các nghiên cứu liên quan

Phần này tập trung vào một số nghiên cứu chính liên quan đến các phương pháp lựa chọn đặc trưng trong dự đoán lỗi phần mềm và đề xuất vấn đề nghiên cứu.

2.2.1 Lựa chọn đặc trưng

Trong những năm gần đây, lựa chọn đặc trưng đã trở thành một trong những hướng nghiên cứu quan trọng nhằm nâng cao hiệu quả của các mô hình dự đoán lỗi phần mềm nhờ khả năng cải thiện độ chính xác dự đoán, giảm chi phí tính toán và rút ngắn thời gian xây dựng mô hình. Việc loại bỏ các đặc trưng dư thừa hoặc không liên quan giúp giảm số chiều dữ liệu, hạn chế hiện tượng quá khớp và nâng cao khả năng tổng quát hóa của mô hình, đặc biệt đối với các tập dữ liệu có không gian đặc trưng lớn và mất cân bằng lớp [44]. Một số nghiên cứu tập trung vào việc áp dụng các thuật toán tối ưu hóa metaheuristic nhằm tìm kiếm tập đặc trưng tối ưu. Các thuật toán này tận dụng khả năng tìm kiếm toàn cục để giải quyết không gian nghiệm có kích thước lớn và tránh rơi vào cực trị cục bộ. Tiêu biểu, Zhu và cộng sự [88] đã đề xuất thuật toán EMWS, kết hợp Whale Optimization Algorithm (WOA) và Simulated Annealing (SA) nhằm nâng cao hiệu quả lựa chọn tập đặc trưng. Tương tự, Pethe và cộng sự [89] sử dụng Bat Optimization Algorithm (BAT) để xác định các đặc trưng quan trọng nhất, qua đó cải thiện độ chính xác dự đoán, giảm số chiều dữ liệu và hạn chế hiện tượng quá khớp. Các nghiên cứu này cho thấy các thuật toán metaheuristic có khả năng tìm kiếm tập đặc trưng tối ưu hiệu quả hơn so với nhiều phương pháp truyền thống.

Tiếp theo, một hướng nghiên cứu khác tập trung vào việc xây dựng các phương pháp lựa chọn đặc trưng lai nhằm kết hợp ưu điểm của các nhóm phương pháp filter và wrapper. Wang và cộng sự [90] đã đề xuất mô hình LASSO-SVM, trong đó LASSO được sử dụng ở giai đoạn tiền xử lý để giảm chiều dữ liệu và loại bỏ các đặc trưng không liên quan trước khi xây dựng mô hình phân loại. Balogun và cộng sự [91] phát triển phương pháp Rank Aggregation-Based Hybrid Multi-Filter Wrapper Feature Selection (RAHMFWS), kết hợp nhiều phương pháp filter để xây dựng danh sách xếp hạng đặc trưng, sau đó sử dụng wrapper dựa trên chiến lược sắp xếp động nhằm lựa chọn tập đặc trưng tối ưu. Gần đây hơn, Malhotra và cộng sự [92] đề xuất mô hình kết hợp các phương pháp filter như Information Gain, Chi-square và ReliefF với thuật toán wrapper Opposition-Based Whale

Optimization Algorithm (OBWOA), đồng thời sử dụng mạng nơ-ron tích chập một chiều kết hợp cơ chế attention để khai thác hiệu quả các đặc trưng đã lựa chọn. Kết quả của các nghiên cứu này cho thấy việc kết hợp filter và wrapper giúp cân bằng giữa chi phí tính toán và chất lượng tập đặc trưng, từ đó cải thiện hiệu quả của mô hình dự đoán lỗi. Ngoài ra một số nghiên cứu khác mở rộng phạm vi của lựa chọn đặc trưng bằng cách kết hợp với các kỹ thuật tiền xử lý dữ liệu, đặc biệt là xử lý mất cân bằng lớp nhằm nâng cao chất lượng dữ liệu đầu vào. Chen và cộng sự [93] đã kết hợp thuật toán ReliefF với kỹ thuật SMOTE và mô hình LSSVM để đồng thời lựa chọn các đặc trưng quan trọng và cân bằng dữ liệu huấn luyện. Kết quả nghiên cứu cho thấy việc kết hợp lựa chọn đặc trưng với các phương pháp xử lý mất cân bằng dữ liệu giúp cải thiện đáng kể hiệu suất phân loại so với chỉ sử dụng riêng lẻ từng kỹ thuật.

Nhìn chung, các nghiên cứu hiện nay đều khẳng định vai trò quan trọng của lựa chọn đặc trưng trong việc nâng cao hiệu quả dự đoán lỗi phần mềm. Xu hướng phát triển chủ yếu tập trung vào ba hướng: (i) ứng dụng các thuật toán tối ưu hóa metaheuristic nhằm nâng cao hiệu quả tìm kiếm tập đặc trưng tối ưu; (ii) xây dựng các mô hình lựa chọn đặc trưng lai kết hợp giữa filter và wrapper để tận dụng ưu điểm của từng phương pháp; và (iii) kết hợp lựa chọn đặc trưng với các kỹ thuật tiền xử lý như cân bằng dữ liệu nhằm nâng cao chất lượng dữ liệu huấn luyện. Tuy nhiên, phần lớn các nghiên cứu mới chỉ tập trung đề xuất hoặc đánh giá một phương pháp lựa chọn đặc trưng cụ thể, hoặc kết hợp với các kỹ thuật lấy mẫu truyền thống như SMOTE. Chưa có nhiều nghiên cứu thực hiện đánh giá một cách hệ thống nhiều phương pháp lựa chọn đặc trưng metaheuristic trên cùng một tập dữ liệu để xác định phương pháp tối ưu, đồng thời sử dụng kết quả đó làm cơ sở kết hợp với các mô hình sinh dữ liệu tiên tiến khác nhằm tối ưu hóa toàn diện dữ liệu đầu vào cho bài toán dự đoán lỗi phần mềm. Đây chính là khoảng trống nghiên cứu mà luận án hướng tới giải quyết.

2.2.2 Vấn đề nghiên cứu

Nhiều nghiên cứu gần đây quan tâm đến vấn đề lựa chọn đặc trưng. Một số công trình đã đề xuất các chiến lược khác nhau dựa trên phương pháp filter và phương pháp wrapper, qua đó số lượng đặc trưng được giảm bớt nhưng vẫn giữ được thông tin quan trọng, góp phần nâng cao hiệu quả của mô hình. Tuy nhiên nghiên cứu chỉ dừng lại ở việc so sánh hiệu suất tổng thể của mô hình. Đáng chú ý, chưa có công trình nào đánh giá hiệu quả của từng phương pháp lựa chọn đặc trưng đến khả năng phát hiện lỗi. Vì vậy, mục tiêu nghiên cứu của luận án trong chương này tập trung vào việc phân tích sâu hơn vai trò của các phương pháp lựa chọn đặc trưng trong nâng cao khả năng dự đoán lỗi phần mềm. Nghiên cứu này hướng tới trả lời hai câu hỏi nghiên cứu chính:

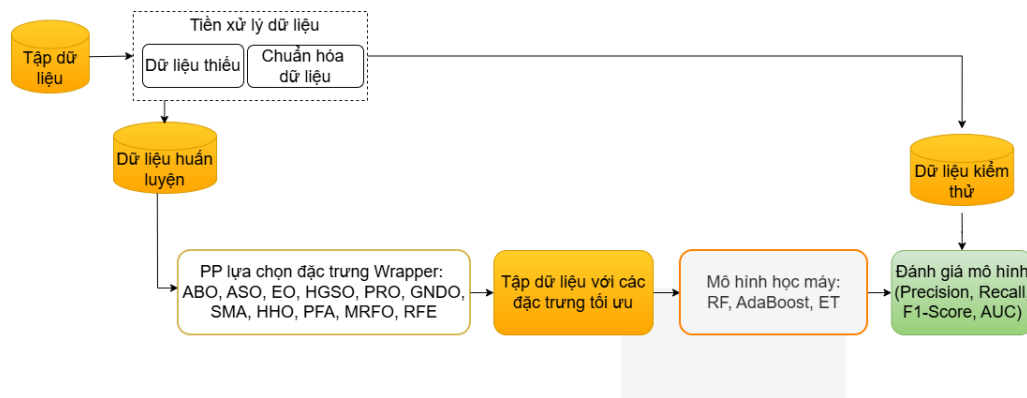
-
1. *Câu hỏi nghiên cứu 1: Các phương pháp lựa chọn đặc trưng được áp dụng hiệu quả như thế nào trong việc nâng cao khả năng dự đoán lỗi phần mềm bằng cách loại bỏ các đặc trưng phần mềm không liên quan hoặc đặc trưng dư thừa?*
 2. *Câu hỏi nghiên cứu 2: Phương pháp lựa chọn đặc trưng wrapper nào hoạt động tốt nhất trong việc chọn ra các đặc trưng tối ưu cho dự đoán lỗi phần mềm?*

Cụ thể, luận án tập trung nghiên cứu chiến lược metaheuristic được áp dụng trong quá trình lựa chọn đặc trưng bao gồm Artificial Butterfly Optimization, Atom Search Optimization, Equilibrium Optimizer, Henry Gas Solubility Optimization, Poor and Rich Optimization, Generalized Normal Distribution Optimization, Slime Mould Algorithm, Harris Hawks Optimization, Path Finder Algorithm, và Manta Ray Foraging Optimization. Các phương pháp metaheuristic được đề xuất được so sánh với mô hình cơ sở là sử dụng trực tiếp các thuật toán học máy trên các bộ dữ liệu lỗi phần mềm gốc. Thử nghiệm được thực hiện trên chín bộ dữ liệu từ kho PROMISE và AEEEM. Để đánh giá hiệu suất phân loại, luận án sử dụng ba mô hình học máy: Random Forest, Extra Trees và AdaBoost. Các độ đo đánh giá bao gồm Precision, Recall, F1-score và Area Under the Curve. Nhằm đánh giá ý nghĩa thống kê của sự chênh lệch hiệu suất giữa các kỹ thuật lựa chọn đặc trưng dạng metaheuristic và mô hình cơ sở, kiểm định Wilcoxon signed-rank đã được thực hiện ở mức ý nghĩa 0,05. Mỗi thử nghiệm được lặp lại mười lần nhằm đảm bảo độ tin cậy, tạo ra mười bộ kiểm thử độc lập. Kết quả thực nghiệm là cơ sở để đánh giá toàn diện hiệu quả của các phương pháp lựa chọn đặc trưng dựa trên metaheuristic, từ đó đề xuất các phương pháp có hiệu suất vượt trội trong việc lựa chọn tập đặc trưng tối ưu, góp phần nâng cao chất lượng dữ liệu đầu vào và cải thiện hiệu quả của các mô hình dự đoán lỗi phần mềm.

2.3 Phương pháp đề xuất

Phần này trình bày phương pháp nghiên cứu và kết quả thực nghiệm của các phương pháp lựa chọn đặc trưng sử dụng các thuật toán metaheuristic đề xuất ở trên. Trong những năm gần đây, các thuật toán metaheuristic đã được ứng dụng rộng rãi trong nhiều bài toán tối ưu hóa, và lựa chọn đặc trưng cũng là một trong những hướng nghiên cứu trọng điểm mà các thuật toán này được khai thác. Vẫn còn một số lượng đáng kể các biến thể của thuật toán metaheuristic chưa được nghiên cứu một cách toàn diện trong lĩnh vực lựa chọn đặc trưng. Điều này cho thấy tiềm năng cho các nghiên cứu tiếp theo trong việc khám phá, cải tiến và kết hợp các thuật toán metaheuristic nhằm nâng cao hiệu quả và tính ổn định của quá trình lựa chọn đặc trưng trong dự đoán lỗi phần mềm. [88]. Vì vậy, trong thử nghiệm này, luận án trình bày đánh giá thử nghiệm về các thuật toán metaheuristic trong khuôn khổ các phương pháp lựa chọn đặc trưng wrapper, nhằm giảm thiểu tính dư thừa của dữ

liệu trong các tập dữ liệu dự đoán lỗi phần mềm. Cụ thể, thử nghiệm tiến hành đánh giá hiệu suất của các kỹ thuật lựa chọn đặc trưng metaheuristic được áp dụng cho các tập dữ liệu lỗi phần mềm, bao gồm Artificial Butterfly Optimization, Atom Search Optimization, Equilibrium Optimizer, Henry Gas Solubility Optimization, Poor and Rich Optimization, Generalized Normal Distribution Optimization, Slime Mould Algorithm, Harris Hawks Optimization, Path Finder Algorithm, Manta Ray Foraging Optimization và Recursive Feature Elimination. Các phương pháp lựa chọn đặc trưng được đề xuất này được đánh giá so sánh với mô hình cơ sở, trong đó thuật toán học máy được áp dụng trực tiếp lên tập dữ liệu lỗi phần mềm gốc mà không thực hiện bước lựa chọn đặc trưng. Hình 2.1 minh họa giải pháp đề xuất này. Sau khi chia dữ liệu thành hai phần: huấn luyện và kiểm thử, luận án tiến hành tiền xử lý dữ liệu với các bước xử lý các giá trị bị thiếu và chuẩn hóa dữ liệu. Tiếp theo, các kỹ thuật lựa chọn đặc trưng dựa trên chiến lược metaheuristic được sử dụng nhằm xác định tập đặc trưng liên quan nhất. Cuối cùng, luận án áp dụng ba mô hình học máy gồm: Random Forest, Extra Trees và AdaBoost. Các độ đo Precision, Recall, F1-score và AUC được dùng nhằm mục đích đánh giá hiệu suất của các phương pháp lựa chọn đặc trưng metaheuristic.



Hình 2.1: Quy trình đánh giá các phương pháp metaheuristic

2.3.1 Tập dữ liệu

Luận án sử dụng chín bộ dữ liệu từ kho PROMISE và AEEEM, cả hai đều được công nhận rộng rãi và thường xuyên được sử dụng trong nghiên cứu dự đoán lỗi phần mềm. Các bộ dữ liệu này bao gồm các biến độc lập như Lines of Code (LOCcode) và Lines of Comments (LOComment), cùng với một biến phụ thuộc cho biết trạng thái của thành phần phần mềm được phân loại thành hai nhóm: lỗi hoặc không lỗi. Một đặc điểm đáng chú ý của bộ dữ liệu là tỷ lệ mẫu không lỗi chiếm ưu thế so với mẫu lỗi, dẫn đến vấn đề mất cân bằng dữ liệu. Chi tiết được trình bày trong Bảng 2.2.

Bảng 2.1: Tập dữ liệu sử dụng thử nghiệm

Dữ liệu	Dự án	Mẫu dữ liệu	Số mẫu lỗi	Số mẫu không lỗi	Tỷ lệ lỗi (%)
CM1	PROMISE	505	48	457	9.50
KC1	PROMISE	2107	325	1782	15.42
KC2	PROMISE	522	107	415	20.50
PC1	PROMISE	1107	76	1031	6.87
EQ	AEEEM	324	129	195	39.81
JDT	AEEEM	997	105	892	10.53
Lucene	AEEEM	691	64	627	9.26
Mylyn	AEEEM	1862	245	1617	13.16
PDE	AEEEM	1497	209	1288	13.96

2.3.2 Các phương pháp lựa chọn đặc trưng metaheuristic

Có nhiều phương pháp lựa chọn đặc trưng metaheuristic đã được áp dụng trong dự đoán lỗi phần mềm, như Binary Genetic Algorithm (BGA), Binary Particle Swarm Optimization (BPSO) và Binary Ant Colony Optimization (BACO) [94]. Luận án xem xét các phương pháp lựa chọn đặc trưng metaheuristic nhằm tối ưu hóa hiệu suất mô hình dự đoán lỗi phần mềm.

- Artificial Butterfly Optimization (ABO), lấy cảm hứng từ hành vi của bướm, được đề xuất bởi Qi và cộng sự [95]. Thuật toán này dựa trên sở thích của loài bướm rừng có đốm, vốn thường tìm đến các khu vực có ánh nắng ấm áp trong rừng và các bãi trống. Thuật toán này mô phỏng hành vi tìm kiếm thức ăn của loài bướm. Trong quá trình di chuyển, mỗi con bướm phát ra một hương thơm với cường độ nhất định, và chính cường độ mùi hương này được sử dụng để định hướng chuyển động của các cá thể trong thuật toán BOA. Dựa trên mức độ cảm nhận hương thơm, nếu một con bướm không cảm nhận được mùi hương từ bất kỳ cá thể nào khác trong không gian tìm kiếm, nó sẽ thực hiện khai thác cục bộ bằng cách di chuyển đến một vị trí ngẫu nhiên mới được chọn. Ngược lại, nếu con bướm cảm nhận được mùi hương của cá thể tốt nhất, nó sẽ di chuyển về phía con bướm đó, và quá trình này được gọi là khám phá toàn cục. BOA khai thác sự cân bằng giữa hai cơ chế khám phá và khai thác, trong đó khám phá giúp mở rộng không gian tìm kiếm toàn cục để tránh rơi vào cực trị cục bộ, còn khai thác tập trung vào việc tinh chỉnh nghiệm hiện tại nhằm cải thiện độ hội tụ đến nghiệm tối ưu. Nhờ sự mô phỏng linh hoạt hành vi tự nhiên này, BOA thể hiện khả năng tìm kiếm toàn cục mạnh mẽ và hiệu quả trong nhiều bài toán tối ưu phức tạp. Trong thuật toán BOA, mùi hương được định nghĩa là một hàm của cường độ kích thích *stimulus intensity* và được xác định theo phương trình như sau:

Bảng 2.2: Mô tả chi tiết của tập dữ liệu từ kho PROMISE

TT	Độ đo	Mô tả
1	loc	Số dòng mã nguồn (McCabe's Line of Code - LOC).
2	v(g)	Độ phức tạp chu trình (Cyclomatic Complexity) theo McCabe, phản ánh số lượng đường đi độc lập trong chương trình.
3	ev(g)	Độ phức tạp thiết yếu (Essential Complexity) theo McCabe, đo mức độ phức tạp của cấu trúc điều khiển sau khi loại bỏ các cấu trúc tuần tự.
4	iv(g)	Độ phức tạp thiết kế (Design Complexity) theo McCabe, phản ánh mức độ phức tạp trong thiết kế của mô-đun.
5	n	Tổng số toán tử và toán hạng theo độ đo Halstead (Total Operators + Operands).
6	v	Program Volume theo Halstead, biểu thị kích thước thông tin của chương trình.
7	l	Độ dài chương trình (Program Length) theo Halstead.
8	d	Độ khó của chương trình (Program Difficulty) theo Halstead.
9	i	Chỉ số thông minh (Program Intelligence) theo Halstead, phản ánh mức độ dễ hiểu của chương trình.
10	e	Nỗ lực lập trình (Programming Effort) theo Halstead, ước lượng công sức cần để phát triển hoặc hiểu chương trình.
11	b	Số lỗi ước lượng (Estimated Number of Bugs) theo Halstead.
12	t	Thời gian lập trình ước lượng (Estimated Programming Time) theo Halstead.
13	lOCode	Số dòng mã nguồn (Lines of Code).
14	lOComment	Số dòng chú thích (Lines of Comments).
15	lOBlank	Số dòng trống (Blank Lines).
16	lOCodeAndComment	Tổng số dòng mã nguồn và dòng chú thích.
17	uniq_Op	Số lượng toán tử duy nhất (Unique Operators).
18	uniq_Opnd	Số lượng toán hạng duy nhất (Unique Operands).
19	total_Op	Tổng số toán tử (Total Operators).
20	total_Opnd	Tổng số toán hạng (Total Operands).
21	branchCount	Số lượng nhánh trong đồ thị luồng điều khiển (Control Flow Graph Branch Count).
22	defects	Nhãn lỗi của mô-đun: false – mô-đun không có lỗi được báo cáo; true – mô-đun có ít nhất một lỗi được báo cáo.

$$f = cI^a \quad (2.1)$$

Trong đó, f là độ lớn của mùi hương *fragrance magnitude* phát ra từ một cá thể bướm nhất định; c là hệ số cảm nhận *sensory modality*, có giá trị nằm trong khoảng $[0,1]$, thể hiện khả năng cảm nhận kích thích của bướm; I là cường độ kích thích *stimulus intensity* tương ứng với mùi hương được phát ra bởi bướm; a là số mũ công suất *power exponent*, phụ thuộc vào hệ số cảm nhận c và có giá trị trong khoảng $[0,1]$. Dựa trên giá trị của f , thuật toán BOA xác định hai cơ chế chính để cập nhật vị trí của các bướm trong quá trình tìm kiếm nghiệm tối ưu. Phương trình 2.2 chịu trách nhiệm cho quá trình tìm kiếm toàn cục *global search*, trong đó các cá thể bướm di chuyển về phía cá thể có giá trị thích nghi tốt nhất trong quần thể, nhằm mở rộng phạm vi khám phá trong không gian tìm kiếm và tăng khả năng thoát khỏi các cực trị cục bộ. Trong khi đó, phương trình 2.3 mô tả quá trình tìm kiếm cục bộ *local search*, nơi các cá thể bướm tinh chỉnh vị trí của mình dựa trên sự tương tác ngẫu nhiên với các cá thể khác, qua đó tăng cường khả năng khai thác khu vực lân cận của nghiệm tiềm năng.

$$x_i^{t+1} = x_i^t + (r^2 \times g^* - x_i^t) \times f_i \quad (2.2)$$

$$x_i^{t+1} = x_i^t + (r^2 \times x_j^t - x_k^t) \times f_i \quad (2.3)$$

Trong đó, g^* biểu thị nghiệm tốt nhất (best solution) đạt được tại vòng lặp hiện tại, phản ánh cá thể bướm có giá trị thích nghi cao nhất trong quần thể. f_i là độ lớn của mùi hương phát ra từ cá thể bướm thứ i được sử dụng để điều chỉnh mức độ di chuyển của bướm trong không gian tìm kiếm. r là một giá trị ngẫu nhiên được chọn trong khoảng $[0,1]$ nhằm tạo ra yếu tố ngẫu nhiên cần thiết giúp thuật toán duy trì khả năng khám phá toàn cục và tránh rơi vào cực trị cục bộ.

- Atom Search Optimisation (ASO) là một thuật toán metaheuristic mới được đề xuất, lấy cảm hứng từ động lực học phân tử [96]. Thuật toán ASO mô phỏng nguyên lý cơ bản của động lực học phân tử và quy luật chuyển động của các nguyên tử, bao gồm đặc trưng của hàm thế năng, lực tương tác và lực ràng buộc hình học. Trong ASO, mỗi nguyên tử duy trì hai vector, bao gồm vị trí và vận tốc như sau:

$$X_i = [X_i^1, X_i^2, \dots, X_i^D] \quad (2.4)$$

$$V_i = [V_i^1, V_i^2, \dots, V_i^D] \quad (2.5)$$

trong đó X_i và V_i lần lượt là vị trí và vận tốc của nguyên tử thứ i , và D là số chiều tối đa của không gian tìm kiếm. Chuyển động của nguyên tử chịu ảnh hưởng mạnh bởi gia tốc của nó. Về mặt toán học, gia tốc của nguyên tử được xác định như sau:

$$a_i = \frac{F_i + G_i}{m_i} \quad (2.6)$$

trong đó F là lực tương tác, G là lực ràng buộc, và m là khối lượng của nguyên tử.

Lực tương tác giữa nguyên tử thứ i và nguyên tử thứ j được xác định dựa trên thế năng Lennard–Jones (L–J) như sau [95]:

$$F_{ij} = \frac{24\varepsilon}{\sigma} \left[2 \left(\frac{\sigma}{r_{ij}} \right)^{13} - \left(\frac{\sigma}{r_{ij}} \right)^7 \right] \frac{r_{ij}}{r_{ij}^d} \quad (2.7)$$

$$F'_{ij} = \frac{24\varepsilon}{\sigma} \left[2 \left(\frac{\sigma}{r_{ij}} \right)^{13} - \left(\frac{\sigma}{r_{ij}} \right)^7 \right] \quad (2.8)$$

trong đó F'_{ij} là độ lớn (mô-đun) của lực tương tác, ε là độ sâu của hố thế năng, σ là tham số tỉ lệ chiều dài, r_{ij} là khoảng cách giữa hai nguyên tử (nguyên tử i và j), và d là số chiều của không gian tìm kiếm.

- Equilibrium Optimizer (EO), được đề xuất bởi Faramarzi và cộng sự [97], là một thuật toán lấy cảm hứng từ vật lý, dựa trên mô hình cân bằng khối lượng thể tích để xác định trạng thái động và cân bằng. Cơ chế này cho phép EO cân bằng hiệu quả giữa khả năng khám phá và khai thác trong quá trình tìm kiếm. Thuật toán EO bao gồm một quần thể ban đầu các vectơ nồng độ trong không gian tìm kiếm, trong đó mỗi vectơ đại diện cho một nghiệm tiềm năng và được xem như vị trí của phần tử đó. Quần thể ban đầu này được khởi tạo nhằm bắt đầu quá trình tối ưu hóa. Quần thể ban đầu này được khởi tạo nhằm bắt đầu quá trình tối ưu hóa, theo công thức sau:

$$C_i^d = LB + rand_i^d \times (UB - LB), \quad i = 1, 2, \dots, N \text{ and } d = 1, 2, \dots, D \quad (2.9)$$

Trong đó, kích thước của quần thể được xác định bởi N , trong khi số chiều của bài

toán được biểu thị bởi D . Giới hạn dưới và giới hạn trên của không gian tìm kiếm lần lượt được ký hiệu là LB và UB . Vector nồng độ ban đầu của cá thể thứ i trong quần thể được biểu diễn bởi C_i^d . Vector $rand_i^d$ có các phần tử ngẫu nhiên được phân bố trong khoảng $[0,1]$.

EO hội tụ về một trạng thái cân bằng, đại diện cho kết quả cuối cùng của quá trình tối ưu hóa. Tuy nhiên, trong suốt quá trình tìm kiếm, chỉ các ứng viên cân bằng (equilibrium candidates) mới được sử dụng để định hướng cho từng cá thể trong mô hình tìm kiếm, trong khi trạng thái cân bằng cuối cùng vẫn chưa được biết trước. Bốn cá thể có giá trị hàm thích nghi cao nhất được xác định trong EO sẽ hình thành tập hợp các ứng viên cân bằng, giúp tăng cường khả năng khám phá của thuật toán. Bên cạnh đó, để nâng cao khả năng khai thác cục bộ, giá trị trung bình của bốn cá thể tốt nhất cũng được bổ sung vào tập này. Tập hợp gồm năm ứng viên cân bằng nói trên được gọi là “equilibrium pool”, và được định nghĩa bằng vector sau:

$$\vec{C}_{eq,pool} = \left\{ \vec{C}_{eq(1)}, \vec{C}_{eq(2)}, \vec{C}_{eq(3)}, \vec{C}_{eq(4)}, \vec{C}_{eq(avg)} \right\}, \quad (2.10)$$

Trong đó,

$$\vec{C}_{eq(avg)} = \frac{\vec{C}_{eq(1)} + \vec{C}_{eq(2)} + \vec{C}_{eq(3)} + \vec{C}_{eq(4)}}{4} \quad (2.11)$$

và

$$f_{C_{eq(1)}} \leq f_{C_{eq(2)}} \leq f_{C_{eq(3)}} \leq f_{C_{eq(4)}}, \quad (2.12)$$

Vec-tơ $\vec{C}_{eq(avg)}$ xác định tập cân bằng (equilibrium pool), trong đó $\vec{C}_{eq(1)}, \vec{C}_{eq(2)}, \vec{C}_{eq(3)}, \vec{C}_{eq(4)}$ là bốn ứng viên cân bằng tốt nhất được xác định cho đến thời điểm hiện tại, và giá trị trung bình của bốn ứng viên này được biểu diễn bởi $\vec{C}_{eq(avg)}$. Trong mỗi vòng lặp, với xác suất lựa chọn ngẫu nhiên như nhau giữa các nghiệm tiềm năng, nồng độ của các cá thể trong quần thể sẽ được cập nhật. Phương trình 2.10 được sử dụng để cập nhật vector nồng độ, được mô tả như sau:

$$C_{new} = C_{eq} + \frac{G}{\lambda}(1 - F) + (C_{old} - C_{eq}) \times F, \quad (2.13)$$

Trong đó, C_{old} và C_{new} lần lượt biểu thị vector nồng độ hiện tại và vector nồng độ mới của các cá thể. Trong tập cân bằng (equilibrium pool), một vector nồng độ C_{eq} được chọn ngẫu nhiên để làm điểm tham chiếu cho quá trình cập nhật. Phương trình (2.11) được sử dụng để tính toán vec-tơ F , thường được xác định như là thành phần

mũ trong quá trình tối ưu hóa.

$$\vec{F} = e^{-\vec{\lambda}(t-t_0)}, \quad (2.14)$$

Trong đó, $\vec{\lambda}$ là một vec-tơ ngẫu nhiên với d chiều, với các phần tử được phân bố trong khoảng $[0,1]$. Ở mỗi vòng lặp, giá trị của tham số t được giảm dần theo số lần lặp, nhằm điều chỉnh tốc độ hội tụ.

- Henry Gas Solubility Optimization (HGSO) [98], dựa trên định luật Henry [99], giải quyết các bài toán tối ưu hóa phức tạp bằng cách mô phỏng quá trình cụm hóa phân tử khí, giúp duy trì sự cân bằng giữa khám phá – khai thác và tránh hội tụ sớm. HGSO khởi tạo quần thể ban đầu gồm N nghiệm ứng viên, trong đó mỗi cá thể biểu diễn một tập con của các đặc trưng cần được lựa chọn để đánh giá. Bước khởi tạo này đóng vai trò quan trọng trong quá trình hội tụ cũng như chất lượng của nghiệm tối ưu thu được. **Bước 1:** Quần thể ban đầu x_i^0 được sinh ngẫu nhiên theo công thức sau:

$$x_i^0 = lb_i + rand_i \times (ub_i - lb_i) \quad (2.15)$$

Trong đó, x_i^0 biểu thị vị trí khởi tạo ban đầu của hạt khí thứ i , lb_i và ub_i lần lượt là giới hạn dưới và giới hạn trên của không gian tìm kiếm tương ứng với nghiệm ứng viên thứ i , $rand_i$ là giá trị ngẫu nhiên được sinh ra để đảm bảo sự phân bố ngẫu nhiên của các nghiệm trong không gian tìm kiếm ban đầu. Vì mỗi hạt khí đều duy trì các đặc tính riêng của nó, nên các đặc tính này cũng được khởi tạo tương ứng trong quá trình mô phỏng như trong công thức 2.16:

$$H_j^0 = l_1 \times rand_1, \quad P_{i,j}^0 = l_2 \times rand_2, \quad C_j^0 = l_3 \times rand_3 \quad (2.16)$$

Trong đó, H_j^0 là giá trị khởi tạo của hằng số Henry đối với loại j , $P_{i,j}^0$ là giá trị khởi tạo của áp suất riêng phần của khí j trong cụm j và C_j^0 là giá trị hằng khởi tạo của loại j . Trong phương trình 2.13, các hằng số l_1, l_2, l_3 là các hằng số có các giá trị lần lượt bằng 5E-02, 100, và 1E-02.

Bước 2: Phân cụm. Quần thể các hạt khí được chia thành k cụm, tương ứng với các loại khí khác nhau. Vì mỗi cụm chỉ bao gồm các hạt khí thuộc cùng một loại, nên mỗi cụm sẽ có một giá trị hằng số Henry H_j khác nhau.

Bước 3: Đánh giá. Các hạt khí trong từng cụm được đánh giá dựa trên giá trị hàm thích nghi và xác định phần tử có giá trị tốt nhất trong cụm, ký hiệu $x_{j,best}$. Mặt khác, trong bước này, tất cả các nghiệm ứng viên đều được sắp hạng nhằm xác định nghiệm tối ưu toàn cục x_{best} trong toàn bộ quần thể.

Bước 4: Cập nhật trọng số Henry rong mỗi vòng lặp, khi áp suất của các hạt khí thay đổi, việc cập nhật hệ số Henry H_i^{t+1} trở nên cần thiết. Hệ số này được tính toán lại theo phương trình 2.17.

$$H_j^{t+1} = H_j^t \times \exp \left[-C_j \times \left(\frac{1}{T^t} - \frac{1}{T^\theta} \right) \right], \quad T^t = \exp \left(-\frac{t}{t_{\max}} \right) \quad (2.17)$$

Trong đó, H_j^t là hệ số Henry của cụm j tại vòng lặp thứ t , T^t biểu thị nhiệt độ tại vòng lặp t , T^θ là giá trị nhiệt độ tham chiếu cố định, được chọn bằng 298.15. Tham số $t_{\max}$ biểu thị tổng số vòng lặp tối đa được thiết lập cho quá trình tối ưu.

Bước 5: Cập nhật độ hòa tan. Độ hòa tan $S_{i,j}^t$ của hạt khí thứ i trong cụm thứ j tại vòng lặp thứ t được cập nhật theo phương trình (2.18):

$$S_{i,j}^t = K \times H_j^{t+1} \times P_{i,j}^t \quad (2.18)$$

Trong đó, $P_{i,j}^t$ là áp suất riêng phần của hạt khí thứ i trong cụm j , H_j^{t+1} là hệ số Henry đã được cập nhật của cụm j và K là một hằng số.

Bước 6: Cập nhật vị trí. Vì các đặc tính của mỗi hạt khí đã được tính toán trong các bước trước, nên ở giai đoạn này, vị trí của hạt khí thứ i trong cụm thứ j tại vòng lặp $(t + 1)$ được cập nhật theo phương trình (2.19):

$$x_{i,j}^{t+1} = x_{i,j}^t + F \times rand_1 \times \gamma \times (x_{j,\text{best}} - x_{i,j}^t) + F \times rand_2 \times \alpha \times (S_{i,j}^t \times x_{\text{best}} - x_{i,j}^t), \quad (2.19)$$

trong đó:

$$\gamma = \beta \times \exp \left(-\frac{F_{t,\text{best}} + \varepsilon}{F_{i,j}^t + \varepsilon} \right), \quad \varepsilon = 0.05 \quad (2.20)$$

Trong đó, F được sử dụng như một hệ số cờ để điều khiển hướng tìm kiếm; γ biểu diễn khả năng của hạt khí so với cụm của nó; α là hệ số điều chỉnh ảnh hưởng của các hạt khí khác lên hạt thứ i ; $rand_1$ và $rand_2$ là hai giá trị thực được sinh ngẫu nhiên trong khoảng $[0, 1]$; ε là một giá trị nhỏ nhằm tránh lỗi chia cho không.

Trong phương trình (2.19), $F_{t,\text{best}}$ và $F_{i,j}^t$ lần lượt biểu diễn giá trị hàm thích nghi tốt nhất toàn cục và của hạt khí thứ i trong cụm j . Cơ chế này giúp cân bằng giữa khả năng khai thác và khám phá trong không gian tìm kiếm, đảm bảo quá trình hội tụ

hiệu quả của thuật toán HGSO.

Bước 7: Thoát khỏi cực trị cục bộ Để triển khai chiến lược tránh rơi vào nghiệm cực trị cục bộ, thuật toán HGSO lựa chọn N_w nghiệm có chất lượng kém nhất để khởi tạo lại, theo phương trình (2.21):

$$N_w = N \times [rand \times (c_2 - c_1) + c_1], \quad c_1 = 0.1, \quad c_2 = 0.2 \quad (2.21)$$

Trong đó, N là kích thước quần thể (population size); $rand$ là giá trị ngẫu nhiên được sinh trong khoảng $[0, 1]$; c_1 và c_2 là hai hệ số điều chỉnh biên dưới và biên trên quy định tỷ lệ phần tử bị khởi tạo lại. Các nghiệm có giá trị thích nghi thấp nhất được chọn ra theo phương trình (2.21) sẽ được khởi tạo lại vị trí ban đầu bằng cách áp dụng lại phương trình (2.16), nhằm tăng khả năng thoát khỏi cực trị cục bộ và mở rộng phạm vi tìm kiếm của thuật toán.

Để áp dụng thuật toán này cho lựa chọn một tập con đặc trưng, cần thực hiện một bước trung gian gọi là chuyển đổi nhị phân trước khi tiến hành đánh giá hàm thích nghi. Do đó, mỗi nghiệm ban đầu x_i^0 phải được chuyển đổi sang dạng nhị phân x_i^{bin} theo công thức 2.22:

$$x_i^{bin} = \begin{cases} 1, & \text{if } x_i^0 > 0.5, \\ 0, & \text{otherwise.} \end{cases} \quad (2.22)$$

Để giải thích chi tiết hơn về quá trình chuyển đổi, ta xét nghiệm x_i gồm sáu phần tử được cho bởi $x_i^0 = [0.6, 0.1, 0.7, 0.43, 0.2, 0.81]$. Phép chuyển đổi được thực hiện dựa trên công thức 2.13 nhằm tạo ra vec-tơ nhị phân $x_i^{bin} = [1, 0, 1, 0, 0, 1]$, trong đó, giá trị 1 biểu thị đặc trưng được chọn, trong khi giá trị 0 biểu thị đặc trưng không được chọn. Điều này có nghĩa là đặc trưng thứ nhất, thứ ba và thứ sáu trong tập dữ liệu gốc là các đặc trưng có liên quan và cần được giữ lại, trong khi các đặc trưng còn lại là không liên quan và cần bị loại bỏ. Sau khi xác định được tập các đặc trưng được chọn, hàm thích nghi sẽ được tính toán cho từng nghiệm x_i^{bin} nhằm xác định chất lượng của các đặc trưng này. Giá trị mục tiêu cho nghiệm thứ i được xác định theo công thức 2.23:

$$Fit_i = w_1 \times Err_i + w_2 \times \frac{d_i}{D} \quad (2.23)$$

Trong đó, $w_1=0.99$ và $w_2 = 1-w_1$. Trọng số w_1 biểu thị hệ số cân bằng được sử dụng nhằm duy trì sự cân đối giữa tỷ lệ lỗi phân loại Err_i và số lượng đặc trưng được chọn d_i , D biểu thị tổng số thuộc tính trong tập dữ liệu gốc.

- Poor and Rich Optimization (PRO), được đề xuất bởi Moosavi và Bardsiri [100], lấy cảm hứng từ hành vi tài chính của các cá nhân trong xã hội, được mô hình hóa thông qua tương tác giữa nhóm giàu hơn và nhóm nghèo hơn với mục tiêu cải thiện vị thế của họ. Nhìn chung, con người trong một xã hội có thể được phân thành hai tầng lớp tài chính. Nhóm thứ nhất bao gồm những người giàu hơn có mức tài sản cao hơn trung bình. Nhóm thứ hai bao gồm những người nghèo hơn có mức tài sản thấp hơn trung bình. Mỗi cá nhân trong hai nhóm này đều mong muốn cải thiện vị thế tài chính của mình trong xã hội. Những người thuộc tầng lớp kinh tế nghèo cố gắng cải thiện tình hình tài chính và thu hẹp khoảng cách giai cấp bằng cách học hỏi từ những người giàu hơn. Ngược lại, những người thuộc tầng lớp kinh tế giàu lại cố gắng mở rộng khoảng cách giai cấp bằng cách quan sát và học hỏi từ những người thuộc tầng lớp nghèo. Trong bài toán tối ưu hóa, mỗi nghiệm cá thể trong quần thể nghèo sẽ di chuyển về phía nghiệm tối ưu toàn cục trong không gian tìm kiếm, thông qua việc học hỏi từ nghiệm giàu trong quần thể giàu.
- Thuật toán Generalized Normal Distribution Optimization (GNDO), được giới thiệu bởi Zhang và cộng sự [101] với ý tưởng được lấy cảm hứng từ lý thuyết phân phối chuẩn, hay còn được gọi là phân phối Gaussian. Ngoài ra, thuật toán này có đặc điểm nổi bật là đơn giản và chỉ yêu cầu một số ít tham số so với các thuật toán khác, cụ thể GNDO chỉ cần kích thước quần thể và số vòng lặp làm tham số đầu vào. Trong GNDO, quá trình tìm kiếm được mô hình hóa thành ba bước chính. Ban đầu, tất cả các nghiệm trong quần thể được khởi tạo với phân bố ngẫu nhiên, không tuân theo phân phối chuẩn. Ở các giai đoạn tiếp theo của quá trình tìm kiếm, các nghiệm này sẽ cập nhật vị trí của mình để tìm nghiệm tối ưu toàn cục thông qua chiến lược khai thác và khai phá. Cuối cùng, các nghiệm sẽ di chuyển dần về nghiệm tốt nhất. Do đó, toàn bộ quá trình tìm kiếm có thể được biểu diễn bằng nhiều phân phối chuẩn khác nhau.

Quá trình khai thác được dẫn hướng bởi giá trị của nghiệm tốt nhất hiện tại và giá trị trung bình của các nghiệm trong quần thể, trong khi quá trình thăm dò dựa trên các nghiệm được chọn ngẫu nhiên. Chi tiết của từng giai đoạn sẽ được trình bày trong các tiểu mục sau.

Giai đoạn khai thác: Trong giai đoạn khai thác của thuật toán GNDO, quá trình tìm kiếm được thực hiện dựa trên mối quan hệ giữa phân bố của các nghiệm cá thể và phân bố chuẩn. Theo đó, mô hình phân bố chuẩn tổng quát của quá trình khai thác được xác định theo phương trình (2.24):

$$v_i^t = \mu_i + \delta_i \times \eta, \quad i = 1, 2, 3, \dots, N \quad (2.24)$$

Trong đó, v_i^t là véc-tơ thử nghiệm tương ứng với nghiệm thứ i trong không gian tìm kiếm tại thời điểm t , μ_i biểu diễn giá trị trung bình tổng quát của nghiệm thứ i trong quần thể, δ_i biểu diễn độ lệch chuẩn tổng quát, và η là hệ số phạt. Các biến được sử dụng trong phương trình (2.24) được xác định lần lượt theo các phương trình (2.25), (2.26), và (2.27) như sau:

$$\mu_i = \frac{1}{3} (x_i^t + x_{\text{best}}^t + M) \quad (2.25)$$

$$\delta_i = \sqrt{\frac{1}{2} [(x_i^t - \mu_i)^2 + (x_{\text{best}}^t - \mu_i)^2 + (M - \mu_i)^2]} \quad (2.26)$$

$$\eta = \begin{cases} \sqrt{-\log(\lambda_1)} \times \cos(2\pi\lambda_2), & \text{nếu } a \leq b, \\ \sqrt{-\log(\lambda_1)} \times \cos(2\pi\lambda_2 + \pi), & \text{ngược lại.} \end{cases} \quad (2.27)$$

Trong đó, a , b , λ_1 và λ_2 là các giá trị ngẫu nhiên được sinh trong khoảng $[0, 1]$. Ngoài ra, x_{best} biểu diễn nghiệm tốt nhất hiện tại, trong khi M là giá trị trung bình của quần thể, được xác định theo phương trình (2.28):

$$M = \frac{\sum_{i=1}^N x_i^t}{N} \quad (2.28)$$

Trong phương trình (2.28), x_i^t là nghiệm thứ i tại vòng lặp t , và N là tổng số nghiệm trong quần thể.

Giai đoạn thăm dò của thuật toán GNDO chủ yếu dựa trên ba nghiệm được chọn ngẫu nhiên, được mô tả chi tiết theo phương trình (2.29):

$$v_i^t = x_i^t + \beta \times (|\lambda_3| \times v_1) + (1 - \beta) \times (|\lambda_4| \times v_2) \quad (2.29)$$

Trong đó, các biến λ_3 và λ_4 là các số ngẫu nhiên tuân theo phân phối chuẩn; β là hệ số điều chỉnh được sinh ngẫu nhiên trong khoảng $[0, 1]$; còn v_1 và v_2 là các véc-tơ thử nghiệm.

Giá trị của v_1 và v_2 được xác định theo phương trình (2.30) và (2.31) như sau:

$$v_1 = \begin{cases} x_i^t - x_{rp1}^t, & \text{nếu } fitness(x_i^t) < fitness(x_{rp1}^t), \\ x_{rp1}^t - x_i^t, & \text{ngược lại.} \end{cases} \quad (2.30)$$

$$v_2 = \begin{cases} x_{rp2}^t - x_{rp3}^t, & \text{nếu } fitness(x_{rp2}^t) < fitness(x_{rp3}^t), \\ x_{rp3}^t - x_{rp2}^t, & \text{ngược lại.} \end{cases} \quad (2.31)$$

Trong đó, p_1, p_2 và p_3 là ba nghiệm ngẫu nhiên được chọn từ danh sách các nghiệm khả dụng có chỉ số từ 1 đến N , thỏa mãn điều kiện $p_1 \neq p_2 \neq p_3 \neq i$. Hàm $fitness()$ được sử dụng để đánh giá giá trị độ thích nghi của mỗi nghiệm.

- Slime Mold Algorithm (SMA), được phát triển bởi Li và cộng sự [102], dựa trên cơ chế dao động của loài nấm nhầy trong tự nhiên, vốn có khả năng tìm ra đường đi tối ưu đến nguồn thức ăn mà không cần hệ thần kinh trung ương. Thuật toán SMA được xây dựng dựa trên hành vi và sự thay đổi hình thái của loài nấm nhầy. Trong quần thể tìm kiếm, các cá thể được chia thành ba nhóm với vai trò khác nhau. Một số cá thể sẽ được lựa chọn ngẫu nhiên từ giai đoạn đầu và tái sinh với một tỷ lệ tương ứng z nhằm duy trì khả năng khám phá không gian tìm kiếm. Dựa trên vị trí hiện tại, một nhóm cá thể khác sẽ tiếp tục thực hiện hoạt động tìm kiếm cục bộ, trong khi nhóm còn lại sẽ được hướng dẫn di chuyển về phía ứng viên có giá trị thích nghi tốt nhất, tức là nghiệm tối ưu hiện tại của quần thể. Các biểu thức toán học chi tiết mô tả cơ chế vận hành của thuật toán SMA được trình bày như sau:

Quá trình khởi tạo: Quy tắc sau đây được đề xuất nhằm mô hình hoá toán học cho phương pháp SMA như được trình bày trong phương trình 2.32:

$$\vec{S}(it+1) = \begin{cases} \vec{S}_b(it) + \vec{v}_b \cdot (\vec{W} \cdot \vec{S}_A(it) - \vec{S}_B(it)), & r < p, \\ \vec{v}_c \cdot \vec{S}(it), & r \geq p, \end{cases} \quad (2.32)$$

Trong đó, $\vec{v}_b$ được xác định như trong Phương trình (4), còn $\vec{v}_c$ giảm tuyến tính từ 1 xuống 0. $\vec{S}_b$ biểu thị vị trí có độ chính xác $fitness$ cao nhất, it là số vòng lặp hiện tại, và $\vec{S}$ đại diện cho vị trí của thể nấm nhầy. Các véc-tơ $\vec{S}_A$ và $\vec{S}_B$ lần lượt biểu diễn hai cá thể được chọn ngẫu nhiên từ quần thể. $\vec{W}$ thể hiện trọng số của nấm nhầy, trong khi p được xác định như trong Phương trình 2.33.

$$p = \tanh |F(j) - bF| \quad (2.33)$$

Trong đó $F(j)$ biểu thị giá trị độ thích nghi của cá thể $\vec{S}$, với $j \in \{1, 2, \dots, n\}$, và bF thể hiện giá trị fitness tốt nhất trong toàn bộ các vòng lặp. Véc-tơ $\vec{v}_b$ được xác định như sau:

$$\vec{v}_b = [a, -a] \quad (2.34)$$

trong đó tham số a được tính theo công thức 2.35:

$$a = \operatorname{arctanh}\left(-\frac{it}{\max_{it}} + 1\right) \quad (2.35)$$

Tại đây, it là chỉ số vòng lặp hiện tại, trong khi $\max_{it}$ đại diện cho tổng số vòng lặp tối đa.

Đánh giá độ thích nghi: Định nghĩa của véc-tơ trọng số $\vec{W}$ được xác định như sau:

$$\vec{W}(F_{\text{sort}}(j)) = \begin{cases} 1 + r \cdot \log\left(\frac{bF - F(j)}{bF - wF} + 1\right), & \text{cond.}, \\ 1 - r \cdot \log\left(\frac{bF - F(j)}{bF - wF} + 1\right), & \text{others}, \end{cases} \quad (2.36)$$

trong đó wF là giá trị fitness thấp nhất, bF là giá trị fitness cao nhất, cond. chỉ ra rằng $F(j)$ thuộc nửa đầu của quần thể (tức có độ thích nghi cao hơn trung bình), F_{sort} là tập hợp các giá trị fitness đã được sắp xếp tăng dần, và r là giá trị ngẫu nhiên trong khoảng $[0, 1]$. Quá trình sắp xếp được mô tả bằng công thức:

$$F_{\text{sort}} = \text{sort}(F) \quad (7)$$

Tuy nhiên, SMA còn tồn tại các vấn đề về tốc độ hội tụ chậm, mất cân bằng giữa khả năng khai phá và khai thác, cũng như dễ bị rơi vào cực trị cục bộ.

- Harris Hawks Optimization (HHO), được giới thiệu bởi Heidari và cộng sự [103], mô phỏng chiến lược săn mồi hợp tác của loài điều hâu Harris. HHO là một thuật toán dựa trên quần thể có hiệu suất cao, đồng thời khả năng khai phá không gian tìm kiếm của nó rất tốt nên HHO được xem là một phương pháp đầy tiềm năng trong việc giải quyết các bài toán tối ưu phức tạp. Thuật toán bắt đầu bằng việc khởi tạo ngẫu nhiên các tác tử “điều hâu” trong không gian tìm kiếm và dẫn dắt chúng qua các giai đoạn khám phá và khai thác. Các cá thể điều hâu Harris sử dụng hai cơ chế chính để khám phá không gian tìm kiếm: (i) khám phá dựa trên vị trí của các cá thể khác trong quần thể tức là mở rộng vùng tìm kiếm mới và (ii) khám phá ngẫu nhiên trong phạm vi giá trị cho phép. Phương trình 2.37 mô tả quá trình tìm kiếm của thuật toán HHO, trong đó hai cơ chế này có cùng xác suất được chọn:

$$X(t+1) = \begin{cases} X_{\text{rand}}(t) - r_1 |X_{\text{rand}}(t) - 2r_2 X(t)|, & q \geq 0.5, \\ (X_{\text{rabbit}}(t) - X_m(t)) - r_3 (LB + r_4 (UB - LB)), & q < 0.5, \end{cases} \quad (2.37)$$

trong đó X biểu diễn véc-tơ vị trí của cá thể điều hâu, $X(t+1)$ là vị trí mới của cá thể tại vòng lặp $t+1$, còn $X_{\text{rabbit}}(t)$ là vị trí con mồi (nghiệm kỳ vọng) tại vòng lặp t . $X_{\text{rand}}(t)$ là vị trí của một cá thể được chọn ngẫu nhiên tại vòng lặp t , và X_m là vị trí trung bình của tất cả các cá thể trong quần thể, được tính như trong Phương trình 2.38. Các biến ngẫu nhiên r_1, r_2, r_3, r_4, q đều nằm trong khoảng $(0, 1)$, trong khi LB và UB lần lượt biểu thị cận dưới và cận trên của không gian tìm kiếm. Phương trình 2.38 trình bày công thức tính vị trí trung bình của quần thể tại vòng lặp t được tính theo công thức:

$$X_m(t) = \frac{1}{N} \sum_{i=1}^N X_i(t), \quad (2.38)$$

với N là tổng số cá thể trong quần thể điều hâu.

Mỗi nghiệm (con mồi) có một giá trị năng lượng E thể hiện khả năng thoát khỏi sự tấn công của điều hâu. Phương trình 2.39 mô tả cách tính E tại vòng lặp t , trong đó T là tổng số vòng lặp tối đa và E_0 là giá trị năng lượng ban đầu được chọn ngẫu nhiên trong khoảng $(-1, 1)$. Giá trị của E quyết định quá trình tìm kiếm mà thuật toán sẽ thực hiện: khi $|E| \geq 1$, thuật toán tiến hành giai đoạn *khám phá*, ngược lại, khi $|E| < 1$, thuật toán chuyển sang giai đoạn *khai thác*.

$$E = 2 \times E_0 \left(1 - \frac{t}{T}\right) \quad (2.39)$$

Giai đoạn chính thứ hai của thuật toán HHO là *giai đoạn khai thác*, liên quan trực tiếp đến giá trị năng lượng E tại vòng lặp t . Thuật toán HHO sử dụng hai cơ chế để thực hiện quá trình khai thác: bao vây mềm *soft besiege* và bao vây cứng *hard besiege*. Quá trình chuyển đổi giữa hai cơ chế này phụ thuộc vào biến ngẫu nhiên r , biểu diễn xác suất con mồi có thể trốn thoát khỏi điều hâu. Một giá trị ngưỡng được sử dụng để xác định xác suất thoát thành công của con mồi (ví dụ, khi $r \geq 0.5$). Trường hợp bao vây mềm xảy ra nếu $r \geq 0.5$ và $|E| \geq 0.5$, tức là con mồi vẫn còn mức năng lượng cao, như được mô tả trong Phương trình 2.39.

Ký hiệu $\Delta X(t)$ biểu thị sự chênh lệch giữa vị trí hiện tại của con mồi và véc-tơ vị

trí tại vòng lặp t , trong khi $J = 2(1 - r_5)$ mô phỏng một bước nhảy ngẫu nhiên dựa trên giá trị ngẫu nhiên r_5 trong khoảng $(0, 1)$.

$$X(t + 1) = \Delta X(t) - E |J X_{\text{rabbit}}(t) - X(t)| \quad (2.40)$$

trong đó:

$$\Delta X(t) = X_{\text{rabbit}}(t) - X(t) \quad (2.41)$$

Ngược lại, quá trình *hard besiege* diễn ra khi năng lượng của con mồi nhỏ hơn 0.5, như được biểu diễn trong Phương trình 2.42. Tại giai đoạn này, các cá thể điều hòa giảm dần khoảng cách trung bình giữa vị trí của chúng và vị trí của con mồi mục tiêu. Các biến Y và Z được xác định lần lượt theo các Phương trình 2.44 và 2.45, trong khi giá trị trung bình của quần thể $X_m(t)$ được tính từ Phương trình 2.38. Sau đó, vị trí của điều hòa được cập nhật dựa trên Phương trình 2.42.

$$X(t + 1) = \begin{cases} Y, & \text{nếu } F(Y) < F(X(t)), \\ Z, & \text{nếu } F(Z) < F(X(t)), \end{cases} \quad (2.42)$$

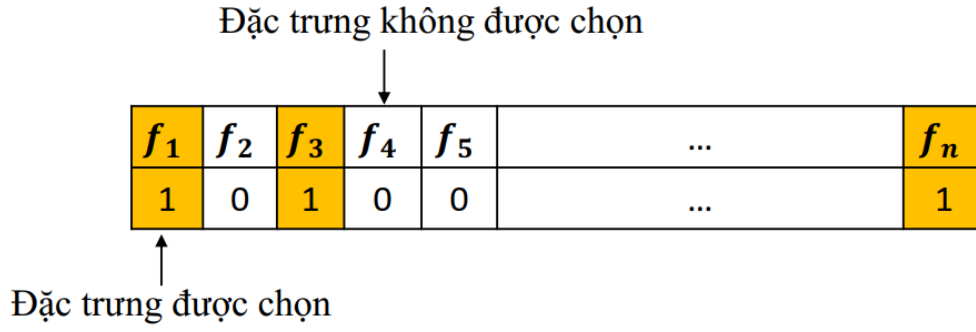
trong đó Y và Z được xác định theo:

$$X(t + 1) = X_{\text{rabbit}}(t) - E |\Delta X(t)| \quad (2.43)$$

$$Y = X_{\text{rabbit}}(t) - E |J X_{\text{rabbit}}(t) - X(t)| \quad (2.44)$$

$$Z = Y + S \times LF(D) \quad (2.45)$$

Trong đó, $\Delta X(t)$ là sai lệch vị trí giữa điều hòa và con mồi, J là hệ số ngẫu nhiên kiểm soát cường độ tấn công, S là véc-tơ bước nhảy ngẫu nhiên, và $LF(D)$ là hàm bước Lévy (Lévy Flight) dùng để mô phỏng sự di chuyển đột ngột và ngẫu nhiên trong không gian tìm kiếm có chiều D . Cơ chế này cho phép thuật toán HHO duy trì sự đa dạng trong quần thể và tăng khả năng thoát khỏi cực trị cục bộ. Thuật toán HHO được áp dụng như một phương pháp lựa chọn đặc trưng nhị phân. Trong đó, mỗi nghiệm trong quần thể được mã hoá dưới dạng một véc-tơ nhị phân, như minh hoạ trong Hình 2.2, trong đó giá trị **0** biểu thị *đặc trưng không được chọn* (ví



Hình 2.2: Biểu diễn giải pháp dưới dạng vector nhị phân

du, f_5), trong khi giá trị **1** biểu thị *đặc trưng được chọn* (ví dụ, f_1). Để đánh giá chất lượng của tập đặc trưng được chọn, thuật toán sẽ sử dụng bộ phân loại làm bộ phân loại nội bộ dựa trên giá trị *resubLoss* làm tiêu chí đánh giá hiệu suất của hàm mục tiêu.

- Pathfinder Algorithm (PFA), được đề xuất bởi Yapici và Cetinkaya [104], là một phương pháp metaheuristic mô phỏng hành vi kiếm ăn tập thể của các nhóm động vật, trong đó các cá thể vừa theo sau một thủ lĩnh, vừa dựa trên nhận thức riêng của mình để khám phá không gian tìm kiếm. Thuật toán PFA có khả năng tránh rơi vào cực trị địa phương và hướng tới đạt cực trị toàn cục. Cơ chế của thuật toán bao gồm việc cập nhật vị trí của các cá thể trong đàn và cá thể dẫn đầu một cách riêng biệt. Quá trình cập nhật vị trí của từng cá thể được xác định bởi Công thức 2.46:

$$x_i^{k+1} = x_i^k + R_1 (x_j^k - x_i^k) + R_2 (x_p^k - x_i^k) + \varepsilon \quad (2.46)$$

Trong đó, k là số vòng lặp hiện tại, x_i biểu diễn vector vị trí của cá thể thứ i , x_j là véc-tơ cá thể thứ j , R_1 và R_2 là véc-tơ ngẫu nhiên được xác định như sau:

$$R_1 = \alpha r_1, \quad R_2 = \beta r_2 \quad (2.47)$$

Hệ số α xác định khoảng di chuyển của một cá thể so với hàng xóm của nó, hệ số hấp dẫn β quy định mức độ hội tụ ngẫu nhiên giữa các cá thể để duy trì sự liên kết của đàn xung quanh cá thể dẫn đầu. Các giá trị α và β được chọn ngẫu nhiên trong khoảng $[1, 2]$ và một phân phối đều trong khoảng $[0, 1]$ cũng được sử dụng để sinh các giá trị ngẫu nhiên r_1 và r_2 . Sau mỗi vòng lặp, ε biểu thị độ dao động được sinh ra, được xác định bởi Công thức 2.48:

$$\varepsilon = \left(1 - \frac{k}{k_{\max}}\right) u_1 \cdot D_{ij}, \quad D_{ij} = x_i - x_j \quad (2.48)$$

Trong đó, u_1 là một vector ngẫu nhiên có giá trị trong khoảng $[-1, 1]$, D_{ij} biểu thị khoảng cách giữa hai cá thể i và j , $k_{\max}$ là số vòng lặp tối đa của thuật toán. Công thức 2.49 được sử dụng để cập nhật vị trí của cá thể dẫn đầu trong quá trình tìm kiếm con mồi:

$$x_p^{k+1} = x_p^k + 2r_3 (x_p^k - x_p^{k-1}) + A \quad (2.49)$$

Trong Công thức (2.49), r_3 là một vector ngẫu nhiên được sinh ra theo phân phối đều trong khoảng $[0, 1]$. Mỗi vòng lặp của thuật toán sử dụng Công thức 2.50 để tính toán giá trị A , được mô tả như sau:

$$A = u_2 e^{-2k/k_{\max}} \quad (2.50)$$

Trong đó, u_2 là một vector ngẫu nhiên có giá trị trong khoảng $[-1, 1]$.

Mặc dù thuật toán PFA có nhiều ưu điểm, nhưng nó cũng tồn tại một số hạn chế. Cụ thể, khi kích thước bài toán lớn, hiệu suất của thuật toán PFA giảm đáng kể. Ở các vòng lặp cuối, việc tìm ra nghiệm mới trở nên bất khó khăn. Khi đó toàn bộ quần thể tiến hóa đến một điểm duy nhất, nghĩa là các cá thể trong quần thể trở nên giống hệt nhau, dẫn đến sự mất đa dạng và dừng tìm kiếm sớm.

- Thuật toán Manta Ray Foraging Optimization (MRFO), được giới thiệu bởi Zhao và cộng sự [105], lấy cảm hứng từ chiến lược kiếm ăn độc đáo của loài cá đuối Manta, vốn có khả năng tìm kiếm trên diện rộng và điều chỉnh linh hoạt vị trí của chúng đến các khu vực có nguồn thức ăn dồi dào hơn. Sinh vật phù du là nguồn thức ăn chính của loài này. Trong chiến lược săn mồi đầu tiên, được gọi là tìm kiếm theo chuỗi, các cá thể Manta di chuyển thành hàng nối tiếp để có thể bắt được những sinh vật phù du bị bỏ sót bởi các cá thể ở phía trước. Trong chiến lược này, các cá thể Manta thực hiện các động tác nhào lộn ngược và chuyển động xoay tròn quanh vị trí con mồi, giúp kéo sinh vật phù du lại gần mang của chúng để dễ dàng thu nhận. Ba cơ chế này mô phỏng hoàn chỉnh hành vi kiếm ăn tự nhiên của loài cá đuối Manta, đồng thời được chuyển hóa thành các phép toán tìm kiếm và cập nhật vị trí trong MRFO nhằm cân bằng giữa hai giai đoạn *khám phá* và *khai thác* không gian tìm kiếm.

Chiến lược tìm kiếm theo chuỗi Chiến lược này được xây dựng sao cho mỗi cá thể trong quần thể cập nhật vị trí của mình dựa trên vị trí tốt nhất hiện có trong quần thể và vị trí của cá thể đứng ngay phía trước nó. Ngoại trừ cá thể dẫn đầu, vị trí của nó chỉ được cập nhật dựa trên nghiệm tốt nhất toàn cục. Do đó, chiến lược tìm kiếm

theo chuỗi được mô hình hóa theo công thức sau:

$$x_i^{t+1} = \begin{cases} x_i^t + r \cdot (G_{\text{best}}^t - x_i^t) + 2r \cdot \sqrt{\log(r)} \cdot (G_{\text{best}}^t - x_i^t), & i = 1, \\ x_i^t + r \cdot (x_{i-1}^t - x_i^t) + 2r \cdot \sqrt{\log(r)} \cdot (G_{\text{best}}^t - x_i^t), & i = 2, \dots, N, \end{cases} \quad (2.51)$$

Trong đó, N là kích thước quần thể, r là véc-tơ ngẫu nhiên có giá trị trong khoảng $(0, 1)$, x_i^t biểu thị vị trí hiện tại của cá thể thứ i tại vòng lặp t , và G_{best} là nghiệm tốt nhất được tìm thấy cho đến thời điểm hiện tại.

Chiến lược tìm kiếm xoáy ốc Tại đây, các cá thể cá đuối manta thực hiện các chuyển động xoáy ốc hướng về vị trí tối ưu toàn cục. Do đó, một phương trình dạng xoáy ốc được sử dụng để mô phỏng hành vi này như sau:

$$x_i^{t+1} = \begin{cases} G_{\text{best}} + r \cdot (G_{\text{best}}^t - x_i^t) + \beta \cdot (G_{\text{best}}^t - x_i^t), & i = 1, \\ G_{\text{best}} + r \cdot (x_{i-1}^t - x_i^t) + \beta \cdot (G_{\text{best}}^t - x_i^t), & i = 2, \dots, N, \end{cases} \quad (2.52)$$

Trong đó, hệ số $\beta = \beta = 2e^{r_1 \frac{T-t+1}{T}} \sin(2\pi r_1)$, N là tổng số lần lặp tối đa, và $r_1 \in [0, 1]$ là một giá trị ngẫu nhiên. Để tăng cường khả năng khám phá trong không gian tìm kiếm, các cá thể Manta thực hiện những dịch chuyển ngẫu nhiên được mô tả như sau: Để tăng cường khả năng khám phá không gian tìm kiếm, các cá thể manta ray thực hiện các dịch chuyển ngẫu nhiên được mô tả bởi phương trình sau:

$$x_i^{t+1} = \begin{cases} x_{\text{rand}} + r (x_{\text{rand}}^t - x_i^t) + \beta (x_{\text{rand}}^t - x_i^t), & i = 1, \\ x_{\text{rand}} + r (x_{i-1}^t - x_i^t) + \beta (x_{\text{rand}}^t - x_i^t), & i = 2, \dots, N, \end{cases} \quad (2.53)$$

Trong đó, x_{rand} là một vị trí được chọn ngẫu nhiên trong không gian tìm kiếm, được xác định theo công thức:

$$x_{\text{rand}} = LB + r \cdot (UB - LB), \quad (2.54)$$

với LB và UB lần lượt là cận dưới và cận trên của không gian tìm kiếm.

- Recursive Feature Elimination (RFE) [57] là một phương pháp lựa chọn đặc trưng thuộc nhóm wrapper, được triển khai dựa trên thuật toán tham lam. Thuật toán khởi đầu với toàn bộ tập đặc trưng và đánh giá chúng thông qua độ chính xác của bộ phân lớp. Sau mỗi vòng lặp, các đặc trưng được xếp hạng và đặc trưng kém quan trọng

nhất (hoặc ít liên quan nhất) sẽ bị loại bỏ. Quy trình này được lặp lại cho đến khi chỉ còn lại các đặc trưng thực sự có liên quan trong mô hình.

2.3.3 Kết quả thử nghiệm

Phần này cung cấp các kết quả thử nghiệm, được trình bày một cách có hệ thống nhằm trả lời hai câu hỏi nghiên cứu đã được đề xuất ở mục vấn đề nghiên cứu đặt ra ở trên.

1. Câu hỏi nghiên cứu 1: Các kỹ thuật lựa chọn đặc trưng được áp dụng hiệu quả như thế nào trong việc nâng cao khả năng dự đoán lỗi phần mềm bằng cách loại trừ các đặc trưng phần mềm không liên quan hoặc đặc trưng dư thừa?

Để trả lời câu hỏi nghiên cứu 1, luận án thực hiện nhiều thử nghiệm nhằm so sánh mười kỹ thuật lựa chọn đặc trưng thông qua việc sử dụng ba mô hình học máy được trình bày. Trong các Bảng 2.3 - 2.5, các kết quả thử nghiệm tốt nhất được đánh dấu in đậm.

Kết quả của Random Forest. Bảng 2.3 cho thấy rằng, trong tất cả các phương pháp lựa chọn đặc trưng được đánh giá, sự kết hợp giữa RFE và RF đạt hiệu suất trung bình cao nhất trên tất cả các độ đo đánh giá, trong khi EO xếp thứ hai. Đáng chú ý, RFE đạt hiệu suất cao nhất với giá trị trung bình của Precision, Recall, F1-score và AUC là 0.72, 0.84, 0.78 và 0.87. Tiếp theo, EO đạt các giá trị trung bình thứ hai trên tất cả các độ đo, với Precision là 0.69, Recall là 0.83, AUC là 0.76 và F1-score là 0.87. PRO đạt hiệu suất trung bình đứng thứ ba, với các giá trị Precision, Recall, F1-score và AUC là 0.68, 0.84, 0.76 và 0.84. Qua phân tích, có thể thấy rằng RFRE, EO và PRO đạt hiệu suất tương đối vượt trội khi được chọn làm phương pháp lựa chọn đặc trưng cho Random Forest, nổi bật hơn so với các phương pháp khác được khảo sát.

Kết quả của AdaBoost. Các kết quả được tóm tắt trong Bảng 2.4 cho thấy RFE đạt hiệu suất cao trên tất cả hầu hết các tập dữ liệu, cụ thể RFE thể hiện vượt trội về giá trị Precision, Recall, F1-score và AUC trung bình, lần lượt đạt 0.75, 0.83, 0.79 và 0.83. Tiếp theo HGSO cũng đạt giá trị Precision và AUC cao thứ hai trên phần lớn các bộ dữ liệu thử nghiệm. Cụ thể, trên các bộ dữ liệu KC1, KC2, PC1, EQ và Lucene, HGSO đạt giá trị AUC cao lần lượt là 0.80, 0.82, 0.89, 0.92 và 0.88. Cuối cùng, EO đạt giá trị Recall và F1-score trung bình cao lần lượt là 0.82 và 0.73.

Kết quả của Extra Trees. Từ kết quả thử nghiệm thu được trong Bảng 2.5, RFE cũng đạt giá trị Recall, F1-score và AUC vượt trội so với các phương pháp lựa chọn

Bảng 2.3: Hiệu suất của các phương pháp lựa chọn đặc trưng metaheuristic khi áp dụng với Random Forest

Tập dữ liệu	Độ đo	Random Forest											
		Cơ sở	ABO	ASO	EO	HGSO	PRO	GNDO	SMA	HHO	PFA	MRFO	RFE
CM1	P	0.50	0.55	0.49	0.50	0.62	0.48	0.54	0.55	0.49	0.61	0.48	0.74
	R	0.56	0.46	0.71	0.96	0.70	0.81	0.63	0.71	0.46	0.96	0.45	0.87
	F1	0.55	0.47	0.62	0.70	0.61	0.72	0.58	0.62	0.47	0.75	0.47	0.85
	AUC	0.61	0.83	0.82	0.87	0.78	0.79	0.77	0.54	0.51	0.72	0.69	0.90
KC1	P	0.58	0.65	0.62	0.65	0.67	0.65	0.62	0.58	0.64	0.63	0.6	0.76
	R	0.65	0.73	0.70	0.81	0.72	0.81	0.70	0.71	0.75	0.69	0.68	0.87
	F1	0.61	0.68	0.66	0.72	0.69	0.72	0.66	0.64	0.69	0.66	0.64	0.81
	AUC	0.74	0.81	0.80	0.82	0.80	0.79	0.81	0.8	0.81	0.82	0.81	0.88
KC2	P	0.65	0.65	0.64	0.67	0.64	0.67	0.56	0.65	0.65	0.73	0.74	0.78
	R	0.76	0.69	0.72	0.71	0.74	0.81	0.59	0.67	0.65	0.77	0.76	0.87
	F1	0.70	0.67	0.67	0.69	0.69	0.72	0.58	0.66	0.65	0.75	0.75	0.83
	AUC	0.81	0.86	0.78	0.83	0.85	0.79	0.56	0.62	0.81	0.81	0.89	0.88
PC1	P	0.51	0.60	0.63	0.69	0.59	0.65	0.62	0.63	0.55	0.59	0.65	0.76
	R	0.52	0.85	0.81	0.78	0.72	0.81	0.72	0.81	0.57	0.72	0.73	0.89
	F1	0.52	0.70	0.71	0.73	0.65	0.72	0.67	0.71	0.56	0.65	0.69	0.77
	AUC	0.60	0.77	0.83	0.94	0.89	0.79	0.84	0.74	0.65	0.83	0.85	0.92
EQ	P	0.73	0.84	0.83	0.90	0.81	0.89	0.74	0.67	0.8	0.77	0.70	0.82
	R	0.72	0.83	0.84	0.90	0.80	0.91	0.74	0.69	0.8	0.76	0.70	0.89
	F1	0.73	0.84	0.83	0.90	0.80	0.90	0.74	0.68	0.8	0.77	0.70	0.83
	AUC	0.76	0.89	0.88	0.93	0.90	0.92	0.75	0.70	0.81	0.77	0.72	0.87
JDT	P	0.70	0.73	0.80	0.80	0.80	0.73	0.73	0.77	0.68	0.69	0.66	0.65
	R	0.75	0.86	0.87	0.89	0.86	0.86	0.78	0.82	0.79	0.79	0.75	0.81
	F1	0.72	0.79	0.84	0.84	0.83	0.79	0.75	0.79	0.73	0.73	0.7	0.72
	AUC	0.83	0.87	0.90	0.90	0.89	0.87	0.84	0.85	0.86	0.87	0.83	0.79
Lucene	P	0.53	0.70	0.63	0.77	0.73	0.73	0.62	0.57	0.68	0.58	0.73	0.72
	R	0.62	0.89	0.97	0.91	0.90	0.97	0.84	0.66	0.7	0.8	0.73	0.81
	F1	0.57	0.79	0.76	0.83	0.80	0.83	0.71	0.61	0.69	0.67	0.73	0.82
	AUC	0.79	0.91	0.85	0.85	0.77	0.89	0.81	0.73	0.64	0.83	0.83	0.89
Mylyn	P	0.59	0.65	0.60	0.68	0.66	0.65	0.60	0.58	0.57	0.59	0.57	0.65
	R	0.74	0.80	0.74	0.82	0.83	0.77	0.75	0.81	0.71	0.72	0.73	0.80
	F1	0.66	0.72	0.66	0.75	0.73	0.71	0.67	0.67	0.63	0.65	0.64	0.72
	AUC	0.88	0.89	0.88	0.90	0.90	0.89	0.85	0.68	0.84	0.84	0.84	0.88
PDE	P	0.56	0.58	0.61	0.59	0.58	0.65	0.57	0.57	0.58	0.55	0.61	0.64
	R	0.61	0.83	0.76	0.69	0.76	0.81	0.74	0.69	0.76	0.72	0.76	0.72
	F1	0.59	0.68	0.67	0.64	0.66	0.72	0.64	0.63	0.66	0.62	0.68	0.67
	AUC	0.58	0.75	0.79	0.76	0.79	0.79	0.80	0.68	0.78	0.75	0.79	0.80
Giá trị trung bình													
	P	0.59	0.66	0.65	0.69	0.68	0.68	0.62	0.62	0.63	0.64	0.64	0.72
	R	0.66	0.77	0.79	0.83	0.78	0.84	0.72	0.73	0.69	0.77	0.70	0.84
	F1	0.63	0.7	0.71	0.76	0.72	0.76	0.67	0.67	0.65	0.69	0.67	0.78
	AUC	0.73	0.84	0.84	0.87	0.84	0.84	0.78	0.70	0.75	0.80	0.81	0.87

đặc trưng khác trên các bộ dữ liệu CM1, KC1, KC2 và PC1. FRE đạt giá trị Precision cao nhất trên các bộ dữ liệu CM1, KC1, KC2, PC1, Mylyn và PDE, với các giá trị lần lượt là 0.66, 0.68, 0.78, 0.70, 0.73 và 0.68. Đối với bộ dữ liệu EQ, EO thể hiện hiệu suất dự đoán tốt hơn tất cả các mô hình lựa chọn đặc trưng khác, với giá trị Recall và AUC cao nhất lần lượt là 0.96 và 0.86. Xét theo giá trị trung bình, RFE

Bảng 2.4: Hiệu suất của các phương pháp lựa chọn đặc trưng metaheuristic khi áp dụng với AdaBoost

Tập dữ liệu	Độ đo	Cơ sở	ABO	ASO	EO	HGSO	PRO	GNDO	SMA	HHO	PFA	MRFO	RFE
CM1	P	0.50	0.55	0.49	0.50	0.62	0.48	0.49	0.5	0.49	0.49	0.49	0.74
	R	0.70	0.71	0.46	0.50	0.86	0.45	0.46	0.45	0.46	0.46	0.46	0.86
	F1	0.60	0.62	0.47	0.60	0.72	0.46	0.47	0.50	0.47	0.47	0.47	0.80
	AUC	0.74	0.76	0.71	0.83	0.72	0.59	0.67	0.63	0.61	0.65	0.68	0.82
KC1	P	0.50	0.54	0.62	0.56	0.53	0.55	0.59	0.58	0.53	0.59	0.57	0.65
	R	0.60	0.62	0.75	0.94	0.68	0.70	0.72	0.68	0.65	0.72	0.79	0.83
	F1	0.60	0.58	0.68	0.70	0.60	0.61	0.65	0.63	0.59	0.65	0.66	0.80
	AUC	0.76	0.77	0.79	0.66	0.80	0.77	0.80	0.80	0.79	0.79	0.82	0.81
KC2	P	0.69	0.75	0.68	0.56	0.67	0.67	0.71	0.68	0.75	0.79	0.83	0.85
	R	0.71	0.78	0.73	0.94	0.71	0.79	0.68	0.75	0.70	0.79	0.79	0.89
	F1	0.70	0.76	0.70	0.70	0.69	0.73	0.70	0.72	0.73	0.79	0.81	0.87
	AUC	0.79	0.88	0.81	0.66	0.82	0.81	0.75	0.71	0.78	0.80	0.9	0.86
PC1	P	0.50	0.50	0.50	0.56	0.59	0.50	0.63	0.65	0.50	0.56	0.53	0.70
	R	0.50	0.47	0.47	0.84	0.72	0.60	0.76	0.73	0.65	0.64	0.59	0.85
	F1	0.50	0.48	0.48	0.70	0.65	0.60	0.69	0.69	0.6	0.59	0.56	0.75
	AUC	0.74	0.82	0.81	0.66	0.89	0.85	0.82	0.66	0.7	0.84	0.82	0.77
EQ	P	0.67	0.52	0.77	0.81	0.88	0.68	0.70	0.70	0.71	0.67	0.66	0.87
	R	0.70	0.88	0.77	0.83	0.87	0.69	0.70	0.70	0.72	0.67	0.67	0.85
	F1	0.68	0.65	0.77	0.82	0.88	0.69	0.70	0.70	0.72	0.67	0.67	0.86
	AUC	0.67	0.65	0.87	0.83	0.92	0.82	0.83	0.71	0.79	0.79	0.79	0.89
JDT	P	0.58	0.71	0.82	0.72	0.68	0.77	0.69	0.69	0.71	0.69	0.63	0.84
	R	0.79	0.77	0.89	0.81	0.71	0.87	0.71	0.8	0.74	0.74	0.70	0.79
	F1	0.67	0.74	0.85	0.76	0.70	0.82	0.70	0.74	0.73	0.72	0.66	0.82
	AUC	0.71	0.82	0.83	0.76	0.81	0.88	0.80	0.84	0.85	0.84	0.81	0.89
Lucene	P	0.57	0.57	0.70	0.75	0.62	0.54	0.55	0.61	0.66	0.66	0.58	0.72
	R	0.88	0.63	0.79	0.88	0.77	0.86	0.56	0.68	0.80	0.80	0.80	0.79
	F1	0.69	0.60	0.74	0.85	0.68	0.69	0.56	0.64	0.72	0.72	0.67	0.76
	AUC	0.71	0.75	0.81	0.84	0.88	0.64	0.74	0.72	0.78	0.80	0.78	0.82
Myllyn	P	0.50	0.54	0.58	0.60	0.58	0.58	0.56	0.53	0.57	0.55	0.59	0.65
	R	0.70	0.72	0.67	0.80	0.71	0.71	0.64	0.74	0.68	0.59	0.68	0.73
	F1	0.60	0.62	0.62	0.72	0.64	0.64	0.60	0.62	0.62	0.57	0.63	0.70
	AUC	0.63	0.80	0.81	0.75	0.71	0.75	0.82	0.69	0.81	0.83	0.80	0.80
PDE	P	0.52	0.60	0.57	0.56	0.59	0.59	0.57	0.52	0.6	0.56	0.68	0.75
	R	0.88	0.74	0.69	0.84	0.67	0.69	0.77	0.93	0.79	0.75	0.76	0.85
	F1	0.65	0.66	0.62	0.70	0.63	0.64	0.65	0.67	0.68	0.64	0.72	0.77
	AUC	0.65	0.68	0.74	0.66	0.75	0.74	0.77	0.65	0.72	0.76	0.72	0.81
Giá trị trung bình													
	P	0.56	0.59	0.64	0.62	0.64	0.60	0.61	0.61	0.61	0.62	0.62	0.75
	R	0.72	0.70	0.69	0.82	0.74	0.71	0.67	0.72	0.69	0.68	0.69	0.83
	F1	0.63	0.63	0.66	0.73	0.69	0.65	0.64	0.66	0.65	0.65	0.65	0.79
	AUC	0.71	0.77	0.80	0.74	0.81	0.76	0.78	0.71	0.76	0.79	0.79	0.83

vượt trội hơn tất cả các phương pháp lựa chọn đặc trưng khác về Precision, Recall, F1-score và AUC, với các giá trị lần lượt là 0.71, 0.80, 0.80 và 0.86.

2. Câu hỏi nghiên cứu 2: Phương pháp lựa chọn đặc trưng wrapper nào hoạt động tốt nhất trong việc chọn ra các đặc trưng tối ưu cho dự đoán lỗi phần mềm?

Bảng 2.5: Hiệu suất của các phương pháp lựa chọn đặc trưng metaheuristic khi áp dụng với Extra Trees

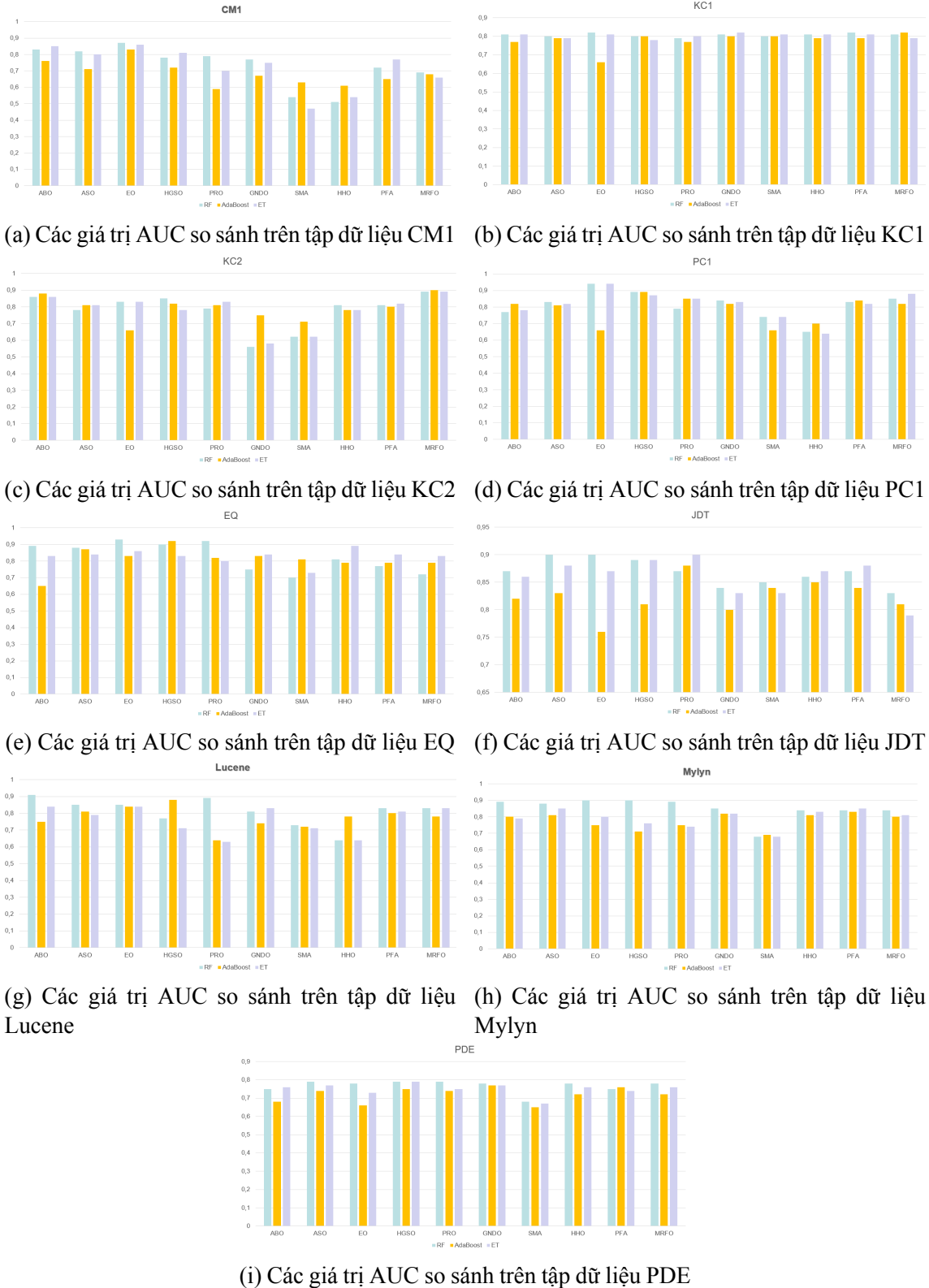
Tập dữ liệu	Độ đo	Cơ sở	ABO	ASO	EO	HGSO	PRO	GNDO	SMA	HHO	PFA	MRFO	RFE
CM1	P	0.55	0.55	0.59	0.61	0.54	0.47	0.6	0.55	0.49	0.61	0.48	0.66
	R	0.71	0.71	0.66	0.96	0.70	0.45	0.71	0.71	0.46	0.96	0.45	0.85
	F1	0.62	0.62	0.63	0.75	0.61	0.46	0.65	0.62	0.47	0.75	0.47	0.86
	AUC	0.85	0.85	0.80	0.86	0.81	0.70	0.75	0.47	0.54	0.77	0.66	0.88
KC1	P	0.65	0.65	0.62	0.66	0.64	0.62	0.64	0.56	0.64	0.63	0.62	0.68
	R	0.70	0.70	0.68	0.75	0.69	0.70	0.69	0.69	0.70	0.67	0.70	0.72
	F1	0.67	0.67	0.65	0.70	0.66	0.66	0.67	0.62	0.67	0.65	0.66	0.69
	AUC	0.81	0.81	0.79	0.81	0.78	0.80	0.82	0.81	0.81	0.81	0.79	0.80
KC2	P	0.70	0.70	0.70	0.72	0.64	0.71	0.56	0.65	0.65	0.72	0.78	0.78
	R	0.75	0.75	0.73	0.75	0.69	0.80	0.59	0.67	0.65	0.75	0.81	0.88
	F1	0.73	0.73	0.71	0.73	0.66	0.75	0.58	0.66	0.65	0.73	0.80	0.84
	AUC	0.86	0.86	0.81	0.83	0.78	0.83	0.58	0.62	0.78	0.82	0.89	0.88
PC1	P	0.52	0.56	0.63	0.69	0.66	0.62	0.59	0.63	0.51	0.62	0.66	0.70
	R	0.55	0.80	0.76	0.78	0.75	0.72	0.66	0.76	0.52	0.72	0.79	0.80
	F1	0.53	0.66	0.69	0.73	0.70	0.67	0.62	0.69	0.52	0.67	0.72	0.75
	AUC	0.60	0.78	0.82	0.94	0.87	0.85	0.83	0.74	0.64	0.82	0.88	0.89
EQ	P	0.77	0.77	0.82	0.61	0.68	0.72	0.77	0.67	0.81	0.81	0.68	0.74
	R	0.76	0.76	0.81	0.96	0.67	0.72	0.77	0.69	0.8	0.8	0.69	0.84
	F1	0.77	0.77	0.82	0.75	0.68	0.72	0.77	0.68	0.81	0.8	0.68	0.81
	AUC	0.83	0.83	0.84	0.86	0.83	0.80	0.84	0.73	0.89	0.84	0.83	0.86
JDT	P	0.78	0.73	0.73	0.72	0.73	0.75	0.72	0.75	0.69	0.73	0.69	0.75
	R	0.82	0.86	0.80	0.81	0.82	0.78	0.77	0.78	0.77	0.84	0.77	0.80
	F1	0.80	0.79	0.77	0.76	0.77	0.76	0.74	0.76	0.73	0.78	0.73	0.72
	AUC	0.87	0.86	0.88	0.87	0.89	0.90	0.83	0.83	0.87	0.88	0.79	0.88
Lucene	P	0.56	0.58	0.74	0.65	0.57	0.61	0.65	0.61	0.68	0.64	0.72	0.73
	R	0.63	0.96	0.81	0.69	0.71	0.76	0.72	0.68	0.7	0.67	0.69	0.84
	F1	0.59	0.73	0.77	0.67	0.63	0.68	0.68	0.64	0.69	0.66	0.70	0.88
	AUC	0.55	0.84	0.79	0.84	0.71	0.63	0.83	0.71	0.64	0.81	0.83	0.87
Myllyn	P	0.55	0.60	0.61	0.53	0.52	0.56	0.61	0.56	0.61	0.61	0.60	0.73
	R	0.63	0.71	0.71	0.59	0.55	0.63	0.7	0.77	0.72	0.71	0.71	0.81
	F1	0.59	0.65	0.66	0.56	0.53	0.59	0.65	0.65	0.66	0.66	0.65	0.76
	AUC	0.54	0.79	0.85	0.80	0.76	0.74	0.82	0.68	0.83	0.85	0.81	0.89
PDE	P	0.57	0.60	0.58	0.57	0.56	0.60	0.61	0.58	0.59	0.59	0.62	0.68
	R	0.61	0.77	0.71	0.62	0.80	0.68	0.78	0.69	0.78	0.75	0.74	0.72
	F1	0.59	0.67	0.64	0.60	0.66	0.64	0.69	0.63	0.67	0.66	0.67	0.70
	AUC	0.60	0.76	0.77	0.73	0.79	0.75	0.77	0.67	0.76	0.74	0.76	0.80
Giá trị trung bình													
	P	0.63	0.64	0.67	0.64	0.62	0.63	0.64	0.62	0.63	0.66	0.65	0.71
	R	0.68	0.78	0.74	0.77	0.71	0.69	0.71	0.72	0.68	0.76	0.71	0.80
	F1	0.65	0.70	0.70	0.70	0.66	0.66	0.67	0.66	0.65	0.71	0.68	0.78
	AUC	0.72	0.82	0.82	0.84	0.80	0.78	0.79	0.70	0.75	0.82	0.80	0.86

Hình 3.2 trình bày hiệu suất AUC của các phương pháp lựa chọn đặc trưng wrapper được đề xuất. Trục x biểu thị tên các thuật toán lựa chọn đặc trưng, trong khi trục y thể hiện các giá trị AUC tương ứng của chúng. Theo biểu đồ, RFE, EO và PRO mang lại kết quả vượt trội trên nhiều bộ dữ liệu. Cụ thể, RFE đạt giá trị AUC cao nhất trên tập dữ liệu CM1, EQ, Lucence, Mylyn và PDE là 0.88, 0.86, 0.87, 0.89 và 0.80. Tiếp theo, EO đạt các giá trị AUC cao thứ hai là 0.78, 0.90, 0.84 và 0.75 trên các bộ dữ liệu KC1, EQ, Lucene và PDE. Tương tự, PRO đạt các giá trị AUC đáng chú ý là 0.88, 0.93 và 0.86 trên các bộ dữ liệu PC1, EQ và JDT. Để phân tích sâu hơn về hiệu suất tổng thể, giá trị AUC trung bình trên các bộ phân loại học máy khác nhau đã được tính toán. Tất cả các phương pháp lựa chọn đặc trưng được đánh giá đều đạt giá trị AUC trung bình trên 0.65. Trong đó, RFE kết hợp với RF đạt giá trị cao nhất là 0.90, EO kết hợp với RF đạt điểm trung bình cao thứ hai là 0.87, tiếp theo là PRO kết hợp với RF đạt 0.84. Ở chiều ngược lại, SMA kết hợp với ET đạt giá trị AUC trung bình thấp nhất, với điểm số là 0.65.

Các kết quả này cho thấy RFE là phương pháp lựa chọn đặc trưng wrapper đạt hiệu quả cao nhất cho dự đoán lỗi phần mềm, giúp cải thiện hiệu suất của các bộ phân loại một cách hiệu quả. EO, PRO và HGSO cũng thể hiện hiệu suất mạnh mẽ và ổn định trong nhiều trường hợp khác nhau. Bảng 2.6 trình bày kết quả xếp hạng độ quan trọng của các đặc trưng, đồng thời trình bày các đặc trưng được lựa chọn. Kết quả cho thấy nhóm đặc trưng liên quan đến kích thước mã nguồn và cấu trúc toán tử/toán hạng, bao gồm `IOComment`, `loc`, `IOBlank`, `v`, `uniq-Op`, `n`, `uniq-Opnd`, `total-Opnd` và `i`, đạt độ quan trọng cao nhất (từ 4.91% đến 7.53%) và đều được lựa chọn. Điều này cho thấy các đặc trưng này có ảnh hưởng đáng kể đến khả năng dự đoán lỗi. Ngược lại, các đặc trưng như `b`, `e`, `total-Op`, `t`, `branchCount` và các chỉ số độ phức tạp (`v(g)`, `iv(g)`, `ev(g)`) có độ quan trọng thấp hơn và không được lựa chọn. Đặc biệt, đặc trưng `locCodeAndComment` có giá trị độ quan trọng bằng 0%, cho thấy đóng góp không đáng kể trong việc phân biệt mô-đun có lỗi và không có lỗi. Điều này có thể từ tính dư thừa thông tin hoặc mức độ tương quan cao với các đặc trưng khác đã được chọn.

3. Kết quả kiểm định thống kê

Để so sánh hiệu suất của các phương pháp lựa chọn đặc trưng metaheuristic (ABO, ASO, EO, HGSO, PRO, GNDO, SMA, HHO, PFA, MRFO và RFE) với mô hình cơ sở, kiểm định Wilcoxon signed-rank được áp dụng với mức ý nghĩa $\alpha = 0.05$. Bảng 2.7 trình bày chi tiết kết quả kiểm định thống kê cho tất cả các mô hình học máy được đánh giá kết hợp với phương pháp chọn đặc trưng metaheuristic liên quan



Hình 2.3: So sánh các giá trị AUC của các phương pháp lựa chọn đặc trưng metaheuristic trên các bộ dữ liệu khác nhau

Bảng 2.6: So sánh độ quan trọng của các đặc trưng

Đặc trưng	Xếp hạng	Độ quan trọng của đặc trưng (%)	Được lựa chọn
i	1	7.53	Chọn
IOComment	1	7.40	Chọn
loc	1	6.85	Chọn
IOBlank	1	6.29	Chọn
v	1	5.90	Chọn
uniq_Op	1	5.66	Chọn
n	1	5.62	Chọn
uniq_Opnd	1	5.25	Chọn
total_Opnd	1	4.98	Chọn
t	1	4.91	Chọn
b	2	5.05	Không
d	3	4.67	Không
total_Op	4	4.77	Không
e	5	5.51	Không
branchCount	6	3.66	Không
IOCode	7	3.99	Không
v(g)	8	3.05	Không
iv(g)	9	4.05	Không
l	10	2.80	Không
ev(g)	11	2.09	Không
locCodeAndComment	12	0.00	Không

đến Precision, Recall, F1-score và AUC, bao gồm giá trị P -value, giá trị effect size r và chênh lệch giá trị trung bình. Dấu hoa thị (*) biểu thị sự khác biệt có ý nghĩa về mặt thống kê, tương ứng với P -value nhỏ hơn 0.05. Giá trị effect size r định lượng mức độ khác biệt hiệu suất giữa các phương pháp so sánh, trong khi chênh lệch trung bình cho biết phương pháp nào đạt điểm trung bình cao hơn. Theo Bảng 2.7, một số phương pháp lựa chọn đặc trưng thể hiện sự cải thiện F1-score đạt mức ý nghĩa thống kê so với phương pháp cơ sở. Ví dụ, ASO đạt cải thiện mang ý nghĩa thống kê với P -value là 0.026 và kích thước hiệu ứng lớn ($r = 0.913$), cùng với chênh lệch trung bình là 0.071. Tương tự, HGSO và PRO cũng đạt mức cải thiện đáng kể với P -value là 0.022 và 0.033, cùng effect size r là 0.520 và 0.561. Tổng thể, ASO, HGSO, PRO, PFA và MRFO được xem là các phương pháp lựa chọn đặc trưng vừa có ý nghĩa thống kê vừa có hiệu quả thực tiễn đối với mô hình Random Forest và nên được ưu tiên triển khai trong các mô hình dự đoán lỗi thực tế.

Bảng 2.7: Kết quả kiểm định thống kê so sánh giữa các phương pháp chọn đặc trưng metaheuristic và phương pháp cơ sở đối với các mô hình Random Forest, Extra Trees và AdaBoost

Nhóm so sánh	Độ đo	Random Forest				Extra Trees				AdaBoost			
		AUC	F1	P	R	AUC	F1	P	R	AUC	F1	P	R
ABO & Mô hình cơ sở	P-value	0.080	0.080	0.416	0.043*	0.080	0.080	0.416	0.043*	0.018*	0.052*	0.236	0.858
	Effect r	0.695	0.782	0.974	-0.062	0.695	0.782	0.974	-0.062	0.589	0.814	0.549	0.481
	Mean-difference	0.097	0.044	0.010	0.096	0.097	0.044	0.010	0.096	0.059	0.002	0.028	0.016
ASO & Mô hình cơ sở	P-value	0.013*	0.109	0.107	0.053*	0.013*	0.109	0.107	0.053*	0.018*	0.362	0.035*	0.590
	Effect r	0.256	0.585	0.714	0.184	0.256	0.585	0.714	0.184	-0.104	0.768	0.703	0.530
	Mean-difference	0.093	0.050	0.041	0.057	0.093	0.050	0.041	0.057	0.087	0.027	0.078	0.027
EO & Mô hình cơ sở	P-value	0.051*	0.293	0.574	0.050*	0.051*	0.293	0.574	0.050*	0.407	0.018*	0.068	0.123
	Effect r	0.205	0.540	0.435	0.525	0.205	0.540	0.435	0.525	-0.284	0.509	0.532	-0.010
	Mean-difference	0.114	0.040	0.012	0.083	0.114	0.040	0.012	0.083	0.028	0.096	0.066	0.102
HGSO & Mô hình cơ sở	P-value	0.208	0.022*	0.189	0.033*	0.208	0.022*	0.189	0.033*	0.011*	0.092	0.011*	0.528
	Effect r	0.446	0.520	0.765	0.123	0.446	0.520	0.765	0.123	0.226	0.401	0.776	-0.013
	Mean-difference	0.079	0.001	-0.012	0.024	0.079	0.001	-0.012	0.024	0.100	0.056	0.081	0.027
PRO & Mô hình cơ sở	P-value	0.284	0.033*	1.000	0.498	0.284	0.033*	1.000	0.498	0.155	0.205	0.207	0.905
	Effect r	0.487	0.561	0.837	0.165	0.487	0.561	0.837	0.165	0.042	0.597	0.677	0.488
	Mean-difference	0.054	0.004	0.001	0.009	0.054	0.004	0.001	0.009	0.050	0.021	0.037	0.011
GNDO & Mô hình cơ sở	P-value	0.372	0.045*	0.028*	0.062	0.372	0.045*	0.028*	0.062	0.050*	0.752	0.024*	0.401
	Effect r	-0.361	0.491	0.685	0.144	-0.361	0.491	0.685	0.144	-0.372	0.113	0.769	-0.210
	Mean-difference	0.062	0.018	0.011	0.026	0.062	0.018	0.011	0.026	0.067	0.003	0.051	0.051
SMA & Mô hình cơ sở	P-value	0.089	1.000	0.026*	0.041*	0.089	1.000	0.026*	0.041*	1.000	0.310	0.028*	0.575
	Effect r	-0.061	0.534	0.757	-0.063	-0.061	0.534	0.757	-0.063	0.199	0.314	0.745	0.326
	Mean-difference	-0.028	0.007	-0.010	0.031	-0.028	0.007	-0.010	0.031	0.001	0.024	0.048	0.000
HHO & Mô hình cơ sở	P-value	0.499	1.000	1.000	1.000	0.499	1.000	1.000	1.000	0.173	0.037*	0.017*	0.373
	Effect r	0.231	0.633	0.780	0.309	0.231	0.633	0.780	0.309	-0.198	0.700	0.900	0.552
	Mean-difference	0.028	-0.002	0.002	-0.007	0.028	-0.002	0.002	-0.007	0.048	0.019	0.054	0.030
PFA & Mô hình cơ sở	P-value	0.023*	0.035*	0.065	0.025*	0.023*	0.035*	0.065	0.025*	0.024*	0.405	0.017*	0.553
	Effect r	0.203	0.786	0.894	0.483	0.203	0.786	0.894	0.483	-0.133	0.686	0.886	0.413
	Mean-difference	0.092	0.052	0.034	0.079	0.092	0.052	0.034	0.079	0.078	0.014	0.059	0.033
MRFO & Mô hình cơ sở	P-value	0.207	0.035*	0.593	0.483	0.207	0.035*	0.593	0.483	0.013*	0.373	0.028*	0.514
	Effect r	-0.207	0.293	0.471	-0.076	-0.207	0.293	0.471	-0.076	0.426	0.671	0.784	0.398
	Mean-difference	0.081	0.021	0.022	0.021	0.081	0.021	0.022	0.021	0.080	0.018	0.059	0.024

2.4 Kết chương

Trong chương này, luận án đã nghiên cứu một cách hệ thống vai trò của các phương pháp lựa chọn đặc trưng trong việc nâng cao hiệu quả dự đoán lỗi phần mềm. Trên cơ sở tổng quan các nghiên cứu liên quan, luận án đã phân tích ưu điểm và hạn chế của ba nhóm phương pháp lựa chọn đặc trưng gồm filter, wrapper và embedded, đồng thời tập trung khai thác tiềm năng của các thuật toán tối ưu hóa metaheuristic trong khuôn khổ phương pháp wrapper nhằm tìm kiếm tập đặc trưng tối ưu.

Để đánh giá hiệu quả của các phương pháp đề xuất, luận án đã tiến hành thực nghiệm

trên chín bộ dữ liệu dự đoán lỗi phần mềm thuộc hai kho dữ liệu PROMISE và AEEEM, sử dụng ba mô hình học máy gồm Random Forest, Extra Trees và AdaBoost. Kết quả thực nghiệm cho thấy các phương pháp lựa chọn đặc trưng dựa trên metaheuristic có khả năng loại bỏ hiệu quả các đặc trưng dư thừa hoặc không liên quan, giảm chiều dữ liệu, đồng thời cải thiện các độ đo đánh giá như Precision, Recall, F1-score và AUC so với mô hình sử dụng toàn bộ tập đặc trưng. Kiểm định Wilcoxon signed-rank cũng xác nhận rằng những cải thiện này có ý nghĩa thống kê, qua đó khẳng định hiệu quả của các phương pháp lựa chọn đặc trưng được đề xuất trong việc nâng cao chất lượng dữ liệu đầu vào và hiệu suất của mô hình dự đoán lỗi phần mềm. Trên cơ sở các kết quả thực nghiệm và kiểm định thống kê, có thể kết luận rằng RFE là phương pháp lựa chọn đặc trưng đạt hiệu quả cao nhất trong số các phương pháp được khảo sát. Phương pháp này vừa đảm bảo khả năng giảm số chiều dữ liệu, vừa duy trì hoặc cải thiện hiệu suất của mô hình dự đoán trên nhiều bộ dữ liệu khác nhau. Do đó, RFE được lựa chọn làm phương pháp lựa chọn đặc trưng trong các nghiên cứu tiếp theo của luận án. Tuy nhiên, mặc dù RFE giúp tối ưu tập đặc trưng đầu vào, hiệu quả dự đoán vẫn chịu ảnh hưởng đáng kể bởi hiện tượng mất cân bằng dữ liệu. Vì vậy, Chương 3 sẽ tập trung nghiên cứu việc kết hợp RFE với các kỹ thuật xử lý mất cân bằng dữ liệu, đặc biệt là các phương pháp sinh mẫu dựa trên GAN, nhằm tối ưu hóa đồng thời cả không gian đặc trưng và phân bố dữ liệu, từ đó nâng cao khả năng phát hiện các mô-đun có lỗi và cải thiện khả năng tổng quát hóa của mô hình dự đoán lỗi phần mềm. Những kết quả đạt được trong chương này là cơ sở khoa học vững chắc để phát triển các mô hình dự đoán lỗi ở cấp độ sâu hơn, chẳng hạn như dự đoán ở mức phương thức hoặc dòng mã. Các kết quả nghiên cứu này đã được công bố trên kỷ yếu Hội nghị khoa học quốc tế trên Tạp chí International Journal of Electrical and Computer Engineering (IJECE).¹

¹Ha Thi Minh Phuong, Dang Thi Kim Ngan, Dao Khanh Duy, Nguyen Thanh Binh “**Enhancing software fault prediction using wrapper-based metaheuristic feature selection methods**”, *International Journal of Electrical and Computer Engineering (IJECE)*, No:5. Vol: 15. ISSN: 2088-8708. Pages: 4803-4812. Year: 2025. URL: <https://doi.org/10.11591/ijece.v15i5.pp4803-4812>

Chương 3

Kết hợp các kỹ thuật lựa chọn đặc trưng và lấy mẫu dữ liệu trong dự đoán lỗi phần mềm

Chương 2 đã trình bày các kỹ thuật lựa chọn đặc trưng để loại bỏ những đặc trưng dư thừa hoặc không liên quan đến lỗi, qua đó nâng cao chất lượng của mô hình dự đoán lỗi phần mềm. Chương này tập trung nghiên cứu các mô hình lấy mẫu dữ liệu nhằm giải quyết vấn đề mất cân bằng dữ liệu từ đó cải thiện độ chính xác và hiệu quả của mô hình dự đoán lỗi.

3.1 Giới thiệu

Để phát triển các mô hình dự đoán lỗi phần mềm, nhiều kỹ thuật học máy đã được nghiên cứu và cho thấy kết quả triển vọng trên các bộ dữ liệu lỗi phần mềm. Trong quá trình huấn luyện, mô hình dự đoán sẽ học các đặc trưng của các dự án phần mềm trước đó và đưa ra dự đoán liệu một mô-đun mới có khả năng bị lỗi hay không. Tuy nhiên, trong nhiều trường hợp, số lượng mô-đun có lỗi trong tập dữ liệu huấn luyện rất nhỏ so với số lượng mô-đun không có lỗi, dẫn đến vấn đề mất cân bằng dữ liệu. Điều này dẫn đến các mô hình dự đoán được xây dựng trên các bộ dữ liệu phần mềm mất cân bằng thường không thể học được các mẫu đặc trưng của các mô-đule lỗi và do đó có xu hướng dự đoán sai nhiều mô-đun lỗi [28]. Do đó, các mô hình dự đoán trở nên thiếu tin cậy và khó có thể được ứng dụng trong thực tế. Vì vậy, việc đề xuất các kỹ thuật hoặc phương pháp nhằm khắc phục vấn đề mất cân bằng lớp trong các bộ dữ liệu lỗi đã và đang được xem là một vấn đề quan trọng trong lĩnh vực dự đoán lỗi phần mềm. Nhiều nghiên cứu trước đây đã được công bố nhằm phát triển các phương pháp khác nhau để giải quyết vấn đề mất cân bằng lớp này [106].

Để khắc phục vấn đề mất cân bằng, có hai hướng tiếp cận chính trong lấy mẫu dữ

liệu, bao gồm tăng cường dữ liệu thiếu số (lấy mẫu tăng) và giảm dữ liệu thiếu số (lấy mẫu giảm). Lấy mẫu tăng tìm cách tạo ra các mẫu mới cho lớp thiếu số nhằm tăng số lượng mẫu và giúp cân bằng phân phối lớp. Các kỹ thuật lấy mẫu tăng phổ biến nhất trong học máy để xử lý mất cân bằng lớp bao gồm SMOTE và các biến thể, chẳng hạn như Borderline SMOTE và ADASYN. Tuy nhiên, các phương pháp lấy mẫu tăng này thường dẫn đến vấn đề quá khớp, do các mẫu tổng hợp được tạo ra có thể quá giống với các mẫu hiện có trong lớp thiếu số. Các kỹ thuật lấy mẫu giảm, ngược lại, giảm kích thước của lớp đa số bằng cách chọn ngẫu nhiên hoặc có chủ đích một tập con của các mẫu không có lỗi. Kỹ thuật lấy mẫu giảm phổ biến nhất là RUS, trong đó các mẫu thuộc lớp đa số được loại bỏ ngẫu nhiên. Tuy nhiên, do việc loại bỏ một phần dữ liệu, các thông tin quan trọng có thể bị mất, dẫn đến sự suy giảm hiệu suất của mô hình phân loại [77]. Ngoài hai phương pháp trên, các cách tiếp cận thay thế như các mô hình học kết hợp (ensemble learning) và học với chi phí nhạy cảm (cost-sensitive) cũng đã được nghiên cứu để xử lý vấn đề mất cân bằng dữ liệu trong dự đoán lỗi phần mềm.

Generative Adversarial Networks (GAN) [107] gần đây đã được áp dụng thành công để giải quyết vấn đề mất cân bằng dữ liệu trong nhiều lĩnh vực, bao gồm tạo ảnh và video, tăng cường dữ liệu và xử lý ngôn ngữ tự nhiên. Các kỹ thuật lấy mẫu lại hiện có như SMOTE chủ yếu dựa vào thông tin cục bộ (tức là các điểm dữ liệu lân cận) để tạo ra các mẫu dữ liệu tổng hợp. Tuy nhiên, phương pháp GAN có khả năng học được phân phối tổng thể của dữ liệu trên các đặc trưng khác nhau cũng như mối tương quan giữa các đặc trưng đó [108]. Nhờ vậy, GAN có thể tạo ra các mẫu dữ liệu tổng hợp có tính chân thực cao hơn so với các kỹ thuật truyền thống. GAN ban đầu được phát triển để tạo ra hình ảnh, dữ liệu dạng bảng và các loại dữ liệu khác [48]. Các phương pháp tiếp cận dựa trên GAN đã được chứng minh có khả năng học phân phối dữ liệu tốt hơn so với các kỹ thuật lấy mẫu dữ liệu truyền thống và giúp giải quyết hiệu quả vấn đề mất cân bằng dữ liệu [48].

Trong chương này, luận án tập trung vào việc đánh giá thử nghiệm các kỹ thuật tiềm năng của GAN và các biến thể của chúng để tạo ra tập dữ liệu lỗi phần mềm dạng bảng cho dự đoán lỗi phần mềm. Ngoài ba mô hình GAN bao gồm VanillaGAN, CTGAN và WGANP được đề cập trong [48], luận án giới thiệu thêm hai biến thể của GAN, cụ thể là CopulaGAN [109] và TGAN [110], nhằm tạo ra các mẫu tổng hợp để cân bằng giữa lớp thiếu số và lớp đa số trong các tập dữ liệu lỗi phần mềm. Phương pháp sinh dữ liệu hoạt động như sau: mô hình GAN bao gồm hai mạng nơ-ron huấn luyện theo cơ chế đối kháng. Bộ sinh (generator) có nhiệm vụ tạo ra dữ liệu gần giống với dữ liệu thực, sau đó dữ liệu này được đưa vào bộ phân biệt (discriminator). Ở mỗi vòng lặp, bộ phân biệt được huấn luyện để phân biệt giữa dữ liệu tổng hợp và dữ liệu gốc [111]. Mục tiêu là tạo ra các mẫu tổng hợp mà bộ phân biệt không thể phân biệt với dữ liệu thực. Các mẫu tổng hợp do phương pháp GAN tạo ra có sự tương đồng cao với phân phối dữ liệu của lớp thiếu số

ban đầu, giúp giải quyết hiệu quả vấn đề mất cân bằng dữ liệu.

Trong chương này, năm biến thể của mạng GAN được sử dụng để tạo ra các mẫu dữ liệu cân bằng. Năm tập dữ liệu lỗi khác nhau từ kho dữ liệu PROMISE đã được tiến hành phân tích thử nghiệm. Bên cạnh đó, sáu thuật toán học máy đã được áp dụng bao gồm Random Forest, eXtreme Gradient Boosting, Histogram-based Gradient Boosting, Multilayer Perceptron, Extra Trees và Adaptive Boosting để xây dựng các mô hình dự đoán. Các phương pháp này được đánh giá bởi các độ đo precision, recall, F1-score, AUC, và MCC. Ngoài ra, thực hiện kiểm định Wilcoxon với mức ý nghĩa thống kê $p < 0.05$ để đánh giá sự khác biệt có ý nghĩa giữa các biến thể của GAN và các phương pháp lấy mẫu lấy mẫu tăng truyền thống. Ngoài ra, hiệu suất giữa các mô hình GAN và các mô hình lấy mẫu cơ sở, bao gồm SMOTE, ADASYN, Borderline-SMOTE và RUS đã được thực hiện so sánh nhằm xác thực hiệu quả của việc tạo dữ liệu tổng hợp trong dự đoán lỗi phần mềm.

3.2 Cơ sở lý thuyết

3.2.1 Các nghiên cứu liên quan

Phần này tập trung vào một số nghiên cứu chính liên quan đến việc kết hợp các kỹ thuật lựa chọn đặc trưng và các kỹ thuật lấy mẫu dữ liệu nhằm nâng cao hiệu quả chất lượng của các mô hình dự đoán lỗi và đề xuất vấn đề nghiên cứu. Elahi và cộng sự [112] đã nghiên cứu và so sánh nhiều chiến lược ensemble nhằm nâng cao hiệu suất của mô hình dự báo. Các tác giả cũng sử dụng các phương pháp lấy mẫu khác nhau, bao gồm SMOTE, RUS và ROS để giải quyết vấn đề mất cân bằng lớp. Bên cạnh đó, các tác giả đã triển khai phương pháp trung bình hóa mô hình bằng cách sử dụng phương pháp bỏ phiếu tổ hợp (voting) và tổng hợp phân tầng (stacking) và kết quả cho thấy phương pháp trung bình hóa mô hình có hiệu suất vượt trội hơn so với các phương pháp khác. Joon và cộng sự [113] đã tối ưu hóa mô hình dự báo lỗi phần mềm bằng cách kết hợp các chiến lược loại bỏ nhiễu, điều chỉnh phân bố lớp mất cân bằng, và lựa chọn các độ đo phần mềm. Các tác giả đã sử dụng kỹ thuật SMOTETomek resampling trong nghiên cứu của mình. Pandey và Tripathi [114] đã thực hiện một nghiên cứu toàn diện gần đây về cách xử lý nhiễu và vấn đề mất cân bằng lớp trong dự báo lỗi phần mềm. Nghiên cứu đã so sánh hiệu suất của các kỹ thuật RUS, SMOTE, Class Balancer và Spread Sub-Sampling và nhận thấy rằng RUS đạt kết quả tốt nhất. Tổng quan các công trình nghiên cứu trước đây cho thấy nhiều hướng tiếp cận khác nhau đã được đề xuất nhằm cải thiện hiệu quả của các mô hình dự đoán lỗi trên dữ liệu mất cân bằng. Bảng 3.1 tổng hợp các nghiên cứu đã giải quyết vấn đề mất cân bằng dữ

Bảng 3.1: Một số nghiên cứu liên quan đến giải quyết vấn đề mất cân bằng lớp trong dự đoán lỗi phần mềm

Tác giả	Tập dữ liệu	Phương pháp thực hiện	Độ đo	Kết quả chính
Malhotra và cộng sự [115]	NASA	Kỹ thuật lấy mẫu: SMOTE, Adaptive Synthetic sampling technique, Safe-Level-SMOTE, Selective Preprocessing of Imbalanced Data (SPIDER), và SPIDER2, SPIDER3. Bộ phân loại: J48, Random Forest, Naïve Bayes, Adaboost, Bagging	ALOC cyclomatic complexity	Phân tích các kỹ thuật lấy mẫu
Joon và cộng sự [113]	NASA, PROMISE	SMOTETomek Simple majority, voting Logistic regression, random forest, SVM, naïve Bayes Lựa chọn đặc trưng: CFS	LOC cyclomatic complexity	So sánh hiệu suất của các kỹ thuật lấy mẫu khác và các kỹ thuật lựa chọn đặc trưng mã nguồn
Balogun và cộng sự [116]	NASA	Lấy mẫu dữ liệu: SMOTE Bộ phân loại: Bayesian Networks và Decision Tree	LOC	thử nghiệm được thực hiện trên các tập dữ liệu vẫn còn giới hạn
Aziz và cộng sự [117]	PROMISE	ANN	CK	Các tập dữ liệu chứa các đặc trưng dư thừa
Tantithamthavorn và cộng sự [60]	101 tập dữ liệu mã nguồn mở	Lấy mẫu dữ liệu: Over-sampling, Under sampling, SMOTE và Bootstrap Random Over-Sampling	CK	Phân tích so sánh tác động của các kỹ thuật chọn đặc trưng khác nhau đến hiệu suất của mô hình lỗi phần mềm.
Turabieh và cộng sự [94]	19 tập dữ liệu từ các dự án thực tế trích xuất từ kho PROMISE	Binary Genetic Algorithm (BGA), Binary Particle Swarm Optimization (BPSO), Binary Ant Colony Optimization (BACO), layered recurrent neural network (L-RNN)	CK	So sánh hiệu suất của các bộ phân loại khác nhau và giải quyết vấn đề mất cân bằng lớp
Tong và cộng sự [118]	AEEEM MetricsRepo	Lấy mẫu dữ liệu: Random under sampling, SMOTER Bộ phân loại: DT Regressor	CK	Trung bình, SHSE cải thiện hiệu suất so với các phương pháp tái lấy mẫu từ 8,7% đến 14,4% và so với các mô hình zero-inflate/hurdle từ 10,3% đến 15,2%.
Nitin và cộng sự [119]	PROMISE ECLIPSE	Lấy mẫu dữ liệu: Random forest, bagging, random subspace, và boosting Bộ phân loại: Decision Trees, Logistic Regression, and K-nearest Neighbors	CK	Các phương pháp dựa trên ensemble đạt hiệu suất đáng kể trong dự đoán lỗi phần mềm nhờ hiệu suất cải thiện so với các kỹ thuật dự đoán lỗi đơn lẻ.
Khuat [120]	PROMISE	Support vector machines (SVM), multilayer perceptron (MLP), Bayesian networks, K-nearest neighbor (KNN), and decision tree J48	Lines Of Code-based measures McCabe Halstead F-score	So sánh hiệu suất với các kỹ thuật lấy mẫu dữ liệu khác và giải quyết vấn đề mất cân bằng lớp
Rathore và cộng sự [48]	PROMISE JIRA Eclipse	VanillaGAN, CTGAN, WGANGP, SMOTE, ADASYN, ROS, RUS, Borderline_SMOTE, và AdaBoost	CK	Hiệu suất của VanillaGAN và CTGAN và WGANGP được cải thiện và có ý nghĩa thống kê hơn so với các kỹ thuật lấy mẫu khác.
Rathi và cộng sự [49]	54 dự án mã nguồn mở	Lấy mẫu dữ liệu: Synthetic Minority Over Sampling Technique (SMOTE), Borderline SMOTE (BSMOTE), SVM-SMOTE (SSMOTE), SMOTE-Edited (SMOTEE), USAM, SMOTE with Tomek's link (SMOTETO), RSAM, and DSAM	CK	Nghiên cứu này hỗ trợ việc áp dụng kỹ thuật lấy mẫu SMOTEE, phương pháp chọn đặc trưng dựa trên tương quan (Correlation-based feature selection) và phương pháp chọn đặc trưng dựa trên kiểm định Chi-squared để phát triển mô hình dự đoán lỗi hiệu quả.

liệu để phát triển mô hình dự đoán lỗi phần mềm.

Nhìn chung, các nghiên cứu hiện nay đều cho thấy rằng việc kết hợp nhiều kỹ thuật tiền xử lý, bao gồm lựa chọn đặc trưng, xử lý nhiễu và cân bằng dữ liệu, có thể cải thiện đáng kể hiệu quả của các mô hình dự đoán lỗi phần mềm. Tuy nhiên, phần lớn các nghiên cứu vẫn chủ yếu dựa trên các kỹ thuật lấy mẫu truyền thống như Random Over Sampling, Random Under Sampling, SMOTE hoặc các biến thể của SMOTE. Mặc dù các phương pháp này góp phần cải thiện sự cân bằng giữa các lớp, chúng vẫn tồn tại những hạn chế như tạo ra các mẫu nội suy có tính đa dạng thấp, dễ gây hiện tượng quá khớp hoặc làm

Bảng 3.2: Một số nghiên cứu liên quan đến giải quyết vấn đề mất cân bằng lớp trong dự đoán lỗi phần mềm (tt)

Tác giả	Tập dữ liệu	Phương pháp thực hiện	Độ đo	Kết quả chính
Shuo Feng [121]	PROMISE	Complexity-based OverSampling Technique SMOTE, Borderline-SMOTE, MWMOTE và MAHAKIL Bộ phân loại: SVM, KNN, RF and MLP	Static Metric	COSTE vượt trội hơn so với các kỹ thuật lấy mẫu lấy mẫu tăng khác. Phân tích thống kê cho thấy khả năng vượt trội của COSTE so với các kỹ thuật lấy mẫu lấy mẫu tăng khác là có ý nghĩa thống kê theo kiểm định Wilcoxon rank sum và Cliff. Mô hình đề xuất vượt trội hơn so với các mô hình dự đoán lỗi khác với điểm AUC cao nhất (= 95,6%), giá trị Độ chính xác tối đa (= 96,9%) và đường cong ROC gần góc trên bên trái nhất. Về mặt thống kê, mô hình này đạt được khả năng dự đoán tốt nhất với mức độ tin cậy là 95%.
Goyal [122]	NASA	Neighbourhood based Under-Sampling (N-US)	CK	BOA khắc phục vấn đề quá khớp, trong khi ADASYN giải quyết vấn đề mất cân bằng lớp dữ liệu cho các lớp thiểu số, qua đó thực hiện quá trình biến đổi dữ liệu một cách đồng đều.
A. Balaram1 và S. Vasundra [106]	PROMISE	Lựa chọn đặc trưng: Butterfly Optimization Algorithm (BOA) Lấy mẫu dữ liệu: Ensemble Random Forest with Adaptive Synthetic Sampling (E-RF-ADASYN)	CK	Sự kết hợp giữa CTGAN với RFE và cặp CTGAN với Autoencoders vượt trội hơn so với các mô hình cơ sở khác trên tất cả các tập dữ liệu, tiếp theo là WGANGP và VanillaGAN.
Ha [123]	PROMISE	Lựa chọn đặc trưng: Recursive Feature Elimination (RFE) Trích xuất đặc trưng: Autocncoders Lấy mẫu dữ liệu: CopulaGAN, VanillaGAN, CTGAN, TGAN và WGANG	CK	Theo phân tích so sánh, các phương pháp lấy mẫu quá mức dựa trên GAN đã cải thiện đáng kể khả năng xử lý vấn đề mất cân bằng dữ liệu trong dự đoán lỗi phần mềm.

mất thông tin của lớp đa số. Bên cạnh đó, việc kết hợp lựa chọn đặc trưng với các kỹ thuật cân bằng dữ liệu còn chưa được nghiên cứu một cách hệ thống. Phần lớn các công trình chỉ xem lựa chọn đặc trưng và cân bằng dữ liệu là hai bước tiền xử lý độc lập, trong khi chưa đánh giá đầy đủ ảnh hưởng của từng phương pháp lựa chọn đặc trưng đối với hiệu quả của các kỹ thuật sinh dữ liệu. Bên cạnh các phương pháp lấy mẫu truyền thống, xu hướng gần đây còn mở rộng sang các phương pháp sinh dữ liệu để cân bằng lớp, nhằm tạo ra các mẫu mới có tính đa dạng và phản ánh tốt hơn phân phối của lớp thiểu số. Tuy nhiên, việc kết hợp có hệ thống các phương pháp lựa chọn đặc trưng tối ưu với các mô hình sinh dữ liệu tiên tiến như mạng sinh đối kháng GAN trong bài toán dự đoán lỗi phần mềm vẫn còn hạn chế, và đây cũng là hướng nghiên cứu mà luận án tập trung khai thác.

3.2.2 Đánh giá hiệu quả của các kỹ thuật lấy mẫu dữ liệu

Các bộ dữ liệu lỗi phần mềm được sử dụng trong nghiên cứu này có dạng bảng, bao gồm nhiều đặc trưng là các biến độc lập và một biến phụ thuộc là nhãn dữ liệu. Việc tạo ra các mẫu dữ liệu tổng hợp mới từ các bộ dữ liệu dạng bảng như vậy là một thách thức, do các đặc trưng có thể tuân theo nhiều phân phối dữ liệu khác nhau. Do đó, việc học và hiểu rõ các phân phối này để sinh ra các mẫu dữ liệu mới không phải là một nhiệm vụ đơn giản.

Các kỹ thuật lấy mẫu tăng tạo ra các mẫu lớp thiểu số tổng hợp mới nhằm tăng số lượng của lớp thiểu số. Hầu hết các kỹ thuật lấy mẫu tăng (oversampling) hiện có sử dụng nhiều chiến lược khác nhau, chẳng hạn như các thước đo khoảng cách, để xác định các mẫu lân cận và nội suy giữa chúng nhằm tạo ra các mẫu dữ liệu tổng hợp mới. Tuy nhiên, hệ quả là các mẫu được sinh ra thường có mức độ đa dạng thấp và dễ bị khái quát hóa quá mức. Ngược lại, các kỹ thuật lấy mẫu giảm loại bỏ một số mẫu của lớp đa số để cân bằng tỷ lệ giữa mẫu lớp thiểu số và đa số [29]. Cả hai nhóm kỹ thuật này đều nhằm mục đích cân bằng sự phân bố của lớp thiểu số và đa số nhằm cải thiện hiệu suất của mô hình dự đoán. Trong dự đoán lỗi phần mềm, các kỹ thuật lấy mẫu tăng thường được sử dụng nhiều hơn so với các kỹ thuật lấy mẫu giảm, vì việc loại bỏ tùy ý các mẫu khỏi tập dữ liệu có thể dẫn đến mất mát thông tin quan trọng [48]. Các kỹ thuật lấy mẫu tăng như SMOTE và các biến thể (ví dụ: ADASYN, Borderline-SMOTE) cùng với kỹ thuật ROS được sử dụng phổ biến. Kỹ thuật ROS chỉ đơn giản sao chép các mẫu của lớp thiểu số để cân bằng tập dữ liệu mà không bổ sung bất kỳ thông tin mới nào [30]. Điều này có thể dẫn đến hiện tượng quá khớp trong mô hình dự đoán, khiến mô hình hoạt động kém hiệu quả đối với các mẫu kiểm thử chưa từng thấy trước đó. Các kỹ thuật dựa trên SMOTE tạo ra các mẫu của lớp thiểu số tổng hợp bằng cách nội suy các mẫu lân cận đã được xác định (dựa trên khoảng cách) để cân bằng tập dữ liệu [124]. Các mẫu mới này không phải là bản sao như trong ROS. Tuy nhiên, do phụ thuộc vào các mẫu lân cận, các kỹ thuật dựa trên SMOTE không thể tạo ra các mẫu trong lớp thiểu số đa dạng, điều này có thể dẫn đến hiện tượng quá khớp làm giảm hiệu suất trên tập kiểm thử. Ngoài ra, các kỹ thuật dựa trên chi phí [125] và các kỹ thuật dựa trên mô hình ensemble [126] cũng đã được sử dụng trong bài toán học mất cân bằng trong dự đoán lỗi phần mềm. Tuy nhiên, Galar và cộng sự [127] phát hiện rằng mặc dù các tập hợp như bagging và boosting giúp cải thiện độ chính xác của mô hình dự đoán lỗi, nhưng chúng không được thiết kế để xử lý các tập dữ liệu mất cân bằng. Từ các nghiên cứu liên quan trên, chúng ta có thể thấy việc tạo ra các mẫu dữ liệu tổng hợp mới từ tập dữ liệu dạng bảng này là một thách thức vì các đặc trưng khác nhau có thể tuân theo các phân phối dữ liệu khác nhau. GAN là một trong những phương pháp tạo dữ liệu tổng hợp đang nổi lên và đầy hứa hẹn nhất [128]. Từ những cơ sở nêu trên, luận án này lựa chọn tiếp cận phương pháp học mất cân bằng dựa trên GAN đối với dữ liệu lỗi phần mềm.

3.3 Các mô hình GAN lấy mẫu dữ liệu trong dự đoán lỗi phần mềm

3.3.1 Mô hình GAN

GAN được Goodfellow và cộng sự [129] giới thiệu vào năm 2014 và được định nghĩa là một trò chơi giữa hai mạng nơ-ron thường được gọi là bộ sinh và bộ phân biệt. Bộ sinh cố gắng học phân phối tỷ lệ của dữ liệu thực và tạo ra các mẫu mới có thể đánh lừa bộ phân biệt. Bộ phân biệt nhằm phân loại các mẫu thực và mẫu sinh tạo một cách chính xác nhất có thể. Cả bộ sinh và bộ phân biệt đều được huấn luyện đồng thời và cũng hoạt động trong cạnh tranh với nhau trong quá trình huấn luyện. Mục tiêu của việc huấn luyện GAN là tìm kiếm cân bằng Nash [130], là trạng thái mà cả hai mạng có thể đạt được hiệu suất tối ưu. Cân bằng Nash trong GAN đạt được khi bộ sinh tạo ra dữ liệu giả không thể phân biệt được với dữ liệu thực và bộ phân biệt không thể cải thiện khả năng phân biệt giữa mẫu thực và do bộ sinh tạo ra. Tại thời điểm này, GAN đã nắm bắt được phân phối của dữ liệu thực và có thể tạo ra các mẫu dữ liệu mới tương tự như dữ liệu thực. GAN đã được áp dụng thành công trong nhiều lĩnh vực và bài toán phức tạp, chẳng hạn như xử lý ngôn ngữ tự nhiên, phân đoạn ngữ nghĩa, thị giác máy tính và phân loại. Zhu và cộng sự [131] đã đề xuất GANCoder để tạo sinh mã ngôn ngữ lập trình có logic và chức năng từ ngôn ngữ tự nhiên. Kết quả của họ cho thấy GANCoder đã vượt trội hơn so với các kỹ thuật tiên tiến khác và cải thiện chất lượng tạo sinh mã nguồn.

3.3.2 Mô hình GAN trong dự đoán lỗi phần mềm

Nhiều nghiên cứu gần đây đã sử dụng các biến thể của GAN để giải quyết vấn đề mất cân bằng dữ liệu trong dự báo lỗi phần mềm. Sun và cộng sự [132] đã chọn Variational Autoencoder (VAE) và GAN để tạo ra các mẫu giả nhằm cân bằng phân phối dữ liệu. SMOTE được áp dụng để thực hiện so sánh với VAE và GAN. Kết quả thử nghiệm chỉ ra rằng cả VAE và GAN đều hiệu quả trong việc xử lý các vấn đề mất cân bằng dữ liệu cho dự đoán lỗi phần mềm. Ying Xing và cộng sự [133] đã đề xuất việc tích hợp GAN và BiLSTM để nắm bắt các đặc trưng ngữ cảnh và thông tin ngữ nghĩa của chương trình. Cụ thể, mô hình GAN được sử dụng để giảm sự khác biệt giữa phân phối dữ liệu của các dự án nguồn và mục tiêu. Kết quả thử nghiệm cho thấy mô hình của họ đạt được hiệu suất tốt hơn so với một số phương pháp dự đoán lỗi dựa trên giá trị AUC và Accuracy. Song và cộng sự [134] đã đề xuất DP-GANPT, sử dụng mô hình GAN và một mô hình mã hai chiều đã được huấn luyện trước cho dự đoán lỗi phần mềm. DP-GANPT đồng thời sử dụng một bộ sinh và Autoencoders đã được huấn luyện trước như hai bộ mã hóa, với bộ phân biệt

hoạt động như một bộ giải mã cho phân loại. Họ đã tiến hành xác thực thử nghiệm trên cả trong cùng dự án và giữa các dự án. Kết quả cho thấy DP-GANPT đạt hiệu suất cao hơn so với các mô hình cơ bản. Zhu và cộng sự [135] đã trình bày BL-GAN, trong đó các bản ghi sửa lỗi giả tương tự như các bản ghi thực được tạo ra bằng cách tìm kiếm cây thư mục dự án để tạo đường dẫn tệp, thay vì duyệt qua nội dung của tất cả các tệp mã. Mô hình BL-GAN đạt hiệu suất vượt trội trên tất cả các chỉ số đánh giá so với các phương pháp hiện có, dựa trên các thử nghiệm quy mô lớn trong thực tế.

Trong luận án này quan tâm đến việc áp dụng các mô hình GAN để giải quyết vấn đề mất cân bằng dữ liệu trong dự đoán lỗi phần mềm. Do đó, luận án đã thực hiện phân tích so sánh năm mô hình GAN, được mô tả chi tiết trong các mục dưới đây.

1. VanillaGAN

VanillaGAN [129] là phiên bản đầu tiên của GAN được công bố. Kiến trúc mô hình GAN bao gồm hai mô hình con: bộ sinh và bộ phân biệt. Được cung cấp một biến nhiễu z làm đầu vào, bộ sinh G tạo ra các mẫu gần với dữ liệu mục tiêu thực. Bộ phân biệt D cố gắng phân biệt xem các mẫu đầu vào đến từ bộ sinh G hay từ dữ liệu thực. Mục tiêu min-max có thể được mô tả như sau:

$$\min_{(G)} \max_{(D)} (E_{x \sim p_D} [\log(D(x))] + E_{z \sim p_z} [\log(1 - D(G(z)))] \quad (3.1)$$

Trong đó, x là dữ liệu đầu vào, p_D và p_z lần lượt đại diện cho phân phối của dữ liệu thực và dữ liệu được tạo ra, $z \sim p_z$ là nhiễu mẫu từ phân phối Gaussian cầu hoặc phân phối đồng nhất. Mục tiêu này dựa trên hàm mất mát Binary-Cross Entropy. Đầu ra của bộ phân biệt D nằm trong khoảng $[0,1]$ bằng cách sử dụng hàm kích hoạt sigmoid. Để học các tham số của GAN, bộ phân biệt D cần được huấn luyện cho đến khi đạt được độ chính xác tối đa trong việc phân biệt dữ liệu đầu vào từ dữ liệu được tạo ra $G(z)$ hoặc dữ liệu thực. Bộ sinh được huấn luyện để đạt được giá trị tối thiểu của $\log(1 - D(G(z)))$. Quá trình huấn luyện được thực hiện luân phiên cho đến khi đạt được giải pháp toàn cục trong trường hợp khi dữ liệu được sinh ra giống dữ liệu thật mà bộ phân biệt không phát hiện được. Mô hình VanillaGAN sử dụng một phương trình toán học đơn giản kết hợp với kỹ thuật hướng điều chỉnh các tham số của mô hình sao cho giá trị của hàm mất mát đạt mức thấp nhất có thể (gradient descent). Mặc dù mô hình VanillaGAN đã đạt được kết quả tiên tiến trong việc xử lý vấn đề mất cân bằng dữ liệu, nhưng mô hình này cũng có một số nhược điểm như sau:

- (a) Cân bằng Nash khó đạt được. Việc huấn luyện GAN dựa vào gradient descent, trong đó bộ sinh và bộ phân biệt được huấn luyện đồng thời để đạt được một

cân bằng Nash. Tuy nhiên, vì cả hai mô hình đều cập nhật hàm mất mát một cách đồng thời và độc lập, sự hội tụ không được đảm bảo [136].

- (b) Vấn đề gradient. Trong quá trình huấn luyện của VanillaGAN, các gradient có thể trở nên cực kỳ nhỏ, làm cho bộ sinh khó khăn trong việc cập nhật trọng số một cách hiệu quả.
- (c) Vấn đề Mode Collapse. VanillaGAN thường gặp phải hiện tượng thu gọn chế độ, trong đó bộ sinh lặp đi lặp lại tạo ra các đầu ra giống nhau. Sự thiếu đa dạng này không thể nắm bắt toàn bộ phạm vi phân phối dữ liệu, do đó hạn chế hiệu quả của GAN trong việc đại diện cho sự phức tạp của dữ liệu trong thế giới thực.

2. WGANP

WGANP (Wasserstein Progressive Growing of Generative Adversarial Networks) là một mô hình GAN, được giới thiệu bởi Karras và cộng sự [137] vào năm 2017. Bằng cách áp dụng một hàm mục tiêu ổn định hơn, khoảng cách Wasserstein, làm nền tảng cho việc huấn luyện, mô hình này nhằm vượt qua những khó khăn về sự không ổn định và sự thu gọn chế độ vốn có trong các GAN truyền thống. Tuy nhiên, trong công thức WGAN gốc, gradient của hàm mất mát của bộ phân biệt có thể gặp phải vấn đề gradient biến mất, trong đó gradient trở nên rất nhỏ và gây ra sự hội tụ chậm hoặc thậm chí dừng lại quá trình huấn luyện. Do đó, bắt đầu với độ phân giải thấp và dần dần mở rộng kích thước đầu vào của mạng bộ sinh, mô hình WGANP được huấn luyện bằng cách tăng dần chỉ cho hình ảnh được tạo ra.

Kiến trúc WGANP có thể được biểu diễn toán học như một trò chơi minimax hai người chơi giữa bộ phân biệt D và bộ sinh G , với mục tiêu là tìm kiếm cân bằng Nash giữa hai mạng này. Dưới đây là biểu thức của hàm mục tiêu để huấn luyện mô hình WGANP:

$$\min_G \max_D (\mathbb{E}_{x \sim p_D} [D(x)] - \mathbb{E}_{z \sim p_Z} [D(G(z))]) - \lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}} [(|| \nabla_{\hat{x}} D(\hat{x}) ||_2 - 1)^2] \quad (3.2)$$

Trong đó, x là một mẫu từ phân phối dữ liệu thực p_D , z là một vec-tơ nhiễu ngẫu nhiên, G là mạng bộ sinh, D là mạng phân biệt và λ là hệ số phạt [138]. WGANP giải quyết các hạn chế của cả VanillaGAN và WGAN [139], như sau:

- (a) Do có thành phần phạt trong hàm mất mát, điều này làm cho tất cả các chuẩn gradient hướng về giá trị 1 trong quá trình tối ưu hóa bộ phân biệt, từ đó đạt được chuẩn gradient đơn vị trong cả phân phối dữ liệu thực và giả. Điều này làm tăng tốc sự hội tụ của WGANP.

- (b) Trong quá trình huấn luyện, WGANP có thể giải quyết các vấn đề không ổn định của VanillaGAN và WGAN, đồng thời tạo ra các mẫu có chất lượng cao hơn.

3. CTGAN

CTGAN (Conditional GAN) [140] là một biến thể của GAN dùng để mô hình hóa phân phối dữ liệu bảng. Để xử lý các cột chứa phân phối không phải Gaussian và đa chế độ, CTGAN thay thế chuẩn hóa min-max bằng cách sử dụng chuẩn hóa theo chế độ cụ thể. Trong chuẩn hóa theo chế độ cụ thể, các giá trị liên tục được chuyển đổi thành các vec-tơ giới hạn, đây là các đầu vào phù hợp cho mạng nơ-ron. CTGAN áp dụng mô hình Variational Gaussian Mixture [141] để dự đoán số lượng chế độ và điều chỉnh một hỗn hợp Gaussian cho mỗi cột liên tục. Hỗn hợp Gaussian học được định nghĩa như sau:

$$\mathbb{P}_{c_i}(c_{i,j}) = \sum_{k=1}^l \mu_k \mathbb{N}(c_{i,j}; \eta_k, k) \quad (3.3)$$

trong đó, μ_k là trọng số size của k là độ lệch chuẩn của một chế độ, trong đó xác suất ρ_k của $c_{i,j}$ được tính toán cho mỗi chế độ. Việc tính toán mật độ xác suất được định nghĩa như sau:

$$\rho_k = \mu_k \mathbb{N}(c_{i,j}; \eta_k, k) \quad (3.4)$$

Một chế độ sau đó được lấy mẫu từ mật độ xác suất đã cho và chế độ được lấy mẫu sẽ được sử dụng để chuẩn hóa giá trị. $c_{i,j}$ được biểu diễn bằng một vec-tơ one-hot $\beta_{i,j}$ có kích thước bằng số lượng chế độ, và một giá trị vô hướng $\alpha_{i,j}$ được sử dụng để biểu diễn giá trị trong chế độ đó. Do đó, một hàng trong tập dữ liệu là sự nối kết giữa các cột liên tục và rời rạc:

$$r_j = \alpha_{1,j} \oplus \beta_{1,j} \oplus \dots \oplus \alpha_{N_c,j} \beta_{N_c,j} \oplus d_{1,j} \oplus \dots \oplus d_{N_d,j} \quad (3.5)$$

trong đó $d_{i,j}$ là một one-hot vector của các giá trị rời rạc.

Để giải quyết vấn đề mất cân bằng dữ liệu trong mô hình phân biệt, CTGAN triển khai bộ sinh có điều kiện và phương pháp huấn luyện qua lấy mẫu. Cụ thể, dữ liệu được lấy mẫu lại một cách hiệu quả sao cho tất cả các danh mục của các biến rời rạc đều được lấy mẫu trong quá trình huấn luyện và dữ liệu thực sẽ được khôi phục trong giai đoạn kiểm tra.

Để giải quyết thách thức dữ liệu mất cân bằng trong các cột rời rạc, mô hình CTGAN

sử dụng một bộ sinh có điều kiện kết hợp với phương pháp huấn luyện qua lấy mẫu [140]. Phương pháp này đảm bảo rằng tất cả các danh mục trong các cột rời rạc đều được lấy mẫu một cách đồng đều trong quá trình huấn luyện, thúc đẩy quá trình học tập cân bằng hơn. Tuy nhiên, CTGAN yêu cầu một lượng lớn dữ liệu bảng thực tế để huấn luyện mô hình và tạo ra dữ liệu tổng hợp có các đặc tính thống kê tương tự như dữ liệu thực.

4. CopulaGAN

Một biến thể của CTGAN là CopulaGAN [109] sử dụng phép biến đổi dựa trên hàm phân phối tích lũy (Cumulative Distribution Function - CDF-based) [142], được định nghĩa như sau:

$$F_x(x) = P[X \leq x] = \int_{-\infty}^x f_X(u) du \quad (3.6)$$

Trong đó, $F_x(x)$ chỉ ra phân phối xác suất của một biến ngẫu nhiên X sẽ đạt được giá trị bằng hoặc nhỏ hơn x . Trong CopulaGAN, các kiểu dữ liệu và định dạng dữ liệu huấn luyện sẽ được học trong suốt quá trình huấn luyện. Dữ liệu có giá trị null hoặc không phải số sẽ được chuyển đổi bằng cách sử dụng phép biến đổi dữ liệu có thể đảo ngược (Reversible Data Transformation - RDT) [143]. Các phân phối xác suất của từng cột trong bảng dữ liệu sẽ được học bởi RDT để tạo ra một bảng dữ liệu hoàn toàn số.

CopulaGAN xây dựng dựa trên CTGAN bằng cách tận dụng khả năng của Copula Gaussian trong việc chuyển đổi dữ liệu sử dụng các hàm phân phối tích lũy. Phép biến đổi này đơn giản hóa quá trình học cho CTGAN, giúp CTGAN nắm bắt cấu trúc dữ liệu cơ bản một cách hiệu quả hơn.

5. TGAN

Xu và Veeramachaneni [110] đã đề xuất mô hình TGAN (Tabular Generative Adversarial Networks) để tổng hợp các mẫu mới cho dữ liệu bảng. Trong giai đoạn chuyển đổi dữ liệu, một biến số được chuyển đổi thành phân phối đa thức và một phạm vi vô hướng $[-1, 1]$ bằng cách áp dụng mô hình hỗn hợp Gaussian và một biến rời rạc D được chuyển đổi thành phân phối đa thức. Trong mô hình bộ sinh, mỗi biến số được sản xuất theo hai bước. Bộ sinh đầu tiên sản xuất giá trị vô hướng v và sau đó sản xuất vec-tơ cụm U . Mô hình bộ sinh sử dụng LSTM để tạo ra dữ liệu tổng hợp. Đầu vào của LSTM trong mỗi bước là một vec-tơ nhúng, một biến ngẫu nhiên z và vec-tơ ẩn trước đó. Kết quả của bộ sinh là các chuyển đổi biến số vô hướng, được sử dụng làm đầu vào cho mô hình phân biệt [144].

Trong bộ phân biệt, TGAN sử dụng mạng nơ-ron Multilayer Perceptron (MLP) đa lớp kết nối đầy đủ với chuẩn hóa theo lô và hàm kích hoạt Leaky-Relu [110]. Đầu vào cho lớp đầu tiên trong bộ phân biệt là một vec-tơ kết hợp giá trị vô hướng V , vec-tơ cụm U và d , là đầu ra của biến rời rạc được tính toán bằng Softmax. Để làm cho mô hình hiệu quả hơn, một hàm mất mát được thêm vào các biến rời rạc và vec-tơ cụm của các vec-tơ liên tục. Công thức cho hàm mất mát trong bộ sinh như sau:

$$L_G = -\mathbb{E}_{z \sim \mathcal{N}(0,1)} \log D(G(z)) + \sum_{i=1}^{n_c} KL(u'_i, u_i) + \sum_{i=1}^{n_d} KL(d'_i, d_i) \quad (3.7)$$

trong đó, dữ liệu được sinh ra là u'_i và d'_i , và dữ liệu thực là u_i và d_i . TGAN sử dụng hàm mất mát Cross-Entropy để tối ưu hóa bộ phân biệt:

$$L_D = -\mathbb{E}_{v_{1:n_c}, u_{1:n_c}, d_{1:n_d} \sim \mathbb{P}(T)} \log D(v_{1:n_c}, u_{1:n_c}, d_{1:n_d}) + \mathbb{E}_{z \sim \mathcal{N}(0,1)} \log D(G(z)) \quad (3.8)$$

Trong mô hình này, theo kiến trúc của Xu và Veeramachaneni [110], mạng LSTM được sử dụng làm bộ sinh và MLP được sử dụng trong bộ phân biệt. Mô hình được huấn luyện bằng cách sử dụng bộ tối ưu Adam. Mặc dù TGAN cung cấp một loạt các ứng dụng, bao gồm mở rộng dữ liệu để huấn luyện mô hình, điền giá trị thiếu trong các tập dữ liệu và tạo dữ liệu kiểm thử/xác thực, nhưng TGAN cũng gặp phải những hạn chế như rủi ro tạo ra dữ liệu thiên lệch hoặc không thực tế nếu dữ liệu huấn luyện không đại diện cho quần thể thực tế.

3.3.3 Vấn đề nghiên cứu

Mặc dù các kỹ thuật học máy đã đạt được nhiều thành công trong việc xây dựng mô hình dự đoán lỗi so với các phương pháp truyền thống, nhưng sự xuất hiện của các đặc trưng không liên quan và sự mất cân bằng giữa các lớp trong tập dữ liệu đang trở thành những thách thức ảnh hưởng đến hiệu quả của các mô hình dự đoán lỗi. Vì vậy, việc xác định các độ đo đặc trưng quan trọng và xử lý vấn đề mất cân bằng lớp sẽ là trọng tâm trong việc xây dựng một mô hình phân loại hiệu quả cho dự đoán lỗi dựa trên các đặc trưng. Các tập dữ liệu dự đoán lỗi phần mềm thường có cấu trúc dạng bảng, bao gồm nhiều thuộc tính mô tả đặc trưng của phần mềm và một biến phụ thuộc cho biết phần mềm có lỗi hay không. Việc tạo ra các mẫu dữ liệu tổng hợp từ các tập dữ liệu dạng bảng như vậy là một thách thức, do các đặc trưng khác nhau có thể tuân theo các phân phối xác suất khác nhau. Do đó, việc học và hiểu các phân phối dữ liệu này để sinh ra các mẫu mới không phải là

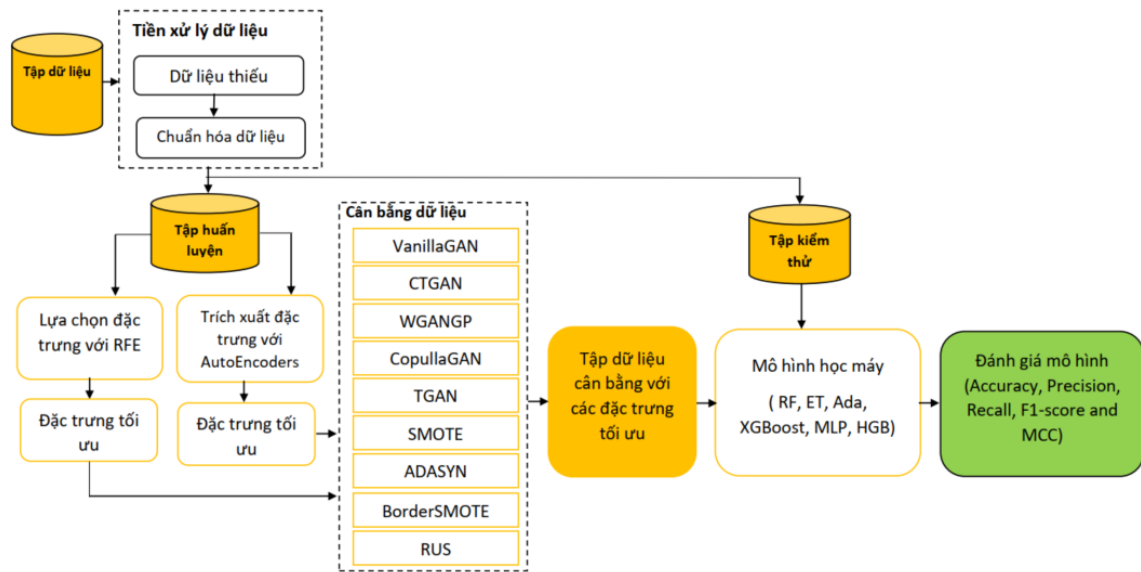
nhiệm vụ đơn giản. Hầu hết các kỹ thuật lấy mẫu tăng hiện nay áp dụng nhiều chiến lược khác nhau, chẳng hạn như sử dụng các phép đo khoảng cách để tìm các mẫu lân cận và nội suy giữa các mẫu này nhằm tạo ra dữ liệu tổng hợp mới. Tuy nhiên, các mẫu được tạo ra theo cách này thường thiếu đa dạng và có xu hướng bị khái quát hóa quá mức [121]. GAN là một trong những phương pháp sinh dữ liệu tổng hợp được đánh giá là đầy triển vọng [128]. Trong các tập dữ liệu lỗi phần mềm được sử dụng, tất cả các đặc trưng đều là dữ liệu dạng số và không có đặc trưng phân loại. Các kỹ thuật lấy mẫu tăng truyền thống như SMOTE thường dựa vào thông tin cục bộ, cụ thể là các điểm lân cận để tạo dữ liệu tổng hợp. Trong khi đó, các phương pháp dựa trên GAN có khả năng học được phân phối tổng thể của dữ liệu và mối tương quan giữa các đặc trưng [108]. Nhờ đó, GAN có thể sinh ra các mẫu dữ liệu tổng hợp chân thực hơn so với các kỹ thuật lấy mẫu tăng truyền thống. Nghiên cứu của Edwards và cộng sự [145] chỉ ra rằng GAN có tiềm năng tạo ra tập dữ liệu tổng hợp phong phú hơn đáng kể so với các phương pháp tiêu chuẩn khác. Trong bối cảnh của các tập dữ liệu lỗi phần mềm, không phải tất cả đặc trưng đều tuân theo cùng một phân phối. Do đó, cách tiếp cận dựa trên GAN được xem là phù hợp cho bài toán này. Xuất phát từ nhận định đó, luận án lựa chọn sử dụng phương pháp học mất cân bằng dựa trên GAN trong xử lý dữ liệu lỗi phần mềm. Trong chương này, nghiên cứu được thiết kế nhằm trả lời các câu hỏi nghiên cứu sau.

1. *Câu hỏi nghiên cứu 1: Các biến thể của GAN có hiệu quả như thế nào trong việc lấy mẫu dữ liệu để xử lý mất cân bằng lớp trong dự đoán lỗi phần mềm?*
2. *Câu hỏi nghiên cứu 2: Trong số các kỹ thuật tiên tiến nhất hiện nay, biến thể nào của GAN vượt trội hơn trong việc xử lý mất cân bằng dữ liệu trong dự đoán lỗi phần mềm?*

3.4 Phương pháp đề xuất

Trong phần này, trước tiên tác giả trình bày các bước chính của thử nghiệm được thực hiện trong luận án này, tiếp theo giới thiệu về tập dữ liệu dự án được sử dụng để thử nghiệm. Sau đó, luận án giới thiệu về các mô hình lựa chọn đặc trưng và trích xuất đặc trưng được áp dụng. Phần tiếp theo trình bày về các tham số huấn luyện của các mô hình học máy. Cuối cùng, các kết quả của thử nghiệm được phân tích và đánh giá.

Các bước của phương pháp thử nghiệm được minh họa trong Hình 3.1. Ở giai đoạn đầu tiên, luận án thu thập năm tập dữ liệu lỗi phổ biến từ kho dữ liệu PROMISE. Mô tả chi tiết về các tập dữ liệu được trình bày trong Bảng 3.1. Sau khi điền các giá trị bị thiếu, bước tiếp theo là chuẩn hóa dữ liệu bằng kỹ thuật chuẩn hóa z (z -normalization) trong giai



Hình 3.1: Phương pháp đề xuất

đoạn tiền xử lý dữ liệu. Các tập dữ liệu sau khi chuẩn hóa được chia thành tập huấn luyện và tập kiểm thử theo tỷ lệ 70:30. Từ kết quả thực nghiệm thu được ở chương 2 đã cho thấy Recursive Feature Elimination là phương pháp lựa chọn đặc trưng đạt hiệu suất tốt nhất, luận án áp dụng phương pháp RFE để lựa chọn đặc trưng, giúp tạo ra các đặc trưng tối ưu từ tập huấn luyện. Ngoài ra, luận án cũng áp dụng Autoencoders, một phương pháp trích xuất đặc trưng hiệu quả, nhằm trích xuất các đặc trưng liên quan và loại bỏ thông tin dư thừa từ tập dữ liệu. Sau đó, luận án áp dụng năm biến thể của GAN, bao gồm VanillaGAN, CTGAN, WGAN, CopulaGAN và TGAN vào hai tập con huấn luyện thu được từ quá trình lựa chọn đặc trưng và trích xuất đặc trưng để tạo ra các tập dữ liệu cân bằng. Các tập dữ liệu huấn luyện được tạo ra sẽ bao gồm các đặc trưng tối ưu và lớp dữ liệu cân bằng sẽ được đưa vào huấn luyện bởi các kỹ thuật học máy bao gồm Random Forest, XGBoost, Multilayer Perceptron, Extra Trees, Adaptive Boosting và HistGradientBoosting để xây dựng các mô hình dự đoán. Các tập dữ liệu kiểm thử được đưa vào các mô hình dự đoán lỗi để đánh giá hiệu suất bằng các chỉ số đo lường khác nhau. Dữ liệu thu được từ các thử nghiệm cho phép đưa ra câu trả lời cho Câu hỏi nghiên cứu 1. Để trả lời Câu hỏi nghiên cứu 2, luận án so sánh hiệu suất của các mô hình dự đoán lỗi khi áp dụng các biến thể GAN để cân bằng dữ liệu với hiệu suất của các mô hình dự đoán trên tập dữ liệu gốc bị mất cân bằng. Các mục dưới đây sẽ mô tả chi tiết về tập dữ liệu lỗi phần mềm, phương pháp lựa chọn đặc trưng, phương pháp trích xuất đặc trưng, các mô hình GAN với những ưu điểm và nhược điểm tương ứng và các độ đo đánh giá hiệu suất được sử dụng trong quá trình này.

- **Tập dữ liệu thử nghiệm**

Năm tập dữ liệu lỗi khác nhau từ kho PROMISE [146] đã được sử dụng trong luận

Bảng 3.1: Chi tiết tập dữ liệu thử nghiệm trước và sau khi cân bằng

Kho dữ liệu	Tập dữ liệu	Số mẫu	Độ đo	Số mẫu không lỗi	Số mẫu lỗi	Tỷ lệ mất cân bằng	Số mẫu sau cân bằng
PROMISE	CM1	498	21	449	49	9.84	628
	KC1	2109	21	1693	326	15.45	2856
	KC2	522	21	415	107	20.49	590
	PC1	1109	21	1032	77	6.9	1422
	JM1	10885	21	8779	2016	18.5	12288

án này. Như thể hiện trong Bảng 3.1, các tập dữ liệu này không cân bằng. Các thuộc tính độc lập là các độ đo về mã nguồn như độ đo McCabe's Cyclomatic Complexity ($v(g)$), LOC, và các thuộc tính phụ thuộc bao gồm nhãn lỗi (1) hoặc không lỗi (0). Bảng 3.1 tóm tắt chi tiết về các tập dữ liệu được sử dụng.

- **Kỹ thuật lựa chọn đặc trưng**

Trong chương 2, luận án đã đánh giá hiệu quả của các phương pháp lựa chọn đặc trưng dựa trên wrapper và xác định rằng RFE là phương pháp đạt hiệu suất cao nhất trong số các kỹ thuật lựa chọn đặc trưng được đánh giá. Phương pháp này không chỉ cải thiện độ chính xác của các mô hình phân loại mà còn giúp giảm chiều dữ liệu, tăng tính ổn định và khả năng tổng quát hóa của mô hình. Ngoài ra, theo nghiên cứu của Mehta và Patnaik [57], nhiều kỹ thuật lựa chọn đặc trưng như RFE, Lasso, Ridge, ElasticNet, Boruta và Correlation-based đã được so sánh nhằm xác định phương pháp phù hợp nhất cho mô hình dự đoán lỗi phần mềm. Kết quả cho thấy RFE vượt trội hơn các kỹ thuật còn lại nhờ cơ chế loại bỏ đệ quy các đặc trưng dư thừa hoặc kém quan trọng, từ đó thu được tập đặc trưng tinh gọn nhưng vẫn đảm bảo khả năng dự báo cao. Do đó, trong thử nghiệm này, luận án áp dụng kỹ thuật RFE để lựa chọn các đặc trưng tối ưu và xây dựng mô hình dự đoán lỗi phần mềm.

- **Kỹ thuật trích xuất đặc trưng**

Malhotra và cộng sự [50] đã thực hiện một nghiên cứu so sánh để đánh giá hiệu quả của bốn kỹ thuật trích xuất đặc trưng, bao gồm Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Kernel-based Principal Component Analysis (K-PCA) và Autoencoders. Kết quả thử nghiệm của nghiên cứu này cho thấy Autoencoders đạt hiệu suất tốt nhất trong số bốn kỹ thuật trích xuất đặc trưng. Do đó, trong thử nghiệm này, luận án sử dụng Autoencoders để trích xuất đặc trưng. luận án tiến hành thử nghiệm đánh giá hiệu suất của các phương pháp lấy mẫu tăng để lấy mẫu dữ liệu, kết hợp với Autoencoders trong trích xuất đặc trưng, sử dụng các bộ phân loại khác nhau.

- **Cấu hình các tham số huấn luyện**

Các bảng 3.2-3.6 mô tả các tham số và giá trị được chọn cho từng mô hình GAN trong thử nghiệm này.

Bảng 3.2: Cấu hình các tham số huấn luyện cho VanillaGAN

Tham số	Định nghĩa	Value
noise_dim	Kích thước của véc-tơ nhiễu đầu vào được cung cấp cho bộ sinh	32
dim	Kích thước không gian đầu ra của bộ sinh	128
batch_size	Số lượng mẫu được sử dụng trong mỗi bước	32
epochs	Số lượng epoch cần thiết để mô hình tối ưu hóa các siêu tham số	5000
learning_rate	Tỷ lệ học của bộ tối ưu hóa	5e-4

Bảng 3.3: Cấu hình các tham số huấn luyện cho CTGAN

Tham số	Định nghĩa	Value
epochs	Số lượng epoch cần thiết để mô hình tối ưu hóa các siêu tham số	300
batch size	Số lượng mẫu được sử dụng trong mỗi bước	500
optimizer	Bộ phân loại	AdamOptimizer
embedding dim	Kích thước của mẫu ngẫu nhiên được cung cấp cho bộ sinh	128
generator dim	Một Residual sẽ được tạo ra trong mỗi lớp cung cấp	(256,256)
discriminator dim	Kích thước của các mẫu đầu ra cho mỗi lớp bộ sinh	(256,256)
generator lr	Kích thước của các mẫu đầu ra cho mỗi lớp bộ phân biệt	2e-4
generator decay	Learning rate của bộ sinh	1e-6
discriminator lr	Sự suy giảm trọng số cho bộ sinh trong bộ tối ưu hóa Adam	2e-4
discriminator decay	Learning rate của bộ phân biệt	1e-6

Bảng 3.4: Cấu hình các tham số huấn luyện cho WGANP

Tham số	Định nghĩa	Giá trị
netG_kwargs	hidden_layer_sizes	Tuple chỉ định kích thước của các lớp ẩn trong mạng bộ sinh (128, 64)
	n-cross-layers	Số lượng lớp chéo (cross layers) được sử dụng trong bộ sinh
	cat-activation	Hàm kích hoạt được sử dụng cho các biến phân loại
	noise-dim	Kích thước của véc-tơ nhiễu đầu vào được cung cấp cho bộ sinh
	activation	Hàm kích hoạt sử dụng trong mỗi lớp ẩn
netD_kwargs	hidden_layer_sizes	Tuple chỉ định kích thước của các lớp ẩn trong bộ sinh (128,64,32)
	n_cross_layers	Số lượng lớp chéo sử dụng trong bộ sinh
	Activation	Hàm kích hoạt sử dụng trong lớp ẩn

3.5 Kết quả thử nghiệm và phân tích kết quả

Từ thiết kế thử nghiệm được trình bày trong mục 3.4, luận án xây dựng các mô hình dự đoán lỗi bằng cách kết hợp các biến thể khác nhau của GANs để tổng hợp các mẫu mới,

Bảng 3.5: Cấu hình các tham số huấn luyện cho CopulaGAN

Tham số	Định nghĩa	Giá trị
epochs	Số lượng epoch cần thiết để mô hình tối ưu hóa các siêu tham số	300
batch size	Số lượng mẫu được sử dụng trong mỗi bước	500
optimizer	Bộ phân loại	AdamOptimizer
embedding_dim	Kích thước của mẫu ngẫu nhiên được cung cấp cho bộ sinh	128
generator_dim	Kích thước của các mẫu đầu ra cho mỗi lớp dư Residual. Một lớp dư Residual sẽ được tạo ra cho mỗi giá trị được cung cấp.	(256,256)
discriminator_dim	Kích thước của các mẫu đầu ra cho mỗi lớp bộ phân biệt	(256,256)
generator_lr	Tỷ lệ Learning rate cho bộ sinh	2e-4
generator_decay	Sự suy giảm trọng số cho bộ sinh trong bộ tối ưu hóa Adam	1e-6
discriminator_lr	Tỷ lệ Learning rate cho bộ phân biệt	2e-4
discriminator_decay	Sự suy giảm trọng số cho bộ phân biệt trong bộ tối ưu hóa Adam	1e-6

Bảng 3.6: Cấu hình các tham số huấn luyện cho TGAN

Tham số	Định nghĩa	Giá trị
epochs	Số lượng epoch cần thiết để mô hình tối ưu hóa các siêu tham số	300
batch size	Số lượng mẫu được sử dụng trong mỗi bước	500
optimizer	Bộ phân loại	AdamOptimizer
z_dim	(Số lượng chiều trong đầu vào nhiễu của bộ sinh	100
l2norm	Hệ số điều chuẩn L2 được sử dụng khi tính toán hàm mất mát	0.00001
learning_rate	Tỷ lệ learning rate của bộ tối ưu	0.001
num_gen_rnn	Số lượng đơn vị RNN bên trong bộ sinh.	400
num_gen_feature	Số lượng đơn vị trong lớp kết nối đầy đủ của bộ sinh	100
num_dis_layers	Số lượng lớp trong bộ phân biệt bộ phân biệt	2
num_dis_hidden	Số lượng đơn vị trong mỗi lớp của bộ phân biệt	200

sau đó sử dụng phương pháp chọn đặc trưng RFE và phương pháp trích xuất đặc trưng Autoencoders để giảm bớt các đặc trưng không liên quan hoặc dư thừa. Luận án đã sử dụng sáu bộ phân loại học máy phổ biến để đánh giá tác động của sự mất cân bằng lớp và sự dư thừa lớp lên hiệu suất của các mô hình này bằng cách sử dụng phương pháp kiểm tra chéo 10 lần cho mỗi thử nghiệm. Hiệu suất của các mô hình dự đoán lỗi phần mềm được đánh giá bằng các chỉ số precision, recall, F1-score, AUC và MCC. Phần này trình bày kết quả thử nghiệm nhằm giải quyết hai câu hỏi nghiên cứu trên.

3.5.1 Trả lời cho Câu hỏi nghiên cứu 1: Các biến thể của GAN có hiệu quả như thế nào trong việc lấy mẫu dữ liệu để xử lý mất cân bằng lớp trong dự đoán lỗi phần mềm?

Luận án thực hiện các thử nghiệm khác nhau để tiến hành nghiên cứu so sánh giữa các biến thể của GAN và các phương pháp lấy mẫu tăng cơ sở. Để giải quyết câu hỏi nghiên cứu 1, luận án đánh giá hiệu quả của hai phương pháp kết hợp: kỹ thuật trích xuất đặc trưng RFE kết hợp với các mô hình lấy mẫu tăng khác nhau và phương pháp trích xuất đặc trưng bằng Autoencoders kết hợp với các phương pháp lấy mẫu tăng khác nhau. Các

mục dưới đây trình bày các đánh giá hiệu suất khác nhau của hai phương pháp kết hợp này.

1. Kết quả thử nghiệm của sự kết hợp giữa RFE và các mô hình GAN

Trong phần này, luận án sẽ đánh giá hiệu suất của các biến thể khác nhau của mô hình GAN trong việc lấy mẫu dữ liệu, kết hợp với phương pháp chọn đặc trưng RFE dựa trên các kỹ thuật học máy khác nhau. Trong các Bảng 3.1 - 3.6, luận án đã tô đậm những kết quả tốt nhất.

(a) Random Forest

Khi so sánh giá trị trung bình của tất cả các phương pháp được trình bày trong

Bảng 3.1: Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng Random Forest

Dữ liệu	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.838	0.841	0.854	0.828	0.864	0.853	0.846	0.839	0.842
	R	0.884	0.805	0.895	0.781	0.781	0.722	0.788	0.730	0.662
	F1	0.853	0.821	0.865	0.794	0.813	0.771	0.818	0.775	0.728
	AUC	0.790	0.728	0.841	0.727	0.749	0.718	0.759	0.688	0.737
	MCC	0.206	0.194	0.251	0.193	0.220	0.162	0.232	0.138	0.207
PC1	P	0.907	0.896	0.898	0.877	0.900	0.891	0.888	0.890	0.883
	R	0.918	0.887	0.927	0.852	0.822	0.817	0.852	0.846	0.702
	F1	0.912	0.891	0.904	0.864	0.859	0.854	0.877	0.873	0.775
	AUC	0.864	0.835	0.861	0.823	0.838	0.846	0.844	0.853	0.800
	MCC	0.272	0.195	0.250	0.199	0.206	0.238	0.194	0.250	0.240
KC1	P	0.797	0.814	0.815	0.792	0.812	0.815	0.807	0.810	0.812
	R	0.839	0.840	0.843	0.758	0.733	0.753	0.768	0.779	0.675
	F1	0.804	0.821	0.814	0.774	0.772	0.778	0.789	0.795	0.719
	AUC	0.797	0.804	0.795	0.755	0.789	0.786	0.795	0.776	0.790
	MCC	0.191	0.275	0.247	0.183	0.225	0.224	0.224	0.211	0.222
KC2	P	0.826	0.817	0.813	0.724	0.809	0.807	0.806	0.814	0.815
	R	0.815	0.841	0.823	0.733	0.776	0.779	0.796	0.798	0.766
	F1	0.82	0.825	0.813	0.728	0.792	0.787	0.806	0.806	0.784
	AUC	0.783	0.807	0.801	0.772	0.779	0.778	0.800	0.779	0.805
	MCC	0.363	0.289	0.412	0.208	0.263	0.303	0.254	0.256	0.248
JM1	P	0.825	0.827	0.829	0.783	0.836	0.828	0.826	0.828	0.829
	R	0.808	0.804	0.807	0.722	0.769	0.72	0.742	0.710	0.668
	F1	0.755	0.769	0.754	0.751	0.714	0.74	0.754	0.731	0.700
	AUC	0.724	0.712	0.714	0.706	0.703	0.72	0.715	0.696	0.721
	MCC	0.174	0.217	0.171	0.126	0.178	0.202	0.168	0.241	0.204
Kết quả trung bình										
	P	0.839	0.839	0.842	0.801	0.844	0.839	0.835	0.836	0.836
	R	0.853	0.835	0.859	0.769	0.776	0.758	0.789	0.773	0.695
	F1	0.829	0.825	0.830	0.782	0.779	0.786	0.809	0.796	0.741
	AUC	0.792	0.777	0.802	0.757	0.774	0.770	0.783	0.758	0.771
	MCC	0.241	0.234	0.266	0.182	0.218	0.226	0.214	0.219	0.224

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: Borderline-SMOTE

Bảng 3.1, WGANGP kết hợp với Random Forest cho thấy hiệu suất cao nhất đối với tất cả các độ đo đánh giá, tiếp theo là VanillaGAN và CTGAN. Cụ thể,

các kết quả trung bình của WGANGP đạt giá trị cao nhất đối với precision, recall, AUC và F1-score, lần lượt là 0.842, 0.859, 0.830, 0.802 và 0.266. Giá trị cao thứ hai được ghi nhận ở VanillaGAN với precision, recall, F1-score, AUC và MCC lần lượt là 0.839, 0.853, 0.829, 0.792 và 0.241. Kết quả phân tích này cho thấy WGANGP và VanillaGAN có hiệu suất tương đối tốt hơn khi kết hợp với Random Forest so với các phương pháp lấy mẫu khác.

(b) **XGBoost**

Phân tích kết quả của XGBoost trong Bảng 3.2 cho thấy WGANGP hoạt

Bảng 3.2: Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng XGBoost

Dataset	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.843	0.863	0.856	0.843	0.867	0.839	0.821	0.831	0.855
	R	0.868	0.880	0.885	0.857	0.857	0.820	0.880	0.864	0.891
	F1	0.852	0.868	0.866	0.849	0.861	0.828	0.849	0.847	0.863
	AUC	0.737	0.761	0.755	0.734	0.735	0.715	0.711	0.694	0.724
	MCC	0.115	0.221	0.181	0.171	0.162	0.220	0.118	0.196	0.153
PC1	P	0.921	0.923	0.926	0.907	0.918	0.921	0.906	0.914	0.903
	R	0.930	0.930	0.936	0.882	0.894	0.925	0.909	0.908	0.906
	F1	0.925	0.926	0.929	0.894	0.911	0.922	0.907	0.910	0.904
	AUC	0.882	0.870	0.888	0.846	0.837	0.873	0.863	0.866	0.838
	MCC	0.382	0.399	0.414	0.309	0.296	0.306	0.333	0.337	0.353
KC1	P	0.822	0.834	0.830	0.823	0.831	0.824	0.821	0.829	0.827
	R	0.841	0.849	0.848	0.835	0.849	0.841	0.832	0.832	0.845
	F1	0.829	0.835	0.835	0.828	0.839	0.832	0.826	0.825	0.819
	AUC	0.796	0.793	0.792	0.783	0.792	0.778	0.780	0.781	0.791
	MCC	0.314	0.356	0.341	0.234	0.296	0.307	0.324	0.329	0.272
KC2	P	0.838	0.936	0.851	0.853	0.826	0.817	0.825	0.817	0.805
	R	0.819	0.849	0.830	0.824	0.831	0.818	0.843	0.836	0.811
	F1	0.824	0.838	0.810	0.838	0.828	0.815	0.833	0.818	0.806
	AUC	0.771	0.793	0.776	0.759	0.792	0.769	0.760	0.680	0.761
	MCC	0.284	0.357	0.243	0.280	0.320	0.290	0.224	0.227	0.246
JM1	P	0.785	0.779	0.772	0.771	0.767	0.779	0.777	0.773	0.773
	R	0.816	0.811	0.807	0.796	0.757	0.803	0.803	0.812	0.812
	F1	0.785	0.783	0.776	0.783	0.761	0.787	0.763	0.761	0.760
	AUC	0.727	0.726	0.717	0.702	0.716	0.707	0.708	0.706	0.705
	MCC	0.277	0.266	0.242	0.234	0.213	0.274	0.213	0.199	0.199
Giá trị trung bình										
	P	0.842	0.867	0.847	0.839	0.842	0.836	0.830	0.833	0.833
	R	0.855	0.864	0.861	0.839	0.838	0.841	0.853	0.850	0.853
	F1	0.843	0.850	0.843	0.838	0.840	0.837	0.836	0.832	0.830
	AUC	0.783	0.789	0.786	0.765	0.774	0.768	0.770	0.745	0.780
	MCC	0.274	0.320	0.284	0.246	0.257	0.279	0.242	0.258	0.245

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: BorderSMOTE

động tốt hơn trên tập dữ liệu PC1 đối với tất cả các độ đo hiệu suất, với các giá trị tốt nhất của precision, recall, F1-score, AUC và MCC lần lượt là 0.926, 0.936, 0.929, 0.888 và 0.414. Ngoài ra, CTGAN cho kết quả tốt hơn so

với các kỹ thuật khác trên các tập dữ liệu CM1, KC1 và KC2. Dựa trên kết quả trung bình của tất cả các phương pháp được trình bày, CTGAN thể hiện hiệu suất tốt nhất đối với các độ đo đánh giá, tiếp theo là WGANGP và VanillaGAN. CopulaGAN và TGAN đều cho hiệu suất tương đương với các phương pháp lấy mẫu cơ sở như SMOTE, Borderline-SMOTE, ADASYN và RUS.

(c) AdaBoost

Như thể hiện trong Bảng 3.3, có thể thấy rằng CTGAN vượt trội hơn so với

Bảng 3.3: Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng AdaBoost

Dataset	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.841	0.851	0.836	0.832	0.863	0.837	0.841	0.830	0.832
	R	0.833	0.858	0.858	0.783	0.783	0.794	0.832	0.794	0.653
	F1	0.836	0.853	0.846	0.780	0.821	0.712	0.836	0.809	0.722
	AUC	0.789	0.792	0.683	0.684	0.709	0.669	0.721	0.654	0.693
	MCC	0.197	0.197	0.187	0.163	0.183	0.191	0.171	0.155	0.186
PC1	P	0.905	0.912	0.917	0.898	0.910	0.912	0.913	0.915	0.907
	R	0.908	0.919	0.922	0.884	0.897	0.900	0.900	0.908	0.732
	F1	0.906	0.915	0.919	0.890	0.903	0.905	0.906	0.911	0.796
	AUC	0.795	0.775	0.804	0.796	0.799	0.803	0.804	0.797	0.795
	MCC	0.263	0.319	0.356	0.182	0.182	0.345	0.323	0.346	0.254
KC1	P	0.825	0.816	0.820	0.818	0.856	0.824	0.824	0.820	0.819
	R	0.850	0.840	0.844	0.808	0.772	0.819	0.816	0.816	0.699
	F1	0.830	0.824	0.827	0.811	0.800	0.821	0.819	0.817	0.737
	AUC	0.756	0.751	0.738	0.742	0.802	0.760	0.760	0.745	0.755
	MCC	0.309	0.286	0.298	0.153	0.291	0.261	0.257	0.244	0.270
KC2	P	0.787	0.816	0.815	0.8712	0.828	0.808	0.804	0.813	0.812
	R	0.784	0.840	0.820	0.772	0.774	0.801	0.798	0.811	0.743
	F1	0.785	0.824	0.816	0.791	0.782	0.802	0.800	0.811	0.764
	AUC	0.700	0.751	0.711	0.727	0.821	0.703	0.720	0.730	0.767
	MCC	0.344	0.286	0.329	0.176	0.286	0.312	0.296	0.230	0.211
JM1	P	0.773	0.778	0.766	0.759	0.749	0.772	0.770	0.769	0.761
	R	0.809	0.808	0.808	0.777	0.799	0.772	0.788	0.788	0.677
	F1	0.790	0.792	0.764	0.746	0.773	0.772	0.777	0.776	0.707
	AUC	0.719	0.714	0.706	0.704	0.700	0.719	0.711	0.718	0.701
	MCC	0.218	0.218	0.202	0.186	0.192	0.218	0.210	0.215	0.202
Trung bình										
	P	0.826	0.835	0.831	0.824	0.841	0.831	0.830	0.829	0.826
	R	0.837	0.853	0.850	0.805	0.807	0.817	0.827	0.823	0.701
	F1	0.829	0.842	0.834	0.804	0.816	0.802	0.828	0.825	0.745
	AUC	0.752	0.757	0.728	0.731	0.766	0.731	0.743	0.729	0.742
	MCC	0.266	0.261	0.274	0.172	0.227	0.265	0.251	0.238	0.225

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: Borderline-SMOTE

các mô hình GAN khác về recall và F1-score trên hầu hết các tập dữ liệu, bao gồm CM1, KC1 và JM1, trong khi WGANGP đạt hiệu suất tốt nhất trên PC1 với các giá trị cao nhất của precision, recall, F1-score, AUC và MCC lần lượt là 0.917, 0.922, 0.919, 0.804 và 0.356. Đáng chú ý, CopulaGAN đạt giá

trị precision cao nhất trên các tập dữ liệu CM1, KC1 và KC2.

(d) **HGBoost**

Như kết quả trong Bảng 3.4 cho thấy, HGBoost có hiệu suất dự đoán tương

Bảng 3.4: Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng HGBoost

Dataset	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.848	0.858	0.845	0.844	0.892	0.777	0.852	0.838	0.860
	R	0.871	0.877	0.874	0.865	0.865	0.602	0.855	0.870	0.634
	F1	0.858	0.866	0.857	0.853	0.853	0.642	0.853	0.852	0.707
	AUC	0.759	0.761	0.748	0.728	0.771	0.704	0.757	0.742	0.686
	MCC	0.263	0.207	0.231	0.192	0.194	0.231	0.178	0.104	0.160
PC1	P	0.917	0.925	0.928	0.902	0.906	0.919	0.912	0.902	0.907
	R	0.928	0.932	0.937	0.880	0.908	0.922	0.901	0.912	0.721
	F1	0.921	0.927	0.929	0.890	0.907	0.920	0.906	0.905	0.789
	AUC	0.891	0.887	0.863	0.833	0.895	0.858	0.875	0.885	0.797
	MCC	0.344	0.409	0.422	0.338	0.383	0.397	0.395	0.402	0.271
KC1	P	0.827	0.835	0.832	0.826	0.830	0.840	0.830	0.837	0.834
	R	0.849	0.851	0.852	0.823	0.849	0.848	0.842	0.848	0.703
	F1	0.833	0.839	0.836	0.824	0.839	0.844	0.837	0.840	0.742
	AUC	0.796	0.799	0.791	0.787	0.801	0.783	0.785	0.776	0.783
	MCC	0.325	0.360	0.342	0.291	0.326	0.294	0.308	0.305	0.285
KC2	P	0.826	0.835	0.819	0.823	0.820	0.806	0.821	0.819	0.835
	R	0.813	0.851	0.830	0.808	0.769	0.805	0.818	0.818	0.768
	F1	0.818	0.840	0.822	0.816	0.793	0.803	0.818	0.816	0.786
	AUC	0.759	0.800	0.789	0.788	0.823	0.761	0.779	0.765	0.781
	MCC	0.361	0.360	0.358	0.303	0.320	0.305	0.307	0.308	0.323
JM1	P	0.782	0.776	0.773	0.785	0.777	0.779	0.778	0.777	0.781
	R	0.815	0.812	0.811	0.765	0.702	0.715	0.802	0.804	0.678
	F1	0.777	0.775	0.768	0.775	0.737	0.737	0.785	0.785	0.709
	AUC	0.750	0.739	0.736	0.745	0.704	0.710	0.735	0.740	0.722
	MCC	0.253	0.241	0.220	0.188	0.231	0.276	0.279	0.275	0.269
Trung bình										
	P	0.840	0.846	0.839	0.836	0.845	0.824	0.839	0.835	0.843
	R	0.855	0.865	0.861	0.828	0.819	0.778	0.844	0.850	0.701
	F1	0.841	0.849	0.842	0.832	0.826	0.789	0.840	0.840	0.747
	AUC	0.791	0.797	0.786	0.776	0.799	0.763	0.786	0.782	0.754
	MCC	0.309	0.315	0.315	0.262	0.291	0.301	0.293	0.279	0.262

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: BorderSMOTE

tự như XGBoost. Đặc biệt, các giá trị precision, recall, F1-score và MCC của WGANGP cao hơn so với các phương pháp khác trên tập dữ liệu PC1, lần lượt đạt 0.928, 0.937, 0.929 và 0.422. Một lần nữa, CTGAN đạt kết quả tốt hơn trên các tập dữ liệu CM1, KC1 và KC2. Dựa trên kết quả trung bình, luận án quan sát thấy rằng CTGAN có đạt kết quả tốt nhất, tiếp theo là WGANGP và VanillaGAN khi sử dụng HGBoost.

(e) **Multilayer Perceptron**

Bảng 3.5: Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng MLP

Dataset	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.860	0.843	0.848	0.850	0.874	0.831	0.828	0.823	0.828
	R	0.722	0.819	0.886	0.768	0.768	0.762	0.663	0.696	0.654
	F1	0.777	0.829	0.862	0.780	0.815	0.800	0.707	0.691	0.722
	AUC	0.701	0.730	0.726	0.704	0.750	0.733	0.666	0.660	0.748
	MCC	0.183	0.224	0.143	0.168	0.182	0.210	0.166	0.106	0.203
PC1	P	0.904	0.894	0.904	0.858	0.924	0.904	0.900	0.893	0.895
	R	0.820	0.893	0.928	0.707	0.753	0.867	0.694	0.744	0.705
	F1	0.853	0.893	0.908	0.775	0.829	0.886	0.750	0.775	0.714
	AUC	0.737	0.745	0.794	0.722	0.774	0.775	0.627	0.628	0.625
	MCC	0.220	0.197	0.201	0.193	0.207	0.185	0.187	0.165	0.170
KC1	P	0.799	0.830	0.827	0.808	0.851	0.823	0.824	0.821	0.814
	R	0.748	0.857	0.852	0.763	0.715	0.736	0.666	0.749	0.73
	F1	0.769	0.830	0.824	0.784	0.777	0.765	0.699	0.751	0.747
	AUC	0.740	0.811	0.789	0.725	0.808	0.770	0.714	0.593	0.657
	MCC	0.225	0.313	0.293	0.238	0.313	0.291	0.253	0.236	0.254
KC2	P	0.826	0.831	0.813	0.795	0.812	0.818	0.810	0.768	0.797
	R	0.755	0.857	0.828	0.756	0.738	0.822	0.775	0.741	0.749
	F1	0.774	0.830	0.811	0.730	0.773	0.827	0.777	0.732	0.762
	AUC	0.784	0.811	0.820	0.732	0.849	0.806	0.712	0.601	0.830
	MCC	0.333	0.313	0.303	0.259	0.279	0.261	0.250	0.280	0.183
JM1	P	0.829	0.835	0.833	0.821	0.848	0.831	0.822	0.811	0.818
	R	0.763	0.848	0.861	0.751	0.725	0.797	0.659	0.710	0.678
	F1	0.786	0.830	0.834	0.778	0.770	0.813	0.684	0.715	0.700
	AUC	0.731	0.764	0.769	0.713	0.768	0.765	0.658	0.605	0.688
	MCC	0.235	0.228	0.229	0.205	0.240	0.245	0.204	0.186	0.197
Trung bình										
	P	0.844	0.847	0.845	0.826	0.862	0.841	0.837	0.823	0.83
	R	0.762	0.855	0.871	0.749	0.740	0.797	0.691	0.728	0.703
	F1	0.792	0.842	0.848	0.769	0.793	0.818	0.723	0.733	0.729
	AUC	0.739	0.772	0.780	0.719	0.792	0.770	0.675	0.617	0.710
	MCC	0.239	0.255	0.234	0.213	0.244	0.238	0.212	0.195	0.201

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: Borderline-SMOTE

Như thể hiện trong Bảng 3.5, kết quả thử nghiệm cho thấy các mô hình GAN khác nhau có hiệu suất tốt hơn trên hầu hết các tập dữ liệu. Cụ thể, trên các tập dữ liệu CM1, PC1, KC1 và JM1, CopulaGAN đạt giá trị precision cao nhất lần lượt là 0.874, 0.924, 0.851 và 0.848. Mặt khác, WGANGP đạt giá trị AUC cao nhất trên các tập dữ liệu PC1, KC2 và JM1. Cuối cùng, trên tập dữ liệu KC1, CTGAN đạt giá trị cao nhất về recall, F1-score, AUC và MCC, trong khi precision cao nhất được ghi nhận ở CopulaGAN. Giá trị MCC cao nhất được ghi nhận ở CTGAN trên các tập dữ liệu KC1 và KC2.

(f) Extra Trees

Như quan sát trong Bảng 3.6, có thể thấy rằng CTGAN vượt trội hơn tất cả các

Bảng 3.6: Kết quả của RFE và các kỹ thuật lấy mẫu tăng sử dụng Extra Trees

Dataset	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.840	0.853	0.843	0.841	0.881	0.839	0.844	0.836	0.841
	R	0.875	0.878	0.892	0.881	0.808	0.815	0.853	0.812	0.641
	F1	0.854	0.862	0.859	0.858	0.827	0.825	0.848	0.822	0.702
	AUC	0.727	0.777	0.736	0.763	0.757	0.741	0.757	0.732	0.713
	MCC	0.194	0.172	0.199	0.152	0.192	0.217	0.135	0.188	0.198
PC1	P	0.912	0.919	0.928	0.874	0.915	0.926	0.913	0.916	0.909
	R	0.927	0.929	0.939	0.854	0.894	0.927	0.916	0.905	0.704
	F1	0.918	0.923	0.927	0.826	0.904	0.926	0.914	0.914	0.776
	AUC	0.856	0.833	0.831	0.796	0.779	0.812	0.826	0.808	0.809
	MCC	0.306	0.370	0.394	0.258	0.256	0.343	0.323	0.349	0.252
KC1	P	0.832	0.840	0.838	0.828	0.851	0.830	0.831	0.833	0.836
	R	0.851	0.857	0.855	0.763	0.782	0.832	0.825	0.829	0.692
	F1	0.838	0.845	0.842	0.791	0.792	0.831	0.827	0.830	0.733
	AUC	0.811	0.830	0.819	0.804	0.798	0.801	0.811	0.803	0.789
	MCC	0.351	0.381	0.368	0.323	0.327	0.355	0.355	0.365	0.316
KC2	P	0.831	0.841	0.812	0.803	0.826	0.804	0.823	0.814	0.833
	R	0.824	0.857	0.826	0.773	0.799	0.807	0.811	0.813	0.751
	F1	0.827	0.845	0.814	0.787	0.812	0.804	0.815	0.812	0.771
	AUC	0.794	0.830	0.817	0.807	0.822	0.804	0.812	0.792	0.808
	MCC	0.379	0.381	0.411	0.284	0.289	0.301	0.352	0.333	0.346
JM1	P	0.788	0.788	0.783	0.783	0.784	0.787	0.784	0.787	0.781
	R	0.818	0.818	0.814	0.798	0.797	0.797	0.787	0.788	0.663
	F1	0.787	0.791	0.786	0.790	0.790	0.791	0.786	0.788	0.697
	AUC	0.746	0.759	0.745	0.732	0.710	0.740	0.744	0.740	0.726
	MCC	0.286	0.295	0.277	0.204	0.201	0.215	0.208	0.218	0.264
Trung bình										
	P	0.841	0.848	0.841	0.826	0.851	0.837	0.839	0.837	0.840
	R	0.859	0.868	0.865	0.814	0.816	0.836	0.838	0.829	0.690
	F1	0.845	0.853	0.846	0.810	0.797	0.835	0.838	0.833	0.736
	AUC	0.787	0.806	0.792	0.780	0.773	0.780	0.792	0.775	0.769
	MCC	0.303	0.320	0.330	0.244	0.253	0.286	0.275	0.291	0.275

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: BorderSMOTE

mô hình GAN khác về recall, F1-score, AUC và MCC trên tập dữ liệu CM1. CopulaGAN đạt hiệu suất tốt nhất về precision trên các tập dữ liệu CM1 và KC1, với các giá trị cao nhất lần lượt là 0.881 và 0.851. Đối với tập dữ liệu PC1, WGANGP cho hiệu suất dự đoán tốt hơn so với tất cả các mô hình GAN khác, với các giá trị precision, recall, F1-score và MCC cao nhất lần lượt là 0.928, 0.939, 0.927 và 0.394. Một lần nữa, đối với các tập dữ liệu KC1, KC2 và JM1, CTGAN vượt trội hơn tất cả các mô hình GAN khác về precision, recall, F1-score và AUC. Ngoài ra, các giá trị MCC giữa các mô hình GAN và các mô hình cơ sở nhìn chung tương đương nhau.

Luận án đã so sánh hiệu quả của các biến thể GAN với bốn phương pháp lấy mẫu tăng cơ sở, bao gồm SMOTE, ADASYN, Borderline-SMOTE và RUS. Hầu hết các phương pháp được nghiên cứu đều được sử dụng để tạo ra một tập dữ liệu cân bằng,

sau đó làm đầu vào cho các bộ phân loại nhằm xây dựng và đánh giá các mô hình dự đoán. Từ phân tích thử nghiệm trên các tập dữ liệu lỗi phần mềm khác nhau, luận án nhận thấy rằng các phương pháp GAN mang lại hiệu suất tốt hơn trên các tập dữ liệu được sử dụng với các kỹ thuật học máy khác nhau, đặc biệt là CTGAN, WGANGP và VanillaGAN so với các phương pháp cơ sở. Mặt khác, Borderline-SMOTE, ADASYN và RUS cũng đạt kết quả tốt hơn trong một số trường hợp khi sử dụng với các mô hình XGBoost, Extra Trees và AdaBoost. Kết quả tổng quan cho thấy rằng hiệu suất của các mô hình GAN khác nhau phụ thuộc nhiều vào đặc điểm của tập dữ liệu cũng như các độ đo hiệu suất được sử dụng. Ngoài ra, các mô hình GAN khác và các độ đo hiệu suất khác cũng nên được xem xét để đạt được kết quả tốt nhất cho từng bài toán cụ thể.

2. Kết quả thử nghiệm của sự kết hợp giữa Autoencoders và các mô hình GAN

Kết quả của các mô hình dự đoán sử dụng sự kết hợp giữa Autoencoders và các phương pháp lấy mẫu tăng được trình bày trên các tập dữ liệu cân bằng được thể hiện trong các bảng sau. Các kết quả tốt nhất trong các Bảng 3.7 - 3.12 được tô đậm.

(a) **Random Forest**

Các mô hình dự đoán lỗi phần mềm sử dụng Autoencoders và CTGAN cho hiệu suất tốt hơn về recall và F1-score trên hầu hết các tập dữ liệu. Như thể hiện trong Bảng 3.7, trong khi CopulaGAN đạt giá trị trung bình cao nhất về recall và F1-score với lần lượt 0.850 và 0.837, thì CopulaGAN lại thể hiện tốt nhất về precision và MCC với các giá trị lần lượt là 0.846 và 0.287.

(b) **XGBoost**

Bảng 3.8 cho thấy rằng CopulaGAN đạt hiệu suất tốt hơn trên các tập dữ liệu CM1, KC2 và JM1. Tương tự, sự kết hợp giữa Autoencoders và CopulaGAN mang lại kết quả tốt nhất về precision, recall, F1-score, AUC và MCC với các giá trị lần lượt là 0.865, 0.844, 0.829, 0.824 và 0.313.

(c) **AdaBoost**

Về độ đo precision, CopulaGAN cũng đạt giá trị cao nhất trên hầu hết các tập dữ liệu. Trong Bảng 3.9, CTGAN cho kết quả tốt hơn về recall và F1-score với các giá trị lần lượt là 0.846 và 0.835.

(d) **HGBoost**

Như quan sát trong Bảng 3.10, CTGAN cho thấy hiệu suất tốt hơn so với các mô hình khác trên các tập dữ liệu CM1 và JM1. Cụ thể, theo kết quả trung

Bảng 3.7: Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy Random Forest

Dataset	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.858	0.889	0.873	0.885	0.893	0.898	0.902	0.902	0.898
	R	0.774	0.897	0.746	0.791	0.711	0.720	0.794	0.726	0.606
	F1	0.809	0.890	0.790	0.825	0.771	0.778	0.832	0.783	0.684
	AUC	0.685	0.667	0.646	0.786	0.730	0.848	0.819	0.799	0.202
	MCC	0.097	0.279	0.160	0.213	0.240	0.256	0.314	0.281	0.202
PC1	P	0.896	0.886	0.853	0.913	0.916	0.905	0.905	0.905	0.909
	R	0.855	0.903	0.436	0.764	0.801	0.721	0.772	0.691	0.654
	F1	0.873	0.894	0.509	0.819	0.845	0.790	0.824	0.769	0.742
	AUC	0.712	0.572	0.487	0.753	0.734	0.734	0.734	0.735	0.707
	MCC	0.125	0.053	0.65	0.201	0.238	0.147	0.167	0.139	0.151
KC1	P	0.818	0.810	0.806	0.831	0.826	0.828	0.827	0.829	0.826
	R	0.786	0.822	0.529	0.732	0.750	0.726	0.728	0.691	0.691
	F1	0.798	0.815	0.579	0.764	0.776	0.758	0.760	0.731	0.730
	AUC	0.780	0.749	0.648	0.774	0.748	0.761	0.772	0.773	0.755
	MCC	0.298	0.272	0.172	0.319	0.312	0.309	0.307	0.296	0.288
KC2	P	0.803	0.830	0.804	0.809	0.831	0.829	0.834	0.838	0.801
	R	0.776	0.835	0.695	0.784	0.776	0.803	0.811	0.797	0.743
	F1	0.785	0.827	0.716	0.792	0.792	0.811	0.819	0.807	0.762
	AUC	0.779	0.767	0.716	0.746	0.758	0.767	0.777	0.773	0.716
	MCC	0.349	0.432	0.322	0.369	0.428	0.433	0.452	0.449	0.335
JM1	P	0.741	0.750	0.599	0.774	0.762	0.768	0.770	0.775	0.774
	R	0.777	0.794	0.334	0.650	0.651	0.623	0.630	0.583	0.639
	F1	0.751	0.758	0.272	0.685	0.685	0.662	0.668	0.626	0.676
	AUC	0.661	0.614	0.498	0.706	0.670	0.699	0.705	0.693	0.701
	MCC	0.162	0.176	0.027	0.242	0.215	0.219	0.226	0.220	0.238
Trung bình										
	P	0.823	0.833	0.787	0.842	0.846	0.846	0.848	0.850	0.842
	R	0.794	0.850	0.548	0.744	0.738	0.718	0.747	0.698	0.666
	F1	0.803	0.837	0.573	0.777	0.774	0.760	0.780	0.743	0.719
	AUC	0.723	0.674	0.599	0.753	0.728	0.762	0.761	0.755	0.616
	MCC	0.206	0.242	0.123	0.269	0.287	0.273	0.293	0.277	0.243

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: Borderline-SMOTE

bình, CTGAN đạt các giá trị cao nhất về recall, F1-score và AUC lần lượt là 0.864, 0.851 và 0.789. Ngoài ra, CopulaGAN cũng đạt giá trị recall, F1-score và AUC cao thứ hai, lần lượt là 0.837, 0.839 và 0.774.

(e) Multilayer Perceptron

Trong Bảng 3.11, MLP cho hiệu suất tương tự như kết quả của HGBost. CTGAN cũng đạt các giá trị cao nhất về recall, F1-score và AUC, lần lượt là 0.845, 0.828 và 0.725.

(f) Extra Trees

Trong Bảng 3.12, trong hầu hết các trường hợp, giá trị precision của tất cả các

Bảng 3.8: Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy XGboost

Dữ liệu	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.870	0.899	0.877	0.853	0.907	0.889	0.851	0.834	0.851
	R	0.843	0.898	0.886	0.892	0.908	0.856	0.831	0.784	0.858
	F1	0.855	0.898	0.876	0.866	0.907	0.886	0.843	0.809	0.843
	AUC	0.734	0.702	0.762	0.780	0.771	0.705	0.697	0.653	0.727
	MCC	0.259	0.372	0.271	0.317	0.315	0.257	0.241	0.276	0.261
PC1	P	0.903	0.899	0.908	0.914	0.910	0.868	0.830	0.859	0.892
	R	0.923	0.879	0.911	0.937	0.870	0.907	0.805	0.845	0.855
	F1	0.907	0.885	0.909	0.867	0.894	0.876	0.769	0.848	0.893
	AUC	0.859	0.731	0.853	0.810	0.813	0.813	0.787	0.797	0.794
	MCC	0.302	0.302	0.333	0.316	0.317	0.310	0.294	0.300	0.295
KC1	P	0.824	0.804	0.830	0.813	0.864	0.856	0.790	0.768	0.799
	R	0.849	0.812	0.837	0.809	0.846	0.764	0.728	0.722	0.717
	F1	0.824	0.808	0.831	0.811	0.854	0.783	0.736	0.760	0.755
	AUC	0.742	0.700	0.774	0.757	0.829	0.798	0.800	0.793	0.768
	MCC	0.265	0.276	0.224	0.340	0.341	0.332	0.265	0.267	0.234
KC2	P	0.845	0.849	0.843	0.818	0.854	0.854	0.781	0.765	0.738
	R	0.843	0.852	0.846	0.770	0.803	0.803	0.687	0.682	0.660
	F1	0.842	0.851	0.838	0.786	0.702	0.702	0.745	0.756	0.723
	AUC	0.726	0.808	0.784	0.801	0.827	0.827	0.792	0.792	0.786
	MCC	0.304	0.290	0.306	0.231	0.311	0.267	0.306	0.269	0.257
JM1	P	0.792	0.745	0.758	0.767	0.789	0.809	0.757	0.743	0.759
	R	0.789	0.751	0.795	0.796	0.793	0.783	0.792	0.992	0.795
	F1	0.789	0.748	0.767	0.776	0.789	0.795	0.769	0.762	0.7744
	AUC	0.840	0.825	0.711	0.670	0.878	0.693	0.690	0.687	0.687
	MCC	0.228	0.206	0.233	0.236	0.282	0.322	0.290	0.234	0.232
Trung bình										
	P	0.847	0.839	0.843	0.833	0.865	0.855	0.802	0.794	0.808
	R	0.849	0.838	0.855	0.841	0.844	0.823	0.769	0.805	0.777
	F1	0.843	0.838	0.844	0.821	0.829	0.808	0.772	0.787	0.798
	AUC	0.78	0.753	0.777	0.764	0.824	0.767	0.753	0.744	0.752
	MCC	0.272	0.289	0.273	0.288	0.313	0.298	0.279	0.269	0.256

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: Borderline-SMOTE

kỹ thuật đều lớn hơn 0.74 với giá trị cao nhất đạt 0.921 trên tập dữ liệu PC1 và giá trị thấp nhất là 0.700 trên tập dữ liệu JM1. Khi so sánh các giá trị trung bình của tất cả các phương pháp lấy mẫu tăng trong Bảng 3.12, WGANGP đạt giá trị cao nhất về precision, recall và F1-score, lần lượt là 0.849, 0.866 và 0.856.

Trong phần này, luận án đã thực hiện so sánh giữa sự kết hợp của Autoencoders và các biến thể GAN với các phương pháp lấy mẫu tăng cơ sở khác. Từ kết quả thử nghiệm sử dụng các bộ phân loại khác nhau, có thể thấy rằng sự kết hợp giữa Autoencoders và CTGAN mang lại hiệu suất tốt nhất trên các tập dữ liệu khác nhau, tiếp theo là cặp Autoencoders và CopulaGAN. Ngoài ra, trích xuất đặc trưng trong dự đoán lỗi phần mềm có thể được thực hiện hiệu quả thông qua việc sử dụng Autoencoders để trích xuất các đặc trưng tối ưu.

Bảng 3.9: Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy AdaBoost

Dữ liệu	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.860	0.884	0.848	0.870	0.907	0.878	0.868	0.862	0.892
	R	0.800	0.874	0.766	0.886	0.760	0.783	0.814	0.783	0.629
	F1	0.823	0.877	0.798	0.877	0.807	0.820	0.835	0.814	0.707
	AUC	0.628	0.736	0.516	0.763	0.757	0.670	0.714	0.675	0.205
	MCC	0.188	0.252	0.136	0.176	0.323	0.192	0.140	0.112	0.205
PC1	P	0.897	0.910	0.897	0.908	0.920	0.907	0.914	0.914	0.910
	R	0.874	0.908	0.471	0.908	0.841	0.838	0.868	0.842	0.663
	F1	0.883	0.907	0.546	0.907	0.872	0.867	0.887	0.871	0.747
	AUC	0.629	0.728	0.530	0.676	0.733	0.655	0.679	0.675	0.690
	MCC	0.130	0.238	0.129	0.225	0.290	0.200	0.263	0.246	0.160
KC1	P	0.826	0.828	0.787	0.819	0.833	0.830	0.827	0.825	0.823
	R	0.805	0.837	0.464	0.799	0.778	0.769	0.773	0.748	0.693
	F1	0.813	0.831	0.501	0.807	0.798	0.791	0.793	0.774	0.732
	AUC	0.757	0.768	0.658	0.721	0.751	0.743	0.740	0.747	0.726
	MCC	0.331	0.339	0.127	0.306	0.348	0.332	0.324	0.308	0.280
KC2	P	0.814	0.812	0.780	0.761	0.842	0.803	0.798	0.788	0.768
	R	0.800	0.816	0.559	0.778	0.797	0.768	0.773	0.749	0.695
	F1	0.802	0.803	0.589	0.767	0.811	0.776	0.779	0.760	0.718
	AUC	0.674	0.642	0.659	0.584	0.777	0.637	0.647	0.654	0.622
	MCC	0.381	0.354	0.207	0.218	0.466	0.342	0.330	0.293	0.230
JM1	P	0.738	0.748	0.628	0.757	0.760	0.764	0.770	0.771	0.763
	R	0.767	0.796	0.423	0.674	0.670	0.673	0.666	0.638	0.651
	F1	0.748	0.756	0.350	0.703	0.700	0.703	0.698	0.675	0.686
	AUC	0.645	0.678	0.551	0.684	0.671	0.697	0.699	0.691	0.682
	MCC	0.156	0.167	0.151	0.209	0.214	0.227	0.240	0.232	0.218
Trung bình										
	P	0.827	0.836	0.788	0.823	0.852	0.836	0.835	0.832	0.831
	R	0.809	0.846	0.537	0.809	0.769	0.766	0.779	0.752	0.666
	F1	0.814	0.835	0.557	0.812	0.798	0.791	0.798	0.779	0.718
	AUC	0.667	0.710	0.583	0.686	0.738	0.680	0.696	0.688	0.585
	MCC	0.237	0.270	0.150	0.227	0.328	0.259	0.259	0.238	0.219

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: Borderline-SMOTE

Các thử nghiệm được thực hiện trên cặp RFE với các mô hình GAN và sự kết hợp giữa Autoencoders với các biến thể GAN cho thấy hiệu suất được cải thiện về precision, recall, F1-score và AUC. Đặc biệt, cặp RFE với CTGAN và Autoencoders với CTGAN đạt hiệu suất tốt nhất, tiếp theo là CopulaGAN, WGANGP và TGAN. Ngoài ra, có thể thấy rằng các phương pháp kết hợp được đề xuất đã cải thiện hiệu suất dự đoán lỗi từ 1–4% so với các mô hình cơ sở. Do đó, khuyến nghị sử dụng sự kết hợp giữa RFE hoặc Autoencoders với các mô hình GAN để xử lý vấn đề dư thừa đặc trưng và mất cân bằng lớp trước khi huấn luyện với các kỹ thuật học máy.

Bảng 3.10: Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy HGBost

Dữ liệu	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.880	0.893	0.861	0.855	0.871	0.886	0.865	0.876	0.900
	R	0.877	0.906	0.820	0.883	0.917	0.886	0.866	0.869	0.651
	F1	0.876	0.897	0.832	0.867	0.890	0.884	0.864	0.870	0.726
	AUC	0.755	0.813	0.677	0.711	0.788	0.778	0.791	0.768	0.237
	MCC	0.219	0.315	0.121	0.070	0.177	0.264	0.134	0.201	0.237
PC1	P	0.922	0.920	0.802	0.905	0.926	0.916	0.917	0.912	0.914
	R	0.924	0.926	0.842	0.926	0.936	0.909	0.919	0.903	0.658
	F1	0.921	0.919	0.837	0.911	0.926	0.912	0.918	0.907	0.743
	AUC	0.769	0.773	0.798	0.704	0.743	0.770	0.781	0.756	0.743
	MCC	0.328	0.300	0.214	0.176	0.342	0.292	0.306	0.260	0.174
KC1	P	0.842	0.825	0.794	0.816	0.827	0.831	0.831	0.830	0.831
	R	0.845	0.836	0.437	0.844	0.849	0.821	0.829	0.822	0.684
	F1	0.842	0.829	0.450	0.822	0.830	0.825	0.830	0.825	0.726
	AUC	0.805	0.806	0.670	0.733	0.782	0.801	0.797	0.797	0.776
	MCC	0.395	0.329	0.161	0.276	0.317	0.353	0.354	0.353	0.298
KC2	P	0.847	0.844	0.807	0.859	0.800	0.822	0.827	0.813	0.828
	R	0.832	0.849	0.711	0.865	0.808	0.805	0.808	0.789	0.762
	F1	0.836	0.840	0.733	0.856	0.801	0.811	0.813	0.798	0.780
	AUC	0.819	0.820	0.777	0.810	0.781	0.803	0.809	0.788	0.783
	MCC	0.494	0.473	0.330	0.525	0.340	0.416	0.425	0.384	0.408
JM1	P	0.766	0.765	0.743	0.745	0.761	0.764	0.765	0.759	0.773
	R	0.799	0.801	0.727	0.783	0.802	0.764	0.764	0.750	0.648
	F1	0.773	0.772	0.721	0.757	0.767	0.764	0.764	0.754	0.684
	AUC	0.732	0.732	0.724	0.670	0.719	0.720	0.722	0.713	0.710
	MCC	0.235	0.227	0.222	0.173	0.210	0.245	0.248	0.231	0.239
Trung bình										
	P	0.851	0.849	0.849	0.836	0.837	0.844	0.841	0.838	0.849
	R	0.855	0.864	0.863	0.860	0.862	0.837	0.837	0.826	0.681
	F1	0.850	0.851	0.851	0.842	0.843	0.839	0.838	0.831	0.732
	AUC	0.776	0.789	0.788	0.726	0.763	0.774	0.780	0.764	0.650
	MCC	0.334	0.329	0.329	0.244	0.277	0.314	0.293	0.286	0.271

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: Borderline-SMOTE

3.5.2 Trả lời cho Câu hỏi nghiên cứu 2: Trong số các kỹ thuật tiên tiến nhất hiện nay, biến thể nào của GAN vượt trội hơn trong việc xử lý mất cân bằng dữ liệu trong dự đoán lỗi phần mềm?

Các phương pháp lấy mẫu dữ liệu khác nhau được biểu diễn bằng biểu đồ trong Hình 3.2, trong đó trục x biểu thị các phương pháp lấy mẫu được sử dụng và trục y biểu thị các giá trị AUC. Từ biểu đồ, luận án quan sát thấy rằng CTGAN kết hợp với Extra Trees vượt trội hơn tất cả các phương pháp lấy mẫu khác với giá trị AUC lần lượt là 0.777, 0.830, 0.830 và 0.759 trên các tập dữ liệu CM1, KC1, KC2 và JM1. Ngoài ra, các giá trị AUC trung bình cũng được so sánh để đánh giá các biến thể của GAN trên các mô hình học máy khác nhau. Kết quả cho thấy giá trị AUC trung bình của các mô hình GAN được đề xuất đều

Bảng 3.11: Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy MLP

Dataset	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.856	0.854	0.856	0.89	0.891	0.897	0.897	0.904	0.885
	R	0.663	0.866	0.703	0.783	0.754	0.683	0.757	0.737	0.609
	F1	0.731	0.859	0.76	0.823	0.799	0.748	0.803	0.791	0.69
	AUC	0.642	0.774	0.705	0.720	0.710	0.767	0.751	0.791	0.755
	MCC	0.161	0.175	0.165	0.254	0.24	0.243	0.265	0.286	0.155
PC1	P	0.894	0.887	0.889	0.916	0.818	0.909	0.914	0.909	0.923
	R	0.824	0.906	0.601	0.847	0.721	0.690	0.760	0.674	0.655
	F1	0.853	0.896	0.658	0.875	0.752	0.767	0.817	0.757	0.741
	AUC	0.765	0.734	0.738	0.71	0.742	0.718	0.713	0.681	0.724
	MCC	0.111	0.163	0.140	0.264	0.275	0.151	0.212	0.154	0.219
KC1	P	0.825	0.801	0.764	0.833	0.779	0.824	0.819	0.825	0.822
	R	0.744	0.823	0.540	0.77	0.765	0.703	0.719	0.636	0.695
	F1	0.765	0.808	0.572	0.792	0.768	0.739	0.751	0.684	0.733
	AUC	0.743	0.692	0.739	0.736	0.607	0.758	0.742	0.743	0.706
	MCC	0.297	0.233	0.108	0.341	0.271	0.287	0.277	0.26	0.278
KC2	P	0.798	0.828	0.801	0.836	0.915	0.832	0.827	0.813	0.799
	R	0.778	0.835	0.692	0.784	0.726	0.814	0.795	0.757	0.738
	F1	0.783	0.823	0.701	0.8	0.793	0.818	0.803	0.773	0.755
	AUC	0.729	0.724	0.764	0.776	0.744	0.746	0.729	0.717	0.651
	MCC	0.333	0.418	0.290	0.442	0.202	0.442	0.421	0.368	0.318
JM1	P	0.728	0.745	0.703	0.765	0.772	0.769	0.767	0.776	0.768
	R	0.734	0.795	0.767	0.673	0.603	0.623	0.612	0.582	0.616
	F1	0.723	0.754	0.726	0.703	0.644	0.661	0.653	0.624	0.653
	AUC	0.705	0.699	0.704	0.670	0.686	0.685	0.682	0.688	0.682
	MCC	0.122	0.158	0.111	0.229	0.221	0.221	0.214	0.221	0.216
Trung bình										
	P	0.820	0.823	0.803	0.848	0.835	0.846	0.845	0.845	0.839
	R	0.749	0.845	0.661	0.771	0.714	0.703	0.729	0.677	0.663
	F1	0.771	0.828	0.683	0.799	0.751	0.747	0.765	0.726	0.714
	AUC	0.717	0.725	0.73	0.722	0.698	0.735	0.723	0.724	0.704
	MCC	0.205	0.229	0.163	0.306	0.242	0.269	0.278	0.258	0.237

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: Borderline-SMOTE

lớn hơn 0.704. CTGAN kết hợp với Extra Trees đạt giá trị AUC trung bình cao nhất, trong khi ADASYN kết hợp với MLP đạt giá trị AUC trung bình thấp nhất trong số chín phương pháp lấy mẫu, lần lượt là 0.806 và 0.605. TGAN và CopulaGAN cho kết quả tương tự như các phương pháp SMOTE, ADASYN, RUS và BorderMOTE.

Do đó, có thể thấy rằng CTGAN là phương pháp hiệu quả nhất để áp dụng trong dự đoán lỗi phần mềm nhằm giải quyết vấn đề mất cân bằng dữ liệu trước khi sử dụng các kỹ thuật học máy, tiếp theo là WGANGP và VanillaGAN. Nguyên nhân tiềm năng dẫn đến hiệu suất vượt trội này là do các biến thể của kiến trúc GAN có khả năng học được phân phối của các lớp lỗi, từ đó tạo ra các mẫu tổng hợp đa dạng và chất lượng cao hơn so với các kỹ thuật lấy mẫu dữ liệu khác.

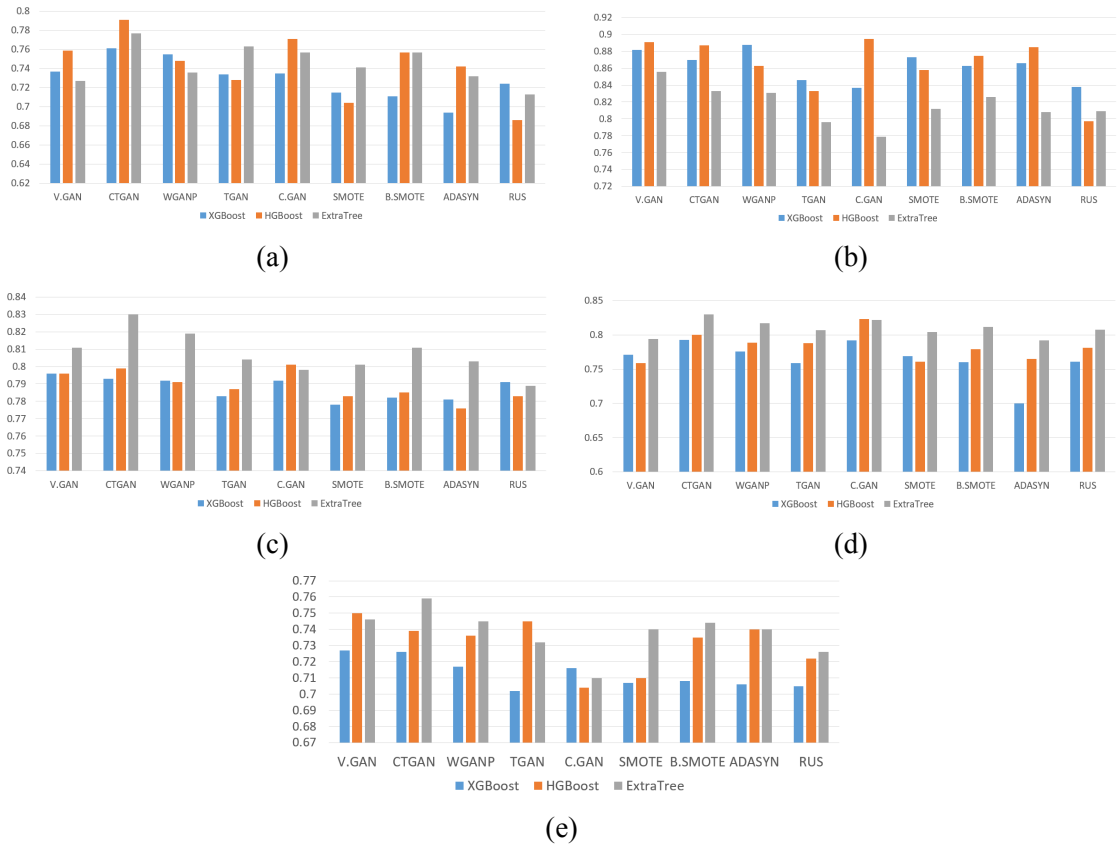
Bảng 3.12: Kết quả Autoencoders và các mô hình lấy mẫu tăng trên mô hình học máy ExTra Trees

Dataset	Độ đo đánh giá	V.GAN	CTGAN	WGANGP	TGAN	C.GAN	SMOTE	B.SMOTE	ADASYN	RUS
CM1	P	0.868	0.892	0.863	0.859	0.88	0.885	0.857	0.856	0.856
	R	0.853	0.89	0.902	0.874	0.879	0.84	0.862	0.807	0.869
	F1	0.863	0.891	0.897	0.863	0.879	0.845	0.859	0.777	0.850
	AUC	0.728	0.793	0.739	0.726	0.844	0.773	0.699	0.714	0.759
	MCC	0.259	0.271	0.313	0.247	0.26	0.32	0.274	0.287	0.292
PC1	P	0.901	0.89	0.921	0.905	0.908	0.879	0.907	0.9	0.855
	R	0.879	0.891	0.93	0.914	0.921	0.861	0.871	0.857	0.87
	F1	0.888	0.891	0.922	0.907	0.913	0.865	0.889	0.875	0.862
	AUC	0.792	0.781	0.773	0.751	0.749	0.743	0.815	0.775	0.789
	MCC	0.34	0.304	0.236	0.338	0.224	0.301	0.302	0.277	0.216
KC1	P	0.798	0.816	0.835	0.834	0.804	0.81	0.808	0.793	0.799
	R	0.782	0.814	0.847	0.845	0.803	0.834	0.865	0.838	0.769
	F1	0.806	0.814	0.837	0.837	0.803	0.826	0.853	0.823	0.792
	AUC	0.795	0.794	0.797	0.798	0.786	0.783	0.795	0.785	0.759
	MCC	0.216	0.303	0.234	0.269	0.205	0.213	0.335	0.306	0.251
KC2	P	0.864	0.833	0.851	0.832	0.832	0.783	0.813	0.783	0.746
	R	0.857	0.866	0.854	0.837	0.837	0.71	0.713	0.774	0.776
	F1	0.858	0.842	0.846	0.83	0.827	0.73	0.762	0.78	0.769
	AUC	0.82	0.793	0.808	0.783	0.784	0.808	0.772	0.766	0.793
	MCC	0.268	0.282	0.229	0.223	0.232	0.198	0.235	0.279	0.253
JM1	P	0.768	0.751	0.770	0.763	0.804	0.798	0.763	0.756	0.709
	R	0.718	0.75	0.796	0.795	0.83	0.759	0.728	0.723	0.756
	F1	0.704	0.751	0.778	0.767	0.803	0.705	0.745	0.734	0.758
	AUC	0.743	0.745	0.752	0.71	0.781	0.724	0.719	0.716	0.717
	MCC	0.198	0.227	0.215	0.274	0.207	0.187	0.262	0.257	0.231
Trung bình										
	P	0.840	0.836	0.849	0.839	0.846	0.831	0.83	0.818	0.793
	R	0.818	0.842	0.866	0.853	0.854	0.801	0.808	0.8	0.808
	F1	0.824	0.838	0.856	0.841	0.845	0.794	0.822	0.798	0.806
	AUC	0.776	0.781	0.774	0.754	0.789	0.766	0.76	0.751	0.763
	MCC	0.256	0.277	0.245	0.27	0.226	0.244	0.282	0.281	0.249

V.GAN: VanillaGAN; C.GAN: CopulaGAN; B.SMOTE: Borderline-SMOTE

3.5.3 Kết quả kiểm định thống kê

Để đánh giá hiệu quả năm biến thể GAN (VanillaGAN, CTGAN, WGANGP, CopulaGAN và TGAN) so với các phương pháp lấy mẫu khác (SMOTE, Borderline-SMOTE, ADASYN và RUS), luận án đã thực hiện tính toán kiểm định Wilcoxon (với mức ý nghĩa $\alpha = 0.05$). Các Bảng 3.13 - 3.17 trình bày kết quả kiểm định thống kê của tất cả các nhóm được xem xét theo các thước đo Precision, Recall, F1-score và MCC. Mỗi bảng bao gồm thông tin về giá trị P-value, hiệu ứng r và chênh lệch trung bình. Đối với giá trị P-value của kiểm định, nếu giá trị này nhỏ hơn 0.05, sự khác biệt về hiệu suất được coi là có ý nghĩa thống kê, được đánh dấu bằng dấu (*), đồng thời bác bỏ giả thuyết null. Giá trị r cho biết mức độ khác biệt về hiệu suất giữa các phương pháp. Giá trị chênh lệch trung bình cho từng nhóm thể hiện phương pháp nào có giá trị trung bình vượt trội hơn.



Hình 3.2: Giá trị AUC của các phương pháp lấy mẫu khác nhau trên các tập dữ liệu khác nhau (3a) CM1, (3b) PC1, (3c) KC1, (3d) KC2, (3e) JM1

Bảng 3.13 trình bày kết quả kiểm định thống kê giữa VanillaGAN và các phương pháp lấy mẫu khác khi sử dụng thuật toán Extra Trees. Các giá trị P-value của hai cặp so sánh VanillaGAN với BorderSMOTE và VanillaGAN với RUS đều nhỏ hơn 0.05, cho thấy sự khác biệt hiệu suất có ý nghĩa thống kê đối với các thước đo Recall, F1-score và MCC. Hơn nữa, giá trị r đều lớn hơn 0.6 trong tất cả các trường hợp, ngoại trừ Recall của nhóm RUS, cho thấy sự khác biệt đáng kể về hiệu suất. Giá trị chênh lệch trung bình dương cho thấy các phương pháp lấy mẫu tăng được đề xuất mang lại hiệu suất tốt hơn so với các mô hình cơ sở. Từ Bảng 3.14, có thể thấy rằng giá trị P-value nhỏ hơn 0.05 đối với tất cả các nhóm so sánh, ngoại trừ nhóm SMOTE. Giá trị r cao trong hầu hết các nhóm và giá trị chênh lệch trung bình dương đối với đa số nhóm, điều này nhấn mạnh hiệu suất vượt trội của phương pháp CTGAN. Bảng 3.15 thể hiện nhóm so sánh giữa WGANP và các phương pháp khác. Từ bảng này, có thể thấy rằng P-value của Recall nhỏ hơn 0.05. Một lần nữa, giá trị r cho thấy sự khác biệt về hiệu suất từ trung bình đến lớn. Trong hầu hết các trường hợp, giá trị chênh lệch trung bình dương minh chứng rằng phương pháp WGANP có hiệu suất tốt hơn so với các kỹ thuật khác. Trong các Bảng 3.16 và 3.17, khi so sánh thống kê giữa CopulaGAN, TGAN và các phương pháp lấy mẫu khác, giá trị

P-value nhỏ hơn 0.05 đối với thước đo Precision và MCC. Tóm lại, từ kết quả kiểm định cho thấy các mô hình sinh dữ liệu dựa trên GAN nhìn chung đạt hiệu suất cao hơn so với các phương pháp lấy mẫu truyền thống. Trong đó, VanillaGAN, CTGAN và WGANGP thể hiện sự khác biệt có ý nghĩa thống kê ($P\text{-value} < 0.05$) với các giá trị r đều cao trong các nhóm so sánh trên giá trị Recall, F1-score và MCC. Tổng thể, CTGAN được xác định là phương pháp sinh mẫu hiệu quả nhất, tiếp theo là CopullaGAN và TGAN, mang lại cải thiện có ý nghĩa thống kê trong mô hình dự đoán lỗi phần mềm.

Bảng 3.13: Kết quả kiểm định thống kê cho so sánh giữa VanillaGAN và các phương pháp lấy mẫu tăng

Nhóm so sánh	Measure	Precision	Recall	F1-Score	MCC
SMOTE	P-value	0.343	0.048*	0.345	0.686
	<i>Effect r</i>	0.971	0.910	0.954	0.698
	Mean-difference	0.003	0.023	0.009	0.017
B.SMOTE	P-value	0.496	0.043*	0.043*	0.025*
	<i>Effect r</i>	0.996	0.990	0.995	0.937
	Mean-difference	0.001	0.020	0.006	0.028
ADASYN	P-value	0.496	0.042*	0.080	0.500
	<i>Effect r</i>	0.987	0.901	0.964	0.836
	Mean-difference	0.003	0.029	0.011	0.013
RUS	P-value	0.893	0.043*	0.043*	0.045*
	<i>Effect r</i>	0.995	-0.118	0.601	0.967
	Mean-difference	0.0006	0.168	0.109	0.028

Bảng 3.15: Kết quả kiểm định thống kê cho so sánh giữa WGANP và các phương pháp lấy mẫu tăng

Nhóm so sánh	Measure	Precision	Recall	F1-Score	MCC
SMOTE	P-value	0.174	0.043*	0.138	0.138
	<i>Effect r</i>	0.996	0.867	0.961	0.844
	Mean-difference	0.003	0.029	0.010	0.043
B.SMOTE	P-value	0.892	0.043*	0.144	0.043*
	<i>Effect r</i>	0.992	0.985	0.994	0.972
	Mean-difference	0.001	0.026	0.007	0.055
ADASYN	P-value	0.225	0.042*	0.104	0.043*
	<i>Effect r</i>	0.999	0.863	0.961	0.935
	Mean-difference	0.003	0.035	0.012	0.039
RUS	P-value	0.492	0.043*	0.043*	0.043*
	<i>Effect r</i>	0.974	-0.153	0.509	0.799
	Mean-difference	0.0008	0.175	0.109	0.054

Bảng 3.16: Kết quả kiểm định thống kê cho so sánh giữa CopulaGAN và các phương pháp lấy mẫu tăng

Nhóm so sánh	Measure	Precision	Recall	F1-Score	MCC
SMOTE	P-value	0.225	0.050*	0.500	0.043*
	<i>Effect r</i>	0.919	0.919	0.931	0.889
	Mean-difference	0.014	-0.019	-0.010	-0.033
B.SMOTE	P-value	0.068	0.050*	0.138	0.225
	<i>Effect r</i>	0.949	0.887	0.948	0.922
	Mean-difference	0.012	-0.022	-0.013	-0.021
ADASYN	P-value	0.225	0.138	0.465	0.080
	<i>Effect r</i>	0.924	0.892	0.931	0.921
	Mean-difference	0.014	-0.0013	-0.008	-0.037
RUS	P-value	0.225	0.043*	0.043*	0.345
	<i>Effect r</i>	0.936	0.119	0.591	0.814
	Mean-difference	0.011	0.125	0.089	-0.022

Bảng 3.14: Kết quả kiểm định thống kê cho so sánh giữa CTGAN và các phương pháp lấy mẫu tăng

Nhóm so sánh	Measure	Precision	Recall	F1-Score	MCC
SMOTE	P-value	0.138	0.043*	0.144	0.223
	<i>Effect r</i>	0.954	0.897	0.924	0.825
	Mean-difference	0.011	0.032	0.017	0.033
B.SMOTE	P-value	0.042*	0.043*	0.043*	0.043*
	<i>Effect r</i>	0.993	0.985	0.980	0.969
	Mean-difference	0.009	0.029	0.015	0.045
ADASYN	P-value	0.043*	0.043*	0.043*	0.104
	<i>Effect r</i>	0.974	0.924	0.945	0.921
	Mean-difference	0.011	0.038	0.020	0.0292
RUS	P-value	0.043*	0.043*	0.043*	0.080
	<i>Effect r</i>	0.998	0.150	0.663	0.835
	Mean-difference	0.008	0.177	0.117	0.044

Bảng 3.17: Kết quả kiểm định thống kê cho so sánh giữa TGAN và các phương pháp lấy mẫu tăng

Nhóm so sánh	Measure	Precision	Recall	F1	MCC
SMOTE	P-value	0.046*	0.345	0.225	0.043*
	<i>Effect r</i>	0.955	0.391	0.401	0.888
	Mean-difference	-0.011	-0.021	-0.025	-0.042
B.SMOTE	P-value	0.042*	0.223	0.225	0.038*
	<i>Effect r</i>	0.963	0.657	0.583	0.968
	Mean-difference	-0.013	-0.024	-0.027	-0.030
ADASYN	P-value	0.176	0.686	0.345	0.043*
	<i>Effect r</i>	0.955	0.348	0.372	0.946
	Mean-difference	-0.011	-0.015	-0.022	-0.046
RUS	P-value	0.044*	0.043*	0.043*	0.225
	<i>Effect r</i>	0.942	-0.565	-0.208	0.856
	Mean-difference	-0.014	0.123	0.074	-0.031

3.6 Kết chương

Chương này tập trung vào hai vấn đề chính: dữ liệu có các đặc trưng dư thừa/không liên quan đến lỗi phần mềm và lớp mất cân bằng là những yếu tố chính ảnh hưởng đáng kể đến hiệu suất của các mô hình dự đoán lỗi phần mềm. Các kỹ thuật chọn đặc trưng, giúp lựa chọn tập con tối ưu của dữ liệu, được áp dụng để giảm bớt các đặc trưng không liên quan và dư thừa. Các phương pháp lấy mẫu dữ liệu, nhằm tạo ra các mẫu lớp mới và cân bằng dữ liệu, đã được nhiều nhà nghiên cứu đề xuất để giải quyết vấn đề này. Trên cơ sở tổng quan các công trình trước, luận án đã chỉ ra rằng các kỹ thuật lấy mẫu truyền thống như SMOTE, ADASYN hay RUS tuy cải thiện được phân phối lớp, nhưng vẫn tồn tại nhiều hạn chế do phụ thuộc vào các mẫu lân cận, dẫn đến hiện tượng quá khớp và thiếu đa dạng dữ liệu. Trên cơ sở phân tích các nghiên cứu liên quan, luận án đã tập trung khảo sát các mô hình tạo sinh dữ liệu dựa trên GAN kết hợp với phương pháp lựa chọn đặc trưng RFE đã được xác định là hiệu quả nhất trong Chương 2. Sự kết hợp này hướng đến giải quyết đồng thời hai thách thức quan trọng của dữ liệu dự đoán lỗi phần mềm là loại bỏ các đặc trưng dư thừa và khắc phục hiện tượng mất cân bằng giữa lớp lỗi và lớp không lỗi.

Cụ thể, luận án trình bày một phân tích so sánh các biến thể của GAN, đại diện cho các kỹ thuật lấy mẫu dữ liệu tiên tiến nhất để xử lý dữ liệu mất cân bằng. Cụ thể, luận án đã đề xuất hướng tiếp cận sử dụng các biến thể của mạng sinh đối kháng GAN bao gồm VanillaGAN, CTGAN, WGANP, CopulaGAN và TGAN cho tổng hợp dữ liệu lỗi phần mềm dạng bảng. Ngoài ra, để chọn một tập đặc trưng tối ưu, luận án sử dụng kỹ thuật lựa chọn đặc trưng RFE và trích xuất đặc trưng Autoencoders trên tập dữ liệu mất cân

bằng, sau đó áp dụng các biến thể GAN để tạo ra các mẫu mới cho lớp thiểu số nhằm cân bằng tập dữ liệu. Tiếp theo, luận án kiểm tra hiệu suất của các thuật toán học máy trên tập dữ liệu đã được cân bằng (Random Forest, XGBoost, MLP, AdaBoost, Extra Trees và HistGradientBoosting), sử dụng tập đặc trưng tối ưu được trích xuất từ các bước trước đó. Luận án đã tiến hành các thử nghiệm trên năm tập dữ liệu lỗi phần mềm và sử dụng các chỉ số đánh giá như precision, recall, F1-score, AUC và MCC để đánh giá hiệu quả của các mô hình GAN trong việc lấy mẫu dữ liệu bổ sung. Ngoài ra, nghiên cứu này đã thực hiện phân tích so sánh giữa hiệu suất của các biến thể GAN và các phương pháp lấy mẫu dữ liệu truyền thống. Kết quả thử nghiệm cho thấy CTGAN, WGANP và VanillaGAN mang lại hiệu suất dự đoán tốt hơn và vượt trội so với các phương pháp lấy mẫu dữ liệu truyền thống được sử dụng trong nghiên cứu, bao gồm SMOTE, ADASYN, RUS và Borderline-SMOTE. Trong đó, CTGAN là phương pháp có hiệu suất tốt nhất, tiếp theo là WGANP và VanillaGAN trong việc tổng hợp mẫu mới để cân bằng dữ liệu. CopulaGAN và TGAN cũng đạt được hiệu quả tương đương hoặc cao hơn so với các phương pháp lấy mẫu truyền thống. Những phương pháp dựa trên GAN này có thể được khai thác rộng rãi để xử lý vấn đề dữ liệu mất cân bằng, góp phần xây dựng các mô hình dự đoán lỗi phần mềm có độ chính xác và hiệu suất cao. Bên cạnh đó, nghiên cứu cũng chỉ ra vai trò quan trọng của bước tiền xử lý dữ liệu và chọn/trích xuất đặc trưng. Kỹ thuật RFE và Autoencoders không chỉ giúp giảm nhiễu và độ dư thừa trong dữ liệu mà còn tăng cường tính ổn định của mô hình khi kết hợp với các phương pháp sinh dữ liệu GAN. Mặc dù các thực nghiệm trong Chương 3 được tiến hành trên các dự án Java, các phương pháp được đề xuất có tính tổng quát cao và không phụ thuộc trực tiếp vào ngôn ngữ lập trình. Quy trình kết hợp lựa chọn đặc trưng và cân bằng dữ liệu bằng các mô hình GAN được xây dựng trên các tập đặc trưng đã được trích xuất từ mã nguồn, do đó có thể mở rộng áp dụng cho các dự án được phát triển bằng các ngôn ngữ lập trình khác khi có tập dữ liệu với cấu trúc và đặc trưng tương tự. Chẳng hạn, các bộ dữ liệu lỗ hổng phần mềm viết bằng ngôn ngữ C/C++ như Asterisk, FFmpeg, LibPNG, LibTiff, Pidgin, VLC Media Player và Xen được tổ chức ở mức hàm (function-level), trong đó mỗi mẫu dữ liệu tương ứng với một hàm của chương trình và được gán nhãn nhị phân (vulnerable hoặc non-vulnerable). Với đặc điểm cũng tồn tại hiện tượng mất cân bằng nghiêm trọng giữa lớp thiểu số và lớp đa số, các phương pháp cân bằng dữ liệu dựa trên GAN được đề xuất trong luận án hoàn toàn có tiềm năng áp dụng để cải thiện chất lượng dữ liệu huấn luyện, từ đó nâng cao hiệu quả của các mô hình phát hiện lỗ hổng phần mềm trên các dự án C/C++. Điều này cho thấy kết quả của Chương 3 không chỉ có ý nghĩa đối với các dự án Java mà còn có khả năng mở rộng sang các dự án sử dụng ngôn ngữ lập trình khác, góp phần nâng cao tính khái quát và khả năng ứng dụng của các phương pháp được đề xuất. Tóm lại, những kết quả đạt được trong chương này khẳng định rằng việc kết hợp lựa chọn đặc trưng và xử lý dữ liệu mất cân bằng là giải

pháp tối ưu để xây dựng tập dữ liệu đầu vào cho các mô hình dự đoán lỗi phần mềm. Tập dữ liệu sau khi được tối ưu hóa không chỉ nâng cao khả năng phát hiện các mô-đun có lỗi mà còn cải thiện khả năng tổng quát hóa và tính ổn định của các mô hình dự đoán. Đây là cơ sở quan trọng để tiếp tục nghiên cứu tương lai, đề xuất và đánh giá các mô hình dự đoán lỗi phần mềm trên tập dữ liệu đã được tối ưu, qua đó nâng cao hiệu quả dự đoán và khẳng định tính khả thi của hướng tiếp cận được đề xuất trong luận án.

Các kết quả nghiên cứu này đã được công bố trên Kỷ yếu Hội nghị Khoa học công nghệ Quốc gia lần thứ XVI về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR) 2023¹ và Tạp chí Applied Intelligence.²

¹Ha Thi Minh Phuong, Nguyen Thanh Long, Nguyen Thanh Binh, “**A combination of feature selection and data sampling techniques for software fault prediction**”, *FAIR - Fundamental and Applied Information Technology*, Đà Nẵng, Pages: 258-265, 2023.

²Ha Thi Minh Phuong, Pham Vu Thu Nguyet, Nguyen Huu Nhat Minh, Le Thi My Hanh, Nguyen Thanh Binh “**A comparative study of handling imbalanced data using generative adversarial networks for machine learning based software fault prediction**”, *Applied Intelligence*, Appl Intell 55, 280 (2025), Springer, Published: 08 January 2025, URL: <https://doi.org/10.1007/s10489-024-05930-z> (SCIE, Q2, IF 3.5).

Chương 4

Dự đoán lỗi phần mềm dựa trên học sâu

Trong chương này, luận án nghiên cứu áp dụng kỹ thuật học sâu để xây dựng mô hình dự đoán lỗi dựa trên việc học được các đặc trưng ngữ nghĩa và cú pháp của các mã nguồn chương trình phần mềm, từ đó nâng cao hiệu quả của việc dự đoán lỗi so với các mô hình dự đoán lỗi truyền thống dựa trên các độ đo phần mềm.

4.1 Giới thiệu

Các mô hình học sâu là một hướng tiếp cận đầy hứa hẹn trong việc học các đặc trưng ngữ nghĩa ẩn nhằm nâng cao hiệu suất của các mô hình dự đoán lỗi. Tuy nhiên, các mạng tuần tự không thể biểu diễn đầy đủ thông tin cú pháp và ngữ nghĩa được trích xuất từ cú pháp trừu tượng, đồng thời gặp khó khăn trong việc nắm bắt các phụ thuộc phức tạp trong mã nguồn [36]. Kiến trúc Transformer ban đầu được phát triển dựa trên cơ chế self-attention và gần đây đã thu hút sự quan tâm đáng kể. Transformer lần đầu tiên được đề xuất bởi Vaswani và cộng sự [147] và sử dụng cơ chế self-attention, cho phép tính toán theo phương pháp song song. Hơn nữa, Transformer cũng giúp khắc phục vấn đề vanishing (gradient biến mất), tức là đạo hàm của hàm mất mát theo trọng số trở nên rất nhỏ trong quá trình lan truyền ngược, dẫn đến việc các trọng số không được cập nhật đủ, hoặc gần như không thay đổi trong quá trình huấn luyện) và exploding gradient (gradient bùng nổ) là hiện tượng đạo hàm của hàm mất mát theo trọng số trở nên rất lớn trong quá trình lan truyền ngược, khiến việc cập nhật trọng số trở nên không ổn định bằng cách sử dụng kỹ thuật chuẩn hóa các lớp để chuẩn hóa thống kê của đầu ra từ các lớp ẩn [148].

Chương này đề xuất mô hình dự đoán lỗi dựa trên Transformer, sử dụng Gensim FastText với nhúng (embedding) các từ tổ (token) từ cây cú pháp trừu tượng (AST). AST

là một cây biểu diễn cấu trúc của mã nguồn [149], được sử dụng để mô tả cấu trúc cú pháp và thông tin ngữ nghĩa của mã nguồn, chẳng hạn như cấu trúc luồng điều khiển với các câu lệnh “while, do, for” cũng như tên của phương thức. Cụ thể, đối với mỗi tệp dự án, trước tiên luận án xây dựng AST từ mã nguồn, sau đó trích xuất các nút AST để tạo các vec-tơ token. Do đó, mỗi tệp mã nguồn được biểu diễn bằng các vec-tơ token. Tiếp theo, luận án sử dụng một mô hình biểu diễn từ dưới dạng vec-tơ đã được huấn luyện trước, có tên Gensim FastText để chuyển đổi các vec-tơ token thành các vec-tơ số có độ dài cố định. Các vec-tơ thu thập được này sau đó được đưa vào mô hình Transformer để thực hiện dự đoán lỗi. Đóng góp chính của chương này là đề xuất mô hình dự đoán lỗi dựa trên FastText-Transformer, có khả năng học các đặc trưng ngữ nghĩa quan trọng của chương trình phần mềm giúp cải thiện đáng kể hiệu quả dự đoán lỗi. Mô hình có thể nắm bắt thông tin ngữ nghĩa trong các biểu diễn vec-tơ của các token mã nguồn, áp dụng cho cả trong cùng dự án và trong liên dự án. Ngoài ra, luận án thực hiện kiểm định Wilcoxon để đánh giá mức độ khác biệt có ý nghĩa thống kê giữa mô hình FastText-Transformer và các mô hình đối sánh. Kết quả đánh giá trên 10 dự án Java cho thấy mô hình được đề xuất của luận án vượt trội hơn so với các mô hình dự đoán lỗi mới hiện nay.

4.2 Các nghiên cứu liên quan

Các nghiên cứu gần đây đã áp dụng các mô hình học sâu để trích xuất các thông tin ngữ nghĩa và cú pháp của mã nguồn. Các chương trình có cú pháp được xác định rõ ràng và có thể được biểu diễn bằng cây cú pháp trừu tượng [150] và đã được sử dụng thành công để nắm bắt các thông tin mã nguồn. Nhiều nghiên cứu đã chỉ ra rằng thông tin ngữ nghĩa của chương trình có ích cho các tác vụ như tự động hoàn thành mã và phát hiện lỗi. Thông tin ngữ nghĩa này cũng có thể hữu ích trong việc mô tả lỗi nhằm cải thiện dự đoán lỗi. Cụ thể, các thông tin ngữ nghĩa này có thể đưa ra các dự đoán chính xác, các đặc trưng cần có tính phân biệt, tức là có khả năng phân biệt các vùng mã khác nhau.

Để trích xuất các đặc trưng ngữ nghĩa của mã nguồn, Wang và cộng sự [151] đã áp dụng DBN giúp nắm bắt các đặc trưng ngữ nghĩa từ các nút AST nhúng trong mã nguồn, dẫn đến hiệu suất tốt hơn so với các mô hình truyền thống. Qiao và Wang [62] đã áp dụng kỹ thuật học sâu để sinh ra các đặc trưng tối ưu cho dự đoán lỗi tức thời. Kết quả thử nghiệm cho thấy mô hình đề xuất đạt được cải thiện đáng kể về giá trị trung bình của recall. Cụ thể, giá trị recall tăng 15,6% và giá trị popt tăng 8,1% so với các mô hình đề xuất tiên tiến. Li và cộng sự [152] đã đề xuất mô hình dự đoán lỗi dựa trên CNN có tên DP-CNN để trích xuất được cả các đặc trưng ngữ nghĩa và cấu trúc từ cây cú pháp trừu tượng và kết quả chỉ ra rằng CNN đạt hiệu quả dự đoán lỗi tốt hơn so với DP-CNN. Liang và cộng sự [153] đã trình bày mô hình LSTM để nắm bắt được các thông tin ngữ nghĩa trong mã

nguồn. Kết quả thử nghiệm của họ đã chứng minh mô hình đề xuất đã đạt kết quả tốt hơn so với ba mô hình dự đoán lỗi được đề xuất gần nhất đối với ngữ cảnh trong cùng một dự án và trong liên dự án. Zhou và cộng sự [36] đã xây dựng một mạng LSTM dựa trên cấu trúc hai chiều và cấu trúc cây LSTM-BT nhằm nâng cao khả năng học đặc trưng và hiệu suất dự đoán. Họ sử dụng lớp embedding để trích xuất đặc trưng và đưa vào LSTM-BT để thực hiện dự đoán. Khleel và Nehéz [67] đã giới thiệu sự kết hợp giữa mạng Bi-LSTM và các kỹ thuật lấy mẫu tăng để giải quyết vấn đề dữ liệu mất cân bằng. Kết quả thử nghiệm cho thấy mô hình đề xuất đạt được những cải thiện đáng kể khi áp dụng trên tập dữ liệu đã cân bằng so với tập dữ liệu ban đầu. Cụ thể, độ chính xác trung bình của mô hình tăng 6% và chỉ số F-score cải thiện 43%. Cheng và cộng sự [68] đã giới thiệu mô hình Deeplinedp nhằm xác định các tệp lỗi và dòng lỗi trong mã nguồn phần mềm bằng cách tự động học các thuộc tính ngữ nghĩa của các token và dòng mã lệnh xung quanh. Để học cấu trúc của chương trình, nhóm nghiên cứu đã sử dụng Hierarchical Attention Network [154]. Thử nghiệm trên 9 dự án phần mềm cho thấy mô hình này cải thiện giá trị AUC và MCC so với các phương pháp ở mức tệp và mức dòng lệnh khác. Trong nghiên cứu của Wang và cộng sự [70], mạng Gated Hierarchical Long Short-Term Memory (GH-LSTMs) đã được sử dụng để nắm bắt cả thông tin ngữ nghĩa từ cây cú pháp trừu tượng AST và các độ đo đặc trưng phần mềm truyền thống từ kho dữ liệu PROMISE. Kết quả thử nghiệm trên 16 bộ dữ liệu từ 10 dự án mã nguồn mở cho thấy GH-LSTMs đạt hiệu suất tương đương về F1-score và P_{opt} so với các mô hình tiên tiến nhất. Uddin và cộng sự [78] đã đề xuất một mô hình sử dụng mạng BiLSTM và BERT-based Semantic Feature (SDP-BB) để dự đoán lỗi phần mềm từ các đặc trưng ngữ nghĩa của mã nguồn. Kết quả trên 10 dự án mã nguồn mở cho thấy mô hình này vượt trội hơn so với các mô hình so sánh.

Batool và cộng sự [71] đã sử dụng ba mô hình học sâu, bao gồm LSTM, BiLSTM và RBFN để dự đoán lỗi phần mềm trên các tập dữ liệu dựa trên độ đo CK. Nhóm nghiên cứu kết luận rằng LSTM và BiLSTM đạt độ chính xác cao nhất lần lượt là 93,53% và 93,75%. Ahmed Abdu và cộng sự [72] đã đề xuất mô hình DH-CNN để sinh thông tin ngữ nghĩa từ đồ thị luồng điều khiển, đồ thị phụ thuộc dữ liệu và đồ thị cú pháp từ AST. Kết quả thử nghiệm trên 9 dự án PROMISE cho thấy DH-CNN đạt hiệu suất vượt trội so với các phương pháp tiên tiến về F1-score và AUC trong dự đoán lỗi trong dự án và trong liên dự án. Lucija và cộng sự [69] đã đề xuất một mô hình dự đoán lỗi dựa trên GCNN, có khả năng nắm bắt toàn bộ thông tin từ các cạnh và nút được trích xuất từ cây cú pháp trừu tượng. Nhóm nghiên cứu kết luận rằng mô hình đề xuất đạt hiệu suất vượt trội so với các mô hình tiên tiến trong việc phân loại các thành phần lỗi bằng cây cú pháp trừu tượng sinh ra từ mã nguồn.

Chen và cộng sự [155] đã đề xuất một phương pháp dự đoán lỗi phần mềm tích hợp mới sử dụng sự kết hợp giữa đa dạng hóa mô hình và mạng nơ-ron. Kết quả thử nghiệm

trên tám tập dữ liệu công khai theo các tiêu chí AUC, F1-score và MCC cho thấy DE-SDP đạt hiệu suất vượt trội so với các phương pháp cơ bản và xử lý hiệu quả vấn đề dữ liệu mất cân bằng. Briciu và cộng sự [156] đã đề xuất một phương pháp giải quyết bài toán dự đoán lỗi phần mềm bằng cách tận dụng các mô hình ngôn ngữ đã được huấn luyện trước. Cụ thể, mô hình dự đoán lỗi của họ được triển khai bằng một mạng nơ-ron nhân tạo sâu với đầu vào là mã nguồn được thực hiện embeddings và học từ các mô hình RoBERTa và CodeBERT-MLM. Kết quả thử nghiệm trên tập dữ liệu Apache Calcite cho thấy mô hình CodeBERT-MLM, khi được huấn luyện trên mã nguồn, đã tạo ra các biểu diễn mã nguồn có khả năng nắm bắt hiệu quả thông tin đặc thù của ngôn ngữ lập trình.

Sahar và cộng sự [157] đã giới thiệu mô hình DP-CCL để dự đoán lỗi trong mã nguồn. Mô hình này sử dụng học tương phản có giám sát kết hợp với CodeBERT để trích xuất hiệu quả các đặc trưng ngữ nghĩa từ mã nguồn. Kết quả thử nghiệm cho thấy các đặc trưng sinh ra từ CodeBERT hiệu quả hơn đáng kể, với mức cải thiện F1-score từ 4,9% đến 14,9% so với các phương pháp hiện tại. Zhou và cộng sự [73] đã giới thiệu phương pháp CGCN, tích hợp sức mạnh của CNN và GNN nhằm trích xuất cả thông tin ngữ nghĩa và cấu trúc từ mã nguồn. Thử nghiệm trên 21 dự án mã nguồn mở cho thấy mô hình này vượt trội so với các mô hình được đề xuất mới nhất, mang lại kết quả ổn định hơn và khả năng biểu diễn đặc trưng mạnh mẽ hơn. Rafi và cộng sự [74] đã đề xuất DepGraph, một GNN mới dành cho định vị lỗi. Họ đã tích hợp thêm các đặc trưng như thông tin thay đổi mã vào đồ thị dưới dạng thuộc tính, giúp mô hình tận dụng hiệu quả dữ liệu lịch sử dự án phong phú. Kết quả đánh giá trên tập dữ liệu Defects4j cho thấy DepGraph đạt hiệu suất vượt trội so với các phương pháp hiện có. Wang và cộng sự [158] đã đề xuất mô hình BugPre với mục tiêu sử dụng phương pháp phân tích kết hợp dựa trên cây lan truyền biến để xác định các mô-đun đã thay đổi trong phiên bản hiện tại. BugPre xây dựng đồ thị dựa trên sự phụ thuộc ngữ cảnh mã nguồn và sử dụng mạng Graph Convolutional Neural Network để học các đặc trưng đại diện của mã, từ đó nâng cao khả năng dự đoán lỗi khi có sự thay đổi phiên bản. Thông qua các thử nghiệm rộng rãi trên các dự án mã nguồn mở Apache, kết quả cho thấy BugPre vượt trội hơn ba phương pháp phát hiện lỗi tiên tiến nhất hiện nay, với giá trị F1-score tăng trên 16%. Liu và cộng sự [159] đề xuất mô hình mới dựa trên kiến trúc Transformer kết hợp với CNN, gọi là PM2-CNN cho bài toán dự đoán lỗi phần mềm. Mô hình này sử dụng ngôn ngữ mô hình hóa tiền huấn luyện và bộ phân loại dựa trên CNN nhằm thu được biểu diễn ngữ cảnh cũng như nắm bắt mối tương quan cục bộ của các chuỗi. Một tập dữ liệu lớn và được sử dụng rộng rãi đã được áp dụng để kiểm chứng tính hiệu quả của phương pháp đề xuất. Tác giả đã chứng minh kết quả cho thấy phương pháp này đạt được sự cải thiện trên nhiều độ đo đánh giá tổng quát so với phương pháp cơ sở tối ưu. Theo đó, thông tin bên ngoài có thể mang lại tác động tích cực đến việc dự đoán lỗi phần mềm, và mô hình đề xuất đã khai thác hiệu quả nguồn thông tin

này để nâng cao hiệu suất phát hiện lỗi. Liu và cộng sự [160] đã giới thiệu mô hình dự đoán lỗi phần mềm mới, gọi là CFG2AT, dựa trên đồ thị Control Flow Graph và mạng Graph Attention Network. CFG2AT được thiết kế chuyên biệt nhằm tự động phát hiện lỗi phần mềm, đồng thời tích hợp đơn vị chú ý có cấu trúc đồ thị để khai thác hiệu quả thông tin luồng điều khiển. Để đánh giá tính hiệu quả của CFG2AT, tác giả đã tiến hành các thí nghiệm trên dữ liệu từ 15 phiên bản thuộc 6 dự án phần mềm mã nguồn mở khác nhau trong cả hai kịch bản dự đoán lỗi trong cùng một dự án và trong liên dự án. Kết quả thử nghiệm cho thấy phương pháp CFG2AT nhìn chung vượt trội hơn so với nhiều phương pháp đối chứng trong dự đoán lỗi phần mềm. Cụ thể, mức cải thiện đạt từ 7,09%–12,80% đối với F1, 1,30%–4,15% đối với AUC, và 6,78%–17,54% đối với hệ số tương quan MCC trong kịch bản dự đoán trong cùng một dự án; đồng thời đạt từ 23,76%–44,79% đối với F1, 8,93%–13,27% đối với AUC, và 36,92%–94,89% đối với MCC trong kịch bản dự đoán trong liên dự án. Siachos và cộng sự [161] đã đề xuất một phương pháp lai, kết hợp ưu điểm của các kỹ thuật word-embedding, mô hình biểu diễn văn bản dựa trên đồ thị và mạng nơ-ron đồ thị nhằm dự đoán lỗi phần mềm. Họ hướng đến việc tận dụng đồng thời cả hai loại thông tin bằng cách biểu diễn mỗi văn bản dưới dạng đồ thị và áp dụng Graph Attention Networks (GATs). Kết quả đánh giá cho thấy phương pháp này mang lại sự cải thiện tổng thể trên nhiều độ đo đánh giá so với nhiều mô hình học máy dựa trên đồ thị hiện có. nhiều nghiên cứu chủ yếu tập trung đánh giá hiệu quả của từng mô hình học sâu hoặc từng dạng biểu diễn mã nguồn riêng lẻ, trong khi chưa khai thác đầy đủ sự bổ sung lẫn nhau giữa các phương pháp biểu diễn và các kiến trúc học sâu khác nhau. Tuy nhiên, phần lớn các phương pháp vẫn tập trung vào việc khai thác một loại biểu diễn mã nguồn hoặc một kiến trúc học sâu riêng lẻ, trong khi việc kết hợp hiệu quả giữa các phương pháp biểu diễn từ vựng và ngữ nghĩa của mã nguồn với khả năng học các quan hệ phụ thuộc dài hạn vẫn còn hạn chế. Đồng thời, việc kết hợp hiệu quả các mô hình học sâu nhằm tận dụng đồng thời khả năng biểu diễn từ vựng, cú pháp, ngữ nghĩa và học các quan hệ phụ thuộc dài hạn của mã nguồn vẫn còn là một hướng nghiên cứu cần được tiếp tục khai thác.

4.3 Vấn đề nghiên cứu

Sự khác biệt giữa giải pháp được đề xuất trong luận án này và các nghiên cứu trên được trình bày như sau. Đầu tiên, các phương pháp truyền thống sử dụng các độ đo phần mềm để xây dựng các mô hình dự đoán. Tuy nhiên, những độ đo này không thể nắm bắt được các đặc trưng cú pháp và ngữ nghĩa của mã nguồn. Giải pháp đề xuất có thể nắm bắt thông tin ngữ nghĩa bằng cách sử dụng FastText, một mô hình học sâu để học các biểu diễn vector từ AST của chương trình. Thứ hai, các nghiên cứu trước đây sử dụng các mô hình học sâu tuần tự như DBN, RNN và LSTM, những mô hình này gặp phải vấn đề về gradient

biến mất và gradient bùng nổ. Bằng cách sử dụng kiến trúc Transformer song song với cơ chế self-attention và có khả năng song song hóa, phương pháp đề xuất của luận án khắc phục được các vấn đề như hiện tượng mất mát gradient và rút ngắn đáng kể thời gian huấn luyện so với các mô hình tuần tự như RNN và LSTM. Tổng thể, phương pháp này tích hợp ba thành phần chính trong một quy trình thống nhất nhằm nâng cao độ chính xác trong dự đoán lỗi. Trước tiên, mã nguồn được phân tích cú pháp thành cây AST để nắm bắt thông tin cấu trúc và cú pháp. Tiếp theo, các token trích xuất từ AST được nhúng bằng FastText để duy trì sự tương đồng ngữ nghĩa, bao gồm cả các mẫu ở cấp độ từ con. Cuối cùng, mô hình Transformer Encoder với cơ chế tự chú ý có khả năng nắm bắt các phụ thuộc dài hạn và mối quan hệ phân cấp giữa các thành phần mã nguồn một cách hiệu quả. Sự kết hợp này cho phép trích xuất hiệu quả các đặc trưng ngữ nghĩa và cấu trúc, vốn đóng vai trò then chốt trong việc xác định các đoạn mã dễ phát sinh lỗi.

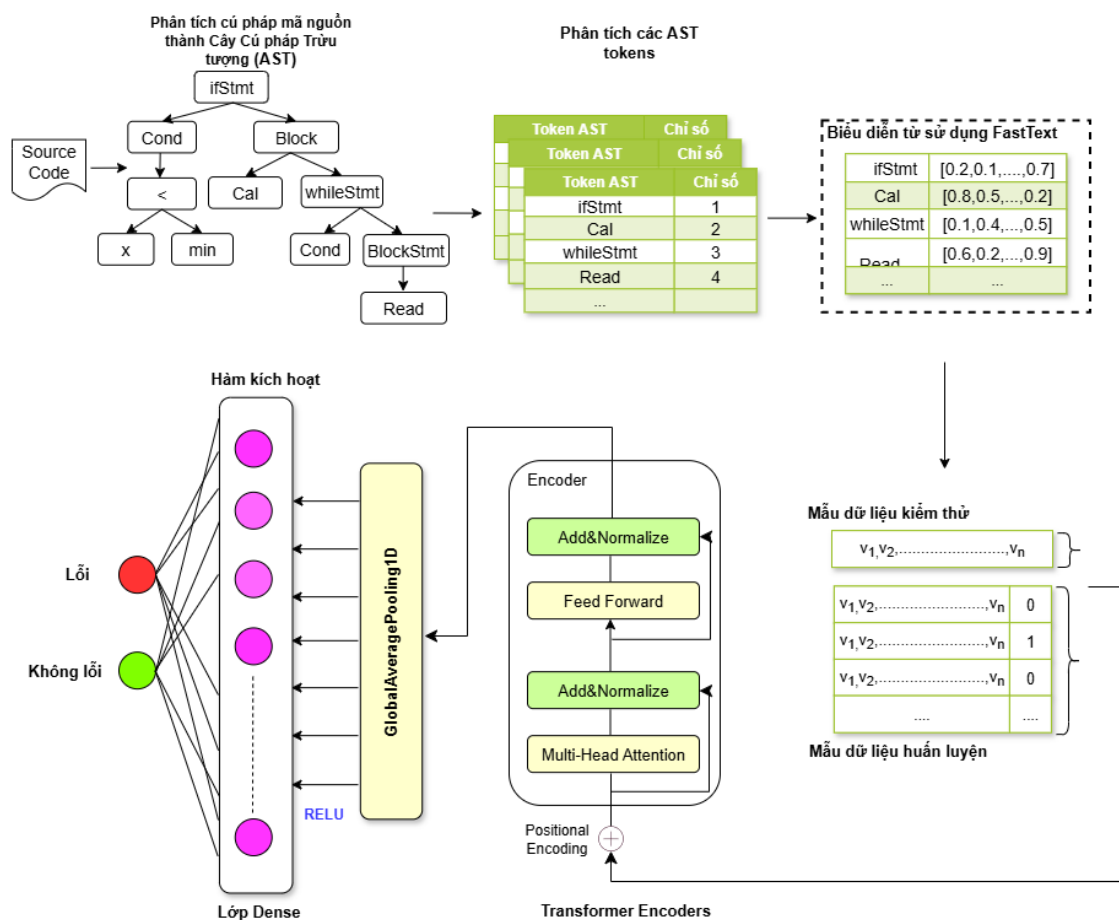
Để khẳng định tính hiệu quả và khả năng ứng dụng của mô hình đề xuất, cần có một quá trình đánh giá toàn diện trên nhiều kịch bản dự đoán khác nhau. Cụ thể, mô hình cần được so sánh với các phương pháp tiên tiến hiện nay nhằm đánh giá không chỉ khả năng cải thiện độ chính xác khi áp dụng cùng một dự án (Within-Project Defect Prediction - WPDP) mà còn khả năng tổng quát hóa khi áp dụng cho dự đoán lỗi trong liên dự án (Cross-Project Defect Prediction - CPDP). Trên cơ sở đó, luận án xây dựng hai câu hỏi nghiên cứu sau để kiểm chứng hiệu quả của mô hình đề xuất FastText-Transformer cho việc dự đoán lỗi phần mềm dựa trên các đặc trưng cú pháp như sau:

1. *Câu hỏi nghiên cứu 1: Mô hình FastText-Transformer được đề xuất có vượt trội hơn so với các mô hình dự đoán lỗi tiên tiến hiện nay đối với dự đoán lỗi trong cùng một dự án không?*
2. *Câu hỏi nghiên cứu 2: Mô hình FastText-Transformer được đề xuất có vượt trội hơn so với các mô hình dự đoán lỗi tiên tiến hiện nay đối với dự đoán lỗi trong liên dự án không?*

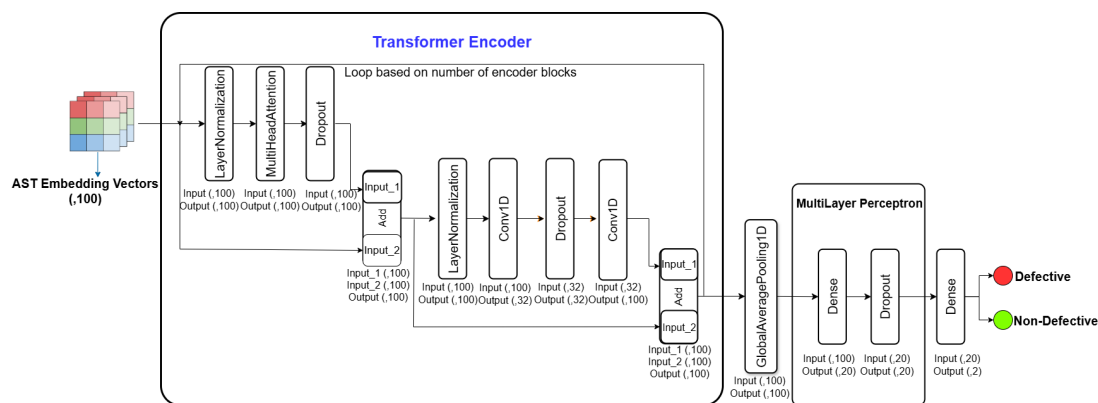
4.4 Phương pháp đề xuất

Luận án này đề xuất mô hình FastText-Transformer để thực hiện dự đoán lỗi phần mềm. Đầu tiên, Gensim FastText được sử dụng để nhúng token nhờ khả năng nắm bắt thông tin của từ con, giúp cho phù hợp với các trường hợp mà mã nguồn có thể chứa các định danh, tên hàm hoặc tên biến không có trong bộ từ vựng chuẩn. Hơn nữa, các FastText có thể xử lý hiệu quả các từ nằm ngoài bộ từ vựng và các biến thể hình thái, đảm bảo việc học các biểu diễn của các chuỗi token ngay cả khi có các từ hoặc token hiếm hoặc chưa từng thấy trong mã nguồn. Thứ hai, Transformer là một mô hình đầy đủ kết nối sử dụng cơ chế

self-attention, phù hợp hơn trong việc nắm bắt các phụ thuộc và các đặc trưng quan trọng trên mã nguồn. Hình 4.1 minh họa kiến trúc tổng thể của mô hình được đề xuất. Đối với mỗi tệp trong tập huấn luyện và tập kiểm thử, một vec-tơ các token được trích xuất từ cây cú pháp trừu tượng của chương trình bằng cách phân tích mã nguồn thành AST và chọn các nút AST. Theo nghiên cứu của Liang và cộng sự [153], chỉ ba loại nút AST được chọn làm token như mô tả trong Bảng 4.1. Cây cú pháp trừu tượng được duyệt và sắp xếp các nút AST vào vec-tơ trình diễn cho các token. Mỗi token vec-tơ sau đó được chuyển đổi thành một embedding vec-tơ và quá trình nhúng token được thực hiện bằng cách sử dụng mô hình FastText. Cuối cùng, các embedding vec-tơ được tạo ra được sử dụng trong bộ dữ liệu huấn luyện để xây dựng một mô hình dựa trên FastText-Transformer để dự đoán lỗi. Mô hình FastText-Transformer đề xuất gồm ba phần chính: 1) phân tích mã nguồn của chương trình thành cây cú pháp trừu tượng và trích xuất các vec-tơ token làm đặc trưng, 2) sử dụng mô hình FastText để học các biểu diễn vec-tơ của các token và ánh xạ các vec-tơ token thành các vec-tơ có giá trị thực, 3) xây dựng mô hình Transformer và thực hiện dự đoán lỗi bằng cách sử dụng các vec-tơ có giá trị thực của tập huấn luyện và tập kiểm thử đã được thu thập sau bước 2.



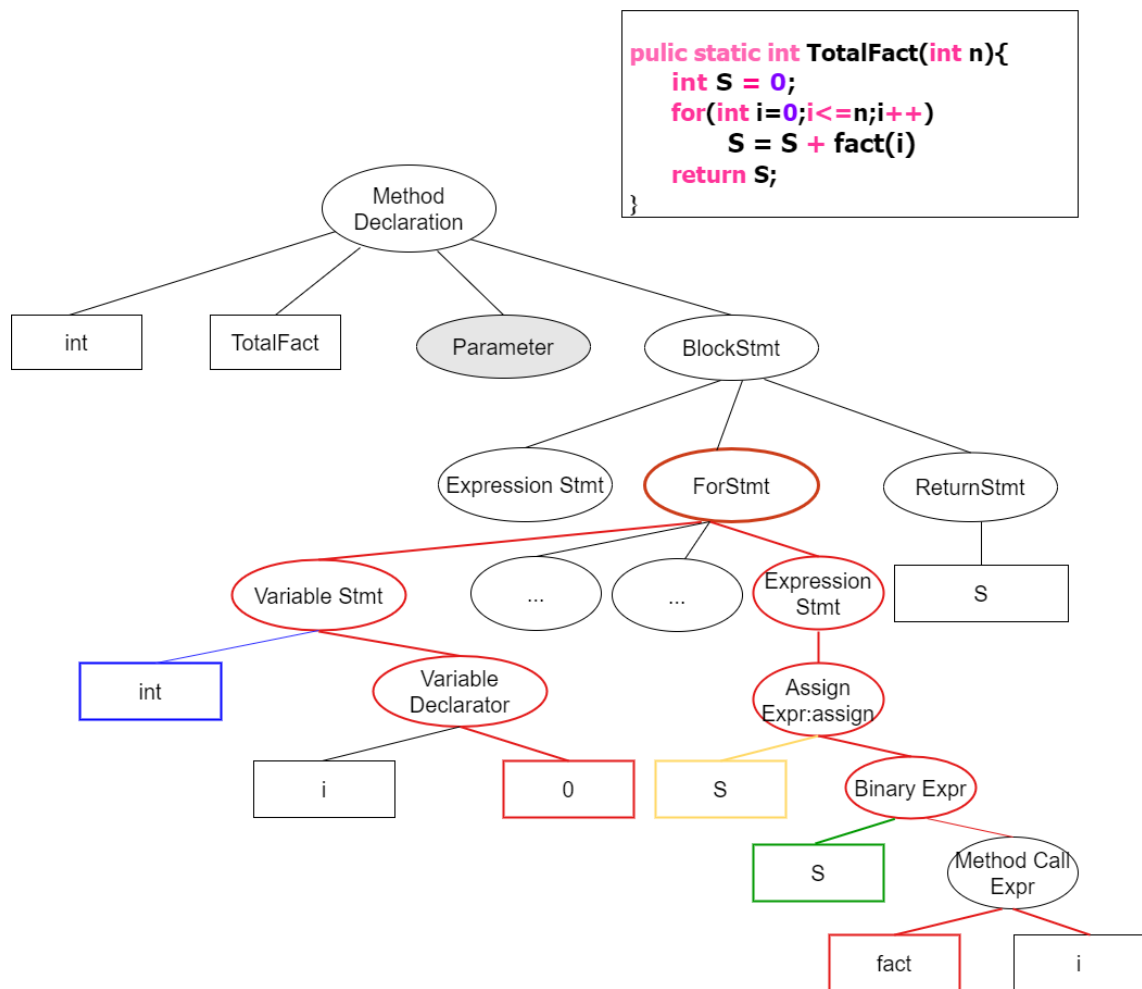
Hình 4.1: Kiến trúc mô hình đề xuất



Hình 4.2: Kiến trúc Transformer Encoder sử dụng

4.4.1 Biểu diễn mã nguồn và trích xuất các đặc trưng

Để nắm bắt các đặc trưng ngữ nghĩa của một chương trình phần mềm, việc trích xuất một vec-tơ các token từ mã nguồn của từng tệp là rất quan trọng. Theo một số nghiên cứu gần



Hình 4.3: Một ví dụ AST cho một chương trình Java

đây [153, 162], các tác giả đã đưa ra bằng chứng thuyết phục về ưu điểm của AST trong mô hình hóa mã nguồn, chứng minh rằng AST có thể được sử dụng để thu được biểu diễn mã nguồn tốt hơn. AST là một cấu trúc dạng cây, trong đó mỗi nút biểu diễn một cấu trúc cú pháp, phản ánh cả ngữ pháp và ngữ nghĩa của mã nguồn.

Biểu diễn AST Trong luận án này, công cụ *javalang* [163] đã được sử dụng để chuyển đổi từng tệp là một lớp Java thành biểu diễn AST. AST thể hiện cấu trúc phân cấp của mã nguồn, trong đó mỗi nút tương ứng với một cấu trúc cú pháp riêng biệt trong chương trình. Quá trình chuyển đổi mã nguồn thành AST bao gồm việc phân tích cú pháp mã Java và xác định các thành phần cú pháp chính, chẳng hạn như định nghĩa lớp, khai báo phương thức, câu lệnh điều khiển luồng, và các cấu trúc chương trình khác. Hình 4.3 minh họa một AST được tạo ra từ một đoạn mã Java.

Phân tách từ Khi cây cú pháp trừu tượng được tạo ra, trong đó tập trung vào ba loại nút cụ thể mang thông tin ngữ nghĩa quan trọng về chương trình. Các nút này được trích xuất

dựa trên mức độ liên quan của chúng đến hành vi và cấu trúc của chương trình. Các danh mục nút được trích xuất dưới đây.

1. **Nút lớp.** Các nút này có liên quan đến tạo thể hiện lớp và triệu gọi phương thức. Đối với mỗi nút lớp, luận án trích xuất tên lớp cũng như tên của bất kỳ phương thức nào được gọi. Ví dụ, trong Hình 4.3, phương thức *fact()* được trích xuất và ghi nhận dưới dạng token *fact()*. Việc trích xuất tên lớp và tên phương thức từ các nút này đóng vai trò quan trọng trong việc hiểu cấu trúc hướng đối tượng của chương trình.
2. **Nút khai báo.** Các nút này tương ứng với các khai báo trong chương trình, chẳng hạn như khai báo *interface*, khai báo lớp, khai báo phương thức và khai báo kiểu dữ liệu. Luận án thực hiện phân tách token cho các nút này bằng cách ghi nhận tên của các thực thể được khai báo. Ví dụ, một khai báo lớp như *public class Factorial* sẽ được chuyển thành token *Factorial*, trong khi một khai báo phương thức sẽ được chuyển thành token dựa trên tên phương thức, chẳng hạn như *fact*.
3. **Nút điều khiển.** Các nút này ghi nhận các cơ chế điều khiển luồng của chương trình, chẳng hạn như *câu lệnh điều khiển* (e.g., câu lệnh *if*), *cấu trúc lặp* (e.g., vòng lặp *while*), and *xử lý ngoại lệ* (e.g., khối *catch*). Đối với mỗi nút điều khiển luồng này, luận án ghi nhận loại của chúng dưới dạng token. Chẳng hạn, một câu lệnh “while” sẽ được chuyển thành token “while”, và một khối “catch” sẽ được chuyển thành token “catch”. Các nút này giúp hiểu rõ cấu trúc logic của chương trình và cách các phần khác nhau của mã nguồn tương tác với nhau.

Ba danh mục này, bao gồm các nút lớp, nút khai báo và nút điều khiển luồng, đóng vai trò cốt lõi trong quá trình phân tách token, vì chúng ghi nhận các đặc trưng cấu trúc và ngữ nghĩa chính của mã nguồn. Các nút được chọn được trình bày chi tiết trong Bảng 4.1.

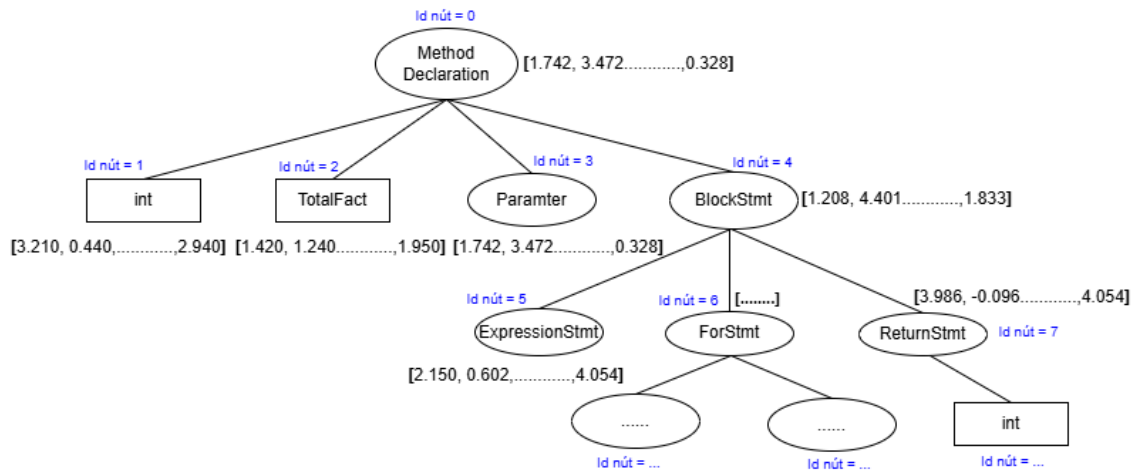
Luận án sử dụng thuật toán Breadth-First Search (BFS) để khám phá có hệ thống cấu trúc của cây cú pháp trừu tượng. Trong quá trình tạo biểu diễn token từ AST, BFS được áp dụng để duyệt cây và ghi nhận các mối quan hệ giữa các token ở các cấp độ khác nhau. Quá trình duyệt bắt đầu từ nút gốc và lần lượt thăm tất cả các nút con trước khi chuyển sang cấp tiếp theo. Ở mỗi cấp độ, các token được thu thập từ các nút lân cận trong một phạm vi cửa sổ xác định, cho biết số lượng nút xung quanh nút hiện tại được xem xét để xây dựng ngữ cảnh.

Bắt đầu từ nút gốc, các nút con được duyệt qua. Đối với mỗi nút con, tiếp tục duyệt sâu xuống các nút con của chúng, hình thành các đường dẫn đại diện cho các *context windows*. Sau khi thu thập các *context windows* này bằng BFS, ánh xạ các token trích xuất được thành các embedding vec-tơ. Ở giai đoạn này, mô hình FastText được áp dụng.

Bảng 4.1: Nút AST được chọn

Danh mục nút	Loại nút
Các nút liên quan đến lớp và lời gọi phương thức	MethodInvocation, SuperMethodInvocation, ClassCreator
Nút khai báo	PackageDeclaration, InterfaceDeclaration, ClassDeclaration, ConstructorDeclaration, MethodDeclaration, VariableDeclarator, FormalParameter
Nút điều khiển	IfStatement, ForStatement, WhileStatement, DoStatement, AssertStatement, BreakStatement, ContinueStatement, ReturnStatement, ThrowStatement, TryStatement, SynchronizedStatement, SwitchStatement, BlockStatement, CatchClauseParameter, TryResource, CatchClause, SwitchStatementCase, ForControl, EnhancedForControl
Các nút khác	BasicType, MemberReference, ReferenceType, SuperMemberReference, StatementExpression

Đối với mỗi token trong AST, mô hình FastText tạo ra các embedding vec-tơ tương ứng. Embedding vec-tơ này biểu diễn token trong không gian vec-tơ liên tục, ghi nhận thông tin ngữ nghĩa dựa trên mã nguồn đầu vào.



Hình 4.4: Ví dụ về quá trình phân tách mã thông qua cây cú pháp trừu tượng cho một chương trình Java

4.4.2 Áp dụng FastText cho Token Embedding

Trong xử lý ngôn ngữ tự nhiên, word-embedding là một kỹ thuật phổ biến được sử dụng để biểu diễn từ hoặc cụm từ dưới dạng các cụm vec-tơ trong một không gian vec-tơ liên tục. Các biểu diễn vec-tơ này nắm bắt được các mối quan hệ ngữ nghĩa và cú pháp giữa các từ, giúp các thuật toán học máy hiểu và xử lý ngôn ngữ tự nhiên tốt hơn. Word2vec [164] nhận vào một tập văn bản lớn và tạo ra một không gian vec-tơ, thường có kích thước hàng trăm chiều, trong đó mỗi từ duy nhất trong tập văn bản sẽ được biểu diễn bằng một vec-tơ tương ứng. Continuous Bag of Words (CBOW) và Skip-Gram là hai phương pháp chính được sử dụng để triển khai word-embedding bằng Word2vec. Phương pháp CBOW nhằm dự đoán một từ mục tiêu dựa trên ngữ cảnh (các từ lân cận). Quá trình so sánh đầu ra dự đoán với đầu ra thực tế sẽ được thực hiện để cập nhật trọng số của mạng. Ngược lại, mô hình Skip-Gram hoạt động theo hướng ngược lại so với CBOW, tức là sử dụng từ mục tiêu để dự đoán ngữ cảnh (các từ xung quanh) [165].

Trong những năm gần đây, Word2Vec đã được ứng dụng rộng rãi trong dự đoán lỗi [36, 162] và định vị lỗi [166]. Trong bối cảnh mã nguồn phần mềm, các token của ngôn ngữ lập trình có thể được xem như các từ trong xử lý ngôn ngữ tự nhiên. Nhiều mô hình word-embedding như GloVe [167], FastText [168] và CBOW [169] đã được áp dụng để học biểu diễn vec-tơ của các token trong mã nguồn, trong đó mỗi mô hình khai thác mối quan hệ ngữ nghĩa từ tập huấn luyện theo những cách khác nhau. GloVe, CBOW và FastText được sử dụng rộng rãi trong các tác vụ xử lý ngôn ngữ tự nhiên nhằm biểu diễn token dưới dạng vec-tơ đặc (dense vector). Luận án đã thực hiện một phân tích so sánh ba mô hình embedding, cụ thể là GloVe, CBOW và FastText, nhằm đánh giá hiệu quả của chúng khi được tích hợp với các kiến trúc dựa trên Transformer trong bài toán dự đoán lỗi phần mềm. So sánh chi tiết về hiệu suất của các kỹ thuật này được trình bày trong Bảng 4.2.

Từ kết quả so sánh, FastText embedding từ thư viện Gensim [170] được sử dụng để học biểu diễn vec-tơ của các token của ngôn ngữ lập trình. Một trong những tính năng quan trọng của Gensim là hỗ trợ Word2Vec, một kỹ thuật học các biểu diễn phân tán của từ các tập văn bản lớn. Các khái niệm cốt lõi của Gensim bao gồm tài liệu, tập dữ liệu văn bản, mô hình không gian vec-tơ, và biến đổi dữ liệu. Trong Word2Vec, mỗi từ được coi là phần tử nhỏ nhất để xây dựng biểu diễn vec-tơ, trong khi FastText coi một từ là tập hợp của các ký tự n-grams. Thay vì đưa từng từ vào mạng nơ-ron, FastText chuyển đổi mỗi từ thành nhiều n-grams (các thành phần con của từ). Lợi ích của FastText là biểu diễn từ dưới dạng tổng của các vec-tơ n-gram [168], nhờ đó có thể tạo ra các biểu diễn vec-tơ cho các từ chưa từng xuất hiện trong tập dữ liệu. Sau khi huấn luyện mạng nơ-ron, word-embedding cho tất cả n-grams trong tập dữ liệu huấn luyện được tạo ra. FastText hỗ

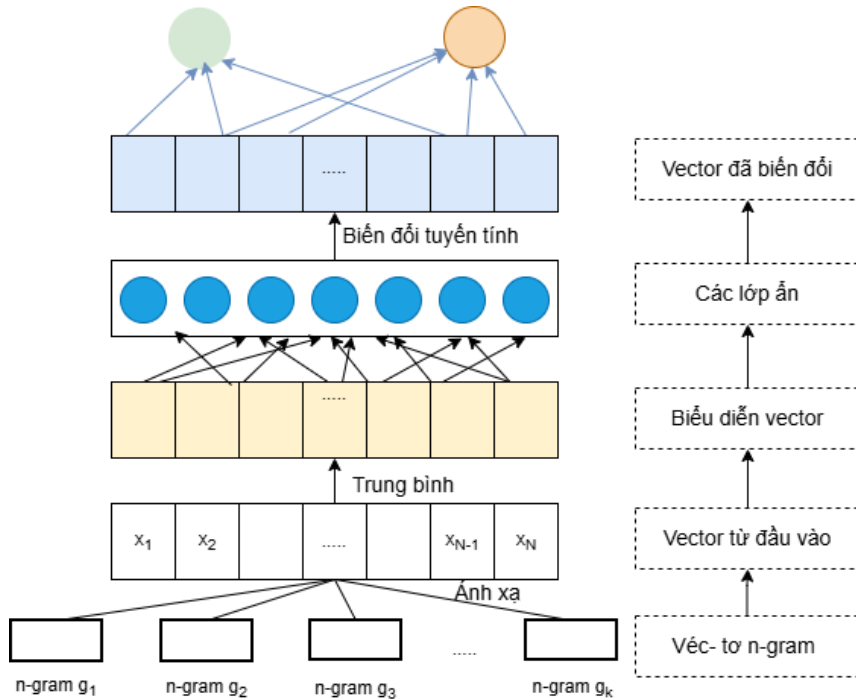
Bảng 4.2: Phân tích so sánh các mô hình dựa trên Transformer với những chiến lược embedding khác nhau

Dự án	Phiên bản	AST - Glove - Transformer				
		Thời gian huấn luyện	Thời gian thử nghiệm	P	R	F1
JEdit	4.0->4.1	2580.12	20.33	64.25	80.16	71.33
Xercer	1.2->1.3	2350.32	23.58	75.53	87.56	81.23
Log4j	1.0->1.1	58.75	0.79	60.49	77.78	68.06
Synapse	1.1->1.2	373.00	1.58	78.12	73.36	75.97
Xalan	2.5->2.6	2124.33	22.01	72.21	73.35	72.28
AST - CBOW- Transformer						
JEdit	4.0->4.1	2280.3	19.23	61.2	80.02	71.21
Xercer	1.2->1.3	3022.67	24.05	74.93	85.14	81.15
Log4j	1.0->1.1	61.87	0.77	70.9	77.04	71.03
Synapse	1.1->1.2	374.75	1.56	79.71	77.60	73.71
Xalan	2.5->2.6	2135.23	18.05	72.21	73.62	70.23
AST - FastText- Transformer						
JEdit	4.0->4.1	2083.49	18.36	63.4	79.6	70.6
Xercer	1.2->1.3	2232.61	18.66	75.90	87.1	72.6
Log4j	1.0->1.1	55.04	0.55	65.8	81.1	72.6
Synapse	1.1->1.2	354.65	1.48	79.5	77.2	73.2
Xalan	2.5->2.6	1401.64	17.36	75.1	73.3	73.8

trợ huấn luyện cả hai mô hình CBOW và Skip-gram, và có thể được xem như một biến thể của Word2Vec với đầu vào là các từ con (đơn vị nhỏ hơn từ được sinh ra từ quá trình phân tách từ vệt đầu vào thành các phần nhỏ hơn) [171]. Hơn nữa, FastText có khả năng sinh ra các véc-tơ từ cố định nhưng không xét đến thông tin về từ con và không thể tạo embedding cho các token ngoài từ vựng (out-of-vocabulary) [172]. Đây là những vấn đề then chốt khi làm việc với mã nguồn, nơi mà các định danh hiếm gặp, tên biến phức hợp và các token mới xuất hiện thường xuyên. Trong luận án này, luận án lựa chọn FastText, vốn xây dựng embedding dựa trên đặc trưng n-gram ký tự. Cách tiếp cận này cho phép tạo véc-tơ cho các token chưa từng gặp dựa trên cấu trúc từ con của chúng, mang lại lợi thế đáng kể trong việc xử lý vốn từ đa dạng và thường mang tính đặc thù lĩnh vực của mã nguồn.

Luận án sử dụng FastText cho tất cả các từ con trong bộ từ vựng huấn luyện, với kiến trúc huấn luyện CBOW. Cụ thể, luận án khai thác mô hình FastText để thu được các biểu diễn véc-tơ cho các n-gram của tokens được trích xuất từ AST của một tập hợp các dự án Java mã nguồn mở. Sau đó, luận án chuyển đổi chuỗi tokens thành chuỗi véc-tơ, có thể được đưa vào mô hình Transformer để huấn luyện. FastText [168] là một thuật toán phân loại văn bản nhanh và hiệu quả, được phát triển bởi Facebook AI Research. FastText sử

dùng hierarchical softmax để giảm thời gian tính toán xác suất trong quá trình huấn luyện. Hình 4.5 minh họa kiến trúc mô hình của FastText.



Hình 4.5: Kiến trúc tổng thể của mô hình FastText

Mỗi từ w được biểu diễn dưới dạng đặc trưng n-gram $x_1, x_2, \dots, x_n$ và sau đó được sử dụng làm đầu vào cho mô hình FastText [173]. FastText thêm ký hiệu ranh giới “<” and “>” vào đầu và cuối từ để phân biệt tiền tố với các chuỗi ký tự khác. Ví dụ, với từ “< fast”, các 3-grams có thể có bao gồm [“< fa”, “fas”, “ast”, “st >”]. Bằng cách kết hợp các đặc trưng n-gram, FastText có thể nắm bắt một phần thông tin về thứ tự từ trong câu, điều mà các mô hình bag-of-words (BoW) truyền thống không thể thực hiện được. Điều này giúp FastText thu được các biểu diễn câu chính xác hơn, đặc biệt là trong các bài toán phân loại văn bản [174].

Ở lớp đầu vào, dữ liệu văn bản được chuyển đổi thành chuỗi ký tự n-gram. Sau đó, các chuỗi từ con này được đưa qua lớp ẩn. Lớp ẩn được tổng hợp từ lớp đầu vào và nhân với ma trận trọng số A , đóng vai trò như một bảng ánh xạ cho biểu diễn từ dạng vec-tơ. Các giá trị của lớp ẩn sau đó được nhân với một ma trận trọng số khác B để tính toán lớp đầu ra. Ở lớp đầu ra, hierarchical softmax dựa trên cây mã hóa Huffman được sử dụng để giảm đáng kể thời gian huấn luyện mô hình. Bằng cách sử dụng mô hình tuyến tính với các ràng buộc về thứ hạng FastText đạt được hiệu suất tương đương với các công nghệ tiên tiến nhất chỉ trong vòng mười phút đối với một tỷ từ [175].

Những ưu điểm chính của FastText gồm [174]: (1) Tốc độ huấn luyện nhanh và hiệu

quả, so với các mô hình phân loại ngôn ngữ lớn khác, trong đó một tỷ từ có thể được huấn luyện trong chưa đầy mười phút mà vẫn đạt được hiệu suất ấn tượng; (2) Khả năng tự huấn luyện để tạo ra vec-tơ từ đầu mà không cần dựa vào các biểu diễn từ đã được huấn luyện sẵn; Tích hợp hai kỹ thuật quan trọng để nâng cao hiệu suất và giảm độ phức tạp: hierarchical softmax và n-grams.

Để nắm bắt hiệu quả các đặc trưng cú pháp và ngữ nghĩa được mã hóa trong các token của AST, luận án sử dụng FastText như một phương pháp embedding. FastText biểu diễn các token dưới dạng tổ hợp của các ký tự $n - gram$, cho phép mô hình hóa thông tin và khái quát hóa tốt hơn đối với những token hiếm hoặc chưa từng xuất hiện, một đặc điểm phổ biến trong AST do tính đa dạng và đặc thù của từ vựng trong miền này. Bằng cách sử dụng FastText, luận án tăng cường khả năng học các biểu diễn ngữ nghĩa phong phú cho các token AST.

Mục này trình bày tổng quan minh họa quá trình huấn luyện và trích xuất vec-tơ từ mô hình FastText bởi công cụ Gensim. Bước đầu tiên là khởi tạo mô hình FastText [170]. Liang và cộng sự [153] đã tiến hành các thử nghiệm trên 5 giá trị độ dài vec-tơ token là 20, 50, 100, 200 và 500. Kết quả cho thấy giá trị tối ưu cho độ dài vec-tơ được chọn là 100. Do đó, luận án chọn kích thước vec-tơ từ là 100, cho phép mô hình embedding học được các đặc trưng. Ngoài ra, các tham số khác như *window size* và *min-count* cũng được thiết lập để kiểm soát phạm vi ngữ cảnh trong biểu diễn từ và ngưỡng tần suất tối thiểu để một từ được đưa vào từ vựng. Tham số *window size* được đặt là 3, nghĩa là xem xét ba từ lân cận (một từ ở mỗi bên của từ hiện tại). Trong một câu, điều này xác định khoảng cách tối đa giữa từ được dự đoán và từ thực tế. Số lần xuất hiện tối thiểu của từ *min-count* được đặt bằng 1, tức là một từ phải xuất hiện ít nhất một lần trong câu để được đưa vào từ điển. Cuối cùng, thuật toán CBOW được sử dụng để huấn luyện mô hình.

Sau khi mô hình FastText được khởi tạo, bước tiếp theo là xây dựng bộ từ vựng. Sau khi hoàn thành việc xây dựng bộ từ vựng, mô hình sẽ được huấn luyện để tạo ra bảng ánh xạ giữa các tokens và các vec-tơ token. Các vec-tơ token được tạo ra này sẽ được đưa vào bộ Transformer Encoder để thực hiện dự đoán lỗi. Hình 4.4 minh họa một ví dụ về quá trình phân tách token từ cây cú pháp trừu tượng. Cụ thể, sau khi duyệt qua tất cả các nút ở cấp độ đầu tiên bắt đầu từ nút gốc, các nút tiếp theo được thu thập như [Node Id = 0, Node Id = 1, Node Id = 2, Node Id = 3, Node Id = 4]. Tiếp theo, mô hình FastText được sử dụng để tạo các embedding vec-tơ tương ứng, cho ra kết quả: [[1.742, 3.472, ..., 0.38], [3.210, 0.440, ..., 2.940], [1.420, 1.240, ..., 1.950], [1.742, 3.472, ..., 0.328], [1.208, 4.401, ..., 1.833]]. Các embedding vec-tơ thu thập được trên sau đó sẽ được đưa vào bộ Transformer Encoder.

4.4.3 Transformer Encoder

Hình 4.2 trình bày kiến trúc của mô hình Transformer Encoder sử dụng để thực hiện dự đoán lỗi với đầu vào là các véc-tơ có giá trị thực. Transformer Encoder đóng vai trò quan trọng trong mô hình bao gồm việc thực hiện cả hai tác vụ là self-attention và feed-forward. Encoder xử lý các véc-tơ của các token theo quy trình sau.

1. **Chuẩn hóa và cơ chế chú ý:** Các đầu vào được đưa qua lớp normalization giúp chuẩn hóa các giá trị trong toàn bộ chuỗi đầu vào. Tiếp theo, một lớp *Multi-Head Attention* được áp dụng, cho phép mô hình chú ý đến các phần khác nhau của chuỗi đầu vào cùng một lúc. Cơ chế *attention* này giúp mô hình nắm bắt được các mối quan hệ phụ thuộc giữa các token. Sau đó, một lớp dropout được sử dụng để giảm thiểu hiện tượng quá khớp. Cuối cùng, đầu ra được tính bằng cách cộng thêm *residual connection*, giúp bảo toàn thông tin từ đầu vào ban đầu.
2. **Feed Forward:** Đầu ra từ sau khi hoàn thành bước trên tiếp tục được xử lý. Lớp chuẩn hóa *normalization* được áp dụng một lần nữa để chuẩn hóa các giá trị. Tiếp theo, một lớp *convolution 1D* với hàm kích hoạt *ReLU* được sử dụng. Một lớp dropout khác cũng được thêm vào để ngăn chặn hiện tượng quá khớp.

Hình 4.2 minh họa kiến trúc Transformer Encoder được sử dụng trong phương pháp dự đoán lỗi của luận án. Kiến trúc mô hình bao gồm nhiều khối Encoder, trong đó mỗi khối tích hợp cơ chế *self-attention* và một đơn vị xử lý *feed-forward*. Cụ thể, mỗi khối Encoder chứa các lớp *multi-head self-attention* nhằm nắm bắt sự phụ thuộc toàn cục giữa các token đầu vào, đồng thời kết hợp với các lớp *convolutional* để trích xuất các đặc trưng ngữ cảnh cục bộ. Các thành phần như *layer normalization*, *residual connections*, và *dropout layers* được chèn xuyên suốt mô hình nhằm nâng cao độ ổn định trong quá trình huấn luyện và giảm thiểu hiện tượng quá khớp. Sau khi đi qua toàn bộ các khối Encoder, mô hình áp dụng phép *global pooling* để gộp các đặc trưng theo thời gian thành một véc-tơ có độ dài cố định. Véc-tơ này sau đó được đưa vào một MLP bao gồm các lớp *dense* cùng với *dropout regularization* để thực hiện nhiệm vụ phân loại cuối cùng. Hàm xây dựng mô hình được sử dụng để thiết lập mô hình Transformer hoàn chỉnh bằng cách xếp chồng nhiều khối Transformer Encoder, trong đó mỗi *block* bao gồm 4 khối *attention heads*, 4 khối *Transformer Encoder*, và 128 MLP đơn vị. Sau khi định nghĩa kiến trúc mô hình, quá trình biên dịch mô hình được thực hiện thông qua phương thức biên dịch. Hàm mất mát được thiết lập là *sparse categorical cross-entropy*, phù hợp cho các bài toán phân loại đa lớp. Bộ tối ưu được sử dụng là Adam với tốc độ học *learning rate* xác định trước. Bên cạnh đó, *sparse categorical accuracy* cũng được áp dụng để đánh giá hiệu suất của mô hình. Kỹ thuật *early stopping* được sử dụng như một *callback* nhằm nâng cao hiệu quả quá trình

huấn luyện. Phương pháp này giám sát giá trị hàm mất mát trên tập thử nghiệm và khôi phục lại trọng số mô hình tại epoch có hiệu suất tốt nhất, từ đó giúp ngăn ngừa hiện tượng quá khớp và tối ưu hóa khả năng khái quát hóa.

Trong mô hình của luận án, phương pháp mã hóa vị trí *positional encoding* trong Transformer được sử dụng để bảo toàn cấu trúc tuần tự của các AST token sau khi chúng được biến đổi thành word-embedding. Mặc dù AST vốn có dạng cây, luận án tiến hành duyệt cây bằng phương pháp BFS trước khi đưa các token đã trích xuất vào Transformer Encoder. Cách tiếp cận này tạo ra một chuỗi token phản ánh trật tự cú pháp của mã nguồn ban đầu. Trong mô hình Transformer, *positional encoding* đóng vai trò thiết yếu nhằm bổ sung thông tin về vị trí của từng token vào các vec-tơ embedding đầu vào. Luận án sử dụng phương pháp *sinusoidal positional encoding* tiêu chuẩn được đề xuất bởi Vaswani [176] để giúp mô hình nắm bắt được thông tin vị trí trong không gian embedding. *Sinusoidal positional encoding* được định nghĩa như sau:

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (4.1)$$

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (4.2)$$

Trong đó, $PE_{(pos, i)}$ là giá trị *positional encoding* tại vị trí pos trên chiều i . Ở đây, pos biểu thị vị trí của token trong chuỗi, i là chiều trong vec-tơ, và d là kích thước của embedding vec-tơ.

Đầu ra cuối cùng của quá trình *positional encoding* bao gồm token embedding và vị trí tương ứng của vec-tơ *positional encoding* tương ứng. Việc bổ sung *positional encoding* giúp mô hình học được vị trí tương đối giữa các thành phần mã nguồn, chẳng hạn như mối quan hệ giữa các cấu trúc điều kiện, vòng lặp và lời gọi hàm. Đây là những yếu tố then chốt trong việc phát hiện các lỗi trong mã nguồn.

4.5 Triển khai thử nghiệm, kết quả thử nghiệm và phân tích kết quả

Việc đánh giá phương pháp đề xuất được thực hiện nhằm xác định mức độ mà mô hình Transformer kết hợp với FastText có thể đạt kết quả vượt trội hơn so với các mô hình tiên tiến trong dự đoán lỗi phần mềm, bao gồm cả dự đoán lỗi trong dự án và dự đoán lỗi trong liên dự án. Luận án tiến hành đánh giá phương pháp đề xuất trên mười tập dữ liệu Apache [177] (Ant, Camel, Jedit, Log4j, Lucene, Poi, Synapse, Ivy, Xalan, và Xerces) được sử

dụng rộng rãi trong nghiên cứu về dự đoán lỗi phần mềm.

Luận án so sánh hiệu suất của phương pháp đề xuất với các mô hình dự đoán lỗi dựa trên phân tích cú pháp và ngữ nghĩa, bao gồm Seml [153], DP-Transformer [178], DBN-CP [151], DTL-DP [179], và TCA+ [180].

4.5.1 Tập dữ liệu thử nghiệm

Mục tiêu của thử nghiệm này là đánh giá hiệu suất của mô hình đề xuất trên tập hợp đa dạng và phong phú của các dự án mã nguồn Java. Để xây dựng bộ dữ liệu, luận án tiến hành hai nhiệm vụ nhằm đảm bảo tính chính xác và đầy đủ. Thứ nhất, luận án thu thập có hệ thống mã nguồn của các tệp chương trình riêng lẻ từ GitHub [177]. Để bảo đảm sự tương ứng chính xác giữa các phiên bản, luận án sử dụng các mã của phiên bản và tên lớp đầy đủ để đối chiếu từng tệp với bản sửa đổi tương ứng. Các tệp sau khi được thu thập sẽ được phân tích cú pháp thành cây cú pháp trừu tượng nhằm phục vụ cho quá trình trích xuất đặc trưng. Thứ hai, để xác định sự tồn tại của lỗi trong các tệp chương trình, luận án sử dụng bộ dữ liệu PROMISE [181], vốn cung cấp nhãn lỗi cùng với 20 đặc trưng mã nguồn cho mỗi tệp. Các tệp được phân loại là “có lỗi” hoặc “không lỗi” dựa trên các nhãn này.

Luận án đã lựa chọn 10 dự án mã nguồn, bao gồm tên dự án, phiên bản phát hành, số lượng tệp trung bình và tỷ lệ lỗi trung bình trích xuất mã nguồn của từng tệp từ GitHub [177]. Các dự án này được chọn dựa trên mức độ liên quan và việc sử dụng trong các nghiên cứu trước đây [36, 153, 162], bao gồm cả phân tích trong dự án và trong liên dự án. Chi tiết các tập dữ liệu Apache được trình bày trong Bảng 4.1.

Bộ dữ liệu Apache bao gồm 10 dự án mã nguồn mở Ant, Camel, JEdit, Log4j, Lucene, Poi, Synapse, Ivy, Xalan và Xerces. Mỗi dự án bao gồm từ hai đến ba phiên bản liên tiếp, với số lượng tệp nguồn trung bình dao động từ 150 đến 792 tệp, phản ánh sự đa dạng về quy mô phần mềm. Tỷ lệ tệp bị lỗi thay đổi từ 15,7% đến 62,49%, cho thấy tỷ lệ mất cân bằng dữ liệu và mức độ phân bố lỗi khác nhau trong liên dự án. Sự đa dạng về quy mô, số phiên bản và tỷ lệ lỗi giúp bộ dữ liệu này đại diện cho nhiều bối cảnh phát triển phần mềm khác nhau, từ các hệ thống có mật độ lỗi thấp đến các hệ thống có tỷ lệ lỗi cao. Điều này tạo điều kiện đánh giá khả năng thích nghi của mô hình trong nhiều điều kiện thực tế. Về tính đại diện, các bộ dữ liệu bao phủ nhiều lĩnh vực ứng dụng khác nhau như công cụ xây dựng phần mềm (Ant), framework tích hợp các thành phần được xây dựng sẵn (Camel), thư viện xử lý tài liệu (POI), công cụ ghi nhật ký (Log4j), công cụ XML (Xalan, Xerces), công cụ tìm kiếm (Lucene) và dịch vụ Web (Synapse). Đồng thời, việc sử dụng nhiều phiên bản của mỗi dự án giúp phản ánh sự thay đổi của phần mềm theo thời gian, mô phỏng sát quy trình phát triển và bảo trì trong thực tế.

Ngoài ra, quan trọng hơn, để xác định sự hiện diện của lỗi trong tệp chương trình, luận án dựa vào nhãn lỗi thu thập từ PROMISE [181]. Một số mẫu dữ liệu được trình bày trong Bảng 4.2. Toàn bộ dữ liệu đều được thu thập từ các dự án mã nguồn mở và đã được công bố công khai trong kho dữ liệu PROMISE đã thể hiện tính khách quan của tập dữ liệu sử dụng. Các nhãn lỗi được xác định dựa trên lịch sử sửa lỗi và hệ thống quản lý phiên bản của dự án thay vì được gán thủ công, nhờ đó hạn chế sai lệch chủ quan trong quá trình xây dựng dữ liệu. Hơn nữa, đây là bộ dữ liệu chuẩn được sử dụng rộng rãi trong nhiều nghiên cứu dự đoán lỗi phần mềm, do đó kết quả thực nghiệm có thể được đối chiếu trực tiếp với các mô hình đề xuất hiện có. Về khả năng khái quát hóa, luận án đã tiến hành thử nghiệm trên dự đoán lỗi phần mềm trong cùng một dự án và trong liên dự án khác nhau. Cụ thể, đối với phân tích trong dự án: mỗi dự án được chia thành hai phiên bản, một phiên bản dùng để huấn luyện và phiên bản còn lại dùng để kiểm thử. Ví dụ, luận án sử dụng phiên bản 1.4 của dự án Ant làm tập huấn luyện và phiên bản 1.6 làm tập kiểm thử. Đối với phân tích trong liên dự án: một dự án được chọn làm tập huấn luyện, trong khi dữ liệu từ các dự án khác được sử dụng làm tập kiểm thử. Chẳng hạn, Apache Poi v2.5 có thể được sử dụng làm tập huấn luyện, trong khi Apache Synapse v1.2 được sử dụng làm tập kiểm thử.

Bảng 4.1: Tập dữ liệu Apache được sử dụng

Tên dự án	Phiên bản	Số tệp trung bình	Tỷ lệ lỗi
Ant	1.4, 1.6, 1.7	425	23.65
Camel	1.2, 1.4, 1.6	792	23.76
Jedit	3.2, 4.0, 4.1	296	27.63
Log4j	1.0, 1.1, 1.2	150	50.44
Lucence	2.0, 2.2, 2.4	258	54.89
Poi	1.5, 2.5, 3.0	351	62.49
Synapse	1.0, 1.1, 1.2	211	23.37
Ivy	1.4, 2.0	622	20.0
Xalan	2.4, 2.5, 2.6	780	36.53
Xerces	1.2, 1.3	447	15.7

Bảng 4.2: Một số mẫu trong tập dữ liệu PROMISE được sử dụng

Tên dự án	Phiên bản	Tên tệp	Độ đo mã nguồn					Lỗi
			wmc	...	LOC	...	cam	
Ant	1.3	apache.tools.ant.taskdefs.ExecuteOn	11	...	395	...	0.232	0
		apache.tools.ant.DefaultLogger	14	...	257	...	0.307	2
Poi	1.5	apache.poi.hssf.record.LineFormatRecord	25	...	429	...	0.352	1
		apache.poi.hssf.record.formula.AttrPtg	20	...	336	...	0.225	6
Synapse	1.1	apache.synapse.registry.AbstractRegistry	7	...	230	...	0.375	0
		apache.synapse.core.axis2.ProxyService	53	...	1164	...	0.134	4
		apache.synapse.SynapseConstants	2	...	81	...	1	3

4.5.2 Các mô hình so sánh

Theo nghiên cứu của Görkem Giray và cộng sự [182], các mô hình CNN, DBN, RNN, LSTM, GRUs và MLP được công nhận rộng rãi là những phương pháp được sử dụng phổ biến nhất trong dự đoán lỗi phần mềm. Bên cạnh đó, các mô hình DBN, RNN, LSTM và GRU đã thể hiện hiệu suất tốt trong việc dự đoán lỗi phần mềm, nhờ khả năng nắm bắt thông tin ngữ nghĩa và thông tin cú pháp của mã nguồn. Nhiều mô hình đề xuất đã sử dụng biểu diễn trung gian, chẳng hạn như AST để chuyển đổi mã nguồn thành định dạng vec-tơ số, giúp mô hình học sâu có thể xử lý hiệu quả hơn [182].

Trong những năm qua, nhiều nghiên cứu đã sử dụng AST để biểu diễn mã nguồn dưới dạng vec-tơ số học, bao gồm các mô hình như Seml [153], DeepCPDP [183], DP-Transformer [151], Li [184], Liu [185], DBN-CP [151], DTL-DP [179], TCA+ [180]. Tuy nhiên, một số mô hình đề xuất đã được đánh giá trên tập dữ liệu hoặc độ đo hiệu suất khác so với tiêu chí trong phương pháp của luận án. Chẳng hạn, mô hình của Liu [185] sử dụng tập dữ liệu Linux và MySQL. Mô hình DeepCPDP sử dụng các độ đo đánh giá khác như AUC. Do đó, luận án đã lựa chọn các mô hình thử nghiệm được tiến hành trên các tập dữ liệu tương đương và được đánh giá bằng cùng một độ đo, nhằm so sánh hiệu suất của mô hình FastText-Transformer trong bài toán dự đoán lỗi phần mềm. Ngoài ra, luận án cũng thực hiện phân tích so sánh giữa mô hình được đề xuất và các mô hình đối sánh đã chọn, được mô tả dưới đây:

- **Seml** [153]: Mô hình dự đoán lỗi học thông tin ngữ nghĩa của mã nguồn bằng cách kết hợp word-embedding và mô hình học sâu.
- **DP-Transformer** [178]: Mô hình dự đoán lỗi phần mềm sử dụng Transformer để học các đặc trưng ngữ nghĩa và ngữ pháp từ các mô-đun phần mềm.
- **DBN-CP** [151]: Mô hình này có khả năng hiểu tốt hơn biểu diễn thông tin ngữ nghĩa để dự đoán lỗi.
- **DTL-DP** [179]: Mô hình học chuyển tiếp dành cho dự đoán lỗi có khả năng nắm bắt thông tin ngữ nghĩa và cấu trúc bằng cách biến mã nguồn thành hình ảnh.
- **TCA+** [180]: Mô hình nhằm điều chỉnh phân phối đặc trưng giữa dự án nguồn và dự án đích.
- **FastText-LSTM**: Mô hình kết hợp FastText và Transformer để dự đoán lỗi phần mềm.

4.5.3 Tham số huấn luyện

Luận án đã tiến hành thử nghiệm nhiều tham số khác nhau để xây dựng một cấu trúc tối ưu cho các mô hình trong luận án này. Các tham số chi tiết được trình bày trong Bảng 4.3 và Bảng 4.4. Trong kiến trúc này, bốn bộ Transformer Encoder được sử dụng, kết hợp với một lớp *Global Average Pooling* để xử lý đầu ra. Lớp *Global Average Pooling* có vai trò quan trọng vì giữ mối quan hệ chặt chẽ với kiến trúc tích chập, giúp kết nối giữa bản đồ đặc trưng và phân loại. Đồng thời cung cấp khả năng chuẩn hóa mô hình *regularization*, giúp giảm thiểu hiện tượng quá khớp [186].

Điều chỉnh tham số là một bước quan trọng để đạt được kết quả tối ưu. Luận án phân tích và điều chỉnh các giá trị siêu tham số cho từng thuật toán trước khi tiến hành thử nghiệm. Danh sách các siêu tham số bao gồm:

1. Epoch. Một epoch đề cập đến một vòng lặp hoàn chỉnh qua toàn bộ tập dữ liệu huấn luyện. Trong các thử nghiệm của luận án, số lượng epochs được điều chỉnh để tìm ra sự cân bằng tối ưu giữa mô hình quá đơn giản và mô hình quá khớp. Cụ thể, luận án đã thử nghiệm với phạm vi epochs từ 50 đến 500. Đối với cả hai mô hình Transformer và LSTM, 200 epochs được xác định là hiệu quả nhất, giúp mô hình học đủ thông tin từ dữ liệu mà không bị quá khớp.
2. Batch-size. Batch size xác định số lượng mẫu huấn luyện được sử dụng trong một vòng lặp của quá trình tối ưu hóa. Batch size nhỏ thường cung cấp ước lượng gradient chính xác hơn nhưng yêu cầu nhiều vòng lặp hơn để hoàn thành một epoch, điều này có thể làm tăng chi phí tính toán. Luận án đã thử nghiệm các giá trị batch size trong khoảng từ 16 đến 64. Dựa trên giới hạn bộ nhớ và kích thước tập dữ liệu, batch size = 16 được xác định là tối ưu, mang lại sự cân bằng tốt giữa tính ổn định khi huấn luyện và hiệu quả tính toán.
3. Drop-out. Để tránh vấn đề quá khớp, luận án áp dụng ngẫu nhiên các tỷ lệ dropout khác nhau [187]. Dropout là một kỹ thuật chuẩn hóa mô hình *regularization* giúp ngăn chặn quá khớp bằng cách loại bỏ ngẫu nhiên một phần nơ-ron trong quá trình huấn luyện. Điều này khuyến khích mô hình tổng quát hóa tốt hơn bằng cách không quá phụ thuộc vào các nơ-ron cụ thể. Luận án đã thử nghiệm các tỷ lệ dropout trong khoảng từ 0.2 đến 0.7. Đối với mô hình Transformer, luận án áp dụng dropout = 0.2 cho lớp MLP và 0.25 trong lớp Encoder. Đối với mô hình LSTM, luận án sử dụng dropout cao hơn (0.5), phản ánh xu hướng quá khớp lớn hơn do sự phức tạp của các phụ thuộc thời gian trong dữ liệu tuần tự.
4. Learning Rate. Kiểm soát mức độ điều chỉnh trọng số của mô hình dựa trên đạo hàm của hàm mất mát. Một tốc độ học tối ưu đảm bảo quá trình hội tụ ổn định và hiệu

quả. Luận án đã thử nghiệm các giá trị learning rate trong khoảng từ $1e-5$ đến $1e-3$. Đối với mô hình Transformer, learning rate = $1e-4$ được lựa chọn. Đối với mô hình LSTM, tốc độ học cao hơn một chút (0.001) được xác định là hiệu quả hơn.

5. Activation Function. Hàm kích hoạt *ReLU* được sử dụng cho cả hai mô hình nhờ vào hiệu quả trong việc giảm thiểu vấn đề đạo hàm biến mất và tăng cường khả năng học. Trong lớp đầu ra của mô hình Transformer, hàm Softmax được sử dụng để hỗ trợ phân loại đa lớp. Trong mô hình LSTM, hàm ReLU được áp dụng xuyên suốt các lớp.
6. Optimizer Function. Luận án sử dụng bộ tối ưu hóa Adam, một phương pháp thuộc SGD. Adam được chọn do khả năng điều chỉnh tốc độ học và hiệu quả tính toán. Cách tiếp cận này mang lại nhiều lợi ích, bao gồm: tính toán hiệu quả, yêu cầu bộ nhớ tối thiểu, khả năng chống chịu với sự thay đổi trong việc chuẩn hóa dữ liệu như đã đề cập trong nghiên cứu của Kingma và Ba [188]. Adam kết hợp ưu điểm của hai phương pháp mở rộng khác của SGD: *AdaGrad* và *RMSProp*, giúp phù hợp với nhiều bài toán học máy khác nhau. Nhờ những đặc điểm này, Adam đặc biệt phù hợp cho các bài toán xử lý tập dữ liệu lớn và có nhiều tham số.

Bảng 4.3: Tham số huấn luyện cho mô hình Transformer

Phần tử	Tham số	Giá trị lựa chọn
Mô hình đề xuất	Transformer Encoders	4
	Số lượng nơ-ron trong MLP	64
	Hàm kích hoạt	RELU
	Tỷ lệ Dropout	0.2
	Hàm kích hoạt đầu ra	Softmax
Encoder	Số lượng heads	4
	Kích thước Head	64
	Tỷ lệ Dropout	0.25
	Kích thước Kernal size (Kích thước của filter)	1
Huấn luyện	Hàm mất mát	sparse_categorical_crossentropy
	Bộ tối ưu hóa Optimizer	keras.optimizers.Adam
	Learning rate	$1e-4$
	Số lượng epochs	200
	Batch size	16

Đối với mô hình Transformer, luận án thiết lập kiến trúc với 04 khối Encoder, mỗi Encoder bao gồm cơ chế multi-head attention với bốn heads. Mỗi head có kích thước 64, đảm bảo khả năng mạnh mẽ trong việc nắm bắt các mẫu dữ liệu phức tạp. Mạng FFN

Bảng 4.4: Tham số huấn luyện cho mô hình LSTM

Phần tử	Tham số	Giá trị lựa chọn
LSTM	Số lượng lớp ẩn	3
	Số lượng nơ-ron trong lớp ẩn thứ nhất	256
	Số lượng nơ-ron trong lớp ẩn thứ hai	256
	Số lượng nơ-ron trong lớp ẩn thứ ba	128
	Số lượng epochs	200
	Batch size	16
	Thuật toán huấn luyện	Adam
	Tỷ lệ Dropout	0.5
	Hàm kích hoạt	Relu
	Learning rate	0.001
	Hàm mất mát	binary_crossentropy

trong Encoder có 64 nơ-ron, sử dụng hàm kích hoạt ReLU để đưa vào tính phi tuyến tính. Để giảm thiểu quá khớp, luận án áp dụng tỷ lệ dropout là 0.2 sau các lớp MLP và 0.25 trong các lớp Encoder. Lớp đầu ra sử dụng hàm softmax để hỗ trợ phân loại đa lớp. Quá trình huấn luyện được thực hiện với hàm mất mát *sparse categorical crossentropy*, tối ưu hóa bằng *Adam optimizer* với *learning rate* = $1e-4$. Mô hình được huấn luyện trong 200 epochs với batch size = 16, đảm bảo sự cân bằng giữa hiệu suất tính toán và sự ổn định trong quá trình hội tụ.

Kiến trúc của mô hình LSTM bao gồm ba lớp ẩn, được thiết kế để nắm bắt các phụ thuộc thời gian trong dữ liệu tuần tự. Hai lớp ẩn đầu tiên có 256 nơ-ron mỗi lớp. Lớp ẩn thứ ba có 128 nơ-ron. Hàm kích hoạt ReLU được sử dụng để ngăn chặn vấn đề đạo hàm biến mất và tăng cường khả năng học. Để giảm thiểu quá khớp, luận án áp dụng tỷ lệ dropout = 0.5 trên các lớp ẩn. Quá trình huấn luyện sử dụng hàm mất mát *binary crossentropy*, với bộ tối ưu hóa *Adam optimizer*, giúp tối ưu hóa hiệu quả tại *learning rate* = 0.001. Tương tự như mô hình Transformer, mô hình LSTM cũng được huấn luyện trong 200 epochs với batch size = 16. Chi tiết các thông số hệ thống để đánh giá hiệu suất của mô hình đề xuất và mô hình đối chứng LSTM được trình bày trong Bảng 4.5.

4.5.4 Trả lời cho Câu hỏi nghiên cứu 1: Mô hình FastText-Transformer được đề xuất có vượt trội hơn so với các mô hình dự đoán lỗi tiên tiến hiện nay đối với dự đoán lỗi trong cùng một dự án không?

Để đánh giá hiệu quả của phương pháp dự đoán lỗi trong cùng một dự án, luận án đã so sánh mô hình FastText-Transformer đề xuất với các mô hình cơ sở FastText-LSTM và bốn mô hình đề xuất sau: Seml [153], DBN-CP [151], DP-Transformer [178] và DTL-

Bảng 4.5: Thông số hệ thống chi tiết

Hệ thống	Thông số kỹ thuật
CPU	Intel(R) Core(TM) i9-12900K CPU@3.4 GHz RAM: 64GB
GPU	NVIDIA GeForce RTX 3090 • NVIDIA CUDA Cores: 1792 • Memory: 24GB GDDR6X • Memory Speed: 8Gbps
Software	OS: Windows 10 Pro (64 bits) • TensorFlow Version: tensorflow-gpu-1.0.1 • CUDA Version: cuda 8.5 • CuDNN Version: cuDNN v5.1 (Jan 20, 2017)

DP [179]. Đối với mỗi dự án, phiên bản trước với tập dữ liệu nhỏ và phiên bản sau với tập dữ liệu lớn được sử dụng làm tập huấn luyện và tập kiểm thử. Các chuỗi vec-tơ được tạo từ phiên bản trước được sử dụng để xây dựng mô hình Transformer, trong khi các chuỗi vec-tơ từ phiên bản sau được dùng để kiểm thử mô hình. Luận án đã thực hiện các thử nghiệm trên 16 tập dữ liệu, được thể hiện trong cột phiên bản của Bảng 4.6 cho bài toán dự đoán lỗi trong phạm vi cùng một dự án. Các phiên bản được ký hiệu dưới dạng $P_s \rightarrow P_t$, trong đó P_s và P_t lần lượt là dự án nguồn và dự án đích. Luận án tiến hành so sánh hiệu suất của mô hình đề xuất với các phương pháp cơ sở dựa trên các độ đo Precision (P), Recall (R) và F1-score (F1). Các giá trị cao nhất của các độ đo được đánh dấu in đậm.

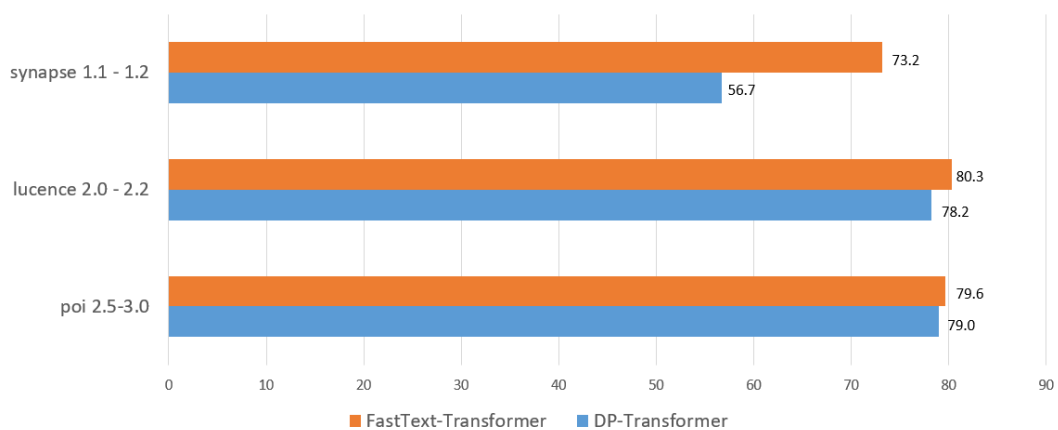
Bảng 4.6 trình bày các giá trị Precision, Recall và F1-score của các phương pháp Seml, DBN-CP, DTL-DP, FastText-LSTM và FastText-Transformer. Nhìn chung, phương pháp của luận án đạt hiệu suất tốt hơn so với các phương pháp cơ sở trên 11 trong số 19 tập dữ liệu. Mô hình FastText-Transformer đạt giá trị trung bình của Precision là 0.762, Recall là 0.807 và F1-score là 0.768. Mô hình FastText-Transformer đạt giá trị trung bình F1-score cao hơn 19.6% so với DTL-DP, 25.4% so với Seml, 19.8% so với DBN-CP và 7% so với mô hình FastText-LSTM. Kết quả cho thấy phương pháp đề xuất có thể cải thiện hiệu suất dự đoán so với các phương pháp Seml, DBN-CP, DTL-DP và FastText-LSTM. Hình 4.6 trình bày kết quả trên 3 tập dữ liệu cùng với kết quả của DP-Transformer. Trục x và trục y lần lượt biểu diễn nhóm dự án và giá trị F1-score được sử dụng để đánh giá. Chẳng hạn, trong nhóm thử nghiệm đầu tiên, Apache Poi 1.5 được sử dụng làm tập huấn luyện và Apache Poi 3.5 làm tập kiểm thử. Mô hình FastText-Transformer đạt hiệu suất cao hơn so với DP-Transformer trên các tập dữ liệu của nhóm Apache Lucene và Apache Synapse. Trong khi đó, kết quả F1-score của hai mô hình trên nhóm Apache Poi là tương đương nhau. Về giá trị trung bình, phương pháp đề xuất đạt giá trị F1-score cao hơn 8.9% so với DP-Transformer. Cách tiếp cận đề xuất cho thấy hiệu quả trong việc dự đoán lỗi phần

Bảng 4.6: Kết quả so sánh hiệu suất của DTL-DP, Seml, DBN-CP, FastText-LSTM và FastText-Transformer đối với dự đoán lỗi trong cùng một dự án

Dự án	Phiên bản	DTL-DP	Seml			DBN-CP			FastText-LSTM			FastText-Transformer		
		F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Camel	1.2->1.4	40.6	37.9	58.0	41.5	96.0	66.4	78.5	83.9	85.4	78.9	72.6	85.2	78.4
JEdit	1.4->1.6	38.5	51.6	54.1	93.4	26.3	64.9	37.4	46.1	50.0	47.9	61.2	50.3	48.7
	3.2->4.0	49.4	46.8	47.4	28.5	46.7	74.7	57.4	51.1	71.5	59.6	72.1	75.6	73.0
	4.0->4.1	68.7	57.6	46.8	0.5	54.4	70.9	61.5	63.2	79.5	70.4	63.4	79.6	70.6
Log4j	1.0->1.1	68.8	69.4	67.6	68.5	67.5	73.0	70.1	60.2	75.6	68.2	65.8	81.1	72.6
Lucene	2.0->2.2	78.3	70.5	66.4	68.4	75.9	56.9	65.1	72.4	80.6	76.4	82.4	78.6	80.3
	2.2->2.4	77.6	66.5	68.8	67.7	66.5	92.1	77.3	62.3	73.2	68.8	71.4	84.5	76.2
Xalan	2.4->2.5	79.4	51.5	58.6	9.9	65.0	54.8	59.5	77.7	88.1	82.6	89.8	88.4	83.3
	2.5->2.6	-	63.0	67.8	6.2	61.6	57.9	59.7	67.2	66.4	66.7	75.1	73.3	73.8
Xerces	1.2->1.3	82.0	33.3	33.7	7.0	40.3	42.0	41.1	43.6	50.0	46.6	75.9	87.1	81.2
Synapse	1.1->1.2	65.9	52.4	35.0	5.8	57.3	59.3	58.3	79.7	89.3	84.2	79.5	77.2	73.2
Poi	1.5->2.5	68.5	78.8	83.7	9.1	76.1	55.2	64.0	69.1	80.0	73.9	61.3	78.3	68.7
	2.5->3.0	42.6	68.4	44.9	2.7	81.6	79.0	80.3	79.7	77.1	78.6	82.4	77.3	79.6
Ant	1.5->1.6	70.0	-	-	-	88.0	95.1	91.4	75.0	77.3	76.1	92.3	96.4	94.6
	1.6->1.7	59.3	-	-	-	98.8	90.1	94.2	84.7	99.2	1.8	90.4	92.3	79.0
Ivy	1.4->2.0	82.9	-	-	-	21.7	90.0	35.0	77.9	78.0	73.2	84.7	87.3	85.0
Trung bình		64.2	57.6	67.4	61.2	63.0	70.7	64.1	68.3	75.9	71.2	76.2	80.7	76.8

mềm bằng cách nắm bắt được các đặc trưng ngữ nghĩa của chương trình, từ đó cải thiện hiệu suất của mô hình dự đoán lỗi trong cùng dự án.

Tuy nhiên, giống như các mô hình học sâu khác, kết quả này vẫn bị ảnh hưởng bởi kích thước tập dữ liệu nhỏ và sự mất cân bằng giữa các lớp. Những hạn chế này cần được xem xét để nâng cao độ chính xác của mô hình dự đoán.



Hình 4.6: So sánh kết quả F1-score giữa FastText-Transformer và DP-Transformer đối với dự đoán lỗi trong cùng một dự án

4.5.5 Trả lời cho Câu hỏi nghiên cứu 2: Mô hình FastText-Transformer được đề xuất có vượt trội hơn so với các mô hình dự đoán lỗi tiên tiến hiện nay đối với dự đoán lỗi trong liên dự án không?

Việc đánh giá phương pháp đề xuất đối với dự đoán trong liên dự án bao gồm so sánh với DTL-DP [179], DBN-CP [151] và TCA+ [180]. Bảng 4.7 trình bày giá trị F1-score của các phương pháp DTL-DP, DBN-CP, TCA+, FastText-LSTM và FastText-Transformer. Những giá trị F1-score tốt nhất của các phương pháp này được đánh dấu in đậm.

Nhìn chung, mô hình FastText-Transformer đạt hiệu suất tốt hơn so với các phương pháp còn lại về giá trị F1-score trên 12 trong số 22 tập thử nghiệm. Chẳng hạn, trong nhóm thử nghiệm sử dụng Apache Ant 1.6 làm tập huấn luyện và Apache Poi 3.0 làm tập kiểm thử, mô hình đề xuất của luận án đạt giá trị F1-score cao nhất là 0.875. Giá trị trung bình F1-score của FastText-Transformer trong CPDP là 0.667, trong khi các mô hình DTL-DP, TCA+, DBN-CP và FastText-LSTM lần lượt đạt 0.618, 0.479, 0.568 và 0.589. Do đó, mô hình đề xuất của luận án có hiệu suất cao hơn các phương pháp trên lần lượt là 7.9%, 39.2%, 17.4% và 13.2%. Ngoài ra, giá trị F1-score của FastText-Transformer cao hơn so với mô hình kết hợp FastText với LSTM trên hầu hết các tập dữ liệu thử nghiệm. Chẳng hạn, dự án nguồn là Apache Poi 2.5 và dự án đích là Apache Synapse 1.2, mô hình FastText-Transformer đạt giá trị F1-score là 0.774, cao hơn 0.274 so với giá trị 0.50 của mô hình FastText-LSTM.

Mô hình được đề xuất có nhiều ưu điểm hơn so với các mô hình đối sánh. Thứ nhất, FastText có khả năng nắm bắt thông tin về các từ con, giúp phù hợp với các tình huống mà mã nguồn có thể chứa định danh, tên hàm hoặc tên biến không có trong từ vựng tiêu

Bảng 4.7: So sánh giá trị F1-score DTL-DP, DBN-CP, TCA+, FastText-LSTM và FastText-Transformer đối với dự đoán lỗi trong liên dự án

Tập nguồn	Tập đích	DTL-DP	TCA+	DBN-CP	FastText-LSTM	FastText-Transformer
Ant1.6	Camel1.4	39.5	29.2	31.6	36.5	39.8
JEdit4.1	Camel1.4	40.7	33.0	69.3	48.1	49.6
Camel1.4	Ant1.6	59.1	61.6	97.9	81.4	89.7
Poi3.0	Ant1.6	69.3	59.8	47.8	63.8	77.0
Camel1.4	JEdit4.1	53.1	53.7	61.5	56.2	59.6
Log4j1.1	JEdit4.1	63.9	41.9	50.3	54.35	65.1
JEdit4.1	Log4j1.1	78.3	57.4	64.5	69.9	80.2
Lucene2.2	Log4j1.1	79.4	57.1	61.8	72.6	78.2
Lucene2.2	Xalan2.5	68.9	53.0	55.0	79.3	79.2
Xerces1.3	Xalan2.5	68.6	58.1	57.2	78.9	78.8
Xalan2.5	Lucene2.2	78.3	56.1	59.4	69.5	71.5
Log4j1.1	Lucene2.2	76.9	52.4	69.2	81.1	86.4
Xalan2.5	Xerces1.3	40.0	39.4	38.6	41.0	46.5
Ivy2.0	Xerces1.3	42.0	39.8	42.6	47.8	48.9
Xerces1.3	Ivy2.0	47.2	40.9	45.3	47.5	50.0
Synapse1.2	Ivy2.0	49.4	38.3	82.4	46.3	48.1
Ivy1.4	Synapse1.1	54.5	34.8	48.9	39.4	49.9
Poi2.5	Synapse1.1	59.7	37.6	42.5	50.0	77.4
Ivy2.0	Synapse1.2	62.0	57.0	43.3	36.6	73.2
Poi3.0	Synapse1.2	62.3	54.2	51.4	68.3	61.3
Synapse1.2	Poi3.0	66.1	65.1	82.7	49.5	69.3
Ant1.6	Poi3.0	61.9	34.3	82.7	77.7	87.5
Trung bình		61.8	47.9	56.8	58.9	66.7

chuẩn. Hơn nữa, embedding của FastText có thể xử lý hiệu quả các từ ngoài từ vựng và biến thể hình thái, đảm bảo học được các biểu diễn ngay cả khi gặp các token hiếm hoặc chưa từng thấy trong mã nguồn. Do đó, phương pháp đề xuất có thể xử lý biểu diễn có ý nghĩa hơn cho các token chưa thấy trước trong từng tệp mã nguồn. Thứ hai, các nghiên cứu gần đây của Gao và cộng sự [189] và Zi và cộng sự [190] đã chứng minh rằng cơ chế attention có thể là một công cụ mạnh mẽ giúp cải thiện dự báo chuỗi thời gian so với các mô hình RNN truyền thống.

Thứ ba, các nghiên cứu gần đây của Xing và cộng sự [191] và Devi và cộng sự [192] đã chứng minh hiệu quả của cơ chế attention trong việc học các đặc trưng ngữ nghĩa của chương trình. Trong phương pháp được đề xuất, mã nguồn thường có các phụ thuộc cấu trúc phức tạp và mối quan hệ phân cấp giữa các thành phần khác nhau. Những đặc điểm này có thể được mô hình hóa hiệu quả nhờ cơ chế self-attention của Transformer. Mô hình Transformer mang lại nhiều lợi thế và thường có hiệu suất cao hơn so với các mô hình đối

chứng truyền thống như CNN, DBN hoặc LSTM trong các bài toán liên quan đến ngôn ngữ, như đã được đề cập trong các nghiên cứu của Shaik và cộng sự [193], Măhdevaswamy và cộng sự [194]. Một lợi thế quan trọng của kiến trúc Transformer là khả năng xử lý song song hóa, cho phép tận dụng tối đa tốc độ xử lý của GPUs và TPUs. Ngoài ra, trong khi LSTM có cơ chế hồi quy giúp mô hình sử dụng mối quan hệ ngữ cảnh dài hạn và multi-head attention để nắm bắt mối quan hệ phức tạp trong dữ liệu quá khứ để đưa ra dự đoán, thì Transformer lại sử dụng cơ chế self-attention để khai thác toàn bộ thông tin trong quá khứ kết hợp với đầu vào hiện tại, giúp tạo ra dự đoán chính xác hơn [195]. Do đó, mô hình FastText-Transformer của luận án được thiết kế để tận dụng những lợi ích kể trên. Phương pháp đề xuất đạt hiệu suất tốt do dựa vào cơ chế self-attention, giúp cải thiện khả năng dự đoán lỗi phần mềm. Ngoài ra, không giống như LSTM hoặc RNN, Transformer không phụ thuộc vào tính toán tuần tự mà có thể xử lý song song, giúp tăng tốc độ thực thi tổng thể.

4.5.6 Kết quả kiểm định thống kê

Kiểm định thống kê được thực hiện để đánh giá mức độ ý nghĩa của hiệu suất mà phương pháp đề xuất đạt được so với các mô hình cơ sở trong quá trình kiểm chứng thử nghiệm. Dựa trên các kiểm định thống kê, luận án đưa ra kết luận chặt chẽ nhằm mô tả hiệu quả của phương pháp đề xuất so với các mô hình khác. Một số kiểm định thống kê thường được sử dụng trong dự đoán lỗi phần mềm bao gồm kiểm định Friedman [196], Cliff's effect size [197], kiểm định Wilcoxon signed-rank sum [198], v.v. Trong chương này, luận án sử dụng kiểm định Wilcoxon với mức ý nghĩa 0.05 để đánh giá sự khác biệt hiệu suất giữa FastText-Transformer và các mô hình đối sánh.

Bảng 4.8 và Bảng 4.9 trình bày kết quả kiểm định thống kê của FastText-Transformer và các mô hình cơ sở theo các tiêu chí Precision, Recall và F1-score. Trong mỗi bảng, nếu giá trị P -value nhỏ hơn 0.05, sự khác biệt về hiệu suất được coi là có ý nghĩa thống kê và được đánh dấu bằng dấu hoa thị (*), dẫn đến việc bác bỏ giả thuyết null. Giá trị của effect size r trong kiểm định thể hiện mức độ khác biệt về hiệu suất. Giá trị r càng cao cho thấy ảnh hưởng càng mạnh, tức là sự khác biệt về hiệu suất giữa các nhóm được so sánh càng lớn. Giá trị chênh lệch trung bình của mỗi nhóm giúp xác định phương pháp nào có hiệu suất trung bình tốt hơn.

Từ kết quả trong Bảng 4.8, giá trị P -value của hầu hết các nhóm so sánh đều nhỏ hơn 0.05, cho thấy sự khác biệt hiệu suất có ý nghĩa thống kê. Hơn nữa, giá trị effect size r cao hơn 0.2 đối với hầu hết các nhóm, ngoại trừ chỉ số Precision trong Sem1 và chỉ số Recall trong DBN-CP, điều này cũng chỉ ra sự khác biệt đáng kể về hiệu suất. Đối với phần lớn

Bảng 4.8: Kết quả kiểm định thống kê so sánh giữa FastText-Transformer và các mô hình đối sánh đối với dự đoán lỗi trong cùng một dự án

Nhóm so sánh	Độ đo	Precision	Recall	F1-score
DTL-DP	P-value	-	-	0.013*
	<i>Effect r</i>	-	-	0.625
	Mean-difference	-	-	8.054
Seml	P-value	0.015*	0.008*	0.010*
	<i>Effect r</i>	-0.257	0.331	0.069
	Mean-difference	15.899	12.980	13.525
DBN-CP	P-value	0.060	0.012*	0.005*
	<i>Effect r</i>	0.207	-0.059	0.458
	Mean-difference	11.408	14.258	12.508
FastText-LSTM	P-value	0.045*	0.239	0.182
	<i>Effect r</i>	0.444	0.500	0.464
	Mean-difference	7.833	4.133	4.427

các nhóm so sánh, FastText-Transformer có giá trị chênh lệch trung bình dương, chứng tỏ mô hình đạt hiệu suất tốt hơn.

Ngoài ra, sự khác biệt hiệu suất có ý nghĩa thống kê giữa FastText-Transformer và các phương pháp khác cũng được quan sát trong tất cả các trường hợp của dự đoán trong liên dự án trong Bảng 4.9. Có thể thấy, giá trị *P*-value nhỏ hơn 0.05, chứng tỏ sự khác biệt về hiệu suất là có ý nghĩa thống kê. Trong hầu hết các trường hợp, giá trị *r* lớn hơn 0.6, chỉ ra mức độ ảnh hưởng lớn của sự khác biệt về hiệu suất. Giá trị chênh lệch trung bình là số dương đối với phần lớn các nhóm so sánh, cho thấy FastText-Transformer có hiệu suất tốt hơn so với DTL-DP, Seml và DBN-CP. Do đó, kết quả kiểm định thống kê xác nhận rằng phương pháp đề xuất của luận án có hiệu suất vượt trội so với các mô hình đối chứng.

4.5.7 So sánh chi phí thời gian huấn luyện và thời gian kiểm thử mô hình giữa FastText-LSTM và FastText-Transformer

Trong khi tiến hành các thực nghiệm, luận án cũng tính chi phí thời gian của FastText-LSTM và FastText-Transformer. Bảng 4.10 trình bày so sánh chi phí thời gian huấn luyện và thử nghiệm giữa FastText-LSTM và FastText-Transformer trên các dự án khác nhau. FastText-Transformer thể hiện rõ lợi thế về hiệu suất, với chi phí thời gian thấp hơn đáng kể cho tất cả các dự án được liệt kê.

Bảng 4.9: Kết quả kiểm định thống kê cho so sánh giữa FastText-Transformer và các mô hình đối sánh đối với dự đoán lỗi trong liên dự án

Nhóm so sánh	Độ đo	F1-score
DTL-DP	P-value	0.012*
	<i>Effect r</i>	0.837
	Mean-difference	6.584
Seml	P-value	0.001*
	<i>Effect r</i>	0.885
	Mean-difference	19.146
DBN-CP	P-value	0.030*
	<i>Effect r</i>	0.643
	Mean-difference	10.576
FastText-LSTM	P-value	0.003*
	<i>Effect r</i>	0.964
	Mean-difference	5.303

Bảng 4.10: So sánh chi phí thời gian huấn luyện và thử nghiệm mô hình giữa FastText-LSTM và FastText-Transformer

Dự án	Phiên bản		FastText-LSTM	FastText-Transformer	FastText-LSTM	FastText-Transformer
	Tập nguồn	Tập đích	Thời gian huấn luyện		Thời gian kiểm thử	
Camel	1.2	1.4	765.81	160.67	8.53	3.36
JEedit	4.0	4.1	2555.55	2083.49	22.68	18.36
Log4j	1.0	1.1	198.12	55.04	1.73	0.55
Lucene	2.0	2.2	4515.87	1771.83	9.68	6.72
Xalan	2.5	2.6	3520.64	1401.64	20.51	17.36
Xercer	1.2	1.3	3350.52	2232.61	24.05	18.66
Synapse	1.1	1.2	538.08	354.65	5.72	1.48
Poi	1.5	2.5	29480.20	19653.47	303.9	136.5
Ant	1.6	1.7	583.71	216.95	6.66	2.37
Ivy	1.4	2.0	3867.08	2055.80	35.2	13.56
Trung bình			4937.558	2973.26	43.87	21.89

Chẳng hạn, đối với dự án Camel, FastText-Transformer chỉ mất 160.67 giây, trong khi FastText-LSTM mất đến 765.81 giây. Đối với dự án Lucene, mô hình FastText-Transformer nhanh hơn đáng kể, với chi phí thời gian là 1771.83 giây so với 4515.87 giây của FastText-LSTM. Tương tự, với dự án Poi, FastText-Transformer có chi phí thời gian là 19653.47 giây, trong khi FastText-LSTM mất 29480.20 giây. Mô hình FastText-Transformer tiếp tục thể hiện hiệu suất vượt trội trên các dự án khác như JEedit (2083.49 giây so với 2555.55 giây) và Lucene (1771.83 giây so với 4515.87 giây). Trong tất cả các dự án, chi phí thời

gian huấn luyện mô hình dự đoán lỗi dao động từ 198.12 giây đến 29480.20 giây đối với FastText-LSTM và từ 55.4 giây đến 29480.20 giây đối với FastText-Transformer trên cả hai tập dữ liệu Log4j và Poi. Trung bình, FastText-Transformer có chi phí thời gian là 2973.26 giây, thấp hơn đáng kể so với mức trung bình 4937.55 giây của FastText-LSTM, nhấn mạnh hiệu suất thời gian vượt trội của FastText-Transformer so với FastText-LSTM. Về bộ nhớ, do kích thước lớn của các chuỗi embedding token thu thập được, bộ nhớ RAM tối đa (24GB) được sử dụng để huấn luyện từng cặp tập dữ liệu. Số lượng tham số của FastText-LSTM và FastText-Transformer lần lượt là 575.689 và 894.554.

Bên cạnh đó, xét về thời gian kiểm thử, mô hình FastText-Transformer luôn đạt hiệu suất vượt trội so với FastText-LSTM trên tất cả các dự án được đánh giá. Thời gian kiểm thử trung bình của FastText-Transformer vào khoảng 21,89 giây, nhanh hơn gần 50% so với FastText-LSTM (43,87 giây). Chẳng hạn, đối với dự án Xerces, FastText-LSTM mất 24,05 giây, trong khi FastText-Transformer rút ngắn còn 18,66 giây. Ngay cả với dự án Ivy, khi FastText-LSTM mất 35,2 giây, FastText-Transformer chỉ mất 13,56 giây, thể hiện sự cải thiện đáng kể. Đáng chú ý, đối với các dự án có quy mô lớn hơn như Poi và Ivy, mức tiết kiệm thời gian còn rõ rệt hơn, nhấn mạnh hiệu quả của cơ chế self-attention song song trong các kiến trúc dựa trên Transformer.

4.5.8 Các mối đe dọa đến tính tin cậy và giá trị của nghiên cứu

1. Độ giá trị ngoại tại

Độ giá trị ngoại tại xem xét mức độ mà kết quả nghiên cứu có thể được tổng quát hóa. Vì chất lượng của kết quả thử nghiệm phụ thuộc vào tập dữ liệu được sử dụng trong luận án, luận án đã chọn tập dữ liệu công khai thường được sử dụng trong các nghiên cứu về dự đoán lỗi phần mềm. Các tập dữ liệu được sử dụng trong thử nghiệm của luận án được trích xuất từ kho PROMISE và ngôn ngữ lập trình của 10 dự án là Java.

Tuy nhiên, tập dữ liệu thử nghiệm của luận án không thể bao gồm tất cả các loại ứng dụng phần mềm hiện có, và sự thiếu đa dạng này có thể dẫn đến sai lệch trong kết quả nghiên cứu. Để làm cho khung phân tích được đề xuất trở nên phổ quát hơn, luận án đang xem xét mở rộng thử nghiệm sang các loại ứng dụng khác, chẳng hạn như ứng dụng di động và ứng dụng web được viết bằng các ngôn ngữ lập trình khác (ví dụ: Python, C++, JavaScript).

2. Độ giá trị nội tại

Do không có sẵn mã nguồn của các nghiên cứu được chọn để đối sánh, gồm của Li và cộng sự [153], Zheng và cộng sự [178], Wang và cộng sự [151], Chen và cộng sự [179] và Nam và cộng sự [179], luận án đã tái tạo các thử nghiệm của họ dựa

trên các thuật toán được trình bày trong nghiên cứu gốc và thực hiện đánh giá trên cùng một tập dữ liệu. Tuy nhiên, luận án không thể đảm bảo rằng việc thực hiện thử nghiệm của luận án hoàn toàn nhất quán với bản gốc. Mặc dù vậy, luận án tin rằng kết quả so sánh các mô hình cơ sở thu được trong luận án này tương đối gần với kết quả được trình bày trong các bài báo gốc.

Trong luận án này, luận án sử dụng các giá trị mặc định cho các siêu tham số như hàm kích hoạt và số lượng lớp của các mô hình học sâu. Điều này có thể tạo ra một mối đe dọa tiềm ẩn đối với kết quả luận án. Trong tương lai, luận án sẽ thực hiện điều chỉnh các tham số mô hình [71] trong phương pháp đề xuất để cải thiện hiệu suất và độ chính xác.

3. Độ giá trị khái niệm

Độ giá trị khái niệm tập trung vào việc lựa chọn các độ đo đánh giá hiệu suất được sử dụng để đo lường mức độ hiệu quả của các mô hình dự đoán. Trong luận án này, để đánh giá hiệu suất của phương pháp đề xuất, luận án sử dụng các độ đo đánh giá phổ biến nhất: precision, recall, và F1-score. Ba độ đo này đã được áp dụng rộng rãi trong nhiều nghiên cứu trước đây để đánh giá hiệu suất tổng thể của mô hình. Ngoài ra, luận án sử dụng kiểm định thống kê Wilcoxon để xác định liệu có sự khác biệt đáng kể giữa phương pháp đề xuất và các phương pháp khác hay không. Luận án cho rằng các độ đo khác cũng có thể được xem xét, nhưng do hạn chế về thời gian, luận án này tập trung vào các độ đo phổ biến nhất đối với bài toán phân loại nhị phân.

4.6 Kết chương

Sự gia tăng không ngừng về quy mô và độ phức tạp của phần mềm hiện đại đặt ra những thách thức lớn trong việc đảm bảo độ tin cậy của hệ thống phần mềm. Tuy nhiên, các nghiên cứu trước đây dựa trên các đặc trưng phần mềm và dữ liệu lỗi hiện có chưa thể khai thác hiệu quả và hiểu được thông tin ngữ nghĩa cũng như cú pháp trong mã nguồn. Chương này đã tập trung nghiên cứu và phát triển một mô hình dự đoán lỗi phần mềm dựa trên học sâu, kết hợp giữa mô hình nhúng từ Gensim FastText và kiến trúc Transformer, nhằm khắc phục các hạn chế của các phương pháp dự đoán lỗi truyền thống vốn chỉ dựa trên độ đo phần mềm hoặc mô hình học máy tuyến tính. Bằng cách khai thác sâu hơn thông tin ngữ nghĩa và cú pháp ẩn chứa trong mã nguồn, mô hình được đề xuất đã chứng minh năng lực nổi bật trong việc biểu diễn mã nguồn dưới dạng véc-tơ đặc trưng và nâng cao đáng kể hiệu quả dự đoán lỗi. Trước hết, luận án đã tiến hành xây dựng cây cú pháp trừu tượng AST cho từng tệp mã nguồn và trích xuất các token quan trọng. Phương pháp này giúp mã nguồn được biểu diễn dưới dạng cấu trúc có ngữ nghĩa rõ ràng, phản ánh

được mỗi quan hệ cú pháp và luồng điều khiển trong chương trình. Sau đó, FastText được áp dụng để biểu diễn các token này thành véc-tơ embedding liên tục, tận dụng các đặc trưng n-gram ký tự nhằm xử lý hiệu quả các từ ngoài từ vựng và các định danh hiếm trong mã nguồn phần mềm. Tiếp theo, mô hình Transformer Encoder được sử dụng trong dự đoán. Với cơ chế multi-head self-attention và khả năng song song hóa trong huấn luyện, Transformer có thể nắm bắt các phụ thuộc dài hạn giữa các thành phần mã nguồn, đồng thời khắc phục hiện tượng vanishing gradient và exploding gradient vốn tồn tại trong các mô hình tuần tự như RNN hay LSTM. Cấu trúc positional encoding được tích hợp giúp bảo toàn thứ tự cú pháp trong chuỗi token được trích xuất từ AST, đảm bảo rằng mỗi quan hệ giữa các thành phần cú pháp và logic của chương trình vẫn được duy trì trong không gian biểu diễn véc-tơ.

Các thử nghiệm thực nghiệm trên 10 dự án Apache (Ant, Camel, Jedit, Log4j, Lucene, Poi, Synapse, Ivy, Xalan và Xerces) đã được tiến hành để đánh giá hiệu quả của mô hình. Kết quả cho thấy mô hình FastText-Transformer đạt hiệu suất cao hơn đáng kể so với các mô hình học sâu hiện nay như Seml, DP-Transformer, DBN-CP, DTL-DP, và TCA+, trên cả hai loại bài toán dự đoán lỗi phần mềm trong cùng một dự án và trong liên dự án. Cụ thể, mô hình của luận án thể hiện độ đo đánh giá accuracy, recall và F1-score cao hơn, đồng thời thời gian huấn luyện ngắn hơn nhờ cơ chế attention song song và tối ưu hóa bằng Adam optimizer. Kết quả kiểm định Wilcoxon signed-rank test ($p < 0.05$) khẳng định rằng sự khác biệt về hiệu năng giữa mô hình được đề xuất và các mô hình so sánh là có ý nghĩa thống kê, chứng minh tính vượt trội và ổn định của phương pháp đề xuất.

Để làm cho mô hình đề xuất của luận án có tính tổng quát cao hơn, các thử nghiệm có thể sử dụng tập dữ liệu từ nhiều dự án khác nhau và ngôn ngữ lập trình khác nhau. Trong tương lai, luận án xem xét việc sử dụng một số kho lưu trữ khác chứa dữ liệu lỗi trong các dự án trên Github, chẳng hạn như Broadleaf Commerce CE, Multi-Model NoSQL DBMS, Elasticsearch, v.v. Hơn nữa, việc xử lý dữ liệu mất cân bằng là một thách thức lớn đối với các mô hình dự đoán lỗi phần mềm. Do đó, để nâng cao chất lượng của tập dữ liệu, phương pháp đề xuất của luận án có thể được mở rộng bằng cách sử dụng các kỹ thuật lấy mẫu tăng [48] và lấy mẫu giảm [118, 122] nhằm cân bằng phân bố các lớp trong tập dữ liệu cũng như tăng tính đa dạng của các mẫu dữ liệu. Ngoài ra, luận án là mở rộng mô hình dự đoán lỗi phần mềm sang các ngôn ngữ lập trình khác như C++ và Python. Mặc dù mỗi ngôn ngữ có cú pháp và đặc điểm ngữ nghĩa riêng, chúng đều có thể được biểu diễn thông qua cây cú pháp trừu tượng hoặc các biểu diễn mã nguồn có cấu trúc tương đương như đồ thị luồng điều khiển, đồ thị phụ thuộc, cho phép khai thác đồng thời thông tin cú pháp và ngữ nghĩa của chương trình. Trên cơ sở đó, các token được trích xuất từ AST hoặc mã nguồn có thể được chuyển đổi thành các biểu diễn ngữ nghĩa bằng các mô hình embedding hoặc các mô hình ngôn ngữ tiền huấn luyện dành cho mã nguồn như CodeBERT, Graph

Neural Network, GraphCodeBERT. Các biểu diễn này sau đó được đưa vào các mô hình học sâu dựa trên Transformer, vốn có khả năng học các quan hệ phụ thuộc dài hạn giữa các thành phần của mã nguồn thông qua cơ chế self-attention, đồng thời nắm bắt hiệu quả các đặc trưng ngữ nghĩa và ngữ cảnh lập trình. Việc kết hợp giữa phân tích cú pháp của ngôn ngữ lập trình và các mô hình Transformer tiền huấn luyện không chỉ giúp nâng cao khả năng biểu diễn mã nguồn mà còn kỳ vọng cải thiện độ chính xác và khả năng tổng quát hóa của mô hình dự đoán lỗi trên nhiều ngôn ngữ lập trình khác nhau. Để nâng cao hơn nữa khả năng khái quát hóa của mô hình được đề xuất, các thử nghiệm trong tương lai có thể sử dụng tập dữ liệu đến từ nhiều dự án khác nhau. Việc này sẽ giúp mô hình được đánh giá toàn diện hơn về tính ổn định, khả năng thích ứng và hiệu quả dự đoán trong các bối cảnh phát triển phần mềm khác nhau, qua đó củng cố tính tổng quát và khả năng ứng dụng thực tiễn của mô hình trong môi trường phát triển phần mềm quy mô lớn.

Các kết quả nghiên cứu này đã được đăng trên Tạp chí Neural Computing and Applications.¹

¹Ha Thi Minh Phuong, Pham Vu Thu Nguyet, Nguyen Huu Nhat Minh, Le Thi My Hanh, John Healy, Nguyen Thanh Binh “**A novel transformer-based software fault prediction using Syntactic Tree and Word Embedding**”, *Neural Computing and Applications*, 2026, (Scopus, Q1)

Kết luận và hướng phát triển

Với mục tiêu ban đầu, Luận án tiến sĩ với tiêu đề: “Dự đoán phần mềm dựa trên mã nguồn” này đã tập trung nghiên cứu và đề xuất các phương pháp nhằm cải thiện hiệu suất của mô hình dự đoán lỗi phần mềm, đặc biệt trong bối cảnh dữ liệu có sự mất cân bằng và đặc trưng ngữ nghĩa phong phú. Luận án đã có các đóng góp chính sau:

- Xây dựng cơ sở lý luận và định hướng nghiên cứu cho bài toán dự đoán lỗi phần mềm dựa trên mã nguồn. Luận án đã tổng quan có hệ thống các phương pháp dự đoán lỗi phần mềm, phân tích ưu điểm, hạn chế của các hướng tiếp cận dựa trên độ đo phần mềm và đặc trưng mã nguồn. Trên cơ sở đó, luận án chỉ ra ba vấn đề cốt lõi còn tồn tại gồm: (i) dữ liệu đầu vào chứa nhiều đặc trưng dư thừa; (ii) sự mất cân bằng giữa lớp lỗi và không lỗi; và (iii) các mô hình dự đoán lỗi truyền thống dựa trên độ đo chưa khai thác hiệu quả thông tin ngữ nghĩa của mã nguồn. Việc xác định rõ các khoảng trống nghiên cứu này là cơ sở để xây dựng một hướng tiếp cận thống nhất nhằm nâng cao hiệu quả dự đoán lỗi phần mềm.
- Đề xuất phương pháp tối ưu hóa dữ liệu đầu vào cho dự đoán lỗi phần mềm thông qua kết hợp lựa chọn đặc trưng, trích xuất đặc trưng và cân bằng dữ liệu. Cụ thể, luận án kết hợp phương pháp Recursive Feature Elimination (RFE) và Autoencoder để tối ưu tập đặc trưng với các mô hình mạng đối kháng sinh (GAN) nhằm giải quyết bài toán mất cân bằng dữ liệu. Khác với các phương pháp lấy mẫu tăng truyền thống như SMOTE và ADASYN, các mô hình GAN có khả năng học phân phối xác suất tổng thể của dữ liệu và mối tương quan giữa các đặc trưng, từ đó sinh ra các mẫu dữ liệu tổng hợp có tính chân thực và đa dạng cao hơn. Luận án nghiên cứu và đánh giá hiệu quả của nhiều biến thể GAN, bao gồm VanillaGAN, CTGAN, WGAN-GP, CopulaGAN và TGAN, đồng thời đề xuất kết hợp chúng với các kỹ thuật lựa chọn và trích xuất đặc trưng. Kết quả thực nghiệm cho thấy sự kết hợp giữa RFE, Autoencoder và các biến thể GAN, đặc biệt là CTGAN, WGAN-GP và VanillaGAN, tạo ra tập dữ liệu huấn luyện chất lượng cao, giúp các mô hình dự đoán lỗi phần mềm đạt hiệu suất vượt trội so với các phương pháp lấy mẫu truyền

thống, đồng thời nâng cao độ chính xác, khả năng khái quát hóa và tính ổn định của mô hình.

- Đề xuất mô hình FastText-Transformer mới dựa trên kiến trúc Transformer kết hợp với biểu diễn từ FastText đã thể hiện khả năng học đặc trưng ngữ nghĩa sâu sắc từ mã nguồn dựa trên cây cú pháp trừu tượng, từ đó cải thiện đáng kể hiệu suất mô hình dự đoán lỗi, đặc biệt trong bối cảnh dự đoán trong cùng và dự đoán giữa các dự án. Mô hình được xây dựng bằng cách kết hợp biểu diễn từ FastText trên cây cú pháp trừu tượng (AST) với kiến trúc Transformer Encoder nhằm học đồng thời đặc trưng cú pháp và ngữ nghĩa của mã nguồn. Kiến trúc FastText-Transformer do luận án đề xuất cho phép mô hình hóa hiệu quả cấu trúc và ngữ nghĩa của mã nguồn, giúp cải thiện đáng kể độ chính xác, khả năng khái quát hóa và ổn định trong dự đoán lỗi. Kết quả thực nghiệm trên nhiều dự án Java trong cả thử nghiệm trong cùng một dự án WPDP và giữa các dự án CPDP cho thấy mô hình đạt hiệu suất vượt trội so với nhiều phương pháp học sâu hiện có. Sự khác biệt được xác nhận bằng kiểm định Wilcoxon với mức ý nghĩa thống kê phù hợp. Kết quả này chứng minh tính đúng đắn, tính ổn định và khả năng khái quát hóa của mô hình, đồng thời mở ra hướng nghiên cứu mới trong việc khai thác biểu diễn ngữ nghĩa của mã nguồn cho các mô hình dự đoán lỗi phần mềm.

Ngoài các kết quả đã đạt được trong luận án, một số vấn đề có thể đặt ra để tiếp tục nghiên cứu:

- Các thuật toán tối ưu hóa meta-heuristic mặc dù cho thấy hiệu quả nhất định trong lựa chọn đặc trưng, nhưng vẫn tồn tại nhiều hạn chế cần được khắc phục. Cụ thể, các mô hình hiện nay thường gặp vấn đề về độ phức tạp tính toán cao, hàm mục tiêu đơn mục tiêu, quy mô tập dữ liệu nhỏ, khởi tạo quần thể ban đầu yếu, cũng như thiếu các so sánh chuẩn hóa với những thuật toán khác. Ngoài ra, một số thử nghiệm cho thấy các thuật toán có thể phát hiện những đặc trưng không dư thừa ngay cả khi đang loại bỏ các đặc trưng dư thừa, dẫn đến mâu thuẫn trong việc xác định tập đặc trưng tối ưu. Vì vậy, hướng nghiên cứu tiếp theo có thể tập trung vào việc tối ưu hóa và cải tiến các thuật toán meta-heuristic trong lựa chọn đặc trưng, nhằm nâng cao tốc độ hội tụ, độ ổn định và khả năng tìm kiếm nghiệm tối ưu toàn cục.
- Các mô hình dự đoán lỗi phần mềm chỉ hoạt động ở mức tệp hoặc mức lớp do đó chưa thể đạt được độ chính xác cao trong việc xác định vị trí lỗi cụ thể trong mã nguồn. Do đó, một hướng nghiên cứu tiềm năng trong tương lai là mở rộng mô hình theo hướng định vị lỗi (fault localization) bằng cách khai thác các biểu diễn ngữ nghĩa của mã nguồn kết hợp với thông tin cấu trúc chương trình để xác định các lớp, phương thức hoặc thậm chí các đoạn mã có khả năng gây lỗi cao nhất. Hướng

tiếp cận này không chỉ giúp thu hẹp phạm vi kiểm tra, giảm thời gian gỡ lỗi mà còn hỗ trợ hiệu quả cho các hoạt động kiểm thử, bảo trì và sửa lỗi tự động trong quy trình phát triển phần mềm.

- Trong luận án này, việc trích xuất đặc trưng mã nguồn dựa trên cây cú pháp trừu tượng chỉ phản ánh cấu trúc cú pháp tĩnh, mà chưa biểu diễn đầy đủ luồng điều khiển và sự phụ thuộc dữ liệu giữa các thành phần mã nguồn. Do đó, các thông tin ngữ nghĩa động và quan hệ ngữ cảnh giữa các khối lệnh chưa được khai thác triệt để. Vì vậy, công việc nghiên cứu tiếp theo có thể mở rộng theo hướng khai thác cấu trúc đồ thị của mã nguồn, bao gồm đồ thị luồng điều khiển (Control Flow Graph), đồ thị phụ thuộc chương trình (Program Dependency Graph) hoặc đồ thị thuộc tính mã nguồn (Code Property Graph) để biểu diễn sâu hơn các mối quan hệ cú pháp và ngữ nghĩa trong chương trình. Cách tiếp cận này cho phép trích xuất đặc trưng cú pháp và đặc trưng ngữ nghĩa từ các đồ thị, từ đó cải thiện khả năng nhận diện các đoạn mã nguồn có khả năng bị lỗi.
- Các mô hình đề xuất hiện tại mới được thử nghiệm trên mã nguồn Java; để tăng tính tổng quát và khả năng ứng dụng thực tế của mô hình, luận án sẽ mở rộng sang nhiều lĩnh vực và ngôn ngữ lập trình khác nhau, chẳng hạn như Python, JavaScript, C++, và ứng dụng cho dự đoán lỗi ở các mức độ chi tiết hơn như lớp, phương thức hay câu lệnh. Đồng thời, cần xem xét cải tiến các chiến lược lấy mẫu và điều chỉnh tham số để mô hình hoạt động hiệu quả hơn trong các môi trường dữ liệu phức tạp và khắt khe hơn.

Các công trình khoa học đã công bố

1. Ha Thi Minh Phuong, Pham Vu Thu Nguyet, Nguyen Huu Nhat Minh, Le Thi My Hanh, John Healy, Nguyen Thanh Binh, “A novel transformer-based software fault prediction using Syntactic Tree and Word Embedding”, *Neural Computing and Applications*, 38.6 (2026): 165, ISSN: 1433-3058, url: <https://doi.org/10.1007/s00521-025-11771-9>, (Scopus, Q1).
2. Ha Thi Minh Phuong, Pham Vu Thu Nguyet, Nguyen Huu Nhat Minh, Le Thi My Hanh, Nguyen Thanh Binh, “A comparative study of handling imbalanced data using generative adversarial networks for machine learning based software fault prediction”, *Applied Intelligence*, ISSN: 1573-7497, 2025, url: <https://doi.org/10.1007>, (SCIE Q2, IF 3.5)
3. Ha Thi Minh Phuong, Dang Kim Ngan, Dao Khanh Duy, Nguyen Thanh Binh, “Enhancing Software Fault Prediction Using Wrapper-Based Metaheuristic Feature Selection Methods. International Journal of Electrical and Computer Engineering”, *International Journal of Electrical and Computer Engineering*, ISSN: 20888708, Vol 15, No.5, pp 4803-4812, 2025, url: <http://doi.org/10.11591/ijece.v15i5.pp4803-4812>.
4. Ha Thi Minh Phuong, Dao Khanh Duy, Nguyen Do Anh Nhu, Hoang Thi Thanh Ha, “Feature selection using the zebra optimization algorithm for software fault prediction: a study on the bughunter dataset”, *The University Of Danang - Journal Of Science And Technology*, ISSN: 1859-1531. Vol: 23, No: 9C, 2025.
5. Ha Thi Minh Phuong, Dang Kim Ngan, Nguyen Thanh Binh, “A comparative study of deep learning techniques in software fault prediction”, *The University Of Danang - Journal Of Science And Technology*, ISSN: 1859-1531. Vol: 22, No: 6B, 2024.
6. Nguyen Thanh Long, Ha Thi Minh Phuong, Nguyen Thanh Binh, “A combination of feature selection and data sampling techniques for software fault prediction”, *Hội nghị Khoa học công nghệ Quốc gia lần thứ XVI về Nghiên cứu cơ bản và ứng dụng*

Công nghệ thông tin (FAIR), Pages: 258–265, Đà Nẵng, 28-29/9/2023.

7. Nguyen Thanh Long, Ha Thi Minh Phuong, Nguyen Thanh Binh, “A Comparative Study of Wrapper Feature Selection Techniques in Software Fault Prediction”, *CITA 2023: The 12th Conference on Information Technology and Its Applications*. Số: Part of the Lecture Notes in Networks and Systems book series (LNNS, volume 734), ISBN: 978-3-031-36886-8. Pages: 62–73. Năm 2023..
8. Ha Thi Minh Phuong, Le Thi My Hanh, Nguyen Thanh Binh, “A Study of Filter-Based Feature Selection in Software Fault Prediction”, *ICIT 2022: Intelligence of Things: Technologies and Applications*, No: 148. Pages: 58-67, 2022.
9. Ha Thi Minh Phuong, Le Thi My Hanh, Nguyen Thanh Binh, “A Comparative Analysis of Filter-based Feature Selection Methods for Software Fault Prediction”, *Chuyên san Các công trình nghiên cứu, phát triển và ứng dụng Công nghệ Thông tin và Truyền thông*, ISSN: 1859-3526. Trang: 1-7. Năm 2021

Tài liệu tham khảo

- [1] Ömer Faruk Arar and Kürşat Ayan. Software defect prediction using cost-sensitive neural network. *Applied Soft Computing*, 33:263–277, 2015.
- [2] B. S. Ainapure. *Software Testing & Quality Assurance*. Technical Publications, Pune, Maharashtra, IND, 1st edition, 2014.
- [3] Rakesh Rana, Mirosław Staron, Jörgen Hansson, and Martin Nilsson. Defect prediction over software life cycle in automotive domain state of the art and road map for future. In *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, pages 377–382. IEEE, 2014.
- [4] A panel of cio’s and senior technology professionals, 2022. <https://www.it-cisq.org/wp-content/uploads/sites/6/2022/11/CPSQ-Report-Nov-22-2.pdf>.
- [5] Fiorenza Brady. Cambridge university study states software bugs cost economy 312billionperyear. *CambridgeUniversity*, 2013.
- [6] Bashar Rajoub. Supervised and unsupervised learning. In *Biomedical signal processing and artificial intelligence in healthcare*, pages 51–89. Elsevier, 2020.
- [7] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, 39(6):757–773, 2012.
- [8] Zhiyuan Wan, Xin Xia, Ahmed E Hassan, David Lo, Jianwei Yin, and Xiaohu Yang. Perceptions, expectations, and challenges in defect prediction. *IEEE Transactions on Software Engineering*, 46(11):1241–1266, 2018.
- [9] Rahul Yedida and Tim Menzies. On the value of oversampling for deep learning in software defect prediction. *IEEE Transactions on Software Engineering*, 48(8):3103–3116, 2021.

-
- [10] Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang, and Ayşe Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17:375–407, 2010.
- [11] Maurice H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., USA, 1977.
- [12] Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.
- [13] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.
- [14] Rachel Harrison, Steve J Counsell, and Reuben V Nithi. An evaluation of the mood set of object-oriented software metrics. *IEEE Transactions on Software Engineering*, 24(6):491–496, 1998.
- [15] Tao Wang and Wei-hua Li. Naive bayes software defect prediction model. In *2010 International conference on computational intelligence and software engineering*, pages 1–4. Ieee, 2010.
- [16] Karim O Elish and Mahmoud O Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, 2008.
- [17] P Reena and R Binu. Software defect prediction system—decision tree algorithm with two level data pre-processing. *International Journal of Engineering Research & Technology (IJERT)*, 3(3), 2014.
- [18] Lan Guo, Yan Ma, Bojan Cukic, and Harshinder Singh. Robust prediction of fault-proneness by random forests. In *15th international symposium on software reliability engineering*, pages 417–428. IEEE, 2004.
- [19] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on software engineering*, 39(9):1208–1215, 2013.
- [20] Sushant Kumar Pandey, Ravi Bhushan Mishra, and Anil Kumar Tripathi. Machine learning based methods for software fault prediction: A survey. *Expert Systems with Applications*, 172:114595, 2021.
- [21] Misbah Ali, Tehseen Mazhar, Tariq Shahzad, Yazeed Yasin Ghadi, Syed Muhammad Mohsin, Syed Muhammad Abrar Akber, and Mohammed Ali. Analysis of feature selection methods in software defect prediction models. *IEEE Access*, 11:145954–145974, 2023.

-
- [22] Chao Ni, Wang-Shu Liu, Xiang Chen, Qing Gu, Dao-Xu Chen, and Qi-Guo Huang. A cluster based feature selection method for cross-project software defect prediction. *Journal of Computer Science and Technology*, 32:1090–1107, 2017.
- [23] Fei Wang, Jun Ai, and Zhuoliang Zou. A cluster-based hybrid feature selection method for defect prediction. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pages 1–9. IEEE, 2019.
- [24] Savina Colaco, Sujit Kumar, Amrita Tamang, and Vinai George Bijju. A review on feature selection algorithms. *Emerging Research in Computing, Information, Communication and Applications: ERCICA 2018, Volume 2*, pages 133–153, 2019.
- [25] Ehsan Elahi, Saima Kanwal, and Ali Nouman Asif. A new ensemble approach for software fault prediction. In *2020 17th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 407–412, 2020.
- [26] Cholmyong Pak, Tian Tian Wang, and Xiao Hong Su. An empirical study on software defect prediction using over-sampling by smote. *International Journal of Software Engineering and Knowledge Engineering*, 28(06):811–830, 2018.
- [27] Lina Gong, Shujuan Jiang, and Li Jiang. Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering. *IEEE Access*, 7:145725–145737, 2019.
- [28] Qinbao Song, Yuchen Guo, and Martin Shepperd. A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering*, 45(12):1253–1269, 2018.
- [29] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.
- [30] Mardhiya Hayaty, Siti Muthmainah, and Syed Muhammad Ghufra. Random and synthetic over-sampling approach to resolve data imbalance in classification. *International Journal of Artificial Intelligence Research*, 4(2):86–94, 2020.
- [31] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.
- [32] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *Advances in Intelligent Computing: International Conference on Intelligent Computing*, pages 878–887. Springer, 2005.

-
- [33] Pooja Paramshetti and DA Phalke. Survey on software defect prediction using machine learning techniques. *International Journal of Science and Research*, 3(12):1394–1397, 2014.
- [34] Guanjun Lin, Jun Zhang, Wei Luo, Lei Pan, Yang Xiang, Olivier De Vel, and Paul Montague. Cross-project transfer representation learning for vulnerable function discovery. *IEEE Transactions on Industrial Informatics*, 14(7):3289–3297, 2018.
- [35] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. Deepfix: Fixing common c language errors by deep learning. In *Proceedings of the aaai conference on artificial intelligence*, volume 31, 2017.
- [36] Xuan Zhou and Lu Lu. Defect prediction via lstm based on sequence and tree structure. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pages 366–373. IEEE, 2020.
- [37] Anum Kalsoom, Muazzam Maqsood, Mustansar Ali Ghazanfar, Farhan Aadil, and Seungmin Rho. A dimensionality reduction-based efficient software fault prediction using fisher linear discriminant analysis (flda). *The Journal of Supercomputing*, 74(9):4568–4602, 2018.
- [38] Md Razu Ahmed, Md Asraf Ali, Nasim Ahmed, Md Fahad Bin Zamal, and FM Javed Mehedi Shamrat. The impact of software fault prediction in real-world application: An automated approach for software engineering. In *Proceedings of 2020 the 6th international conference on computing and data engineering*, pages 247–251, 2020.
- [39] Md Nasir Uddin, Bixin Li, Md Naim Mondol, Md Mostafizur Rahman, Md Suman Mia, and Elizabeth Lisa Mondol. Sdp-ml: an automated approach of software defect prediction employing machine learning techniques. In *2021 International Conference on Electronics, Communications and Information Technology (ICECIT)*, pages 1–4. IEEE, 2021.
- [40] NC Shrikanth and Tim Menzies. Assessing practitioner beliefs about software defect prediction. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, pages 182–190, 2020.
- [41] Cong Pan, Minyan Lu, Biao Xu, and Houleng Gao. An improved cnn model for within-project software defect prediction. *Applied Sciences*, 9(10):2138, 2019.
- [42] Jiehan Deng, Lu Lu, and Shaojian Qiu. Software defect prediction via lstm. *IET software*, 14(4):443–450, 2020.
- [43] Santosh S Rathore and Sandeep Kumar. A study on software fault prediction techniques. *Artificial Intelligence Review*, 51:255–327, 2019.

-
- [44] Santosh Singh Rathore and Atul Gupta. A comparative study of feature-ranking and feature-subset selection techniques for improved fault prediction. In *Proceedings of the 7th India software engineering conference*, pages 1–10, 2014.
- [45] Brian H Sellers. Object-oriented metrics: measures of complexity. *PH PTR, New Jersey*, 1996.
- [46] Lionel C Briand, Jürgen Wüst, John W Daly, and D Victor Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of systems and software*, 51(3):245–273, 2000.
- [47] Y Lee. Measuring the coupling and cohesion of an object-oriented program based on information flow. In *Proc. Int’l Conf. Software Quality, 1995*, 1995.
- [48] Santosh Singh Rathore, Satyendra Singh Chouhan, Dixit Kumar Jain, and Aakash Gopal Vachhani. Generative oversampling methods for handling imbalanced data in software fault prediction. *IEEE Transactions on Reliability*, 71(2):747–762, 2022.
- [49] Sonika Chandrakant Rathi, Sanjay Misra, Ricardo Colomo-Palacios, R Adarsh, Lalita Bhanu Murthy Neti, and Lov Kumar. Empirical evaluation of the performance of data sampling and feature selection techniques for software fault prediction. *Expert Systems with Applications*, 223:119806, 2023.
- [50] Ruchika Malhotra and Kishwar Khan. A study on software defect prediction using feature extraction techniques. In *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pages 1139–1144. IEEE, 2020.
- [51] Satyendra Singh Chouhan and Santosh Singh Rathore. Generative adversarial networks-based imbalance learning in software aging-related bug prediction. *IEEE Transactions on Reliability*, 70(2):626–642, 2021.
- [52] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3(null):1157–1182, March 2003.
- [53] Zhou Xu, Jin Liu, Xiapu Luo, Zijiang Yang, Yifeng Zhang, Peipei Yuan, Yutian Tang, and Tao Zhang. Software defect prediction based on kernel pca and weighted extreme learning machine. *Information and Software Technology*, 106:182–200, 2019.
- [54] Rizgar Zebari, Adnan Abdulazeez, Diyar Zeebaree, Dilovan Zebari, and Jwan Saeed. A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction. *Journal of Applied Science and Technology Trends*, 1(1):56–70, 2020.

-
- [55] Utkarsh Mahadeo Khaire and R Dhanalakshmi. Stability of feature selection algorithm: A review. *Journal of King Saud University-Computer and Information Sciences*, 34(4):1060–1073, 2022.
- [56] Yahaya Zakariyau Bala, Pathiah Abdul Samat, Khaironi Yatim Sharif, and Noridayu Manshor. Improving cross-project software defect prediction method through transformation and feature selection approach. *IEEE Access*, 11:2318–2326, 2022.
- [57] Sweta Mehta and K Sridhar Patnaik. Improved prediction of software defects using ensemble machine learning techniques. *Neural Computing and Applications*, 33:10551–10562, 2021.
- [58] Richard Bellman and Robert Kalaba. Dynamic programming and statistical communication theory. *Proceedings of the National Academy of Sciences*, 43(8):749–751, 1957.
- [59] Zeeshan Ali Rana, Mian M Awais, and Shafay Shamail. Impact of using information gain in software defect prediction models. In *International Conference on Intelligent Computing*, pages 637–648. Springer, 2014.
- [60] Chakkrit Tantithamthavorn, Ahmed E Hassan, and Kenichi Matsumoto. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Transactions on Software Engineering*, 46(11):1200–1219, 2018.
- [61] Manzura Jorayeva, Akhan Akbulut, Cagatay Catal, and Alok Mishra. Machine learning-based software defect prediction for mobile applications: A systematic literature review. *Sensors*, 22(7):2551, 2022.
- [62] Lei Qiao and Yan Wang. Effort-aware and just-in-time defect prediction with neural network. *PloS one*, 14(2):e0211359, 2019.
- [63] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. Mode: Automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018*, page 175–186, New York, NY, USA, 2018. Association for Computing Machinery.
- [64] Vincent J. Hellendoorn, Christian Bird, Earl T. Barr, and Miltiadis Allamanis. Deep learning type inference. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of*

Software Engineering, ESEC/FSE 2018, page 152–162, New York, NY, USA, 2018. Association for Computing Machinery.

- [65] Cheng Zeng, Chun Ying Zhou, Sheng Kai Lv, Peng He, and Jie Huang. Gcn2defect : Graph convolutional networks for smotetomek-based software defect prediction. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 69–79, 2021.
- [66] Jiaqi Yan, Guanhua Yan, and Dong Jin. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 52–63, 2019.
- [67] Nasraldeen Alnor Adam Khleel and Károly Nehéz. Software defect prediction using a bidirectional lstm network combined with oversampling techniques. *Cluster Computing*, pages 1–24, 2023.
- [68] Chanathip Pornprasit and Chakkrit Kla Tantithamthavorn. Deeplinedp: Towards a deep learning approach for line-level defect prediction. *IEEE Transactions on Software Engineering*, 49(1):84–98, 2022.
- [69] Lucija Šikić, Adrian Satja Kurdiya, Klemo Vladimir, and Marin Šilić. Graph neural network for source code defect prediction. *IEEE access*, 10:10402–10415, 2022.
- [70] Hao Wang, Weiyuan Zhuang, and Xiaofang Zhang. Software defect prediction based on gated hierarchical lstms. *IEEE Transactions on Reliability*, 70(2):711–727, 2021.
- [71] Iqra Batool and Tamim Ahmed Khan. Software fault prediction using deep learning techniques. *Software Quality Journal*, 31(4):1241–1280, 2023.
- [72] Ahmed Abdu, Zhengjun Zhai, Hakim A Abdo, and Redhwan Algabri. Software defect prediction based on deep representation learning of source code from contextual syntax and semantic graph. *IEEE Transactions on Reliability*, 73(2):820–834, 2024.
- [73] Chunying Zhou, Peng He, Cheng Zeng, and Ju Ma. Software defect prediction with semantic and structural information of codes based on graph neural networks. *Information and Software Technology*, 152:107057, 2022.
- [74] Md Nakhla Rafi, Dong Jae Kim, An Ran Chen, Tse-Hsun Chen, and Shaowei Wang. Towards better graph neural network-based fault localization through enhanced code representation. *Proceedings of the ACM on Software Engineering*, 1(FSE):1937–1959, 2024.

-
- [75] Shivkumar Shivaji, E James Whitehead, Ram Akella, and Sunghun Kim. Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering*, 39(4):552–569, 2012.
- [76] Bartosz Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in artificial intelligence*, 5(4):221–232, 2016.
- [77] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.
- [78] Md Nasir Uddin, Bixin Li, Zafar Ali, Pavlos Kefalas, Inayat Khan, and Islam Zada. Software defect prediction employing bilstm and bert-based semantic feature. *Soft Computing*, 26(16):7877–7891, 2022.
- [79] Jiaxi Xu, Jun Ai, Jingyu Liu, and Tao Shi. Acgdp: An augmented code graph-based system for software defect prediction. *IEEE Transactions on Reliability*, 71(2):850–864, 2022.
- [80] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.
- [81] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63:3–42, 2006.
- [82] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995.
- [83] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- [84] Aleksei Guryanov. Histogram-based algorithm for building gradient boosting ensembles of piecewise linear decision trees. In *Analysis of Images, Social Networks and Texts: 8th International Conference, Kazan, Russia*, pages 39–50. Springer, 2019.
- [85] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.
- [86] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.
- [87] Taghi M Khoshgoftaar, Alireza Fazelpour, Huanjing Wang, and Randall Wald. A survey of stability analysis of feature subset selection techniques. In *2013 IEEE 14th International Conference on Information Reuse & Integration (IRI)*, pages 424–431. IEEE, 2013.

-
- [88] Kun Zhu, Shi Ying, Nana Zhang, and Dandan Zhu. Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. *Journal of Systems and Software*, 180:111026, 2021.
- [89] Yoginee Surendra Pethe, Mahendra Kumar Gourisaria, Pradeep Kumar Singh, and Himansu Das. Fsboa: feature selection using bat optimization algorithm for software fault detection. *Discover Internet of Things*, 4(1):6, 2024.
- [90] Kechao Wang, Lin Liu, Chengjun Yuan, and Zhifei Wang. Software defect prediction model based on lasso-svm. *Neural Computing and Applications*, 33:8249–8259, 2021.
- [91] Abdullateef O Balogun, Shuib Basri, Saipunidzam Mahamad, Luiz Fernando Capretz, Abdullahi Abubakar Imam, Malek A Almomani, Victor E Adeyemo, and Ganesh Kumar. A novel rank aggregation-based hybrid multifilter wrapper feature selection method in software defect prediction. *Computational intelligence and neuroscience*, 2021(1):5069016, 2021.
- [92] Ruchika Malhotra, Sonali Chawla, and Anjali Sharma. Software defect prediction based on multi-filter wrapper feature selection and deep neural network with attention mechanism. *Neural Computing and Applications*, pages 1–28, 2025.
- [93] Li-qiong Chen, Can Wang, and Shi-long Song. Software defect prediction based on nested-stacking and heterogeneous feature selection. *Complex & Intelligent Systems*, 8(4):3333–3348, 2022.
- [94] Hamza Turabieh, Majdi Mafarja, and Xiaodong Li. Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. *Expert systems with applications*, 122:27–42, 2019.
- [95] Xiangbo Qi, Yunlong Zhu, and Hao Zhang. A new meta-heuristic butterfly-inspired algorithm. *Journal of computational science*, 23:226–239, 2017.
- [96] Weiguo Zhao, Liying Wang, and Zhenxing Zhang. Atom search optimization and its application to solve a hydrogeologic parameter estimation problem. *Knowledge-Based Systems*, 163:283–304, 2019.
- [97] Afshin Faramarzi, Mohammad Heidarinejad, Brent Stephens, and Seyedali Mirjalili. Equilibrium optimizer: A novel optimization algorithm. *Knowledge-based systems*, 191:105190, 2020.
- [98] Fatma A Hashim, Essam H Houssein, Mai S Mabrouk, Walid Al-Atabany, and Seyedali Mirjalili. Henry gas solubility optimization: A novel physics-based algorithm. *Future Generation Computer Systems*, 101:646–667, 2019.

-
- [99] Yuhong Li, Wen Zhang, Li Wang, Fenghua Zhao, Wei Han, and Gaochao Chen. Henry's law and accumulation of weak source for crust-derived helium: a case study of weihe basin, china. *Journal of Natural Gas Geoscience*, 2(5-6):333–339, 2017.
- [100] Seyyed Hamid Samareh Moosavi and Vahid Khatibi Bardsiri. Poor and rich optimization algorithm: A new human-based and multi populations algorithm. *Engineering applications of artificial intelligence*, 86:165–181, 2019.
- [101] Yiying Zhang, Zhigang Jin, and Seyedali Mirjalili. Generalized normal distribution optimization and its applications in parameter extraction of photovoltaic models. *Energy Conversion and Management*, 224:113301, 2020.
- [102] Shimin Li, Huiling Chen, Mingjing Wang, Ali Asghar Heidari, and Seyedali Mirjalili. Slime mould algorithm: A new method for stochastic optimization. *Future generation computer systems*, 111:300–323, 2020.
- [103] Ali Asghar Heidari, Seyedali Mirjalili, Hossam Faris, Ibrahim Aljarah, Majdi Mafarja, and Huiling Chen. Harris hawks optimization: Algorithm and applications. *Future generation computer systems*, 97:849–872, 2019.
- [104] Hamza Yapici and Nurettin Cetinkaya. A new meta-heuristic optimizer: Pathfinder algorithm. *Applied soft computing*, 78:545–568, 2019.
- [105] Weiguo Zhao, Zhenxing Zhang, and Liying Wang. Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications. *Engineering Applications of Artificial Intelligence*, 87:103300, 2020.
- [106] A Balaram and S Vasundra. Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm. *Automated Software Engineering*, 29(1):6, 2022.
- [107] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.
- [108] Yuan Xie and Tao Zhang. Imbalanced learning for fault diagnosis problem of rotating machinery based on generative adversarial networks. In *2018 37th Chinese Control Conference (CCC)*, pages 6017–6022. IEEE, 2018.
- [109] CopulaGAN. Copulagan model, 2023. Available:<https://docs.sdv.dev/sdv/single-table-data/modeling/synthesizers/copulagansynthesizer>.

-
- [110] Lei Xu and Kalyan Veeramachaneni. Synthesizing tabular data using generative adversarial networks, 11 2018.
- [111] Ying Sun, Xiao-Yuan Jing, Fei Wu, Juanjuan Li, Danlei Xing, Haowen Chen, and Yanfei Sun. Adversarial learning for cross-project semi-supervised defect prediction. *IEEE Access*, 8:32674–32687, 2020.
- [112] Ehsan Elahi, Saima Kanwal, and Ali Nouman Asif. A new ensemble approach for software fault prediction. In *2020 17th international Bhurban conference on applied sciences and technology (IBCAST)*, pages 407–412. IEEE, 2020.
- [113] Ankush Joon, Rajesh Kumar Tyagi, and Krishan Kumar. Noise filtering and imbalance class distribution removal for optimizing software fault prediction using best software metrics suite. In *2020 5th international conference on communication and electronics systems (ICCES)*, pages 1381–1389. IEEE, 2020.
- [114] Sushant Kumar Pandey and Anil Kumar Tripathi. An empirical study toward dealing with noise and class imbalance issues in software defect prediction. *Soft Computing*, 25(21):13465–13492, 2021.
- [115] Ruchika Malhotra and Shine Kamal. An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data. *Neurocomputing*, 343:120–140, 2019.
- [116] Abdullateef O Balogun, Fatimah B Lafenwa-Balogun, Hammed A Mojeed, Fatimah E Usman-Hamza, Amos O Bajeh, Victor E Adeyemo, Kayode S Adewole, and Rasheed G Jimoh. Data sampling-based feature selection framework for software defect prediction. In *International Conference on Emerging Applications and Technologies for Industry 4.0 (EATI'2020) Emerging Applications and Technologies for Industry 4.0*, pages 39–52. Springer, 2021.
- [117] Syed Rashid Aziz, Tamim Khan, and Aamer Nadeem. Experimental validation of inheritance metrics’ impact on software fault prediction. *IEEE Access*, 7:85262–85275, 2019.
- [118] Haonan Tong, Wei Lu, Weiwei Xing, Bin Liu, and Shihai Wang. Shse: A subspace hybrid sampling ensemble method for software defect number prediction. *Information and Software Technology*, 142:106747, 2022.
- [119] Nitin, Kuldeep Kumar, and Santosh Singh Rathore. Analyzing ensemble methods for software fault prediction. In *Advances in Communication and Computational Technology: Select Proceedings of ICACCT 2019*, pages 1253–1267. Springer, 2021.

-
- [120] TK Tung and ML Hanh. Ensemble learning for software fault prediction problem with imbalanced data [j]. *International Journal of Electrical and Computer Engineering (IJECE)*, 9(4), 2019.
- [121] Shuo Feng, Jacky Keung, Xiao Yu, Yan Xiao, Kwabena Ebo Bennin, Md Alamgir Kabir, and Miao Zhang. Coste: Complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction. *Information and Software Technology*, 129:106432, 2021.
- [122] Somya Goyal. Handling class-imbalance with knn (neighbourhood) under-sampling for software defect prediction. *Artificial Intelligence Review*, 55(3):2023–2064, 2022.
- [123] Ha Thi Minh Phuong, Pham Vu Thu Nguyet, Nguyen Huu Nhat Minh, Le Thi My Hanh, and Nguyen Thanh Binh. A comparative study of handling imbalanced data using generative adversarial networks for machine learning based software fault prediction. *Applied Intelligence*, 55(4):1–34, 2025.
- [124] Haibo He, Yang Bai, Eduardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks*, pages 1322–1328. IEEE, 2008.
- [125] Kai Ming Ting. An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):659–665, 2002.
- [126] Shamsul Huda, Kevin Liu, Mohamed Abdelrazek, Amani Ibrahim, Sultan Alyahya, Hmood Al-Dossari, and Shafiq Ahmad. An ensemble oversampling model for class imbalance problem in software defect prediction. *IEEE access*, 6:24184–24195, 2018.
- [127] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2011.
- [128] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.
- [129] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Advances in neural information processing systems. *Curran Associates, Inc*, 27:2672–2680, 2014.

-
- [130] Lillian J Ratliff, Samuel A Burden, and S Shankar Sastry. Characterization and computation of local nash equilibria in continuous games. In *2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 917–924. IEEE, 2013.
- [131] Yabing Zhu, Yanfeng Zhang, Huili Yang, and Fangjing Wang. Gancoder: an automatic natural language-to-programming language translation approach based on gan. In *Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China*, pages 529–539. Springer, 2019.
- [132] Yuanyuan Sun, Lele Xu, Lili Guo, Ye Li, and Yongming Wang. A comparison study of vae and gan for software fault prediction. In *Algorithms and Architectures for Parallel Processing: 19th International Conference, ICA3PP 2019, Melbourne, VIC, Australia, December 9–11, 2019, Proceedings, Part II 19*, pages 82–96. Springer, 2020.
- [133] Ying Xing, Xiaomeng Qian, Yu Guan, Bin Yang, and Yuwei Zhang. Cross-project defect prediction based on g-lstm model. *Pattern Recognition Letters*, 160:50–57, 2022.
- [134] Wei Song, Lu Gan, and Tie Bao. Software defect prediction via generative adversarial networks and pre-trained model. *International Journal of Advanced Computer Science & Applications*, 15(3), 2024.
- [135] Ziye Zhu, Hanghang Tong, Yu Wang, and Yun Li. Bl-gan: Semi-supervised bug localization via generative adversarial network. *IEEE Transactions on Knowledge and Data Engineering*, 35(11):11112–11125, 2023.
- [136] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.
- [137] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. 10 2017.
- [138] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 5769–5779, 2017.
- [139] Karthika. S and M. Durgadevi. Generative adversarial network (gan): a general review on different variants of gan and applications. In *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, pages 1–8, 2021.
- [140] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. *Modeling Tabular Data Using Conditional GAN*. Curran Associates Inc., 2019.

-
- [141] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.
- [142] Jasbir Singh Arora. Introduction to optimum design (fourth edition). Academic Press, Boston, fourth edition edition, 2017.
- [143] Chen-Yi Lin. A reversible data transform algorithm using integer transform for privacy-preserving data mining. *J. Syst. Softw.*, 117(C):104–112, jul 2016.
- [144] Sankha Subhra Mullick, Shounak Datta, and Swagatam Das. Generative adversarial minority oversampling. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1695–1704, 2019.
- [145] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.
- [146] Jelber Sayyad Shirabad and Tim Menzies. The promise repository of software engineering databases. 2005.
- [147] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [148] Mustafa Abed, Monzur Alam Imteaz, Ali Najah Ahmed, and Yuk Feng Huang. A novel application of transformer neural network (tnn) for estimating pan evaporation rate. *Applied Water Science*, 13(2):31, 2023.
- [149] Ira D Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant’Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pages 368–377. IEEE, 1998.
- [150] Abram Hindle, Earl T Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu. On the naturalness of software. *Communications of the ACM*, 59(5):122–131, 2016.
- [151] Song Wang, Taiyue Liu, and Lin Tan. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering*, pages 297–308, 2016.
- [152] Jian Li, Pinjia He, Jieming Zhu, and Michael R Lyu. Software defect prediction via convolutional neural network. In *2017 IEEE international conference on software quality, reliability and security (QRS)*, pages 318–328. IEEE, 2017.

-
- [153] Hongliang Liang, Yue Yu, Lin Jiang, and Zhuosi Xie. Seml: A semantic lstm model for software defect prediction. *IEEE Access*, 7:83812–83824, 2019.
- [154] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.
- [155] Jinfu Chen, Jiaping Xu, Saihua Cai, Xiaoli Wang, Haibo Chen, and Zhehao Li. Software defect prediction approach based on a diversity ensemble combined with neural network. *IEEE Transactions on Reliability*, 2024.
- [156] Anamaria Briciu, Gabriela Czibula, and Mihaela Lupea. A study on the relevance of semantic features extracted using bert-based language models for enhancing the performance of software defect classifiers. *Procedia Computer Science*, 225:1601–1610, 2023.
- [157] Sadia Sahar, Muhammad Younas, Muhammad Murad Khan, and Muhammad Umer Sarwar. Dp-ccl: A supervised contrastive learning approach using codebert model in software defect prediction. *IEEE Access*, pages 22582 – 22594, 2024.
- [158] Zixu Wang, Weiyuan Tong, Peng Li, Guixin Ye, Hao Chen, Xiaoqing Gong, and Zhanyong Tang. Bugpre: an intelligent software version-to-version bug prediction system using graph convolutional neural networks. *Complex & Intelligent Systems*, 9(4):3835–3855, 2023.
- [159] Jingyu Liu, Jun Ai, Minyan Lu, Jie Wang, and Haoxiang Shi. Semantic feature learning for software defect prediction from source code and external knowledge. *Journal of Systems and Software*, 204:111753, 2023.
- [160] Haiyang Liu, Zhiqiang Li, Hongyu Zhang, Xiao-Yuan Jing, and Jinhui Liu. Cfg2at: Control flow graph and graph attention network-based software defect prediction. *IEEE Transactions on Reliability*, pages 1–15, 2024.
- [161] Ilias Siachos, Nikos Kanakaris, and Nikos Karacapilidis. Software bug prediction using graph neural networks and graph-based text representations. *Expert Systems with Applications*, 259:125290, 2025.
- [162] Ziyi Cai, Lu Lu, and Shaojian Qiu. An abstract syntax tree encoding method for cross-project defect prediction. *IEEE Access*, 7:170844–170853, 2019.
- [163] Chris Thunes. javalang: pure python java parser and tools, 2020.

-
- [164] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.
- [165] Z Li. A beginner’s guide to word embedding with gensim word2vec model. *Towards Data Science*. <https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92> (accessed Jun. 27, 2022), 2019.
- [166] Xuan Huo, Ming Li, Zhi-Hua Zhou, et al. Learning unified features from natural and programming languages for locating buggy source code. In *IJCAI*, volume 16, pages 1606–1612, 2016.
- [167] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [168] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.
- [169] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.
- [170] 2022. <https://radimrehurek.com/gensim/models/fasttext.html>.
- [171] Daniel Vasić and Emil Brajković. Development and evaluation of word embeddings for morphologically rich languages. In *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–5, 2018.
- [172] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.
- [173] Abhinav Kumar, Sunil Saumya, and Jyoti Prakash Singh. Nitp-ai-nlp@ hasoc-dravidian-codemix-fire2020: A machine learning approach to identify offensive languages from dravidian code-mixed text. In *FIRE (Working notes)*, pages 384–390, 2020.

-
- [174] Tengjun Yao, Zhengang Zhai, and Bingtao Gao. Text classification model based on fasttext. In *2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIS)*, pages 154–157. IEEE, 2020.
- [175] Yifan Zhou. A review of text classification based on deep learning. In *Proceedings of the 2020 3rd international conference on geoinformatics and data analysis*, pages 132–136, 2020.
- [176] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [177] Apache. The apache software foundation. <https://github.com/apache>.
- [178] Wei Zheng, Lijuan Tan, and Chengbin Liu. Software defect prediction method based on transformer model. In *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pages 670–674. IEEE, 2021.
- [179] Jinyin Chen, Keke Hu, Yue Yu, Zhuangzhi Chen, Qi Xuan, Yi Liu, and Vladimir Filkov. Software visualization and deep transfer learning for effective software defect prediction. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20*, page 578–589, New York, NY, USA, 2020. Association for Computing Machinery.
- [180] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. Transfer defect learning. In *2013 35th international conference on software engineering (ICSE)*, pages 382–391. IEEE, 2013.
- [181] Marian Jureczko and Lech Madeyski. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE '10*, New York, NY, USA, 2010. Association for Computing Machinery.
- [182] Görkem Giray, Kwabena Ebo Bennin, Ömer Köksal, Önder Babur, and Bedir Tekinerdogan. On the use of deep learning in software defect prediction. *Journal of Systems and Software*, 195:111537, 2023.
- [183] Deyu Chen, Xiang Chen, Hao Li, Junfeng Xie, and Yanzhou Mu. Deepcpdp: Deep learning based cross-project defect prediction. *IEEE Access*, 7:184832–184848, 2019.
- [184] Jian Li, Pinjia He, Jieming Zhu, and Michael R. Lyu. Software defect prediction via convolutional neural network. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 318–328, 2017.

-
- [185] Qinchen Liu, Jianwen Xiang, Bin Xu, Dongdong Zhao, Wenhua Hu, and Jian Wang. Aging-related bugs prediction via convolutional neural network. In *2020 7th International Conference on Dependable Systems and Their Applications (DSA)*, pages 90–98, 2020.
- [186] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.
- [187] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [188] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [189] Junyi Gao, Rakshith Sharma, Cheng Qian, Lucas M Glass, Jeffrey Spaeder, Justin Romberg, Jimeng Sun, and Cao Xiao. Stan: spatio-temporal attention network for pandemic prediction using real-world evidence. *Journal of the American Medical Informatics Association*, 28(4):733–743, 2021.
- [190] Wenjie Zi, Wei Xiong, Hao Chen, and Luo Chen. Tagcn: Station-level demand prediction for bike-sharing system via a temporal attention graph convolution network. *Information Sciences*, 561:274–285, 2021.
- [191] Ying Xing, Xiaomeng Qian, Yu Guan, Bin Yang, and Yuwei Zhang. Cross-project defect prediction based on g-lstm model. *Pattern Recognition Letters*, 160:50–57, 2022.
- [192] M Chitra Devi and T Dhiliphan Rajkumar. A novel attention based deep learning model for software defect prediction with bidirectional word embedding system. *Soft Computing*, 29(4):2171–2188, 2025.
- [193] Amjan Shaik, B Aruna Devi, R Baskaran, Satish Bojjawar, P Vidyullatha, and Prasanalakshmi Balaji. Recurrent neural network with emperor penguin-based salp swarm (rnn-eps2) algorithm for emoji based sentiment analysis. *Multimedia Tools and Applications*, 83(12):35097–35116, 2024.
- [194] UB Mahadevaswamy and P Swathi. Sentiment analysis using bidirectional lstm network. *Procedia Computer Science*, 218:45–56, 2023.
- [195] Kamesh Korangi, Christophe Mues, and Cristián Bravo. A transformer-based model for default prediction in mid-cap corporate markets. *European Journal of Operational Research*, 308(1):306–320, 2023.

-
- [196] Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The annals of mathematical statistics*, 11(1):86–92, 1940.
- [197] Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, 114(3):494, 1993.
- [198] Guisheng Fan, Xuyang Diao, Huiqun Yu, Kang Yang, and Liqiong Chen. Software defect prediction via attention-based recurrent neural network. *Scientific Programming*, 2019(1):6230953, 2019.