

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



ĐỒ ÁN
TỐT NGHIỆP ĐẠI HỌC

**Đề tài : “ PHƯƠNG PHÁP TÍCH HỢP LỰA
CHỌN ĐẶC TRƯNG VÀ HỌC MÁY HIỆN ĐẠI
CHO PHÁT HIỆN URL GIẢ MẠO”**

Người hướng dẫn : THS. ĐÀM MINH LINH

Sinh viên thực hiện : BÙI HỮU TRÍ - N21DCAT058

ĐINH QUỐC TOÀN - N21DCAT057

Lớp : D21CQAT01-N

Hệ : Đại học chính quy

TP. HỒ CHÍ MINH, NĂM 2025

BÙI HỮU TRÍ
ĐINH QUỐC TOÀN

MSSV: N21DCAT058
MSSV: N21DCAT057

CHUYÊN NGÀNH: AN TOÀN THÔNG TIN
CHUYÊN NGÀNH: AN TOÀN THÔNG TIN

KHOÁ: 2021-2026
KHOÁ: 2021-2026

LỚP: D21CQAT01-N
LỚP: D21CQAT01-N

TP.
HCM
2025

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



ĐỒ ÁN

TỐT NGHIỆP ĐẠI HỌC

Đề tài : “ PHƯƠNG PHÁP TÍCH HỢP LỰA
CHỌN ĐẶC TRƯNG VÀ HỌC MÁY HIỆN ĐẠI
CHO PHÁT HIỆN URL GIẢ MẠO”

Người hướng dẫn : THS. ĐÀM MINH LINH

Sinh viên thực hiện : BÙI HỮU TRÍ - N21DCAT058

ĐINH QUỐC TOÀN - N21DCAT057

Lớp : D21CQAT01-N

Hệ : Đại học chính quy

TP. HỒ CHÍ MINH, NĂM 2025

LỜI CẢM ƠN

Đồ án tốt nghiệp là một cột mốc quan trọng đánh dấu quá trình học tập và rèn luyện của mỗi sinh viên, giúp chúng em vận dụng kiến thức lý thuyết vào thực tiễn, rèn luyện tư duy nghiên cứu, kỹ năng làm việc nhóm và tác phong chuyên nghiệp.

Trong suốt quá trình thực hiện, nhóm xin chân thành cảm ơn Ban Giám hiệu Học viện Công nghệ Bưu chính Viễn thông cùng quý thầy cô Khoa Công nghệ Thông tin 2 đã tận tâm giảng dạy, truyền đạt cho chúng em những kiến thức và định hướng quý báu.

Đặc biệt, nhóm xin gửi lời tri ân sâu sắc đến thầy Thạc sĩ Đàm Minh Linh, giảng viên hướng dẫn, người đã tận tình định hướng, góp ý chuyên môn, hỗ trợ chúng em trong suốt quá trình thực hiện đồ án tốt nghiệp và thầy luôn bổ sung kiến thức cần thiết và giải đáp kịp thời những vướng mắc. Thầy giúp nhóm tiếp cận các phương pháp hiện đại, từ trích xuất đặc trưng đến huấn luyện mô hình AI. Nhờ vậy, chúng em hiểu sâu hơn về chuyên ngành an toàn mạng và cách kết hợp trí tuệ nhân tạo để giải quyết bài toán thực tiễn.

Mặc dù đã nỗ lực hoàn thiện, nhưng đồ án khó tránh khỏi những thiếu sót. Nhóm rất mong nhận được những góp ý quý báu từ quý thầy cô để tiếp tục hoàn thiện và nâng cao năng lực chuyên môn trong tương lai.

Cuối cùng, nhóm xin chân thành cảm ơn thầy hướng dẫn vì sự tận tâm và nhiệt huyết. Kính chúc thầy luôn mạnh khỏe, nhiều niềm vui và thành công trong công tác giảng dạy cũng như nghiên cứu. Nhóm hy vọng sẽ tiếp tục nhận được sự đồng hành và chỉ dẫn của thầy trong những chặng đường học tập và làm việc tiếp theo.

TP.HCM, tháng 12 năm 2025

MỤC LỤC

MỞ ĐẦU	24
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT	3
 1.1 Tổng quan về URL độc hại.....	3
1.1.1 Khái niệm về URL	3
1.1.2 Các thành phần của URL	4
1.1.3 Một số kỹ thuật tấn công dựa trên URL	5
1.1.4 Dấu hiệu và hành vi của URL Phishing	8
1.1.5 Bối cảnh nghiên cứu	9
1.1.6 Bài toán đặt ra	10
 1.2 Tổng quan về những đặc trưng phát hiện url giả mạo	11
1.2.1 Nhóm đặc trưng ký tự	11
1.2.2 Nhóm đặc trưng dựa trên máy chủ	11
1.2.3 Nhóm đặc trưng ngữ nghĩa và văn bản	12
 1.3 Tìm hiểu và đề xuất các mô hình	13
1.3.1 Các mô hình học máy	13
1.3.2 Các mô hình học sâu	19
1.3.3 Các mô hình học máy hiện đại	30

1.4 Nhữn^g tiêu chí, thu^{ật} toán cho tiền xử lý bộ dữ liệu	36
1.4.1 Làm sạch và chuẩn hóa dữ liệu	37
1.4.2 Cân bằng dữ liệu và chia tập huấn luyện	37
1.4.3 Chọn lọc đặc trưng và đánh giá chất lượng bộ dữ liệu	38
1.5 Các tiêu chí đánh giá định lượng	39
1.5.1 Precision	39
1.5.2 Recall	40
1.5.3 Accuracy	40
1.5.4 F1-score	41
1.5.5 PR-AUC	41
1.5.6 ROC-AUC	41
1.5.7 Confusion Matrix	42
1.6 Giải thích mô hình với SHAP và LIME	42
1.6.1 SHAP	43
1.6.2 LIME	46
CHƯƠNG 2: ĐỀ XUẤT BỘ ĐẶC TRƯNG VÀ CÁC MÔ HÌNH	47
2.1 Thu thập và chuẩn bị tập dữ liệu URL	47
2.2 Trích xuất đặc trưng và tiền xử lý dữ liệu	50

2.2.1	Trích xuất thêm đặc trưng	50
2.2.2	Loại bỏ dữ thừa dữ liệu, các URL trùng lặp	54
2.2.3	Xử lý mất bằng dữ liệu	54
2.2.4	Tinh gọn đặc trưng bằng ma trận tương quan	57
2.2.5	Huấn luyện Word2Vec và embedding URL	60
2.2.6	Huấn luyện Char2Vec và embedding URL	63
2.3	Cấu hình, huấn luyện và đánh giá các mô hình	65
2.3.1	Machine learning	65
2.3.2	Deep learning	79
2.3.3	SLM/LLM	121
2.4	Giải thích mô hình với SHAP/LIME	139
2.3.1	XGBoost	139
2.3.2	RandomForest	154
2.3.3	WordCNN	168
2.3.4	CNN-LSTM	188
2.3.5	ALBERT	197
2.5	Kiểm thử dự đoán với các mô hình đã huấn luyện	218
CHƯƠNG 3: KẾT LUẬN VÀ ĐÁNH GIÁ	219	

3.1 Kết quả thực nghiệm	220
3.2 Thảo luận	220
KẾT LUẬN VÀ KIẾN NGHỊ	223
4.1 Kết luận	223
4.2 Hướng phát triển	223
TÀI LIỆU THAM KHẢO	224

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

Từ viết tắt	Ý nghĩa
AI: Artificial Intelligence	Trí tuệ nhân tạo
ALBERT: A Lite BERT	Phiên bản nhẹ của BERT
AUC: Area Under the Curve	Diện tích dưới đường cong
BERT: Bidirectional Encoder Representations from Transformers	Mô hình Transformer hai chiều tiền huấn luyện
CM: Confusion Matrix	Ma trận nhầm lẫn
CNN: Convolutional Neural Network	Mạng nơ-ron tích chập
CSV: Comma-Separated Values	Dữ liệu phân cách bằng dấu phẩy
DL: Deep Learning	Học sâu
DNS: Domain Name System	Hệ thống phân giải tên miền
DOM: Document Object Model	Mô hình đối tượng tài liệu
F1-Score	Chỉ số cân bằng giữa Precision và Recall
FN: False Negative	Dự đoán sai âm tính - Bỏ sót
FP: False Positive	Dự đoán sai dương tính - Báo động giả
GBM: Gradient Boosting Machine	Máy học tăng cường độ dốc
HTML: HyperText Markup Language	Ngôn ngữ đánh dấu siêu văn bản

HTTP: Hypertext Transfer Protocol	Giao thức truyền tải siêu văn bản
HTTPS: Hypertext Transfer Protocol Secure	Giao thức truyền tải siêu văn bản bảo mật
IDN: Internationalized Domain Name	Tên miền quốc tế
IETF: Internet Engineering Task Force	Lực lượng chuyên trách kỹ thuật Internet
IP: Internet Protocol	Giao thức Internet
LIME: Local Interpretable Model-agnostic Explanations	Giải thích cục bộ không phụ thuộc mô hình
LLM: Large Language Models	Mô hình ngôn ngữ lớn
LoRA: Low-Rank Adaptation	Thích ứng hạng thấp - Kỹ thuật tinh chỉnh mô hình
LR: Logistic Regression	Hồi quy Logistic
LSTM: Long Short-Term Memory	Bộ nhớ ngắn-dài hạn
ML: Machine Learning	Học máy
NLP: Natural Language Processing	Xử lý ngôn ngữ tự nhiên
NSP: Next Sentence Prediction	Dự đoán câu tiếp theo
PEFT: Parameter-Efficient Fine-Tuning	Tinh chỉnh tham số hiệu quả
PR-AUC: Precision-Recall Area Under Curve	Diện tích dưới đường cong Precision-Recall
QLoRA: Quantized Low-Rank Adaptation	LoRA lượng tử hóa

ReLU: Rectified Linear Unit	Hàm kích hoạt chỉnh lưu tuyến tính
RF: Random Forest	Rừng ngẫu nhiên
RNN: Recurrent Neural Network	Mạng nơ-ron hồi quy
ROC: Receiver Operating Characteristic	Đường cong đặc tính hoạt động của bộ thu
SHAP: SHapley Additive Explanations	Giải thích dựa trên giá trị Shapley
SMOTE: Synthetic Minority Over-sampling Technique	Kỹ thuật sinh mẫu thiểu số nhân tạo
SLM: Small Language Model	Mô hình Ngôn ngữ Nhỏ
SOP: Sentence Order Prediction	Dự đoán trật tự câu
SSL: Secure Sockets Layer	Lớp công bảo mật
SVM: Support Vector Machine	Mô hình máy vector hỗ trợ
TF-IDF: Term Frequency - Inverse Document Frequency	Trọng số tần suất – nghịch tài liệu
TLD: Top-Level Domain	Tên miền cấp cao nhất
TLS: Transport Layer Security	Bảo mật tầng giao vận
TN: True Negative	Dự đoán đúng âm tính
TP: True Positive	Dự đoán đúng dương tính
URL: Uniform Resource Locator	Định vị tài nguyên thống nhất
UX: User Experience	Trải nghiệm người dùng

WHOIS: WHOIS Lookup Service	Dịch vụ tra cứu thông tin đăng ký tên miền
XGBoost: Extreme Gradient Boosting	Thuật toán tăng cường độ dốc cực trị

DANH MỤC CÁC BẢNG VỀ

Bảng 2.1 Nhóm đặc trưng về cấu trúc URL được thêm	51
Bảng 2.2 Nhóm đặc trưng về máy chủ URL được thêm	52
Bảng 2.3 Nhóm Brand/Typosquatting	52
Bảng 2.4 Nhóm đặc trưng Content/DOM đã thêm	53
Bảng 2.5 Nhóm đặc trưng máy chủ mở rộng	54

DANH MỤC CÁC HÌNH VẼ

Hình 1.1 Mô tả các thành phần cơ bản của một cấu trúc URL	4
Hình 1.2 Minh họa kỹ thuật tấn công Phishing URL	5
Hình 1.3 Minh họa kỹ thuật tấn công Typosquatting	6
Hình 1.4 Minh họa kỹ thuật Homograph sử dụng Punycode để đánh lừa người dùng	6
Hình 1.5 Minh họa kỹ thuật Open redirect abuse	7
Hình 1.6 Minh họa kỹ thuật URL shorting abuse	7
Hình 1.7 Minh họa kỹ thuật tấn công Obfuscation	8
Hình 1.8 Minh họa kỹ thuật Malware distribution URL	8
Hình 1.9 Minh họa thông tin Whois của một website đáng ngờ	9
Hình 1.10 Minh họa embedding chuyển đổi token trong URL thành vector số học	12
Hình 1.11 Mô tả nguyên lý hoạt động của mô hình XGBoost	14
Hình 1.12 Mô tả nguyên lý hoạt động mô hình Random Forest	17
Hình 1.13 Pipeline char-embedding character-level sequence	20
Hình 1.14 Pipeline char-embedding URL-token sequence	20
Hình 1.15 Cơ chế hoạt động của mô hình CNN	21
Hình 1.16 Mô tả minh họa kiến trúc 1D-CNN cho phân loại URL Phishing	22

Hình 1.17 Kiến trúc hệ thống của mô hình CNN-LSTM	24
Hình 1.19 Kiến trúc hoạt động của mô hình CharCNN	26
Hình 1.20 Kiến trúc mô hình CNN-LSTM	27
Hình 1.21 Kiến trúc mô hình CNN-Hybrid	29
Hình 1.22 So sánh sự khác nhau giữa kiến trúc mô hình BERT truyền thống và MobileBERT	32
Hình 1.23 So sánh kỹ thuật tối ưu giữa BERT truyền thống và ALBERT	33
Hình 1.24 Mô tả quá trình xử lý dữ liệu đầu vào mô hình TinyLlama	36
 <hr/>	
Hình 2.1 Thống kê số lượng nhãn Phishing/Legitimate	48
Hình 2.2 Tổng quan một số đặc trưng dataset	48
Hình 2.3 Tỷ lệ phân bố nhãn của dataset	49
Hình 2.4 Tỷ lệ phân bố nhãn trong dataset cân bằng	57
Hình 2.5 Một số URL không gắn được nhãn	57
Hình 2.6 Số lượng nhãn Phishing và Legitimate	57
Hình 2.7 Ma trận tương quan của dataset trước khi tinh gọn đặc trưng có tương quan cao	58
Hình 2.8 Một số đặc trưng có tương quan cao	59

Hình 2.9 Ma trận tương quan sau khi tinh gọn đặc trưng	60
Hình 2.10 Accuracy Graph của model XGBoost numerical	67
Hình 2.11 Loss graph của model XGBoost numerical	67
Hình 2.12 PR-AUC graph của model XGBoost numerical	68
Hình 2.13 Confusion matrix của model XGBoost numerical	68
Hình 2.14 Đánh giá mô hình XGBoost numerical	69
Hình 2.15 Accuracy graph của model XGBoost embedding	70
Hình 2.16: Loss graph của model XGBoost embedding	70
Hình 2.17 PR-AUC của model XGBoost embedding	71
Hình 2.18 Confusion matrix của model XGBoost embedding	71
Hình 2.19 Đánh giá model XGBoost embedding	72
Hình 2.20 Confusion matrix của model Random Forest numerical	74
Hình 2.21 Đánh giá models Random Forest numerical	74
Hình 2.22 ROC-Curve của model Random Forest numerical	75
Hình 2.23 Precision/Recall Curve của model Random Forest numerical	75
Hình 2.24 Top 20 đặc trưng quan trọng trong huấn luyện Random Forest model	76
Hình 2.25 Confusion matrix của model Random Forest embedding	77

Hình 2.26 Đánh giá model RF với nhiều tiêu chí khác nhau	77
Hình 2.27 ROC Curve của model Random Forest embedding	78
Hình 2.28 Precisio/Recal Curve của model Random Forest embedding	78
Hình 2.29 Kiến trúc model WordCNN numerical	81
Hình 2.30 Kiến trúc model WordCNN embedding	82
Hình 2.31 Accuracy graph của model WordCNN numerical	85
Hình 2.32 Loss graph của model WordCNN numerical	85
Hình 2.33 Confusion matrix của model WordCNN numerical	86
Hình 2.34 Precision/Recall Curve của model WordCNN numerical	86
Hình 2.35 ROC Curve của model WordCNN numerical	87
Hình 2.36 Accuracy graph của model WordCNN embedding	88
Hình 2.37 Loss graph của model WordCNN embedding	88
Hình 2.38 Confusion matrix của model WordCNN embedding	89
Hình 2.39 Precisio/Recall Curve của model WordCNN embedding	89
Hình 2.40 ROC Curve của model WordCNN embedding	90
Hình 2.41 Kiến trúc mô hình của CNN-LSTM numerical	93
Hình 2.42 Kiến trúc mô hình CNN-LSTM embedding	94

Hình 2.43 Accuracy graph của model CNN-LSTM numerical	95
Hình 2.44 Loss graph của model CNN-LSTM numerical	96
Hình 2.45 Confusion matrix cẩu model CNN-LSTM numerical	96
Hình 2.46 Precision/Recall Curve CNN-LSTM numerical	97
Hình 2.47 ROC Curve của model CNN-LSTM numerical	97
Hình 2.48 Accuracy graph của model CNN-LSTM embedding	98
Hình 2.49 Loss graph của model CNN-LSTM embedding	99
Hình 2.50 Confusion matrix của model CNN-LSTM embedding	99
Hình 2.51 Precision/Recall curve của model CNN-LSTM embedding	100
Hình 2.52 ROC Curve của model CNN-LSTM embedding	100
Hình 2.53 Kiến trúc mô hình CharCNN	102
Hình 2.54 Accuracy graph của model CharCNN	104
Hình 2.55 Loss graph của model CharCNN	105
Hình 2.56 Confusion matrix cẩu model CharCNN	105
Hình 2.57 Precision/Recall Curve CharCNN	106
Hình 2.58 ROC Curve của model CharCNN	106
Hình 2.59 Kiến trúc mô hình CharCNN-LSTM	108

Hình 2.60 Accuracy graph của model CharCNN-LSTM	110
Hình 2.61 Loss graph của model CharCNN-LSTM	111
Hình 2.62 Confusion matrix cẩu model CharCNN-LSTM	111
Hình 2.63 Precision/Recall Curve CharCNN-LSTM	112
Hình 2.64 ROC Curve của model CharCNN-LSTM	112
Hình 2.65 Đánh giá tổng quan model CNN-Hybrid được huấn luyện	117
Hình 2.66 Accuracy graph của model CNN-Hybrid	117
Hình 2.67 Loss graph của model CNN-Hybrid	118
Hình 2.68 Confusion matrix cẩu model CNN-Hybrid	118
Hình 2.69 Precision Curve CNN-Hybrid	119
Hình 2.70 Recall Curve của model CNN-Hybrid	119
Hình 2.71 Prediction distribution của model CNN-Hybrid	120
Hình 2.72 Đánh giá tổng quan model CNN-Hybrid với nhiều chỉ số khác nhau	120
Hình 2.73 Đánh giá bộ test mô hình ALBERT	125
Hình 2.74 Training/validation Loss graph của model ALBERT	126
Hình 2.75 Validation F1-score/epoch của model ALBERT	126
Hình 2.76 Confusion matrix cia model ALBERT	127

Hình 2.77 ROC Curve của model ALBERT	127
Hình 2.78 Precision/Recall Curve của model ALBERT	128
Hình 2.79 Đánh giá model MobileBERT với nhiều tiêu chí đánh giá khác nhau	129
Hình 2.80 Training/Validation Loss graph của model MobileBERT	130
Hình 2.81 Validation F1 over Epoch model MobileBERT	130
Hình 2.82 Confusion matrix của model MobileBERT	131
Hình 2.83 ROC Curve cẩu model MobileBERT	131
Hình 2.84 Precision/Recall Curve của model MobileBERT	132
Hình 2.85 Đánh giá model TinyLlama với nhiều tiêu chí đánh giá khác nhau	136
Hình 2.86 Training/Validation Loss graph của model TinyLlama	137
Hình 2.87 Confusion matrix của model TinyLlama	137
Hình 2.88 ROC Curve cẩu model TinyLlama	138
Hình 2.89 Precision/Recall Curve của model TinyLlama	138
Hình 2.90 Tính SHAP value những đặc trưng tác động đến model XGBoost	140
Hình 2.91 Vẽ shap.force_plot shap_values_xgb[57,:] XGB model	141
Hình 2.92 Vẽ shap.plots.waterfall XGB model	141
Hình 2.93 Vẽ shap.force_plot shap_values_xgb[38,:] XGBoost model	142

Hình 2.94 Vẽ shap.plots.waterfall XGBoost model	142
Hình 2.95 Vẽ shap.dependence_plot đặc trưng LineOfCode XGBoost model	143
Hình 2.96 Vẽ shap.dependence_plot đặc trưng NoOfExternalRef XGBoost model	143
Hình 2.97 Vẽ shap.dependence_plot đặc trưng NoOfSelfRef XGBoost model	144
Hình 2.98 Vẽ shap.dependence_plot IsHTTPS XGBoost model	144
Hình 2.99 Vẽ shap.dependence_plot đặc trưng nb_slash XGBoost model	145
Hình 2.100 Vẽ shap.dependence_plot đặc trưng nb_www x LineOfCode XGBoost model	145
Hình 2.101 shap.dependence_plot đặc trưng nb_www x NoOfExternalRef XGBoost model	146
Hình 2.102 Vẽ shap.dependence_plot đặc trưng HasSocialNet x NoOfExternalRef XGBoost model	146
Hình 2.103 Vẽ shap.dependence_plot IsHTTPS x nb_www XGBoost model	147
Hình 2.104 Vẽ shap.dependence_plot đặc trưng IsHTTPS x nb_slash XGBoost model	147
Hình 2.105 Tính shap value XGBoost embedding model	149
Hình 2.106 Vẽ shap.force_splot XGBoost embedding	150
Hình 2.107 Vẽ shap.plots.waterfall XGB embedding mode	150
Hình 2.108 Vẽ shap.force_plot XGBoost embedding	151

Hình 2.109 Vẽ shap.plots.waterfall XGB embedding model	151
Hình 2.110 Vẽ shap.dependence_plot Feature 40 XGB embedding model	152
Hình 2.111 Vẽ shap.dependence_plot Feature 9 XGB embedding model	152
Hình 2.112 Vẽ shap.dependence_plot Feature 46 XGBoost embedding model	153
Hình 2.113 Vẽ shap.dependence_plot Feature 6 XGB embedding model	153
Hình 2.114 Vẽ shap.dependence_plot Feature 18 XGBoost embedding model	154
Hình 2.115 Vẽ shap.force_plot RF model	155
Hình 2.116 Vẽ shap.plots.waterfall Random Foreset numerical model	156
Hình 2.117 Vẽ shap.dependence_plot shortest_word_raw_path_len RF model	157
Hình 2.118 Vẽ shap.dependence_plot NoOfImage RF model	157
Hình 2.119 shap.dependence_plot nb_slash RF model	158
Hình 2.120 Vẽ shap.force_plot RF model	158
Hình 2.121 Vẽ shap.plots.waterfall RandomForest model	159
Hình 2.122 Vẽ shap.dependence_plot NoOfImage RandomForest model	160
Hình 2.123 Vẽ shap.dependence_plot NoOfExternalRef	160
Hình 2.124 Vẽ shap.dependence_plot HasPasswordField RF model	161
Hình 2.125 Vẽ shap.summary_plot RF embedding model	162

Hình 2.126 Vẽ shap.force_plot RandomForest embedding model	163
Hình 2.127 Vẽ shap.plots.waterfall RandomForest embedding model	163
Hình 2.128 Vẽ shap_values_rf_class1 Feature 45 RF embedding model	164
Hình 2.129 Vẽ shap.dependence_plot Feature 25 RF embedding model	164
Hình 2.130 Vẽ shap.force_plot RandomForest embedding model	165
Hình 2.131 Vẽ shap.plots.waterfall RandomForest embedding model	165
Hình 2.132 Vẽ shap.dependence_plot Feature 19 RandomForest embedding model	166
Hình 2.133 Vẽ shap.dependence_plot Feature 8 RandomForest embedding model	166
Hình 2.134 Vẽ shap.dependence_plot Feature 19 RandomForest embedding model	167
Hình 2.135 Vẽ shap.dependence_plot Feature 34 RF embedding model	167
Hình 2.136 SHAP value impact output of WordCNN model	171
Hình 2.137 Shap.force_plot WordCNN model	172
Hình 2.138 shap.plots.waterfall WordCNN model	172
Hình 2.139 Shap.dependence_plot Feature NoOfimage WordCNN model	173
Hình 2.140 Shap.dependence_plot Feature nb_slash WordCNN model	173
Hình 2.141 Shap.dependence_plot Feature NoOfSelfRedirect WordCNN model	174
Hình 2.142 Shap.dependence_plot IsHTTPS WordCNN model	174

Hình 2.143 Shap.dependence_plot avg_word_raw_len WordCNN model	175
Hình 2.144 Shap.force_plot WordCNN model	175
Hình 2.145 Shap.plots.waterfall WordCNN model	176
Hình 2.146 Shap.dependence_plot nb_slash WordCNN model	177
Hình 2.147 Shap.dependence_plot NoOfImage WordCNN model	177
Hình 2.148 Shap.dependence_plot isResponsive WordCNN model	178
Hình 2.149 Shap.dependence_plot shortest_word_raw_path_len WordCNN model	178
Hình 2.150 Shap.dependence_plot Feature Bank WordCNN model	179
Hình 2.151 SHAP value impact output of WordCNN embedding model	181
Hình 2.152 Shap.force_plot sample 57 WordCNN embedding model	182
Hình 2.153 Vẽ shap.plots.waterfall WordCNN embedding model	183
Hình 2.154 Shap.force_plot sample 33 WordCNN embedding model	184
Hình 2.155 Vẽ shap.plots.waterfall WordCNN embedding model	185
Hình 2.156 Vẽ shap.dependence_plot Feature 36 WordCNN embedding model	186
Hình 2.157 Vẽ shap.dependence_plot Feature 26 WordCNN embedding model	186
Hình 2.158 Vẽ shap.dependence_plot Feature 49 WordCNN embedding model	187
Hình 2.159 Vẽ shap.dependence_plot Feature 35 WordCNN embedding model	187

Hình 2.160 exp.show_in_notebook CNN-LSTM	189
Hình 2161 exp = explainer.explain_instance CNN-LSTM model	190
Hình 2.162 LIME contribution sample 0 CNN-LSTM model	195
Hình 2.163 LIME contribution sample 1 CNN-LSTM model	196
Hình 2.164 LIME contribution sample 2 CNN-LSTM model	196
Hình 2.165 Vẽ SHAP value ALBERT model	201
Hình 2.166 Vẽ SHAP plot [42:] ALBERT model	202
Hình 2.167 Vẽ shap.plots.waterfall(sample_shap[:, 1]) ALBERT model	203
Hình 2.168 Top 15 tokens impact sample 42 ALBERT model	205
Hình 2.169 Phân tích chi tiết ảnh hưởng của các token đến model ALBERT	206
Hình 2.170 Mức độ ảnh hưởng các token đến ALBERT model	208
Hình 2.171 SHAP value heatmap sample 42 ALBERT model	210
Hình 2.172 Decision Plot - Sample 42 ALBERT model	211
Hình 2.173 Phishing sample SHAP ALBERT model	212
Hình 2.174 Vẽ shap.plots.waterfall(sample_shap[:, 0]) ALBERT model	213
Hình 2.175 Top 15 tokens impact sample 125 ALBERT model	213
Hình 2.176 Phân tích chi tiết ảnh hưởng của các tokens đến model ALBERT	214

Hình 2.177 SHAP value heatmap sample 125 ALBERT model	215
Hình 2.178 Biểu đồ mức độ ảnh hưởng của các tokens đến model ALBERT	216
Hình 2.179 Decision Plot sample 125 Phishing Class ALBERT model	217
Hình 2.180 Extension tích hợp trên trình duyệt web sử dụng những models đã huấn luyện để phát hiện URL phising real-time	218
Hình 2.181 Giao diện demo dự đoán với url legitimate	219
Hình 2.182 Giao diện demo dự đoán với url phishing	219

PHÂN CÔNG CÔNG VIỆC

Thành viên	MSSV	Nội dung lý thuyết	Nội dung thực nghiệm
Bùi Hữu Trí	N21DCAT058	<ul style="list-style-type: none"> - Nghiên cứu tổng quan về URL Phishing và các kỹ thuật tấn công. - Thu thập bộ dữ liệu từ nhiều nguồn - Tìm hiểu các tiêu chuẩn đánh giá mô hình - Khảo sát các phương pháp Lựa chọn đặc trưng và thuật toán ML truyền thống. 	<ul style="list-style-type: none"> - Thu thập, gán nhãn và xử lý mất cân bằng dữ liệu. - Trích xuất đặc trưng thủ công và thực hiện chọn lọc đặc trưng. - Huấn luyện mô hình ML cơ sở. - Xây dựng Extension để demo phát hiện thời gian thực.
Đinh Quốc Toàn	N21DCAT057	<ul style="list-style-type: none"> - Nghiên cứu kiến trúc Deep Learning & LLM. - Tìm hiểu cơ chế Attention và các phương pháp Embedding. - Nghiên cứu lý thuyết SHAP/LIME để giải thích mô hình. 	<ul style="list-style-type: none"> - Thực hiện Tokenization và Vector hóa dữ liệu URL. - Xây dựng và huấn luyện mô hình lai (Hybrid Model): kết hợp Word + Char - Tích hợp SHAP/LIME để phân tích, giải thích kết quả dự đoán.

MỞ ĐẦU

Trong kỷ nguyên số, Internet đã trở thành nền tảng thiết yếu trong mọi lĩnh vực của đời sống và công việc. Tuy nhiên, cùng với sự phát triển mạnh mẽ đó là hàng loạt rủi ro về an ninh mạng. Một trong những mối đe dọa phổ biến và nguy hiểm nhất là các URL độc hại, được kẻ tấn

công sử dụng để lừa đảo, phát tán phần mềm độc hại, tấn công thay đổi giao diện hay phát tán thư rác. Các phương pháp phát hiện truyền thống như blacklist ngày càng kém hiệu quả do kẻ tấn công liên tục thay đổi chiến thuật để qua mặt hệ thống. Do đó, việc xây dựng các giải pháp phát hiện URL phishing tự động, sớm và chính xác là nhu cầu cấp thiết trong bối cảnh hiện nay.

Trong đồ án tốt nghiệp này, nhóm sẽ tiếp cận bài toán phát hiện URL giả mạo dưới dạng phân loại nhị phân, với mục tiêu phân biệt giữa hai loại URL: hợp pháp và giả mạo. Điểm nổi bật của đề tài là sử dụng nhiều phương pháp học máy và học sâu hiện đại, áp dụng một số mô hình ngôn ngữ lớn đồng thời kết hợp giữa lựa chọn, tinh gọn đặc trưng hiệu quả và biểu diễn ngữ nghĩa từ chuỗi URL. Cụ thể, mô hình được chia thành ba nhánh chính:

- Machine Learning: khai thác các đặc trưng số đã được chọn lọc.
- Deep Learning: học biểu diễn trực tiếp từ embedding URL.
- Large Language Models: tận dụng khả năng học ngữ cảnh và đặc trưng chuỗi của Transformer.

Ngoài ra, đồ án còn sử dụng các kỹ thuật chọn lọc đặc trưng như kiểm tra tương quan và lựa chọn những đặc trưng quan trọng, nhằm loại bỏ yếu tố dư thừa và tối ưu hóa dữ liệu huấn luyện. Việc đánh giá mô hình được thực hiện bằng bộ tiêu chí toàn diện gồm Macro-F1, PR-AUC, ROC-AUC, Ma trận nhầm lẫn, Ma trận tương quan. Hơn nữa, nhóm cũng áp dụng SHAP và LIME để giải thích quyết định của mô hình, giúp tăng tính minh bạch và độ tin cậy của hệ thống.

Báo cáo được cấu trúc thành bốn chương chính:

- Chương 1: Cơ sở lý thuyết: Trình bày tổng quan lý thuyết về URL phishing, các loại đặc trưng, mô hình ML/DL/LLM và các tiêu chí đánh giá.
- Chương 2: Đề xuất bộ đặc trưng và các mô hình: Mô tả quy trình thu thập, tiền xử lý, trích xuất và chọn lọc đặc trưng, cùng pipeline huấn luyện.
- Chương 3: Kết luận và đánh giá: Trình bày kết quả huấn luyện và so sánh mô hình.

Mục tiêu cuối cùng của đồ án là xây dựng một hệ thống phát hiện URL giả mạo hiệu quả, giải thích được quyết định mô hình, đồng thời so sánh độ hiệu quả giữa các mô hình học máy, học sâu truyền thống so với các mô hình học máy hiện đại. Đồ án không chỉ mang giá trị ứng

dụng cao trong phát hiện và phòng chống tấn công phishing, mà còn đóng góp vào hướng nghiên cứu trí tuệ nhân tạo ứng dụng trong an toàn thông tin.

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1 Tổng quan về URL độc hại

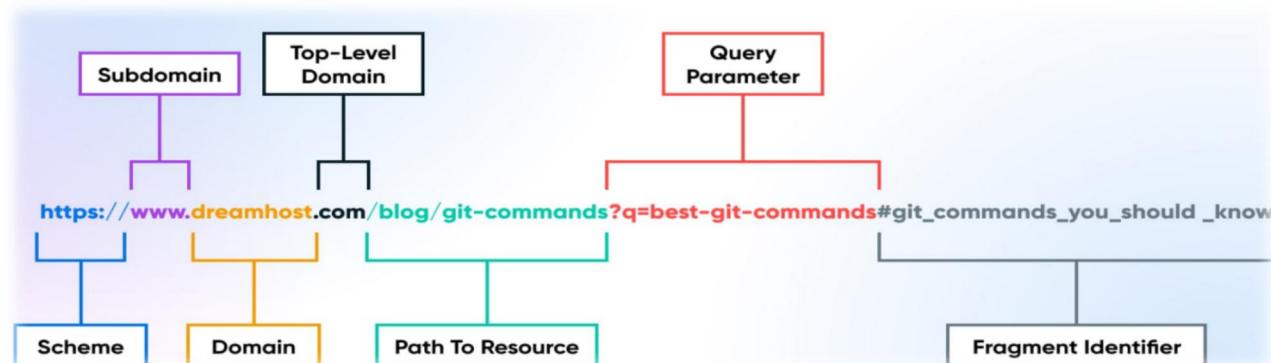
1.1.1 Khái niệm về URL

URL là một địa chỉ dùng để xác định và truy cập tài nguyên trên Internet, chẳng hạn như trang web, hình ảnh, tệp tin hoặc dịch vụ trực tuyến. URL được xem như địa chỉ của tài nguyên trong không gian mạng, giúp trình duyệt hoặc ứng dụng biết nơi cần đến và cách truy cập. Mỗi URL thường bao gồm các thành phần cơ bản như giao thức truy cập, tên miền, đường dẫn, và tham số truy vấn.

Về bản chất, URL là cầu nối giữa người dùng và tài nguyên trên mạng. Khi người dùng nhập URL vào trình duyệt, hệ thống sẽ gửi yêu cầu đến máy chủ chứa tài nguyên tương ứng và hiển thị kết quả. Cấu trúc và định dạng của URL tuân theo tiêu chuẩn do Tổ chức IETF ban hành, giúp đảm bảo khả năng định danh thống nhất và truy cập chính xác đến mọi tài nguyên trên Internet.

1.1.2 Các thành phần của URL

Một URL được cấu tạo từ nhiều phần, mỗi phần giữ vai trò xác định vị trí và cách truy cập đến tài nguyên trên Internet. Cấu trúc cơ bản của một URL có dạng như sau:



Hình 1.1 Mô tả các thành phần cơ bản của một cấu trúc URL

Giải thích các thành phần URL:

- Scheme: Xác định phương thức truy cập tài nguyên, ví dụ http, https, ftp... Trong đó, https được sử dụng phổ biến nhất hiện nay nhờ khả năng mã hóa và bảo mật cao.

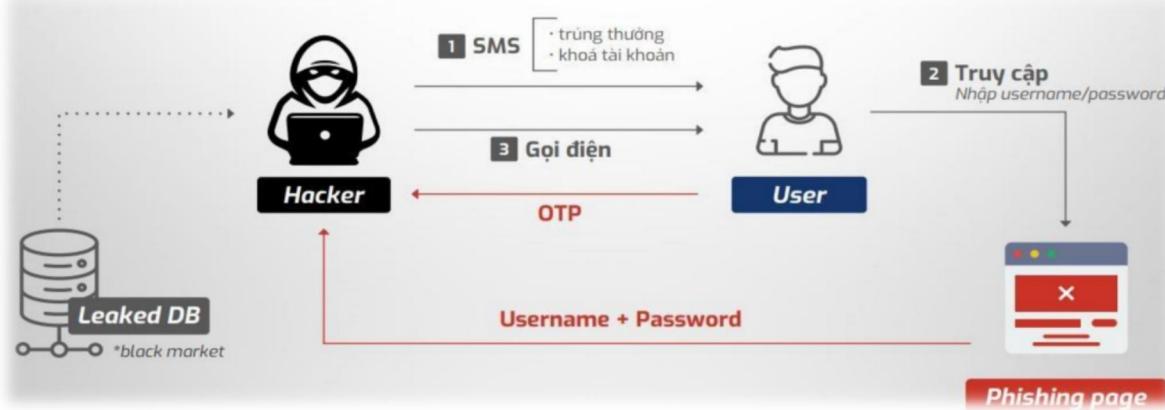
- Domain: Là địa chỉ của máy chủ chứa tài nguyên, có thể bao gồm tên miền chính và các subdomain.
- Port: Chỉ định cổng dịch vụ mà máy chủ sử dụng, thường là 80 cho HTTP và 443 cho HTTPS.
- Path: Xác định vị trí cụ thể của tài nguyên trên máy chủ.
- Query: Chứa các cặp khóa và giá trị truyền dữ liệu động.
- Fragment: Dùng để trỏ đến một phần cụ thể trong tài nguyên, như một vị trí trong trang web.

Nhờ sự kết hợp của các thành phần trên, URL có thể mô tả chính xác cách truy cập đến bất kỳ tài nguyên nào trên Internet một cách thống nhất và linh hoạt.

1.1.3 Một số kỹ thuật tấn công dựa trên URL

Các tấn công dựa trên URL khai thác cấu trúc chuỗi và lòng tin của người dùng để đánh lừa hoặc thực thi mã độc. Một số kiểu tiêu biểu gồm:

- Phishing: Tạo URL hoặc trang web giả mạo giống với dịch vụ hợp pháp để lừa người dùng cung cấp thông tin nhạy cảm như tài khoản, mật khẩu hoặc thông tin thẻ.



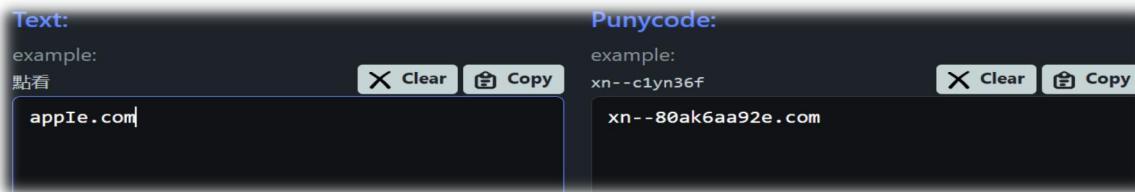
Hình 1.2 Minh họa kỹ thuật tấn công Phishing URL

- Typosquatting: Đăng ký tên miền gần giống tên miền thật, tận dụng sai sót của người dùng để dẫn họ tới trang lừa đảo.



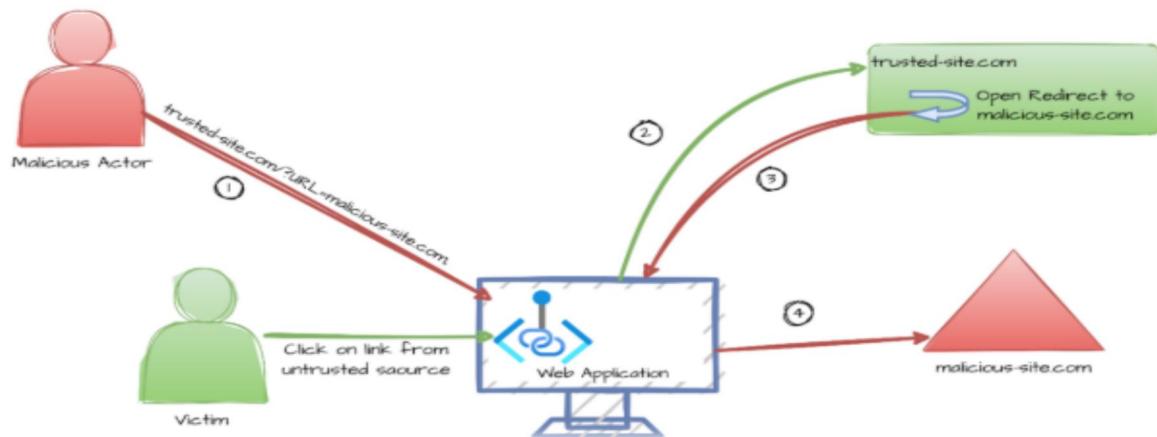
Hình 1.3 Minh họa kỹ thuật tấn công Typosquatting

- Homograph/IDN attacks: Sử dụng ký tự giống nhau từ các bảng mã khác như Punycode, ký tự Unicode để tạo tên miền giả mạo mà mắt thường khó phân biệt.



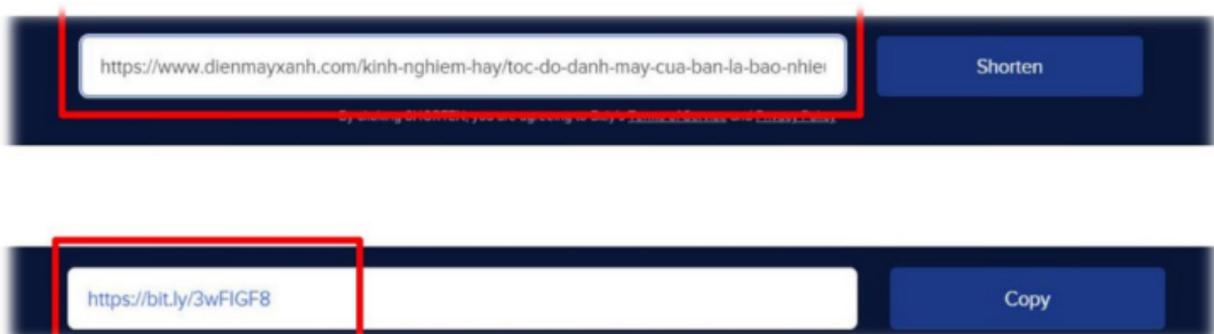
Hình 1.4 Minh họa kỹ thuật Homograph sử dụng Punycode để đánh lừa người dùng

- Open redirect abuse: Lợi dụng tham số chuyển hướng trên trang hợp pháp để dẫn người dùng sang địa chỉ độc hại, làm tăng độ tin cậy ban đầu của URL.



Hình 1.5 Minh họa kỹ thuật Open redirect abuse

- URL shortening abuse: Che giấu đích đến thực sự bằng dịch vụ rút gọn như: bit.ly, tinyurl khiến người dùng và hệ thống khó xác minh trước khi nhấp.



Hình 1.6 Minh họa kỹ thuật URL shorting abuse

- Obfuscation and redirect chains: Mã hóa hoặc làm rối chuỗi URL hoặc chuyển hướng nhiều bước nhằm né kiểm duyệt và làm phức tạp việc phân tích.

```

1 http://www.trustedsite.com/search.html?type=<><<sCript>alert
2 (document.cookie)</sCript><"<<sCript>alert(document.cookie)
3 </sCript>
4 http://www.trustedsite.com/search.html?type=%3C%22%3C%3C
5 %73%43%72%49%70%54%3E%61%6C%65%72%74%28%64
6 %6F%63%75%6D%65%6E%74%2E%63%6F%6B%69%
7 65%29%3C%2F%73%43%72%49%70%54%3E%3C%22%3C
8 %3C%73%43%72%49%70%54%3E%61%6C%65%72%74%28
9 %64%6F%63%75%6D%65%6E%74%2E%63%6F%6B%
10 69%65%29%3C%2F%73%43%72%49%70%54%3E

```

Hình 1.7 Minh họa kỹ thuật tấn công Obfuscation

- Malware distribution URL: Dẫn người dùng tới trang chứa exploit hoặc file tải về có mã độc.



Hình 1.8 Minh họa kỹ thuật Malware distribution URL

Những kỹ thuật trên thường kết hợp với social engineering và các biện pháp kỹ thuật khác như: WHOIS ẩn hoặc sử dụng máy chủ tạm thời để tăng hiệu quả và giảm khả năng phát hiện, do đó việc phát hiện URL độc hại đòi hỏi cả phân tích chuỗi, ngữ cảnh và hành vi khi truy cập.

1.1.4 Dấu hiệu và hành vi của URL Phishing

Từ những URL phishing thường có những đặc điểm bất thường trong cấu trúc và hành vi nhằm đánh lừa người dùng hoặc hệ thống phát hiện. Về cấu trúc chuỗi, các URL này thường dài

bất thường, chứa nhiều ký tự đặc biệt như: “-”, “_”, “%”, “@”... hoặc chuỗi ký tự ngẫu nhiên, sử dụng subdomain để giả mạo tên miền thật, hoặc thay thế ký tự bằng số tương tự . Nhiều URL còn che giấu đích đến thật thông qua dịch vụ rút gọn liên kết hoặc mã hóa trong phần query.

Về hành vi, khi người dùng truy cập, các URL phishing thường dẫn đến trang web yêu cầu nhập thông tin nhạy cảm như: tên tài khoản, mật khẩu, thẻ tín dụng ngay lập tức, chứa nhiều iframe, script lạ, hoặc form giả mạo. Một số trang sử dụng chuyển hướng nhiều tầng để né hệ thống lọc hoặc hiển thị giao diện bắt chước gần giống trang thật nhưng có chứng chỉ bảo mật không hợp lệ hoặc không khớp tên miền. Ngoài ra, các tên miền phishing thường mới đăng ký, có thời gian tồn tại ngắn, thông tin WHOIS bị ẩn, và máy chủ đặt tại khu vực rủi ro cao.

Domain Whois record

Queried **whois.internic.net** with "dom **spotifycalendar.com**"...

```

Domain Name: SPOTIFYCALENDAR.COM
Registry Domain ID: 2982642961_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.cosmotown.com
Registrar URL: http://www.cosmotown.com
Updated Date: 2025-05-12T09:15:42Z
Creation Date: 2025-05-12T09:14:51Z
Registry Expiry Date: 2026-05-12T09:14:51Z
Registrar: Cosmotown, Inc.

```

Hình 1.9 Minh họa thông tin Whois của một website đáng ngờ

Từ những đặc điểm trên, cho thấy việc phát hiện URL phishing hiệu quả đòi hỏi kết hợp nhiều lớp phân tích: đặc trưng chuỗi, domain, WHOIS, chứng chỉ TLS và hành vi động khi truy cập, nhằm xây dựng mô hình học máy, học sâu có khả năng nhận biết các mẫu tấn công tinh vi và biến đổi liên tục trên không gian mạng.

1.1.5 Bối cảnh nghiên cứu

Trong lĩnh vực an toàn thông tin, phát hiện URL phishing là một trong những bài toán trọng tâm nhằm ngăn chặn tấn công lừa đảo trực tuyến. Bản chất của bài toán được mô hình hóa dưới dạng phân loại nhị phân, trong đó mỗi URL được gán nhãn là hợp pháp hoặc giả mạo dựa

trên các đặc trưng được trích xuất. Mục tiêu của hệ thống là học được ranh giới phân tách giữa hai nhóm URL, giúp dự đoán chính xác các liên kết độc hại chưa từng xuất hiện trước đó.

Quá trình nhận diện được thực hiện thông qua việc thu thập và chuẩn hóa dữ liệu URL từ nhiều nguồn, trích xuất đặc trưng chuỗi URL, ngữ nghĩa và hành vi, sau đó huấn luyện bằng các mô hình Machine Learning, Deep Learning và Large Language Models. Các đặc trưng như độ dài URL, ký tự đặc biệt, entropy, tên miền, chứng chỉ TLS, hoặc mẫu token nghi vấn được dùng làm đầu vào cho mô hình. Kết quả đầu ra là xác suất hoặc nhãn dự đoán là: 0 là phishing, 1 là legitimate.

Bài toán này mang tính thực tiễn cao vì các URL phishing liên tục thay đổi để né tránh phát hiện. Do đó, hướng nghiên cứu hiện đại tập trung vào nhiều phương pháp học máy, tối ưu chọn lọc đặc trưng, và giải thích quyết định mô hình bằng SHAP, LIME nhằm đạt được độ chính xác cao, khả năng tổng quát tốt và đảm bảo tính minh bạch trong ứng dụng thực tế.

1.1.6 Bài toán đặt ra

Từ bối cảnh trên, bài toán phát hiện URL phishing được xác định là một bài toán phân loại nhị phân trong lĩnh vực học máy và an toàn thông tin. Đầu vào của hệ thống là tập hợp các chuỗi URL thu thập từ nhiều nguồn khác nhau, được chuẩn hóa và biểu diễn dưới dạng đặc trưng số hoặc embedding. Đầu ra là nhãn dự đoán tương ứng, trong đó 0 đại diện cho URL giả mạo và 1 đại diện cho URL hợp pháp.

Mục tiêu của bài toán là xây dựng một mô hình tự động có khả năng học và phân biệt được các đặc trưng của URL phishing và legitimate, ngay cả khi chúng có hình thức tương tự nhau. Để đạt được điều này, cần kết hợp nhiều phương pháp học máy và học sâu hiện đại, đồng thời áp dụng kỹ thuật chọn lọc đặc trưng nhằm loại bỏ yếu tố dư thừa, xử lý mất cân bằng dữ liệu, và đánh giá mô hình bằng các chỉ số như Precision, Recall, F1-score, PR-AUC và Confusion Matrix.

Ngoài việc đạt độ chính xác cao, hệ thống cần đảm bảo khả năng giải thích kết quả dự đoán thông qua các phương pháp như SHAP và LIME, giúp mô hình trở nên minh bạch, dễ hiểu và đáng tin cậy khi ứng dụng vào môi trường thực tế, đặc biệt trong các hệ thống phát hiện URL độc hại hoạt động thời gian thực.

1.2 Tổng quan về những đặc trưng phát hiện url giả mạo

Việc trích xuất đặc trưng là bước quan trọng nhất quyết định độ chính xác của mô hình học máy. Đối với bài toán phát hiện URL giả mạo, các đặc trưng thường được chia thành bốn nhóm chính: đặc trưng từ vựng, đặc trưng dựa trên máy chủ, đặc trưng nội dung và đặc trưng ngữ nghĩa hiện đại.

1.2.1 Nhóm đặc trưng ký tự

Đây là nhóm đặc trưng truyền thống dựa trên việc phân tích chuỗi ký tự của URL mà không cần kết nối internet. Các đặc trưng này rất hiệu quả trong việc phát hiện các kỹ thuật giả mạo bì mặt. Một số đặc trưng ký tự như:

- Đặc trưng độ dài:
 - + Độ dài URL: Các URL lừa đảo thường rất dài để che giấu phần địa chỉ đích thực sự trên thanh địa chỉ của trình duyệt.
 - + Độ dài Hostname: Tên miền lừa đảo thường dài hơn do chứa nhiều từ khóa nhồi nhét.
- Đặc trưng ký tự:
 - + Số lượng dấu chấm “.”: Một URL hợp lệ thường chỉ có từ 2-3 dấu chấm. Nếu số lượng dấu chấm lớn, khả năng cao đó là URL giả mạo dùng nhiều subdomain.
 - + Ký tự đặc biệt “@, //, -“: Dấu @ thường bị lợi dụng để trình duyệt bỏ qua tất cả các ký tự phía trước nó. Dấu gạch ngang - thường được dùng trong kỹ thuật Typosquatting để gây nhầm lẫn thị giác.
 - Mô hình URL bất thường:
 - + Sử dụng địa chỉ IP: Nếu phần domain là một địa chỉ IP, đây là dấu hiệu khả nghi cao vì các trang web uy tín luôn dùng tên miền.
 - + Dịch vụ rút gọn: Kẻ tấn công thường dùng bit.ly, tinyurl để che giấu URL gốc.

1.2.2 Nhóm đặc trưng dựa trên máy chủ

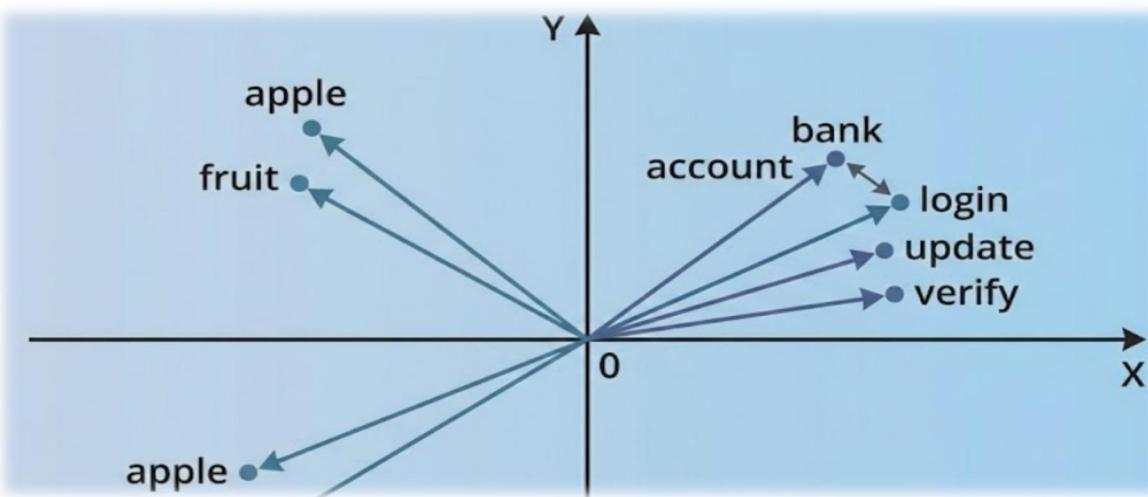
Nhóm đặc trưng này khai thác các thông tin về hạ tầng mạng và định danh của tên miền. Tuy tồn thời gian truy vấn hơn đặc trưng từ vựng nhưng cung cấp độ tin cậy cao. Một số đặc trưng nổi bật dựa vào máy chủ như:

- Tuổi đời tên miền: Các trang web lừa đảo thường có vòng đời rất ngắn. Ngược lại, các trang web hợp pháp thường tồn tại lâu năm.
- Thông tin WHOIS: Kiểm tra tính đầy đủ của thông tin đăng ký, ngày đăng ký và ngày hết hạn.
- Chứng chỉ bảo mật: Trước đây, HTTPS là dấu hiệu của trang web an toàn. Tuy nhiên, hiện nay kẻ tấn công có thể sử dụng các chứng chỉ SSL miễn phí. Do đó, cần phân tích kỹ đơn vị phát hành chứng chỉ.

1.2.3 Nhóm đặc trưng ngữ nghĩa và văn bản

Sự phát triển của Xử lý ngôn ngữ tự nhiên và các mô hình ngôn ngữ lớn đã mở ra hướng tiếp cận mới: coi URL là một văn bản cần hiểu nghĩa thay vì chỉ đếm ký tự.

- Từ khóa nhạy cảm: Sự xuất hiện của các từ khóa như "login", "verify", "account", "update", "banking" trong URL là dấu hiệu quan trọng. Kẻ tấn công dùng chúng để thao túng tâm lý người dùng.
- Biểu diễn nhúng: Các phương pháp như Word2Vec, FastText hoặc GloVe chuyển đổi các token trong URL thành các vector số học. Ví dụ, vector của từ "bank" sẽ nằm gần vector của từ "account" trong không gian số, giúp mô hình học được mối quan hệ từ vựng.



Hình 1.10 Minh họa embedding chuyển đổi token trong URL thành vector số học

- Biểu diễn ngữ cảnh: Khác với Embedding tĩnh, các mô hình hiện đại như BERT, ALBERT có khả năng hiểu ngữ cảnh của từ. Ví dụ, mô hình có thể phân biệt sự khác nhau giữa apple.com và apple-store-verify.com dựa trên cơ chế attention, giúp phát hiện các mẫu tấn công tinh vi mà phương pháp đếm ký tự truyền thống bỏ sót.

1.3 Tìm hiểu và đề xuất các mô hình

Để giải quyết bài toán phát hiện URL giả mạo, việc lựa chọn và kết hợp các mô hình học máy, học sâu và mô hình ngôn ngữ lớn đóng vai trò quan trọng trong việc nâng cao độ chính xác và khả năng khái quát của hệ thống. Mỗi nhóm mô hình có ưu thế riêng trong việc xử lý các loại đặc trưng khác nhau, từ đặc trưng số học đến chuỗi ký tự phức tạp của URL. Để tài đề xuất ba hướng mô hình chính tương ứng với các phương pháp học hiện đại: Machine Learning, Deep Learning, và Large Language Models nhằm so sánh độ hiệu quả giữa các mô hình từ đó đưa ra đánh giá và nhận định.

1.3.1 Các mô hình học máy

Trong bài toán phát hiện URL giả mạo, các mô hình học máy đóng vai trò quan trọng trong việc phân loại URL dựa trên bộ đặc trưng số được trích xuất từ URL. Những đặc trưng này có thể bao gồm: độ dài URL, số lượng ký tự đặc biệt, tỷ lệ chữ số, chiều dài domain, độ sâu thư mục, số lượng tham số truy vấn, hoặc các đặc trưng thống kê khác phản ánh sự bất thường của URL.

Các mô hình ML phù hợp với dạng dữ liệu cấu trúc, có ưu điểm:

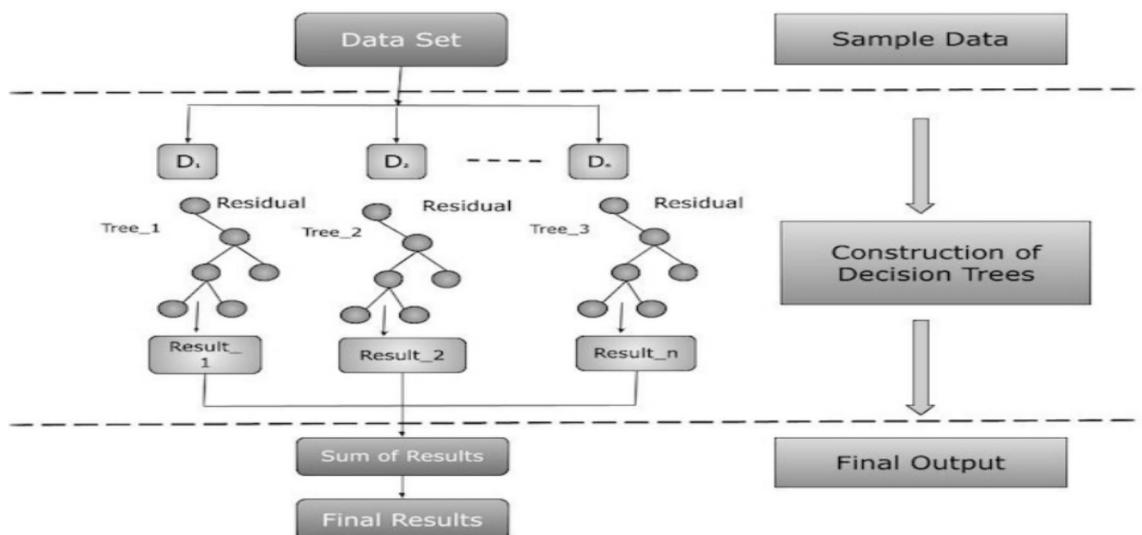
- Tốc độ huấn luyện nhanh
- Đễ diễn giải
- Khả năng xử lý tốt dữ liệu phi tuyến
- Hoạt động hiệu quả khi đặc trưng được chuẩn bị và chọn lọc hợp lý

Trong phần này, một số mô hình học máy phổ biến và có tính ứng dụng cao trong lĩnh vực an ninh mạng, đặc biệt là phát hiện URL giả mạo, sẽ được trình bày: Random Forest, và XGBoost.

1.3.1.1 XGBoost

1.3.1.1.1 Tổng quan và nguyên lý hoạt động

XGBoost là một kỹ thuật học máy thuộc họ học tăng cường, được thiết kế để tối ưu hóa tốc độ tính toán và hiệu năng mô hình. Khác với các mô hình đơn lẻ như cây quyết định, XGBoost hoạt động dựa trên nguyên lý học kết hợp, cụ thể là kết hợp hàng loạt các mô hình yếu để tạo thành một mô hình mạnh. Cơ chế cốt lõi của XGBoost là quá trình huấn luyện cộng. Các cây quyết định không được xây dựng độc lập mà được thêm vào một cách tuần tự. Cây thứ t được sinh ra với mục tiêu học và sửa chữa các sai số do cây thứ t-1 để lại. Kết quả dự đoán cuối cùng là tổng trọng số của tất cả các cây trong mô hình.



Hình 1.11 Mô tả nguyên lý hoạt động của mô hình XGBoost

Giải thích:

- Data Set tạo thành các tập D₁, D₂, D_n:
- + Toàn bộ dữ liệu ban đầu được đưa vào quá trình huấn luyện.
- + Ở mỗi vòng lặp, XGBoost tạo một tập dữ liệu tương ứng D₁, D₂

- + Tập dữ liệu trong mỗi vòng sẽ phản ánh mức độ dự đoán của từng mẫu.
- Tính phần sai số:
- + Trước khi xây cây thứ m, mô hình đã có dự đoán từ những cây trước.
- + XGBoost tính phần sai số còn lại mà mô hình chưa dự đoán tốt.
- + Phần sai số trong XGBoost chính là gradient của hàm loss giúp cây mới tập trung học vào các mẫu bị dự đoán sai.
- Xây dựng các cây quyết định:
- + Với mỗi phần sai số XGBoost sẽ huấn luyện một cây quyết định nhỏ.
- + Cây không dự đoán nhãn trực tiếp mà dự đoán giá trị sửa lỗi.
- + Các nút tròn trong cây thể hiện các lần tách thuộc tính.
- Kết quả:
- + Output của mỗi cây là giá trị cần cộng thêm vào mô hình hiện tại để cải thiện dự đoán.
- + Mỗi cây đóng góp một phần nhỏ vào mô hình cuối cùng.

1.3.1.1.2 Cơ sở toán học và hàm mục tiêu

Điểm khác biệt lớn nhất giúp XGBoost vượt trội hơn các thuật toán Gradient Boosting truyền thống nằm ở hàm mục tiêu. XGBoost tích hợp thành phần điều chỉnh trực tiếp vào hàm mất mát để kiểm soát độ phức tạp của mô hình, giúp hạn chế tối đa hiện tượng overfitting. Hàm mục tiêu tại bước lặp thứ t được định nghĩa như sau:

$$Obj^{(t)} = \sum_{i=1}^n L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

Giải thích các tham số trong công thức:

- n: Số lượng mẫu dữ liệu huấn luyện.
- L: Hàm mất mát, đo lường sự sai lệch giữa nhãn thực tế y_i và giá trị dự đoán $\hat{y}_i^{(t-1)}$
- $f_t(x_i)$: Kết quả dự đoán của cây mới thứ t.
- $\Omega(f_t)$: Hàm điều chỉnh, để phạt độ phức tạp của cây mới.

Công thức của hàm điều chuẩn $\Omega(f_t)$ được tính toán dựa trên số lượng lá và trọng số của các lá trong cây:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \|\omega\|^2 = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T \omega_j^2$$

Giải thích các tham số có trong công thức:

- T : Số lượng lá của cây.
- ω_j : Trọng số tại các lá.
- γ : Tham số phạt dựa trên số lượng lá. Giá trị γ càng lớn thì thuật toán càng thận trọng trong việc tạo thêm lá mới giúp cắt tia cây.
- λ : Tham số phạt L2 regularization trên trọng số lá, giúp trọng số không bị quá lớn, làm mượt mô hình

Để tối ưu hóa hàm mục tiêu này nhanh chóng, XGBoost sử dụng khai triển Taylor bậc hai để xấp xỉ hàm matsu. Điều này cho phép thuật toán sử dụng thêm thông tin từ đạo hàm bậc hai, giúp hội tụ nhanh hơn và chính xác hơn so với việc chỉ sử dụng đạo hàm bậc nhất như các mô hình truyền thống:

$$Obj^{(t)} \approx \sum_{i=1}^n \left[L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

Trong đó:

g_i và h_i lần lượt là đạo hàm bậc nhất và đạo hàm bậc hai của hàm matsu.

1.3.1.2 Random Forest

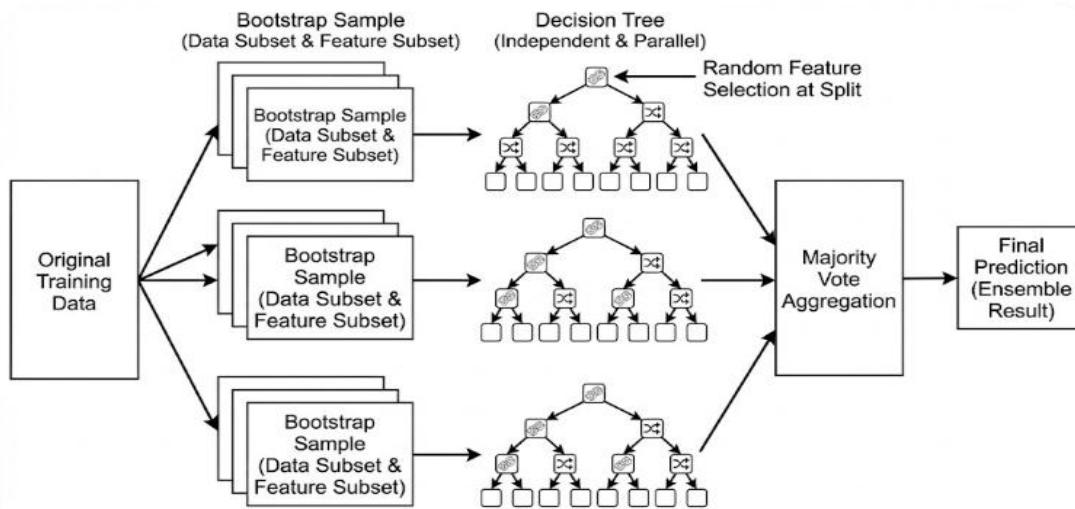
1.3.1.2.1 Tổng quan và nguyên lý hoạt động

Định nghĩa: Random Forest là một thuật toán học máy thuộc nhóm Học kết hợp, cụ thể là kỹ thuật Bagging.

Ý tưởng cốt lõi: Thay vì phụ thuộc vào một cây quyết định duy nhất dễ bị sai lệch, Random Forest xây dựng một rừng cây gồm nhiều cây quyết định hoạt động song song và độc lập. Kết quả cuối cùng được quyết định dựa trên cơ chế bầu chọn số đông.

Sự ngẫu nhiên: Thuật toán đưa sự ngẫu nhiên vào ở hai khâu:

- Lấy mẫu dữ liệu: Mỗi cây được huấn luyện trên một tập con ngẫu nhiên của dữ liệu gốc.
- Lấy mẫu đặc trưng: Tại mỗi nút phân chia, chỉ một tập con ngẫu nhiên các đặc trưng được xem xét, giúp các cây trong rừng không bị tương quan.



Hình 1.12 Mô tả nguyên lý hoạt động mô hình Random Forest

Giải thích nguyên lý hoạt động mô hình Random Forest:

- Giai đoạn lấy mẫu:
 - + Từ tập dữ liệu huấn luyện gốc, thuật toán tạo ra nhiều tập dữ liệu con khác nhau thông qua kỹ thuật lấy mẫu có hoàn lại.
 - + Điểm đặc biệt được thể hiện trong sơ đồ là tại mỗi tập mẫu, thuật toán không chỉ lấy ngẫu nhiên các dòng dữ liệu mà còn lấy ngẫu nhiên một tập con các đặc trưng. Điều này nhằm đảm bảo tính đa dạng cho các cây quyết định được sinh ra sau này.
- Giai đoạn huấn luyện song song:
 - + Với mỗi tập mẫu Bootstrap được tạo ra, một cây quyết định sẽ được xây dựng tương ứng.

- + Quá trình xây dựng các cây này diễn ra độc lập và song song.
- + Tại mỗi nút phân chia trong cây, thuật toán chỉ xem xét một tập hợp ngẫu nhiên các đặc trưng để tìm ra điểm phân chia tốt nhất, thay vì xem xét toàn bộ đặc trưng như cây quyết định truyền thống. Cơ chế này giúp giảm sự tương quan giữa các cây trong rừng, hạn chế hiện tượng quá khớp.
- Giai đoạn tổng hợp kết quả:
- + Sau khi huấn luyện xong, mỗi cây quyết định sẽ đưa ra một kết quả dự đoán riêng biệt cho dữ liệu đầu vào mới.
- + Hệ thống sẽ thu thập tất cả các kết quả này để tiến hành tổng hợp. Trong bài toán phân loại, phương pháp được sử dụng là Bầu chọn đa số.
- Dự đoán cuối cùng:
- + Kết quả cuối cùng được xác định là nhãn lớp nhận được nhiều phiếu bầu nhất từ các cây thành viên. Việc kết hợp ý kiến của nhiều cây giúp mô hình đạt được độ ổn định cao hơn và sai số thấp hơn so với một cây đơn lẻ.

1.3.1.2.2 Cơ sở toán học

Trong mô hình Random Forest, các cây quyết định thành viên được xây dựng dựa trên thuật toán CART. Cơ sở toán học của mô hình tập trung vào hai thành phần chính: Tiêu chí phân chia nút dựa trên chỉ số Gini và cơ chế tổng hợp kết quả.

Chỉ số Gini: Để đánh giá chất lượng của một phép phân chia tại mỗi nút, Random Forest sử dụng chỉ số Gini để đo lường độ bất định của dữ liệu. Mục tiêu của thuật toán là cực tiểu hóa chỉ số này, tức là làm cho các nút con sinh ra có độ tinh khiết cao nhất. Công thức tính chỉ số Gini tại một nút t được định nghĩa như sau:

$$Gini(t) = 1 - \sum_{i=1}^c (p_{i,t})^2$$

Giải thích các tham số có trong công thức:

- t: Nút hiện tại đang xét trên cây

- $p_{i,t}$ là xác suất xuất hiện của lớp i có thể là Phishing hoặc Legitimate tại nút t đó. Giá trị này được tính bằng tỉ lệ số mẫu thuộc lớp i trên tổng số mẫu tại nút đó.
- c là tổng số lớp trong bài toán phân loại.

Nếu $Gini(t) = 0$, thì nút đó hoàn toàn tinh khiết

Cơ chế bầu chọn: Random Forest là một phương pháp học kết hợp. Sau khi huấn luyện N cây quyết định độc lập trên các tập mẫu Bootstrap, kết quả dự đoán cuối cùng cho một mẫu dữ liệu x được quyết định thông qua cơ chế bầu chọn đa số. Công thức toán học cho kết quả dự đoán \hat{y} :

$$\hat{y} = \arg \max_c \left(\sum_{n=1}^N I(T_n(x) = c) \right)$$

Giải thích các tham số trong công thức:

- x: Vector đặc trưng của URL đầu vào.
- N: Tổng số cây quyết định trong rừng.
- $T_n(x)$: Kết quả dự đoán nhãn của cây thứ n đối với mẫu x.
- c: Nhãn lớp đang xét Với $c \in \{0: \text{Phishing}, 1: \text{Legitimate}\}$
- I(): Hàm chỉ thị, trả về 1 nếu điều kiện đúng nếu cây dự đoán đúng lớp c và 0 nếu sai.
- argmax: Hàm chọn ra nhãn lớp c có tổng số phiếu bầu cao nhất.

1.3.2 Các mô hình học sâu

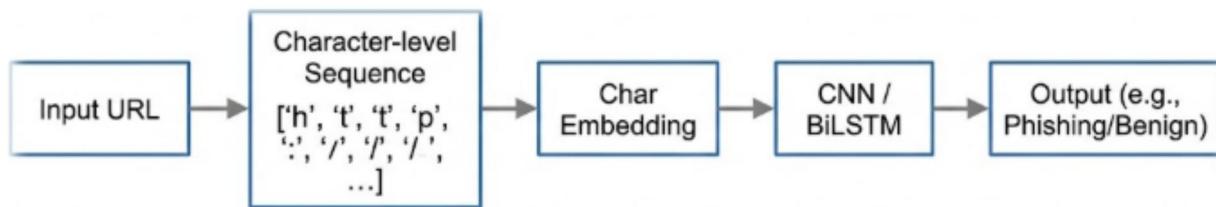
Các mô hình học sâu cho phép hệ thống học trực tiếp biểu diễn ngữ nghĩa từ chuỗi URL thông qua các tầng embedding và mạng nơ-ron nhiều lớp. Khác với các phương pháp học máy truyền thống vốn phụ thuộc vào đặc trưng số học được trích xuất thủ công, các mô hình DL có khả năng tự động học đặc trưng từ dữ liệu thô như chuỗi ký tự hoặc chuỗi token.

Trong bài toán phát hiện URL phishing, DL đặc biệt hiệu quả vì URL thường chứa các mẫu ký tự tinh vi, chuỗi bị xáo trộn, kỹ thuật che giấu, homograph IDN, typosquatting và các

mẫu từ khoá đặc trưng. DL có khả năng học những mẫu này mà không cần xây dựng đặc trưng thủ công.

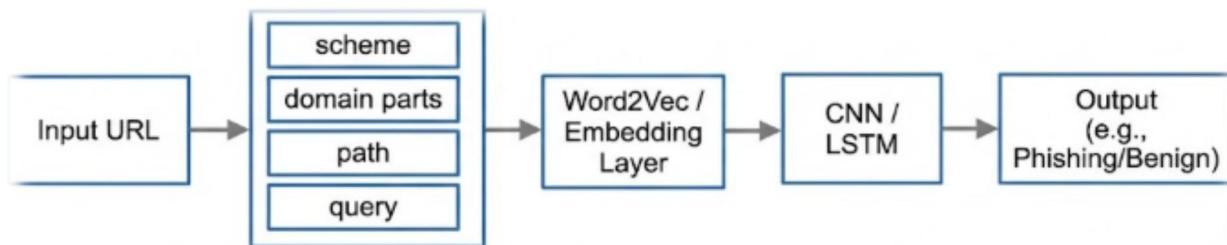
Dữ liệu đầu vào trong mô hình học sâu URL tùy pipeline thì đầu vào có hai dạng:

- Character-level sequence → Char embedding → CNN/BiLSTM



Hình 1.13 Pipeline char-embedding character-level sequence

- Tokenized URL như scheme, domain parts, path, query → Word2Vec/Embedding Layer → CNN, LSTM



Hình 1.14 Pipeline char-embedding URL-token sequence

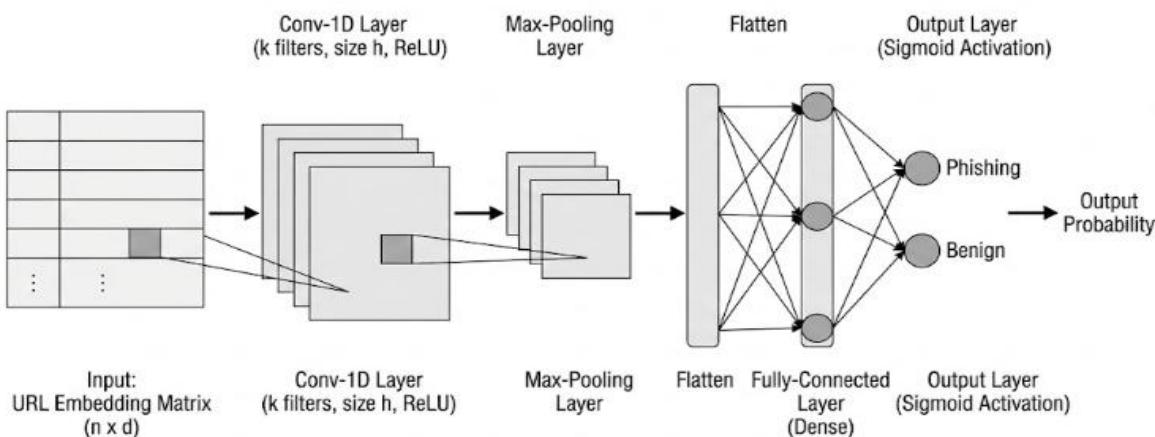
1.3.2.1 CNN

1.3.2.1.1 Tổng quan

Định nghĩa: CNN là một kiến trúc nơ-ron sâu được lấy cảm hứng từ vỏ não thị giác sinh học. Mặc dù CNN thường được biết đến với các thành tựu trong Thị giác máy tính xử lý ảnh 2

chiều, nhưng trong bài toán xử lý ngôn ngữ tự nhiên và phân loại URL, biến thể 1D-CNN lại phát huy hiệu quả vượt trội.

Cơ chế hoạt động: Thay vì xử lý ma trận điểm ảnh, 1D-CNN trượt các bộ lọc dọc theo chuỗi ký tự hoặc chuỗi từ của URL. Điều này cho phép mô hình tự động phát hiện các mẫu cục bộ quan trọng như các cụm từ "secure", "account", "login" hay các chuỗi ký tự ngẫu nhiên bất kể chúng xuất hiện ở vị trí nào trong URL.



Hình 1.15 Cơ chế hoạt động của mô hình CNN

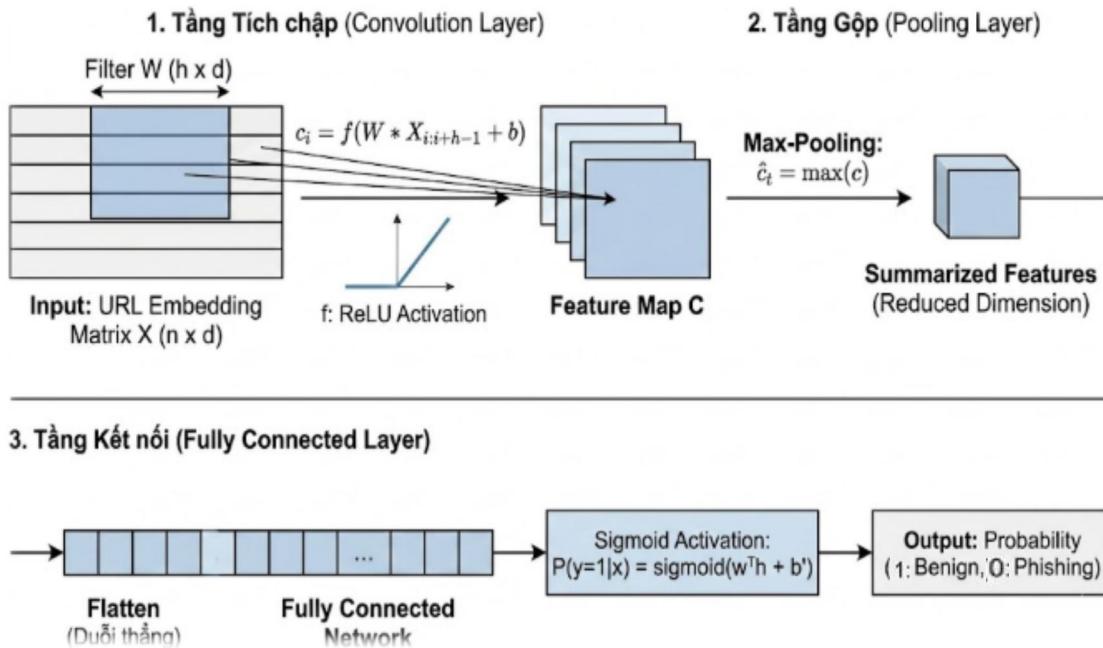
CNN là mô hình nơ-ron tích chập được thiết kế nhằm trích xuất đặc trưng cục bộ thông qua các bộ lọc quét ngang chuỗi embedding của URL.

Cấu trúc điển hình:

- Embedding layer: biểu diễn mỗi ký tự hay token thành vector.
- 1D Convolution layers: phát hiện các mẫu ký tự như “verify”, “confirm”, “login”, “secure”.
- Pooling layer: giảm chiều, giữ lại đặc trưng quan trọng nhất.
- Dense layer: phân loại phishing / legitimate.

1.3.2.1.2 Kiến trúc chi tiết của 1D-CNN cho phân loại URL

Một mô hình CNN tiêu chuẩn dùng cho phân loại URL bao gồm ba tầng chính: Tầng tích chập, Tầng gộp và Tầng kết nối.



Hình 1.16 Mô tả minh họa kiến trúc 1D-CNN cho phân loại URL Phishing

- Tầng tích chập: Đây là nơi trích xuất đặc trưng quan trọng nhất. Giả sử URL được biểu diễn dưới dạng ma trận nhúng X kích thước $n \times d$. Một ma trận trọng số của bộ lọc ω có kích thước cửa sổ h sẽ qua X để tạo ra một đặc trưng mới c_i theo công thức: $c_i = f(\omega \cdot x_{i:i+h-1} + b)$

Trong đó:

- $x_{i:i+h-1}$: Đoạn cửa sổ cục bộ của URL từ vị trí i đến $i+h-1$.
- b : Tham số định kiển, giúp điều chỉnh ngưỡng kích hoạt.
- f : Hàm kích hoạt phi tuyến sử dụng ReLU để giúp mô hình học các tương quan phức tạp và tránh tiêu biến đạo hàm.

- Tầng gộp: Sau khi tích chập, đầu ra thường có chiều lớn. Tầng Pooling giúp giảm chiều dữ liệu, giảm chi phí tính toán và quan trọng nhất là tạo ra tính bất biến dịch chuyển. Kỹ thuật phổ biến nhất là Max-Pooling: $\hat{c} = \max\{c\}$
- + Kỹ thuật này chọn ra giá trị lớn nhất trong bản đồ đặc trưng, nghĩa là nó chỉ quan tâm xem dấu hiệu lừa đảo có xuất hiện hay không, chứ không quá quan trọng việc nó nằm ở đâu hay cuối URL.
- Tầng kết nối đầy đủ: Các đặc trưng sau khi được gộp sẽ được duỗi thẳng thành một vector và đưa vào mạng nơ-ron truyền thống. Lớp cuối cùng sử dụng hàm kích hoạt Sigmoid để đưa ra xác suất 0|1. Với công thức dưới ra:

$$P(y=1|x) = \text{sigmoid}(\omega^T h + b')$$

Trong đó:

- $P(y=1|x)$: Xác suất để đầu vào x thuộc lớp 1.
 - h : Vector đặc trưng cấp cao là đầu ra của tầng Fully Connected trước đó.
 - ω^T : Trọng số của lớp đầu ra.
 - b' : Bias của lớp đầu ra.
 - sigmoid: Hàm kích hoạt Sigmoid giúp ép giá trị về khoảng (0, 1) để biểu thị xác suất.
- Công thức:

$$\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

1.3.2.2 CNN–LSTM kết hợp

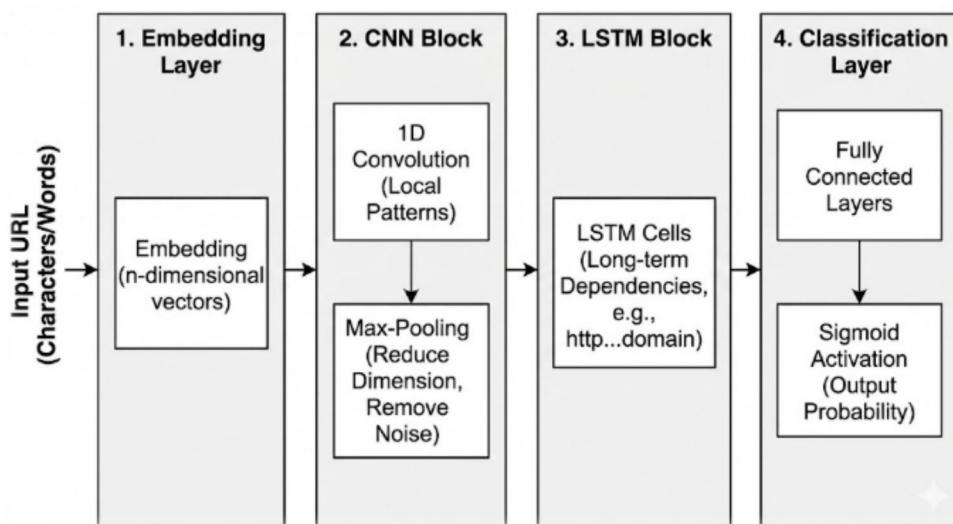
1.3.2.2.1 Tổng quan

Trong bài toán phát hiện URL giả mạo, chúng ta đối mặt với hai thách thức đồng thời. Thứ nhất, cần phát hiện các từ khóa lừa đảo rời rạc, ví dụ: "login", "update" là thế mạnh của CNN. Thứ hai, cần hiểu cấu trúc sắp xếp tuần tự của các từ khóa đó trong URL, ví dụ: "paypal"

đứng trước "com" là chính xác, nhưng "paypal" đứng trước "update" là giả mạo chính là thế mạnh của LSTM.

Giải pháp Hybrid: Đề xuất kiến trúc lai ghép CNN-LSTM. Tư tưởng chủ đạo là sử dụng CNN như một bộ trích xuất đặc trưng để xử lý thông tin từ chuỗi URL thô, sau đó đưa chuỗi đặc trưng cấp cao này vào LSTM để phân tích quy luật thời gian. Cách tiếp cận này giúp mô hình vừa nhìn thấy chi tiết, vừa nhìn thấy tổng thể.

1.3.2.2.2 Kiến trúc chi tiết của hệ thống



Hình 1.17 Kiến trúc hệ thống của mô hình CNN-LSTM

Mô hình lai ghép được xây dựng theo kiến trúc phân tầng gồm 4 khối chức năng chính: Tầng nhúng, khối

- Tầng nhúng: Chuyển đổi chuỗi ký tự/từ trong URL thành các vector số học trong không gian n-chiều .
- Khối CNN:
- + Thực hiện tích chập 1 chiều lên đầu vào để phát hiện các mẫu cục bộ.

- + Áp dụng lớp Max-Pooling ngay sau đó. Bước này cực kỳ quan trọng giúp giảm chiều dữ liệu, loại bỏ nhiễu và giữ lại những đặc trưng nổi bật nhất. Nhờ đó, chuỗi đầu vào cho LSTM sẽ ngắn hơn và "sạch" hơn.
- Khối LSTM:
- + Nhận đầu vào là chuỗi các bản đồ đặc trưng từ khối CNN.
- + Thực hiện lan truyền qua các cell LSTM để học mối quan hệ phụ thuộc dài hạn giữa các đặc trưng. Ví dụ: LSTM sẽ học được mối liên kết giữa giao thức http ở đầu chuỗi với một tên miền lạ ở giữa chuỗi.
- Tầng phân loại:
- + Vector trạng thái ẩn cuối cùng của LSTM được đưa qua các lớp kết nối đầy đủ để tổng hợp thông tin.
- + Lớp cuối cùng sử dụng hàm kích hoạt Sigmoid để đưa ra xác suất dự đoán W_0 .

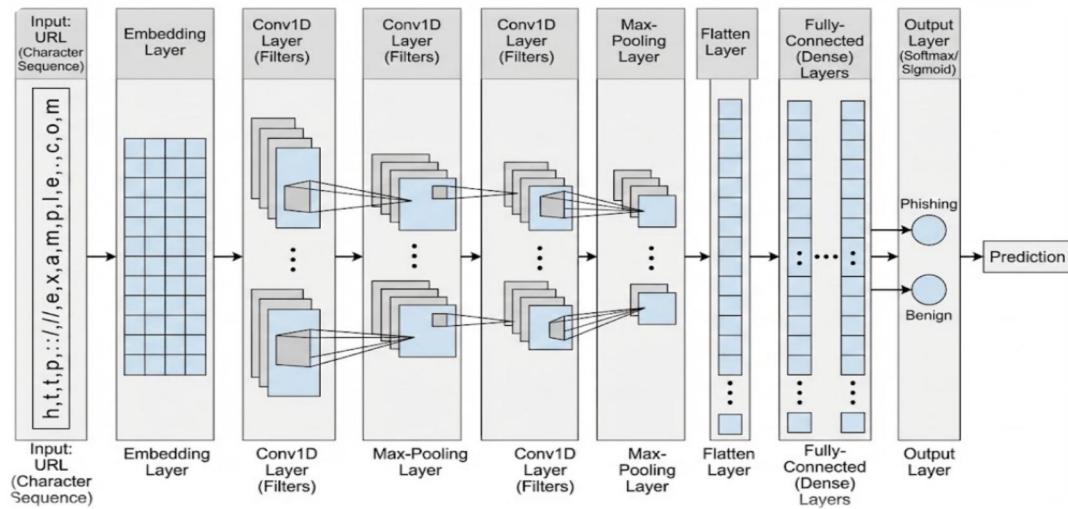
1.3.2.3 CharCNN

1.3.2.3.1 Tổng quan

Hạn chế của mô hình cấp độ từ: Các mô hình xử lý ngôn ngữ tự nhiên truyền thống thường dựa trên từ điển. Tuy nhiên, URL là các chuỗi ký tự hỗn loạn, chứa nhiều tên miền ngẫu nhiên, mã thông báo và các từ sai chính tả có chủ đích. Khi gặp các từ không có trong từ điển, mô hình cấp độ từ thường thất bại.

Giải pháp CharCNN: CharCNN coi văn bản là một chuỗi các ký tự thô thay vì các từ. Mô hình học cách biểu diễn dữ liệu trực tiếp từ các ký tự, do đó không bao giờ gặp vấn đề từ lạ và có khả năng nắm bắt cấu trúc hình thái của URL.

1.3.2.3.2 Kiến trúc và cơ chế hoạt động



Hình 1.19 Kiến trúc hoạt động của mô hình CharCNN

Quá trình xử lý của CharCNN được thực hiện qua các bước sau:

- Lượng tử hóa ký tự:
- + Đầu tiên, xây dựng một tập bảng chữ cái C bao gồm các ký tự hợp lệ trong URL ví dụ: 26 chữ cái thường, 10 chữ số và 32 ký tự đặc biệt. Kích thước C = m.
- + Mỗi URL đầu vào được cắt hoặc đệm để có độ dài cố định l_0 .
- + Sử dụng kỹ thuật One-hot Encoding: Mỗi ký tự được chuyển thành một vector có độ dài m, với giá trị 1 tại vị trí của ký tự đó và 0 ở các vị trí còn lại.
- + Kết quả: Một URL được biểu diễn thành ma trận X có kích thước $l_0 \times m$
- Tích chập mức ký tự:
- + Thực hiện phép tích chập 1 chiều giữa ma trận đầu vào X và các bộ lọc.
- + Điểm đặc biệt: Các bộ lọc này sẽ học các đặc trưng hình thái của chuỗi ký tự. Ví dụ: nó có thể học được rằng chuỗi "paypal" rất giống với "paypal" về mặt cấu trúc, điều mà Word2Vec không làm được.
- Tầng Gộp và Phân loại:

- + Sử dụng Max-Pooling để giảm chiều dữ liệu và tìm ra các đặc trưng nổi bật nhất bát biến với vị trí.
- + Cuối cùng, các đặc trưng được đưa qua các lớp kết nối dày đủ để phân loại.

1.3.2.4 CharCNN-LSTM

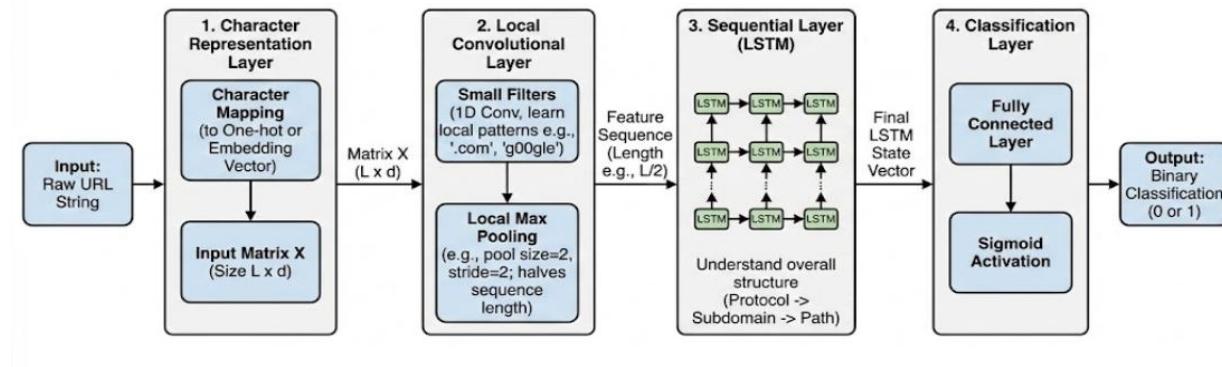
1.3.2.4.1 Tổng quan

Điểm hạn chế của mô hình đơn lẻ:

- CharCNN thuần túy: Tuy rất giỏi trong việc phát hiện các đặc trưng hình thái cục bộ, nhưng CharCNN thường sử dụng lớp Max-Pooling toàn cục, làm mất đi thông tin về vị trí và thứ tự xuất hiện của các đặc trưng trong chuỗi URL dài.
- LSTM thuần túy: Khi làm việc trực tiếp trên ký tự, chuỗi đầu vào trở nên cực kỳ dài. Điều này khiến LSTM gặp khó khăn trong việc hội tụ và tốn kém tài nguyên tính toán.

Giải pháp CharCNN-LSTM: Kiến trúc này tận dụng CNN như một bộ lọc thông minh để giảm chiều dữ liệu và trích xuất các n-gram ký tự quan trọng, sau đó đưa chuỗi đặc trưng đã được xử lý này vào LSTM để học ngữ cảnh. Đây là sự kết hợp tối ưu giữa khả năng chi tiết của CNN và khả năng xâu chuỗi của LSTM.

1.3.2.4.2 Kiến trúc chi tiết và luồng dữ liệu



Hình 1.20 Kiến trúc mô hình CNN-LSTM

- Tầng biểu diễn ký tự:
- + Đầu vào là chuỗi ký tự thô của URL.
- + Mỗi ký tự được ánh xạ thành một vector one-hot hoặc đi qua một lớp Character Embedding có thể huấn luyện được.
- + Ma trận đầu vào X có kích thước $L \times d$ với L là độ dài tối đa của URL.
- Tầng Tích chập cục bộ:
- + Sử dụng các bộ lọc có kích thước nhỏ để trượt qua chuỗi ký tự.
- + Mục đích: Học các mẫu ký tự liền kề như ".com", "http", "secure", hoặc các biến thể lừa đảo như "g00gle".
- + Khác biệt quan trọng: Ở bước này, không dùng Global Pooling để nén toàn bộ URL thành 1 vector. Thay vào đó, dùng Local Max Pooling, ví dụ: pool size = 2, stride = 2 để giảm chiều dài chuỗi đi một nửa nhưng vẫn giữ được tính tuần tự của dữ liệu.
- Tầng Tuân tự:
- + Chuỗi đặc trưng đầu ra từ tầng CNN có độ dài $L/2$ hoặc $L/3$ được đưa vào mạng LSTM.
- + LSTM sẽ duyệt qua chuỗi đặc trưng này để hiểu cấu trúc tổng thể: Giao thức nằm ở đầu - Subdomain nằm ở giữa - Path nằm ở cuối.
- Tầng phân loại:
- + Vector trạng thái cuối cùng của LSTM được đưa qua lớp Fully Connected và hàm Sigmoid để đưa ra quyết định.

1.3.2.5 CNN Hybrid

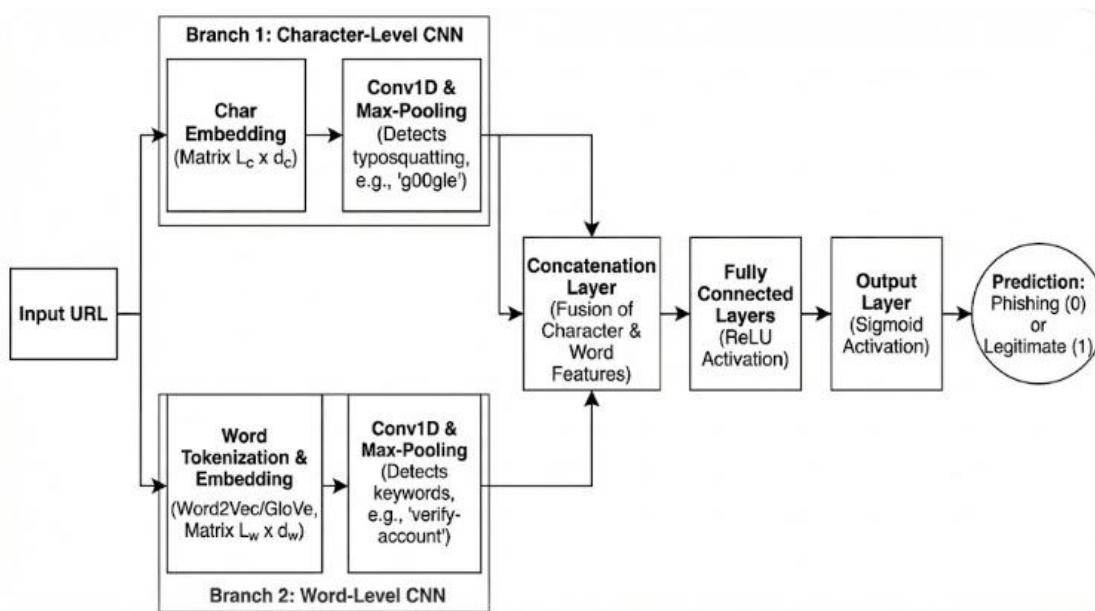
1.3.2.5.1 Tổng quan

Mô hình CNN-Hybrid được xây dựng dựa trên kiến trúc mạng nơ-ron đa nhánh với khả năng xử lý đa đầu vào. Thay vì chỉ dựa vào một dạng biểu diễn dữ liệu duy nhất, mô hình này tiếp nhận đồng thời hai luồng dữ liệu riêng biệt từ cùng một URL:

- Luồng ký tự: Nhằm nắm bắt cấu trúc hình thái, phát hiện các lỗi chính tả có ý hoặc các ký tự gây nhiễu mà không phụ thuộc vào từ điển.
 - Luồng từ vựng: Nhằm nắm bắt ngữ nghĩa của các từ khóa quan trọng, ví dụ: "login", "bank", "secure" và mối quan hệ ngữ cảnh giữa chúng.
-

Hai luồng này được xử lý song song bởi các mạng con chuyên biệt để trích xuất đặc trưng. Sau đó, các vector đặc trưng đầu ra được hợp nhất để cung cấp một cái nhìn toàn diện nhất cho lớp phân loại cuối cùng.

1.3.2.5.2 Kiến trúc chi tiết và cơ chế hoạt động



Hình 1.21 Kiến trúc mô hình CNN-Hybrid

Kiến trúc của mô hình bao gồm hai nhánh xử lý song song trước khi hợp nhất tại tầng Fully Connected:

- Nhánh 1: Xử lý đặc trưng ký tự
 - + Đầu vào: Chuỗi URL được xem như một chuỗi các ký tự liền mạch.
 - + Embedding: Sử dụng lớp Character Embedding để chuyển đổi mỗi ký tự thành một vector số học, tạo thành ma trận là $P \in [0,1]$, với L_c là độ dài chuỗi ký tự.
 - + Convolution: Áp dụng các bộ lọc tích chập kích thước nhỏ để quét qua chuỗi ký tự nhằm phát hiện các mẫu hình thái cục bộ, ví dụ: chuỗi "g00gle" thay vì "google".

- Nhánh 2: Xử lý đặc trưng Từ vựng
 - + Đầu vào: URL được tách thành các từ dựa trên các ký tự phân cách đặc biệt, như /, ., -, ?.
 - + Embedding: Sử dụng lớp Word Embedding như Word2Vec hoặc GloVe để chuyển đổi mỗi từ thành một vector ngữ nghĩa, tạo thành ma trận $L_w \times d_w$.
 - + Convolution: Áp dụng các bộ lọc tích chập để học các cụm từ n-grams mang tính chất lừa đảo, ví dụ: cụm "verify-account" hoặc "security-check".
- Tầng Hợp nhất
 - + Các đặc trưng trích xuất từ hai nhánh sau khi đi qua lớp Max-Pooling để giảm chiều được nối lại thành một vector tổng hợp duy nhất:
$$V_{final} = [V_{char} \oplus V_{word}]$$
 - + Vector này chứa đựng cả thông tin về cấu trúc vì mô và ngữ nghĩa vĩ mô.
- Tầng Phân loại
 - + Vector tổng hợp được đưa qua các lớp kết nối đầy đủ với hàm kích hoạt ReLU để học các mối quan hệ phi tuyến phức tạp.
 - + Lớp cuối cùng sử dụng hàm Sigmoid để trả về xác suất dự đoán $P \in [0,1]$, quyết định URL là Phishing hay Legitimate.

1.3.3 Các mô hình học máy hiện đại

Sự phát triển mạnh mẽ của kiến trúc Transformer đã mở ra một thế hệ mô hình học máy hiện đại, đặc biệt là các mô hình ngôn ngữ lớn. Khác với các mô hình học sâu truyền thống như CNN/LSTM vốn học cục bộ hoặc tuần tự trên chuỗi, LLM có khả năng:

- Học quan hệ ngữ cảnh dài hạn giữa các ký tự/tokens
 - Nắm bắt cấu trúc tổng thể của URL
 - Hiểu ngữ nghĩa sâu hơn nhờ cơ chế Self-Attention
-

- Hoạt động tốt trên các chuỗi bất quy tắc như URL - vốn đã kết hợp chữ, số, ký tự đặc biệt, phân cấp domain và path.

Nhờ đó, Transformer và LLM trở thành hướng tiếp cận mạnh mẽ cho bài toán phát hiện URL giả mạo, đặc biệt khi URL ngày càng chứa nhiều kỹ thuật che giấu và biến thể phức tạp.

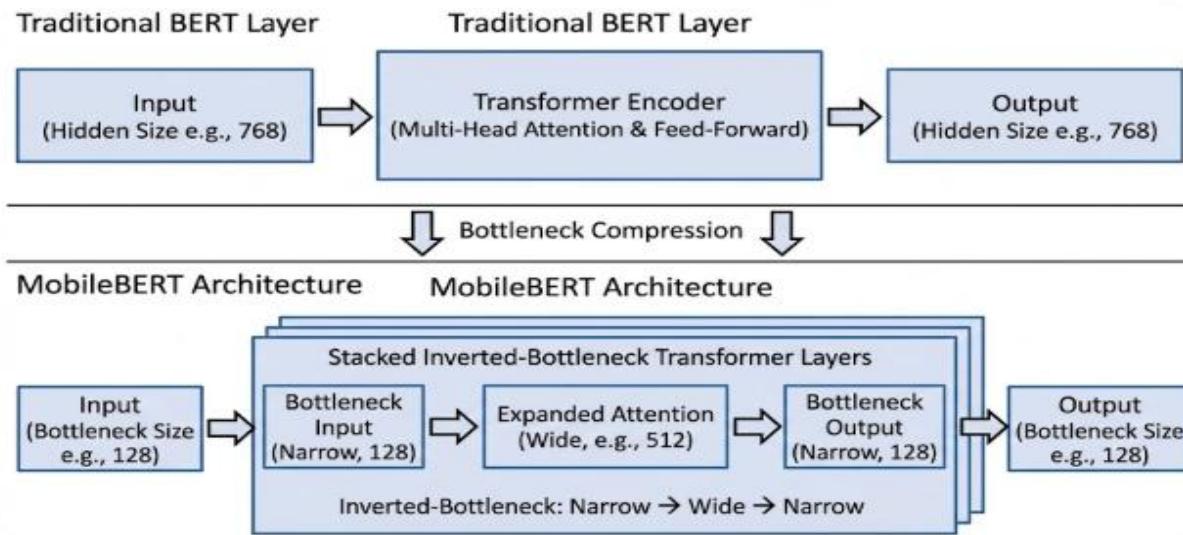
1.3.3.1 MobileBERT

1.3.3.1.1 Tổng quan

MobileBERT là một biến thể từ mô hình BERT truyền thống. Các mô hình như BERT-Base hay BERT-Large dù đạt độ chính xác rất cao trong hiểu ngữ nghĩa URL, nhưng có nhược điểm là quá nặng với hàng trăm triệu tham số và tốc độ suy diễn chậm. Điều này khiến việc tích hợp vào các hệ thống phát hiện lừa đảo thời gian thực hoặc trên các thiết bị tài nguyên hạn chế như gateway, thiết bị di động trở nên bất khả thi.

Với MobileBERT: Được đề xuất năm 2020, MobileBERT là một phiên bản nén của BERT-Large. Mục tiêu của nó là giảm kích thước mô hình và độ trễ tính toán nhưng vẫn duy trì hiệu năng tương đương.

1.3.3.1.2 Kiến trúc



Hình 1.22 So sánh sự khác nhau giữa kiến trúc mô hình BERT truyền thống và MobileBERT

Cấu trúc nút thắt cỗ chai:

- Trong BERT thông thường, kích thước của lớp ẩn rất lớn, ví dụ: 768 hoặc 1024 để chứa thông tin ngữ nghĩa phong phú.
- MobileBERT áp dụng cấu trúc nút thắt cỗ chai để nén vector biểu diễn này xuống kích thước nhỏ hơn ví dụ: 128 hoặc 256 tại đầu vào và đầu ra của mỗi lớp Transformer, giúp giảm khối lượng tính toán ma trận.

Nút thắt cỗ chai đảo ngược:

- Từ MobileNet, MobileBERT sử dụng cấu trúc "Hẹp - Rộng - Hẹp". Dữ liệu đi vào sẽ hẹp, được mở rộng ra bên trong khối Attention để xử lý chi tiết, rồi lại được nén lại ở đầu ra.
- MobileBERT sử dụng các lớp Stacked Inverted-Bottleneck thay thế cho các lớp Transformer truyền thống. Điều này cho phép mô hình nhiều lớp nhưng vẫn ít tham số tại mỗi lớp.

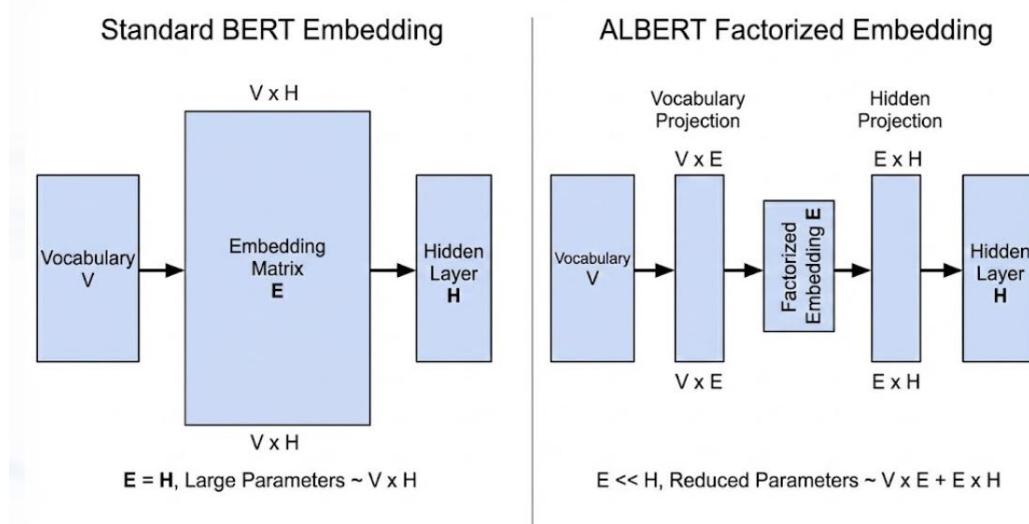
1.3.3.2 ALBERT

1.3.3.2.1 Tổng quan

Các mô hình ngôn ngữ lớn như BERT thường có hàng trăm triệu, thậm chí hàng tỷ tham số. Việc tăng quy mô mô hình thường đi đôi với việc tăng độ chính xác, nhưng lại gây ra vấn đề về bộ nhớ và thời gian huấn luyện kéo dài.

ALBERT được thiết kế để giải quyết bài toán về hiệu năng và kích thước mô hình. ALBERT có số lượng tham số ít hơn BERT-Base nhưng vẫn đạt kết quả tương đương trên các tác vụ hiểu ngôn ngữ.

1.3.3.2.2 Các kỹ thuật tối ưu



Hình 1.23 So sánh kỹ thuật tối ưu giữa BERT truyền thống và ALBERT

Tham số hóa lớp nhúng phân rã:

- Vấn đề: Trong BERT, kích thước của lớp nhúng từ vựng E và kích thước lớp ẩn H được gắn chặt với nhau $E=H$. Điều này không tối ưu vì lớp nhúng chỉ cần nắm bắt ý nghĩa từ vựng cục bộ, trong khi lớp ẩn cần không gian lớn hơn để nắm bắt ngữ cảnh toàn cục.

- Giải pháp: ALBERT tách ma trận nhúng lớn $V \times H$ thành hai ma trận nhỏ hơn: $V \times E$ và $E \times H$.
- Hiệu quả: Giảm độ phức tạp tính toán từ $O(V \times H)$ xuống $O(V \times E + E \times H)$. Với $E \ll H$, ví dụ: $E=128$, $H=768$, số lượng tham số giảm đi đáng kể.

Chia sẻ tham số xuyên lớp:

- Cơ chế: Đây là kỹ thuật giảm tham số mạnh nhất. Thay vì mỗi lớp Transformer trong tổng số 12 hoặc 24 lớp có một bộ trọng số riêng như BERT, ALBERT buộc tất cả các lớp phải dùng chung một bộ trọng số duy nhất.
- Hiệu quả: Giúp mô hình trở nên nhẹ hơn về mặt lưu trữ bộ nhớ, đồng thời đóng vai trò như một cơ chế điều chuẩn giúp ổn định quá trình huấn luyện.

Dự đoán trật tự câu:

BERT sử dụng loss Next Sentence Prediction để học mối quan hệ giữa các câu. Tuy nhiên, các nghiên cứu chỉ ra rằng NSP quá dễ và không thực sự hiệu quả trong việc hiểu sự liên kết mạch lạc. ALBERT thay thế NSP bằng SOP. Điều này buộc mô hình phải thực sự hiểu sự phụ thuộc và mạch văn logic giữa các câu thay vì chỉ so sánh chủ đề, giúp cải thiện hiệu suất trong các tác vụ hiểu ngôn ngữ.

1.3.3.3 TinyLLaMA

1.3.3.3.1 Tổng quan mô hình và huấn luyện tinh chỉnh

Fine-tuning là quá trình lấy một mô hình đã được huấn luyện trước trên một tập dữ liệu lớn và tiếp tục huấn luyện nó trên một tập dữ liệu nhỏ hơn, chuyên biệt hơn. Đối với TinyLlama với 1.1 tỷ tham số, fine-tuning cho phép mô hình học các mẫu ngôn ngữ cụ thể, tri thức chuyên ngành hoặc thay đổi phong cách phản hồi mà không cần huấn luyện lại từ đầu.

Trong bối cảnh này, phương pháp fine-tuning được áp dụng là Instruction Tuning. Mục tiêu là biến đổi TinyLlama từ một mô hình hoàn thành văn bản thành một trợ lý có khả năng tuân theo chỉ dẫn.

1.3.3.3.2 Kỹ thuật Parameter-Efficient Fine-Tuning

Vì việc fine-tuning toàn bộ tham số của TinyLlama là tốn kém và không hiệu quả.

Nguyên lý: Thay vì cập nhật tất cả 1.1 tỷ tham số, PEFT chỉ cập nhật một phần nhỏ các tham số được thêm vào hầu hết các tham số gốc và chỉ huấn luyện các lớp chuyên đổi.

Cơ sở toán học kỹ thuật Low-Rank Adaptation: Giả sử ΔW là ma trận trọng số của một lớp trong mô hình pre-trained. Khi fine-tuning, thay vì cập nhật trực tiếp W , LoRA phân rã ma trận cập nhật ΔW thành tích của hai ma trận có hạng thấp:

$$W = W_0 + \Delta W = W_0 + BA$$

Trong đó:

- $B \in \mathbb{R}^{d \times r}$ và $A \in \mathbb{R}^{r \times k}$ với $r \ll \min(d, k)$.
- r là siêu tham số, quyết định số lượng tham số cần huấn luyện.
- Trong quá trình huấn luyện, W_0 được đóng băng, chỉ có A và B được cập nhật.

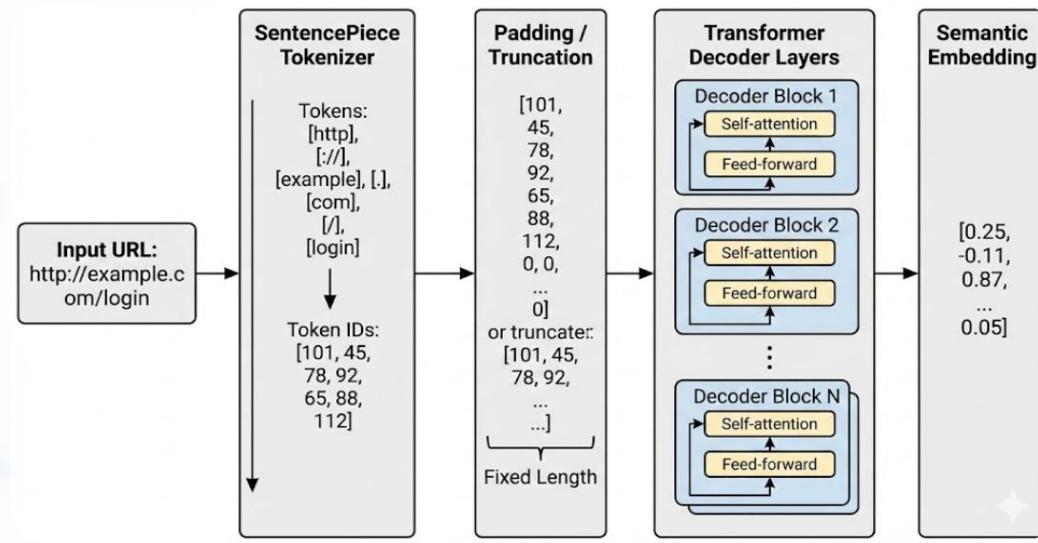
Lợi ích của LoRA:

- Giảm bộ nhớ: Số lượng tham số cần huấn luyện giảm đi hàng nghìn lần so với full fine-tuning.
- Không tăng độ trễ: Sau khi huấn luyện, trọng số BA có thể được cộng gộp lại vào W_0 , giúp tốc độ suy luận không đổi.

QLoRA

- 4-bit NormalFloat: TinyLlama được nén về độ chính xác 4-bit để giảm bộ nhớ VRAM tải model.
- Double Quantization: Lượng tử hóa cả các hằng số lượng tử hóa để tiết kiệm thêm bộ nhớ.
- Paged Optimizers: Sử dụng tính năng của NVIDIA CUDA để quản lý bộ nhớ GPU hiệu quả hơn, tránh lỗi Out of Memory.

1.3.3.3.3 Kiến trúc dữ liệu đầu vào



Hình 1.24 Mô tả quá trình xử lý dữ liệu đầu vào mô hình TinyLlama

URL được đưa qua:

- tokenizer SentencePiece → token IDs
- padding/truncation

Sau đó qua các lớp decoder Transformer để tạo embedding ngữ nghĩa.

1.4 Những tiêu chí, thuật toán cho tiền xử lý bộ dữ liệu

Trước khi huấn luyện mô hình, dữ liệu cần được tiền xử lý và đánh giá chất lượng nhằm đảm bảo tính chính xác, toàn vẹn và khả năng học hiệu quả của mô hình. Giai đoạn này bao gồm các bước làm sạch, chuẩn hóa, cân bằng, tách tập dữ liệu và đánh giá thống kê giúp giảm sai lệch, tránh rò rỉ dữ liệu và tăng độ tin cậy khi mô hình hóa.

1.4.1 Làm sạch và chuẩn hóa dữ liệu

Loại bỏ bản ghi trùng lặp hoặc lỗi: Các URL trùng hoặc chứa ký tự không hợp lệ được loại bỏ để tránh:

- Overfitting: Mô hình sẽ học quá mức vào các mẫu xuất hiện nhiều lần có thể gây thiên vị khi dự đoán.
- Rò rỉ dữ liệu: Nếu bản ghi trùng lặp xuất hiện ở cả tập huấn luyện và tập kiểm tra, kết quả đánh giá sẽ cao một cách giả tạo, không phản ánh đúng khả năng tổng quát hóa.

Dữ liệu thiếu thường xảy ra ở các đặc trưng thu thập từ bên thứ ba, ví dụ: thông tin WHOIS như ngày đăng ký tên miền, quốc gia đăng ký vì vậy cần xử lý giá trị thiếu bằng cách thay thế bằng giá trị trung bình, trung vị hoặc loại bỏ nếu tỷ lệ thiếu quá cao.

Các đặc trưng trích xuất thủ công thường có thang đo rất khác nhau. Ví dụ: đặc trưng "độ dài URL" có thể dao động từ 10 đến 100, trong khi "số lượng ký tự đặc biệt" chỉ từ 0 đến 5. Sự chênh lệch này khiến thuật toán Gradient Descent hội tụ chậm và mô hình có xu hướng ưu tiên các đặc trưng có giá trị lớn vì vậy cần chuẩn hóa dữ liệu số học bằng cách áp dụng một số kỹ thuật:

- Min-Max Scaling: Kỹ thuật này nén dữ liệu về đoạn [0, 1]. Phù hợp với các đặc trưng có phân phối không chuẩn hoặc cần giữ nguyên hình dạng phân phối gốc. Với công thức:

$$X_{norm} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

1.4.2 Cân bằng dữ liệu và chia tập huấn luyện

Trong các bài toán an ninh mạng, đặc biệt là phát hiện lừa đảo, dữ liệu thường gặp hiện tượng mất cân bằng lớp nghiêm trọng. Số lượng URL an toàn thường chiếm đa số so với URL độc hại. Nếu không xử lý, mô hình sẽ có xu hướng thiên vị lớp đa số để đạt độ chính xác cao nhưng lại bỏ sót các mối đe dọa thực sự. Đồng thời, cách chia dữ liệu không đúng cũng dẫn đến đánh giá sai lệch hiệu năng mô hình. Một số cách xử lý mất cân bằng và chia tập dữ liệu:

- Xử lý mất cân bằng dữ liệu bằng SMOTE bằng cách áp dụng SMOTE để tăng tỷ lệ mẫu phishing/legitimate để tránh việc mất cân bằng dẫn đến thiên vị khi dự đoán:
- + Cơ sở lý thuyết: SMOTE không sao chép dữ liệu cũ mà tạo ra các mẫu dữ liệu nhân tạo mới dựa trên sự tương đồng của các đặc trưng trong không gian vector.
- + Nguyên lý hoạt động: Với mỗi điểm dữ liệu x_i thuộc lớp thiểu số, thuật toán tìm k láng giềng gần nhất. Một điểm mới x_{new} sẽ được tạo ra bằng cách nội suy tuyến tính giữa x_i và một láng giềng x_{zi} được chọn ngẫu nhiên theo công thức:

$$x_{new} = x_i + \lambda \times (x_{zi} - x_i)$$

- + Trong đó λ là một vector ngẫu nhiên trong khoảng [0, 1].
- Chia tách tập dữ liệu hợp lý: Sử dụng Stratified Split theo domain để chia dữ liệu thành train / test chia theo tỷ lệ 80/20 đảm bảo không có rò rỉ domain giữa các tập.

1.4.3 Chọn lọc đặc trưng và đánh giá chất lượng bộ dữ liệu

Sau khi tiền xử lý sơ bộ, bước tiếp theo là phân tích sâu các đặc tính thống kê của dữ liệu để loại bỏ nhiễu và các đặc trưng thừa. Quy trình này bao gồm ba công đoạn chính: Phân tích tương quan, đánh giá phân bố nhãn và lọc độ biến thiên.

- Phân tích ma trận tương quan: Việc tồn tại các đặc trưng có mối tương quan tuyến tính quá mạnh sẽ làm tăng chi phí tính toán không cần thiết và gây nhiễu cho mô hình vì vậy cần xác định đặc trưng trùng lặp hoặc có mối liên hệ cao để loại bỏ:
- + Phương pháp: Sử dụng hệ số tương quan Pearson r để đo lường mối quan hệ tuyến tính giữa hai biến X và Y. Công thức:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

- + Trong đó: giá trị r nằm trong khoảng [-1, 1].

+ Thực hiện tinh gọn đặc trưng: Thiết lập ngưỡng cắt, ví dụ: 0.9. Nếu hai đặc trưng có độ tương quan $|r| > 0.9$, ví dụ: "độ dài URL" và "số lượng ký tự", ta sẽ loại bỏ một trong hai vì chúng mang lượng thông tin gần như trùng lặp.

- Trong tập dữ liệu URL, có những đặc trưng mang giá trị gần như không đổi đối với mọi mẫu dữ liệu vì vậy cần kiểm tra độ biến thiên để giúp nhận diện các đặc trưng có giá trị gần như cố định hoặc không đóng góp cho quá trình học:

+ Cơ sở toán học: Phương sai σ^2 được tính theo công thức: $\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$

+ Chiến lược: Loại bỏ các đặc trưng có phương sai thấp hơn một ngưỡng xác định trước. Điều này giúp giảm chiều dữ liệu, giúp mô hình nhẹ hơn và hội tụ nhanh hơn.

Việc áp dụng quy trình trên, tập dữ liệu đầu vào trở nên sạch, cân bằng và ổn định, đảm bảo mô hình học máy hoặc học sâu có thể hội tụ nhanh, đạt độ chính xác cao và khả năng tổng quát tốt. Đây là bước nền tảng không thể thiếu trong quy trình xây dựng hệ thống phát hiện URL phishing thông minh và đáng tin cậy.

1.5 Các tiêu chí đánh giá định lượng

Đánh giá định lượng là bước quan trọng nhằm đo lường hiệu quả và độ tin cậy của mô hình phát hiện URL phishing. Các chỉ số được sử dụng phải phản ánh toàn diện khả năng phân loại giữa hai nhãn phishing và legitimate, đặc biệt trong trường hợp dữ liệu mất cân bằng. Những tiêu chí chính bao gồm Precision, Recall, F1-score, PR-AUC, ROC-AUC và Confusion Matrix.

1.5.1 Precision

Phản ánh tỷ lệ URL được mô hình dự đoán là phishing mà thực sự đúng là phishing.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Chỉ số này thể hiện mức độ tin cậy của mô hình khi đưa ra cảnh báo. Precision cao giúp giảm cảnh báo sai, phù hợp cho các hệ thống yêu cầu độ chính xác cao.

Trong đó:

- TP: Mẫu thuộc lớp dương được dự đoán đúng.
- FP: Mẫu thuộc lớp âm bị dự đoán nhầm là dương.
- Ý nghĩa: Cho biết trong số các mẫu được mô hình dự đoán là dương tính, có bao nhiêu phần trăm là đúng.
- Ứng dụng: Phản ánh mức độ tin cậy của mô hình; phù hợp với các ứng dụng cần giảm cảnh báo sai.

1.5.2 Recall

Đo lường khả năng mô hình phát hiện đúng các URL phishing trong toàn bộ dữ liệu thật.

$$Recall = \frac{TP}{TP + FN}$$

Recall cao giúp mô hình không bỏ sót các URL độc hại, đặc biệt quan trọng trong lĩnh vực an toàn thông tin.

Trong đó:

- FN: Mẫu dương bị bỏ sót.
- Ý nghĩa: Cho biết trong tổng số mẫu thật sự dương, mô hình nhận ra được bao nhiêu phần trăm.

1.5.3 Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Ý nghĩa: Tỉ lệ tổng số dự đoán đúng trên toàn bộ mẫu.

Hạn chế: Không phản ánh đúng hiệu năng khi dữ liệu mất cân bằng.

1.5.4 F1-score

Là trung bình điều hòa giữa Precision và Recall, dùng để cân bằng giữa hai mục tiêu đối nghịch: phát hiện đủ và tránh báo sai.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Trong bài toán phishing detection, Macro-F1 được ưu tiên vì dữ liệu thường lệch nhãm.

Ý nghĩa: Cân bằng giữa độ chính xác và độ bao phủ; thích hợp khi dữ liệu mất cân bằng.

F1 cao nghĩa là mô hình vừa ít bỏ sót vừa ít cảnh báo sai.

1.5.5 PR-AUC

$$PR - AUR = \int_0^1 (Precision(Recall) d(Recall))$$

Đo diện tích dưới đường cong Precision–Recall, phản ánh hiệu năng tổng thể của mô hình với dữ liệu mất cân bằng. PR-AUC cao cho thấy mô hình duy trì Precision và Recall ổn định ở nhiều ngưỡng dự đoán khác nhau.

Ý nghĩa: đo sự cân bằng giữa độ chính xác và độ bao phủ trên toàn bộ ngưỡng phân loại.

Giá trị cao gần 1 → mô hình vừa có ít cảnh báo sai, vừa phát hiện được nhiều mẫu thật.

Đặc biệt hữu ích khi dữ liệu có tỉ lệ positive nhỏ.

1.5.6 ROC-AUC

$$ROC - AUR = \int_0^1 TPR(FPR^{-1}(x)) dx$$

Đo khả năng phân biệt giữa hai lớp thông qua quan hệ giữa True Positive Rate và False Positive Rate. ROC-AUC = 1 thể hiện mô hình phân loại hoàn hảo, tuy nhiên trong bài toán phishing, chỉ số PR-AUC thường mang ý nghĩa thực tế hơn.

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Ý nghĩa: đo khả năng mô hình phân biệt giữa hai lớp positive / negative.

Ưu điểm: Không phụ thuộc vào ngưỡng phân loại cụ thể.

Hạn chế: Khi dữ liệu mất cân bằng nặng, ROC-AUC có thể đánh giá quá lạc quan, do FPR rất nhỏ nhưng không phản ánh số lượng FP thực tế.

1.5.7 Confusion Matrix

Biểu diễn trực quan mối quan hệ giữa giá trị dự đoán và thực tế, gồm bốn thành phần:

- TP: Dự đoán đúng phishing.
- TN: Dự đoán đúng legitimate.
- FP: Dự đoán sai – URL hợp pháp bị gán là phishing.
- FN: Dự đoán sai – URL phishing bị bỏ sót.

Phân tích Confusion Matrix giúp nhận diện lỗi của mô hình và điều chỉnh ngưỡng dự đoán hoặc chiến lược cân bằng dữ liệu cho phù hợp.

1.6 Giải thích mô hình với SHAP và LIME

Trong các mô hình học máy và học sâu, việc giải thích kết quả dự đoán là cần thiết nhằm đảm bảo tính minh bạch, khả năng kiểm chứng và độ tin cậy của hệ thống phát hiện URL phishing. Hai phương pháp giải thích phổ biến và hiệu quả nhất hiện nay là SHAP và LIME. Cả

hai đều thuộc nhóm phương pháp giải thích hậu mô hình, có thể áp dụng cho bất kỳ thuật toán nào, từ XGBoost đến CNN hoặc Transformer.

1.6.1 SHAP

SHAP là phương pháp giải thích mô hình học máy dựa trên lý thuyết giá trị Shapley trong Game Theory. SHAP được xem là tiêu chuẩn hiện đại cho bài toán Model Interpretability, đặc biệt trong các mô hình phức tạp như XGBoost, Random Forest, CNN hoặc Transformer.

1.6.1.1 Nguyên lý – Lý thuyết giá trị Shapley

SHAP dựa trên khái niệm giá trị Shapley trong lý thuyết trò chơi hợp tác.

Mỗi đặc trưng được xem như một người chơi và dự đoán mô hình là lợi ích cần được phân chia cho từng người chơi.

Mục tiêu: Đo lường đóng góp trung bình của mỗi đặc trưng khi tham gia vào tất cả các tổ hợp đặc trưng có thể có.

Công thức giá trị Shapley cho đặc trưng i :

$$\Phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} [f(S \cup \{i\}) - f(S)]$$

Trong đó:

- F là tập tất cả đặc trưng
- S là tập con của đặc trưng
- $f(S)$ là dự đoán mô hình khi chỉ dùng đặc trưng S
- ϕ_i là shap value, mức đóng góp của đặc trưng i
- $|F|$ là tổng số lượng đặc trưng trong tập F
- $|S|$ là Số lượng đặc trưng trong tập con S .
- $f(S)$ là Giá trị dự đoán của mô hình khi chỉ sử dụng các đặc trưng trong tập S

- $f(S \cup \{i\}) - f(S)$ là đóng góp biên của đặc trưng i. Đây là sự chênh lệch kết quả dự đoán khi thêm đặc trưng i vào tập hợp S so với khi chưa có nó.
- $\frac{|S|!(|F|-|S|-1)!}{|F|!}$ là trọng số, đại diện cho xác suất xuất hiện của tập hợp S trong tất cả các hoán vị có thể có của các đặc trưng.

1.6.1.2 Cơ chế hoạt động trong mô hình học máy

SHAP tính toán đóng góp của từng đặc trưng vào đầu ra mô hình theo dạng:

$$f(x) = \phi_0 + \sum_{i=1}^n \phi_i$$

Trong đó:

- $f(x)$ là xác suất dự đoán URL nhãn x
- n là tổng số lượng đặc trưng
- ϕ_0 là giá trị dự đoán trung bình
- ϕ_i là giá trị đóng góp của đặc trưng i

Ý nghĩa của dấu:

- $\phi_i > 0$: đặc trưng làm tăng xác suất nhãn x
- $\phi_i < 0$: đặc trưng làm giảm xác suất nhãn x
- $\phi_i = 0$: đặc trưng không ảnh hưởng

Khi áp dụng vào bài toán URL phishing, SHAP phân tích từng đặc trưng như:

- Độ dài domain
- Entropy của hostname
- Số lượng ký tự “-”, “/”, “@”
- Số subdomain

- Tỉ lệ chữ số
- Đặc trưng mức độ ngẫu nhiên

1.6.1.3 Ứng dụng của SHAP trong phân tích và giải thích mô hình

SHAP cung cấp các công cụ trực quan hóa mạnh mẽ giúp chuyển đổi các giá trị Shapley trừu tượng thành các biểu đồ dễ hiểu, phục vụ cho việc phân tích ở cả cấp độ toàn cục và cục bộ.

1.6.1.3.1 SHAP Summary Plot

Đây là công cụ quan trọng nhất để đánh giá mức độ ảnh hưởng của các đặc trưng trên toàn bộ tập dữ liệu huấn luyện.

- Hiển thị tầm quan trọng: Các đặc trưng được sắp xếp theo trục tung y dựa trên tổng giá trị Shapley tuyệt đối trung bình. Đặc trưng ở trên cùng có ảnh hưởng lớn nhất đến mô hình.
- Phân bố tác động: Mỗi điểm trên đồ thị đại diện cho một mẫu dữ liệu.
- Ý nghĩa màu sắc:
 - + Màu đỏ: Giá trị của đặc trưng đó cao, ví dụ: Độ dài URL lớn.
 - + Màu xanh: Giá trị của đặc trưng đó thấp, ví dụ: Độ dài URL nhỏ.
- Kết luận: Giúp xác định mối tương quan: Ví dụ nếu các điểm màu đỏ có giá trị cao nằm về phía dương của trục hoành tăng khả năng Phishing, ta kết luận đặc trưng đó tỷ lệ thuận với nguy cơ lừa đảo.

1.6.1.3.2 SHAP Force Plot

Đây là công cụ dùng để giải thích quyết định của mô hình đối với từng trường hợp cụ thể, giúp minh bạch hóa lý do tại sao một URL bị gán nhãn là Phishing. Với:

- Cơ chế đối kháng: Biểu đồ hiển thị sự cân bằng giữa các lực đẩy và lực kéo của các đặc trưng.
- Màu đỏ: Các đặc trưng đẩy giá trị dự đoán tăng lên làm tăng nguy cơ là Phishing.

- Màu xanh: Các đặc trưng kéo giá trị dự đoán giảm xuống làm tăng khả năng là Legitimate.
- Kết quả: Điểm cân bằng giữa hai lực này chính là giá trị dự đoán cuối cùng $f(x)$ của mô hình.

1.6.2 LIME

LIME là phương pháp giải thích mô hình tập trung vào từng dự đoán cụ thể. Khác với SHAP giải thích toàn cục, LIME xây dựng một mô hình tuyến tính đơn giản để mô phỏng hành vi của mô hình gốc trong khu vực lân cận của mẫu cần giải thích.

1.6.2.1 Nguyên lý hoạt động

LIME tạo ra các biến thể của URL bằng cách:

- Thay đổi ký tự
- Bỏ bớt token
- Che một phần chuỗi
- Hoán đổi domain/path
- Thêm nhiều ngẫu nhiên

Mô hình gốc dự đoán các mẫu nhiễu: LIME thu thập xác suất phishing/legitimate mà mô hình gốc trả về.

Xây dựng mô hình tuyến tính cục bộ: LIME huấn luyện một mô hình tuyến tính đơn giản để mô phỏng hành vi mô hình gốc trong vùng lân cận điểm đang xét.

Đánh giá độ quan trọng đặc trưng: Trọng số của mô hình tuyến tính cục bộ biểu diễn mức độ ảnh hưởng của từng đặc trưng.

Kết quả là một bảng tập đặc trưng giải thích trực quan cho riêng URL đang được phân tích.

1.6.2.2 Cơ sở toán học

Mục tiêu của LIME là tìm ra một mô hình giải thích đơn giản g sao cho nó xấp xỉ tốt nhất mô hình gốc f tại vùng lân cận của mẫu dữ liệu x , đồng thời giữ độ phức tạp của g ở mức thấp nhất. LIME tối ưu bài toán sau:

$$\xi(x) = \arg \min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

Trong đó:

- f là hàm dự đoán của mô hình phức tạp cần giải thích
- g là mô hình đơn giản, dễ diễn giải thường là Hồi quy tuyến tính hoặc Cây quyết định nhỏ, thuộc tập hợp các mô hình giải thích G
- π_x là phân bố trọng số của các điểm nhiễu gần điểm x
- L là độ lệch giữa mô hình gốc và mô hình tuyến tính
- $\Omega(g)$ là độ phức tạp của mô hình

CHƯƠNG 2: ĐỀ XUẤT BỘ ĐẶC TRƯNG VÀ CÁC MÔ HÌNH

2.1 Thu thập và chuẩn bị tập dữ liệu URL

Bộ dữ liệu PhiUSIIL Phishing URL - UCI ID 967 được lưu trữ tại kho dữ liệu UCI Machine Learning Repository vào ngày 3 tháng 3 2024.

Dataset này gồm toàn bộ 235,795 mẫu URL, trong đó 134,850 mẫu legitimate với mã nhị phân là 1 và 100,945 mẫu url phishing với mã nhị phân là 0.

```
df["label"].value_counts()
count
label
1    134850
0    100945
dtype: int64
```

Hình 2.1 Thống kê số lượng nhãn Phishing/Legitimate

Dữ liệu được thiết kế cho bài toán phân loại nhị phân legitimate và phishing, không có giá trị thiếu và định dạng bản ghi dạng bảng với 54 đặc trưng đã trích xuất.

	FILENAME	URL	URLLength	Domain	DomainLength	IsDomainIP	TLD	URLSimilarityIndex	CharContinuationRate	TLDLegitimateProb	URLCharProb	TLDLength	NoOfSubdomain	HasObfuscation	NoOfObfuscatedChar
0	521848.txt	https://www.southbankmosaics.com	31	www.southbankmosaics.com	24	0	.com	100.0	1.000000	0.522907	0.061933	3	1	0	0
1	31372.txt	https://www.uni-mainz.de	23	www.uni-mainz.de	16	0	.de	100.0	0.666667	0.032650	0.050207	2	1	0	0
2	597387.txt	https://www.voicefmradio.co.uk	29	www.voicefmradio.co.uk	22	0	.uk	100.0	0.866667	0.028555	0.064129	2	2	0	0
3	554095.txt	https://www.sfhmjournal.com	26	www.sfhmjournal.com	19	0	.com	100.0	1.000000	0.522907	0.057606	3	1	0	0
4	151578.txt	https://www.rewildingargentina.org	33	www.rewildingargentina.org	26	0	.org	100.0	1.000000	0.079963	0.059441	3	1	0	0
5	23107.txt	https://www.globalreporting.org	30	www.globalreporting.org	23	0	.org	100.0	1.000000	0.079963	0.060614	3	1	0	0
6	23034.txt	https://www.saffronart.com	25	www.saffronart.com	18	0	.com	100.0	1.000000	0.522907	0.063549	3	1	0	0
7	696732.txt	https://www.nerdscandy.com	25	www.nerdscandy.com	18	0	.com	100.0	1.000000	0.522907	0.060486	3	1	0	0
8	739255.txt	https://www.hyderabadonline.in	29	www.hyderabadonline.in	22	0	.in	100.0	1.000000	0.005084	0.056980	2	1	0	0
9	14486.txt	https://www.aap.org	18	www.aap.org	11	0	.org	100.0	1.000000	0.079963	0.070497	3	1	0	0

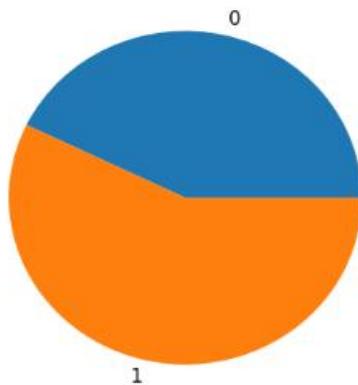
Hình 2.2 Tổng quan một số đặc trưng dataset

Mỗi bản ghi trong dataset bao gồm đường dẫn URL và nhãn phân loại: 0 là phishing, 1 là legitimate cùng với các đặc trưng như: tên miền, TLD, độ dài URL, độ dài tên miền, TLDLegitimateProb, CharContinuationRate, chỉ số tương đồng URL và nhiều biến khác.

Đánh giá tổng quan bộ dataset:

- Quy mô lớn và phân bố tương đối cân bằng giữa hai lớp khoảng 57 % hợp pháp và 43 % giả mạo giúp giảm lệch nhẫn và phù hợp cho huấn luyện mô hình phân loại nhị phân.

```
[<matplotlib.patches.Wedge at 0x7d0439faf4a0>,
 <matplotlib.patches.Wedge at 0x7d0439f739e0>],
 [Text(0.2463445108531288, 1.0720608107612082, '0'),
 Text(-0.2463442893361636, -1.0720608616638614, '1')])
```



Hình 2.3 Tỷ lệ phân bổ nhãn của dataset

- Tính đa dạng nguồn dữ liệu và đặc trưng đã trích xuất sẵn 54 đặc trưng nên giúp triển khai nhanh các mô hình ML, DL và LLM mà không cần bước trích xuất đặc trưng quá lớn từ đầu.
- Không có missing values, định dạng sẵn, thuận lợi cho pipeline tiền xử lý, giảm sai sót do dữ liệu thiếu.
- Giấy phép sử dụng rõ ràng có DOI tham chiếu “10.1016/j.cose.2023.103545” giúp trích dẫn nghiên cứu và mở rộng.

Nhờ những đặc điểm trên, bộ dataset PhiUSIIL là nguồn dữ liệu nền tảng thích hợp cho đồ án, nhóm sẽ sử dụng để thực hiện các bước tiền xử lý, trích xuất thêm đặc trưng, huấn luyện các mô hình học máy, học sâu, mô hình ngôn ngữ lớn.

2.2 Trích xuất đặc trưng và tiền xử lý dữ liệu

Trích xuất thêm đặc trưng và tiền xử lý dữ liệu để tăng khả năng học của mô hình và phù hợp với yêu cầu thực nghiệm của đề tài. Mục tiêu của giai đoạn này là chuyển các URL thành tập đặc trưng số học và ngữ nghĩa, phản ánh đầy đủ hành vi, cấu trúc và rủi ro tiềm ẩn của từng URL.

2.2.1 Trích xuất thêm đặc trưng

2.2.1.1 Nhóm đặc trưng về cấu trúc URL

Tên đặc trưng	Kiểu	Ý nghĩa
url_len, hostname_len	int	Độ dài URL/toàn host phản ánh độ phức tạp và khả năng che giấu
entropy	float	Mức hỗn loạn ký tự của URL
nb_fragments, nb_dots, nb_hyphens, nb_at,...	int	Đếm các ký tự/dấu đặc biệt hay gấp trong phishing
nb_http_token	int	Đếm “http/https” bị nhúng thêm trong path/query
nb_phish_hints	int	Đếm số “từ khóa nghi vấn”: login, admin, signin, ...
has_path_txt_extension, has_path_exe_extension	bin	Path kết thúc bằng .txt/.exe : nghi phát tán/payloader
nb_subdomain	int	Số lớp subdomain nhiều lớp hay giả mạo

has_prefix_suffix, has_abnormal_subdomain	bin	Mẫu something-something hoặc www-/ww\d bất thường
nb_www, nb_com	int	Số lần từ www/com trong các token có dấu hiệu giả mạo
char_repeat	int	Lặp ký tự liên tiếp như: aa, 111, ... độ dài 2–5
Thống kê độ dài word (word_raw_len, shortest_word_raw_len, avg_word_raw_len, ...)	float/int	Thống kê độ dài/cấu trúc token theo toàn URL/host/path

Bảng 2.1 Nhóm đặc trưng về cấu trúc URL được thêm

2.2.1.2 Nhóm đặc trưng về máy chủ URL

Tên đặc trưng	Kiểu	Ý nghĩa
has_ip	bin	Domain là địa chỉ IP thay vì tên miền
has_https	bin	Còn “không dùng HTTPS”, 0 nếu https, 1 nếu không
has_sus_tld	bin	TLD nằm trong danh sách “nghi vấn”
has_tld_in_path, has_tld_in_subdomain, tld_in_bad_position	bin	TLD xuất hiện sai vị trí tại path/subdomain
has_punycode	bin	Host dùng punycode xn--
has_port	bin	Có chỉ định port trong domain

has_short_svc	bin	Dùng dịch vụ rút gọn như: bit.ly, tinyurl, ...
has_statistical_report	bin	Domain/IP khớp với báo cáo blacklist
ratio_digits_url, ratio_digits_host	float	Tỷ lệ chữ số trong toàn URL/host

*Bảng 2.2 Nhóm đặc trưng về máy chủ URL được thêm***2.2.1.3 Nhóm Brand/Typosquatting**

Tên đặc trưng	Kiểu	Ý nghĩa ngắn gọn
has_domain_in_brand	bin	Domain thuộc danh sách thương hiệu chuẩn
has_domain_in_brand1	bin	Domain gần giống brand
has_brand_in_path, has_brand_in_subdomain	bin	Brand xuất hiện ở path/subdomain

Bảng 2.3 Nhóm Brand/Typosquatting

2.2.1.4 Nhóm đặc trưng nội dung website

Tên đặc trưng	Kiểu	Ý nghĩa
avg_sentence_len, avg_words_per_sentence, avg_paragraph_len	float	Độ đọc/độ dài câu/đoạn phản ánh nội dung thật/giả
readability_score, txt_to_html_ratio, txt_to_tag_ratio	float	Thước đo đọc hiểu và mật độ thẻ/HTML
link_density, img_density	float	Mật độ liên kết/hình ảnh spam/SEO/đánh lừa
form_complexity	float	Độ phức tạp form
unique_words_ratio, punctuation_density	float	Tỉ lệ từ duy nhất và mật độ dấu câu
has_viewport_meta, has_responsive_imgs, has_mobile_css	bin	Dấu hiệu responsive
has_popup_ad, has_comment_sections, has_rating_system, has_search_box	bin	Thành phần giao diện thường thấy ở site hợp pháp

Bảng 2.4 Nhóm đặc trưng Content/DOM đã thêm

2.2.1.5 Nhóm đặc trưng máy chủ mở rộng

Tên đặc trưng	Kiểu	Ý nghĩa
whois_reg_domain	str/bin	Domain đăng ký lấy từ WHOIS
domain_reg_len, domain_age	int	Số ngày còn hiệu lực / tuổi domain
dns_record	bin	Có bản ghi DNS hoạt động
google_index	bin	Có xuất hiện trong Google index
page_rank	int	Chỉ số xếp hạng toàn cục với fallback = -1

Bảng 2.5 Nhóm đặc trưng máy chủ mở rộng

2.2.2 Loại bỏ dữ liệu trùng lặp

Trong các bộ dữ liệu lớn như PhiUSIIL, cùng một domain hoặc URL có thể xuất hiện nhiều lần với nội dung giống nhau, đặc biệt khi hợp nhất dữ liệu từ nhiều nguồn như Kaggle, PhishTank. Việc tồn tại các bản ghi trùng lặp dẫn đến:

- Giảm tính đại diện của dữ liệu huấn luyện
- Sai lệch đánh giá dẫn đến leakage URL trùng xuất hiện ở cả tập train và test
- Tăng kích thước và thời gian huấn luyện không cần thiết

2.2.3 Xử lý mất bằng dữ liệu

Cân bằng đặc trưng số học bằng thuật toán SMOTE tùy chỉnh:

```
def smote_like(X, n_new, k=5, seed=1234):
    rng = np.random.default_rng(seed)
    n = X.shape[0]; k = min(k, n)
```

```

if n < 2: raise ValueError("Thiếu data phishing để SMOTE (>=2).")
nbrs = NearestNeighbors(n_neighbors=k).fit(X)
out = []
for _ in range(n_new):
    i = rng.integers(0, n); x = X[i]
    Ns = nbrs.kneighbors(x.reshape(1,-1), return_distance=False)[0]
    j = int(rng.choice(Ns[1:] if len(Ns)>1 else Ns))
    lam = rng.random()
    out.append(x + lam*(X[j]-x))
return np.vstack(out)

```

- Đối với các cột dữ liệu số, ví dụ: độ dài URL, số lượng dấu chấm, số lượng ký tự đặc biệt..., cần cài đặt thuật toán SMOTE dựa trên nguyên lý lảng giềng gần nhất k-Nearest Neighbors:
- + Cơ chế hoạt động: Với mỗi điểm dữ liệu x thuộc lớp thiểu số Phishing, thuật toán tìm k lảng giềng gần nhất k=5. Một điểm dữ liệu mới được tạo ra bằng cách nội suy tuyến tính:

$$x_{new} = x_i + \lambda \times (x_{zi} - x_i)$$

- + Hậu xử lý: Điểm đặc biệt trong cài đặt này là việc áp dụng các ràng buộc thực tế sau khi sinh dữ liệu:
 - Làm tròn số nguyên: Các đặc trưng đếm được làm tròn về số nguyên để đảm bảo tính logic.
 - Ràng buộc không âm: Đảm bảo không có giá trị âm được sinh ra cho các đặc trưng vật lý.

Sinh dữ liệu chuỗi URL giả lập:

```

def gen_url():
    brands = ["paypal", "apple", "microsoft", "facebook", "google", "banksecure", "account-verify"]
    actions = ["login", "confirm", "update", "secure", "verify", "reset", "billing"]
    tld = random.choice(["invalid", "test", "example"])
    subs = []
    for _ in range(random.randint(2,4)):
        token = ''.join(random.choices(string.ascii_lowercase+string.digits,

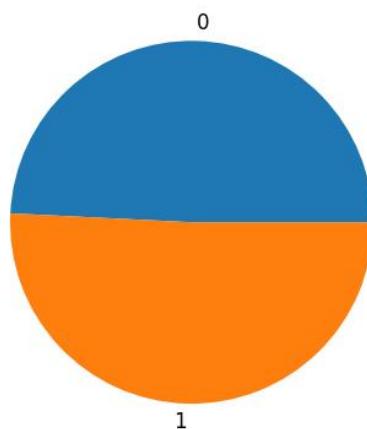
```

```

k=random.randint(5,10)))
    if random.random()<0.5: token += "-"+"".join(random.choices(string.ascii_lowercase,
k=random.randint(2,5)))
        subs.append(token)
brand = random.choice(brands)
host = ".".join(subs+[brand,tld])
if random.random()<0.25:
    host = f"{"}.join(random.choices(string.ascii_lowercase+string.digits,k=6))}@{host}"
path = "/" +random.choice(actions)+"/"+uuid.uuid4().hex[:16]
q = f"?session={uuid.uuid4().hex[:8]} &id={random.randint(10000,99999)}"
return f"{random.choice(['http','https'])://{host}{path}{q}}"

```

- Chiến lược: Các URL mới được tạo ra bằng cách lắp ghép ngẫu nhiên các thành phần thường thấy trong các cuộc tấn công lừa đảo, bao gồm:
 - + Thương hiệu giả mạo: paypal, apple, google, banksecure...
 - + Hành động kích động: login, verify, update, secure...
 - + Cấu trúc lừa đảo: Chèn thêm các token ngẫu nhiên, sử dụng subdomain dài hoặc chèn ký tự đặc biệt như @ hoặc -.
 - Mục đích: Việc này giúp lắp đầy cột url cho các dòng dữ liệu số vừa được sinh ra bởi SMOTE, đảm bảo dataset sau khi cân bằng có đầy đủ cả thông tin số học và văn bản.
Sau quá trình xử lý, tập dữ liệu đạt trạng thái khá cân bằng giữa hai lớp nhãn, sẵn sàng cho giai đoạn huấn luyện mô hình.



Hình 2.4 Tỷ lệ phân bổ nhãn trong dataset cân bằng

Ngoài ra cần loại bỏ một số URL lỗi , không gắn được nhãn

df[df['label'].isna()]		
	url	label
142670	http://www.tomshardware.com/reviews/gigabit-et...	NaN
142804	http://vim.wikia.com/wiki/Copy	NaN
143038	http://www.biologyjunction.com/Viruses	NaN
143173	http://www.tomshardware.com/reviews/gpu-hierarchy	NaN
143443	http://www.tomsitpro.com/articles/next-generat...	NaN
...
485177	https://get-workspace-resources-jsz.s3.ap-sout...	NaN
485797	https://international-2022-ifv.s3.us-east-1.am...	NaN
485920	https://managed-sync-2021.s3.us-west-1.amazona...	NaN
486292	https://now-streamlined-files-40y.s3.us-west-1...	NaN
486619	https://streamlined-analytics-files-jk.s3.eu-w...	NaN

308 rows × 2 columns

Hình 2.5 Một số URL không gắn được nhãn

```
df["label"].value_counts()
```

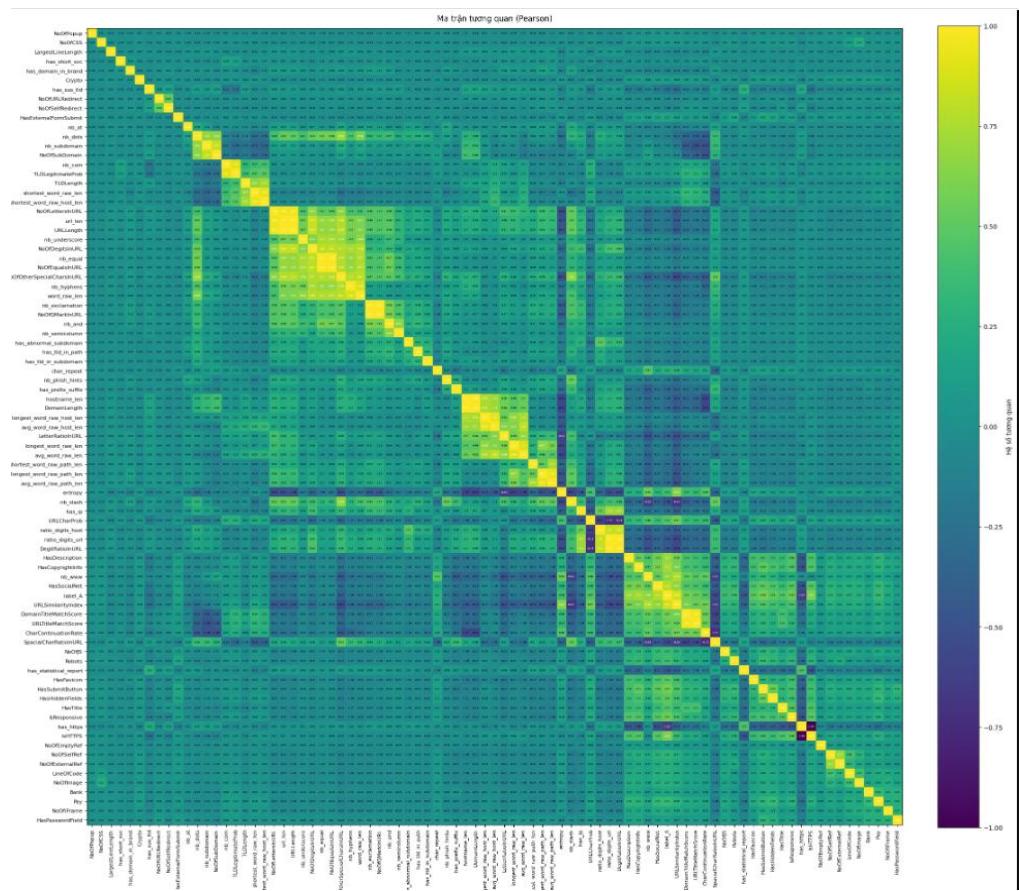
count	
label	count
1	247326
0	239718

dtype: int64

Hình 2.6 Số lượng nhãn Phishing và Legitimate

2.2.4 Tinh gọn đặc trưng bằng ma trận tương quan

Ma trận tương quan ban đầu của dataset có khá nhiều cặp đặc trưng tương đồng có chỉ số corr cao.



Hình 2.7 Ma trận tương quan của dataset trước khi tinh gọn đặc trưng có tương quan cao

Ta phân tích ma trận tương quan và tính chỉ số corr để tinh gọn đặc trưng cho dataset:

```
# ===== LOẠI THEO ALIAS (quy tắc) nếu thật sự trùng mạnh (|corr|>0.9) =====
# Tính corr nhanh chỉ cho các cột có trong alias map
if len(X.columns) > 1:
    corr = X.corr().abs()
else:
    corr = pd.DataFrame(np.eye(len(X.columns))), index=X.columns, columns=X.columns)

dropped_by_alias = []
for keep, alist in ALIAS_PREF.items():
```

```

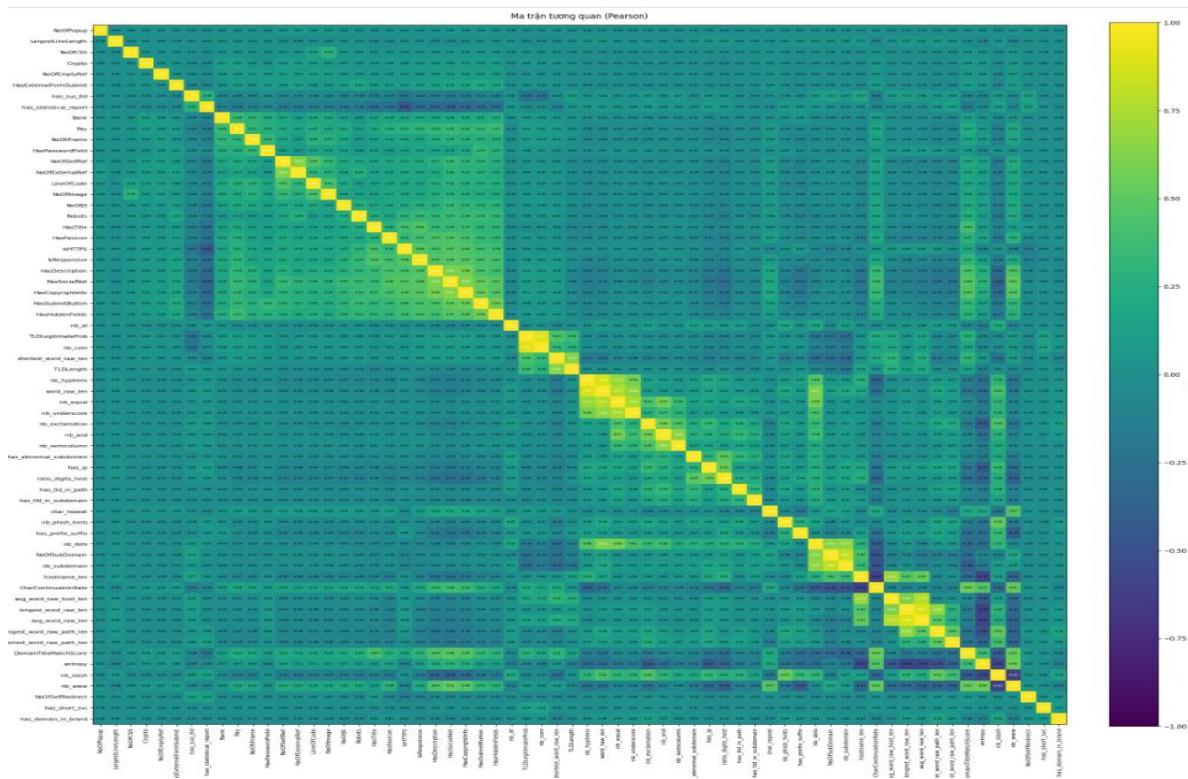
if keep not in X.columns:
    continue
for a in alist:
    if a in X.columns:
        if corr.loc[keep, a] >= 0.9:
            X.drop(columns=[a], inplace=True)
            dropped_by_alias.append(a)

Top 30 cặp |corr| cao:
      level_0           level_1      corr
      hostname_len     DomainLength  0.999831
      url_len          URLLength    0.999528
      ratio_digits_url DigitRatioInURL 0.999409
      nb_equal          NoOfEqualsInURL 0.998651
      nb_exclamation   NoOfQMarkInURL 0.998500
      has_https         IsHTTPS      0.993840
      DomainTitleMatchScore URLTitleMatchScore 0.963391

```

Hình 2.8 Một số đặc trưng có tương quan cao

Ta loại bỏ những đặc trưng có tương quan cao, $\text{corr} > 0.9$, sau khi tinh gọn đặc trưng ta được ma trận tương quan:



Hình 2.9 Ma trận tương quan sau khi tinh gọn đặc trưng

2.2.5 Huấn luyện Word2Vec và embedding URL

2.2.5.1 Phân chia dữ liệu

Tập dữ liệu đầu vào D gồm các URL và nhãn tương ứng được phân chia thành tập huấn luyện và tập kiểm tra bằng kỹ thuật phân chia train-test có phân tầng.

$$D \rightarrow (D_{train}, D_{test})$$

Cụ thể, dữ liệu URL X và nhãn Y được phân chia với tỉ lệ kiểm tra là t, ví dụ: t=0.2:

$$(X_{train}, X_{test}, Y_{train}, Y_{test}) = \text{StratifiedSplit}(X, Y, t)$$

BÙI HỮU TRÍ D21COAT01-N

ĐỊNH QUỐC TOÀN D21CQAT01-N

2.2.5.2 Tiền xử lý và Token hóa

Mỗi URL thô được chuyển đổi thành một chuỗi các token có ý nghĩa, $T_i = \{t_{i,1}, t_{i,2}, \dots, t_{i,L_i}\}$ thông qua hàm tiền xử lý Φ :

$$T_i = \phi(x_i)$$

Hàm $\phi(x)$ bao gồm các bước:

Chuẩn hóa: Chuyển URL về chữ thường và giải mã nếu cần

Phân tích cấu trúc: Sử dụng cấu trúc URL như: Scheme, Netloc, Path, Query để tách các thành phần chính thành token, ví dụ: http, www, example, com, path, param=value:

$$T_i = \{scheme\} \cup Netloc_Parts \cup Path_Parts \cup Query_Parts$$

Lọc: Loại bỏ các ký tự không hợp lệ/đặc biệt khỏi các token đã.

Tập hợp tất cả các chuỗi token từ tập huấn luyện được gọi là Corpus huấn luyện C_{train} :

$$C_{train} \{T_i | x_i \in X_{train}\}$$

2.2.5.3 Nhúng URL

Quá trình nhúng được thực hiện qua hai bước: Huấn luyện mô hình Word2Vec và Tính toán Vector Đại diện URL:

- Huấn luyện mô hình Word2Vec
- + Một mô hình Word2Vec được huấn luyện trên Corpus Huấn luyện để học các vector biểu diễn từ cho từng token. Phương pháp Skip-gram được sử dụng.

$$M_{w2v} = Word2Vec(C_{train}; d, w, m)$$

- + Trong đó:
- d: Kích thước vector nhúng , d=50.
- w: Kích thước cửa sổ ngữ cảnh, w=5.
- m: Nguồn tần suất tối thiểu, m=1.

Mô hình học một hàm ánh xạ $Vec : V \rightarrow R^d$, trong đó V là từ vựng của các token và $Vec(t)$ là vector d-chiều của token t.

- Vector đại diện URL:
- + Mỗi URL x_i được chuyển thành một vector đại diện duy nhất $v_i \in R^d$ bằng hàm URL2Vec(Ψ).
- + Hàm Ψ tính toán vector trung bình của tất cả các vector nhúng của các token hợp lệ trong chuỗi tương ứng:

$$v_i = \psi(T_i, M_{w2v}) = \frac{1}{|T_i^{valid}|} \sum_{t \in T_i^{valid}} Vec(t)$$

- + Trong đó: $\{V_i \mid x_i \in X_{test}\}$ là tập hợp các token trong URL có mặt trong từ vựng của mô hình Word2Vec. Nếu T_i^{valid} rỗng, v_i là một vector zero $0 \in R^d$
- + Cuối cùng, dữ liệu được chuyển đổi thành vector đại diện mật độ:

$$X_{train_embed} = \{v_i \mid x_i \in X_{train}\}$$

$$X_{test_embed} = \{v_i \mid x_i \in X_{test}\}$$

2.2.6 Huấn luyện Char2Vec và embedding URL

2.2.6.1 Phân chia dữ liệu

Tập dữ liệu đầu vào D gồm các URL $x_i \in D$ và nhãn $y_i \in \{0,1\}$ tương ứng được phân chia thành tập huấn luyện D_{train} và D_{test} tập kiểm tra bằng kỹ thuật phân chia train-test có phân tầng.

$$D \rightarrow (D_{train}, D_{test})$$

Cụ thể, dữ liệu URL X và nhãn Y được phân chia với tỉ lệ kiểm tra là t, ví dụ: t=0.2:

$$(X_{train}, X_{test}, Y_{train}, Y_{test}) = StratifiedSplit(X, Y, t)$$

2.2.6.2 Chuẩn hóa URL

Mỗi URL x_i được chuẩn hóa bằng hàm `clean_url()` trước khi token hóa:

- Chuyển đổi chữ thường: Chuyển đổi tất cả ký tự cả ký tự kí tự thành chữ thường.
- Loại bỏ khoảng trắng: Loại bỏ tất cả các ký tự khoảng trắng thừa.

$$x_i = clean_url(x_i)$$

2.2.6.3 Token hóa và xây dựng từ vựng

Quá trình token hóa được thực hiện ở cấp độ ký tự:

- Xây dựng Từ vựng: Một bộ từ vựng V được xây dựng từ các ký tự duy nhất có trong tập URL huấn luyện đã được chuẩn hóa.

$$V = \{c_1, c_2, \dots, c_{|V|}\}$$

- + Mỗi ký tự trong V được ánh xạ với một chỉ số nguyên duy nhất $k \in \{1, 2, \dots, |V| - 1\}$, sử dụng Tokenizer của Keras. Một chỉ số Padding/Out-of-Vocabulary giữ lại cho các mục đích đệm và xử lý ký tự không xác định.

$$Map : V \rightarrow \{1, \dots, |V| - 1\}$$

- + Kích thước từ vựng được xác định là $|V| = 106$.
- Chuyển đổi thành chuỗi số nguyên: Mỗi URL chuẩn hóa x_i được chuyển thành một chuỗi số nguyên $S_i = \{k_{i,1}, k_{i,2}, \dots, k_{i,L_i}\}$ thông qua ánh xạ Map.

$$S_i = \text{text_to_sequences}(x_i)$$

2.2.6.4 Đệm chuỗi

Các chuỗi số nguyên S_i có độ dài khác nhau được chuyển đổi thành các vector có độ dài cố định $L_{\max} = 200$ bằng kỹ thuật đệm chuỗi.

$$V_i = \text{pad_sequences}(S_i; \text{max len} = L_{\max}; \text{padding} = 'post', \text{truncating} = 'post')$$

Trong đó:

- V_i là vector đại diện cuối cùng của URL x_i .
- Post-padding: Các vị trí thừa được điền bằng chỉ số 0 ở cuối chuỗi.
- Post-truncating: Chuỗi dài hơn L_{\max} sẽ bị cắt bỏ ở cuối.

Dữ liệu cuối cùng được chuẩn bị cho mô hình là:

$$\mathbf{X}_{\text{train}}_{\text{embed}} = \{V_i \mid x_i \in X_{\text{train}}\}, \text{shape} = (392807, 200)$$

$$\mathbf{X}_{\text{test}}_{\text{embed}} = \{V_i \mid x_i \in X_{\text{test}}\}, \text{shape} = (98202, 200)$$

2.3 Cấu hình, huấn luyện và đánh giá các mô hình

2.3.1 Machine learning

2.3.1.1 XGBoost

Cấu hình:

```
xgb = XGBClassifier(
    objective='binary:logistic',
    learning_rate=0.08,
    n_estimators=300,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_lambda=1.5,
    reg_alpha=0.5,
    eval_metric=['logloss', 'error', 'aucpr'],
    random_state=42,
    n_jobs=-1
)
```

Trong đó:

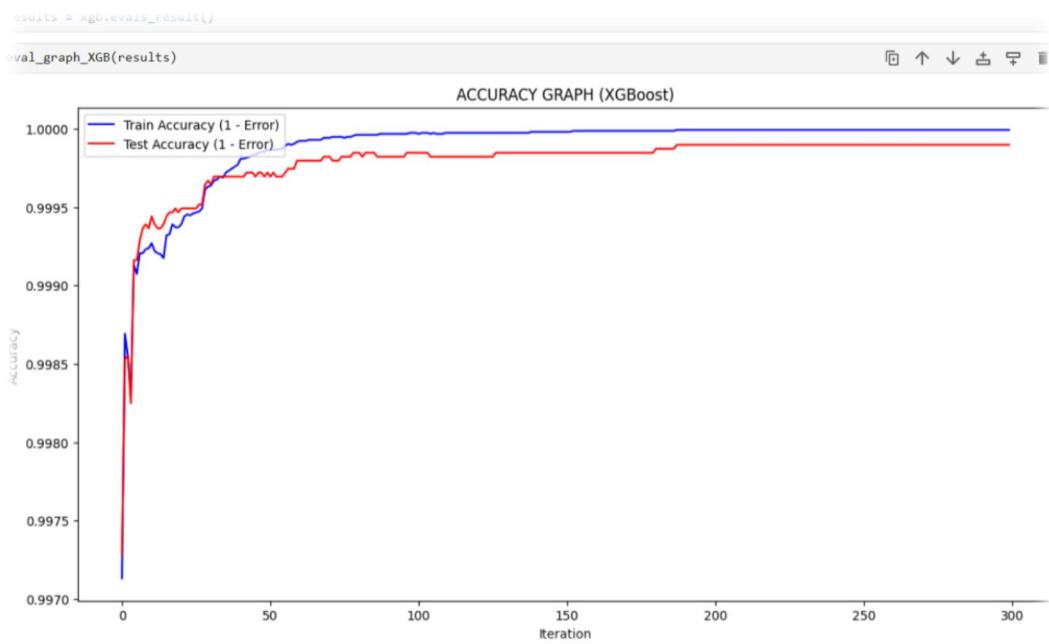
- `objective='binary:logistic'`: Xác định loại bài toán là phân loại nhị phân, đầu ra là xác suất (0,1) qua hàm sigmoid
- `learning_rate=0.08`: Tốc độ học hay còn gọi là eta. Giá trị nhỏ giúp mô hình học chậm nhưng ổn định hơn, giảm nguy cơ overfitting
- `n_estimators=300`: Số lượng cây quyết định được huấn luyện. Mô hình càng nhiều cây thì càng mạnh, nhưng có thể tốn thời gian và dễ overfit nếu quá cao
- `max_depth=6`: Độ sâu tối đa của mỗi cây. Giới hạn này giúp kiểm soát độ phức tạp của mô hình càng sâu, mô hình càng có khả năng học quan hệ phi tuyến nhưng dễ overfit
- `subsample=0.8`: Tỷ lệ mẫu được chọn ngẫu nhiên để huấn luyện mỗi cây. Giúp tăng khả năng tổng quát hóa và giảm overfitting
- `colsample_bytree=0.8`: Tỷ lệ đặc trưng được lấy ngẫu nhiên khi xây mỗi cây. Tương tự như Random Forest, giúp giảm tương quan giữa các cây

- reg_lambda=1.5: Hệ số regularization L2, giúp phạt các trọng số lớn, làm mô hình mượt mà hơn
- reg_alpha=0.5: Hệ số regularization L1, giúp mô hình chọn lọc đặc trưng giúp loại bỏ đặc trưng không quan trọng
- eval_metric=['logloss', 'error', 'aucpr']: Các chỉ số đánh giá trong quá trình huấn luyện:
- logloss: mất mát logistic để đo độ chính xác xác suất
- error: tỷ lệ sai
- aucpr: diện tích dưới đường cong Precision-Recall, phù hợp với dữ liệu mất cân bằng
- random_state=42: Có định seed để kết quả có thể tái lập được
- n_jobs=-1: Sử dụng tất cả các lõi CPU khả dụng để tăng tốc huấn luyện.

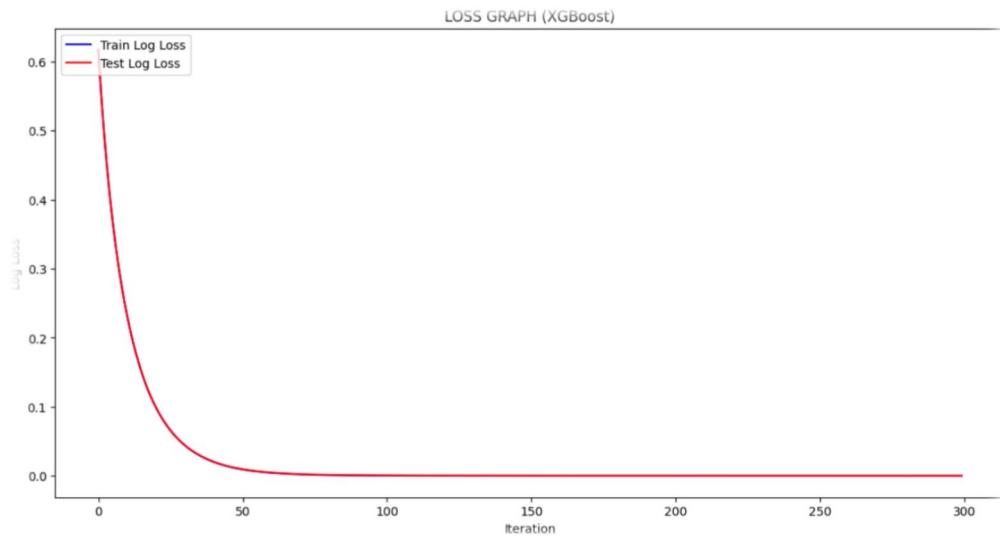
Huấn luyện mô hình với dữ liệu đầu vào là đặc trưng số:

```
eval_set = [(X_train1, Y_train1), (X_test1, Y_test1)]  
xgb.fit(  
    X_train1, Y_train1,  
    eval_set=eval_set,  
    verbose=50)
```

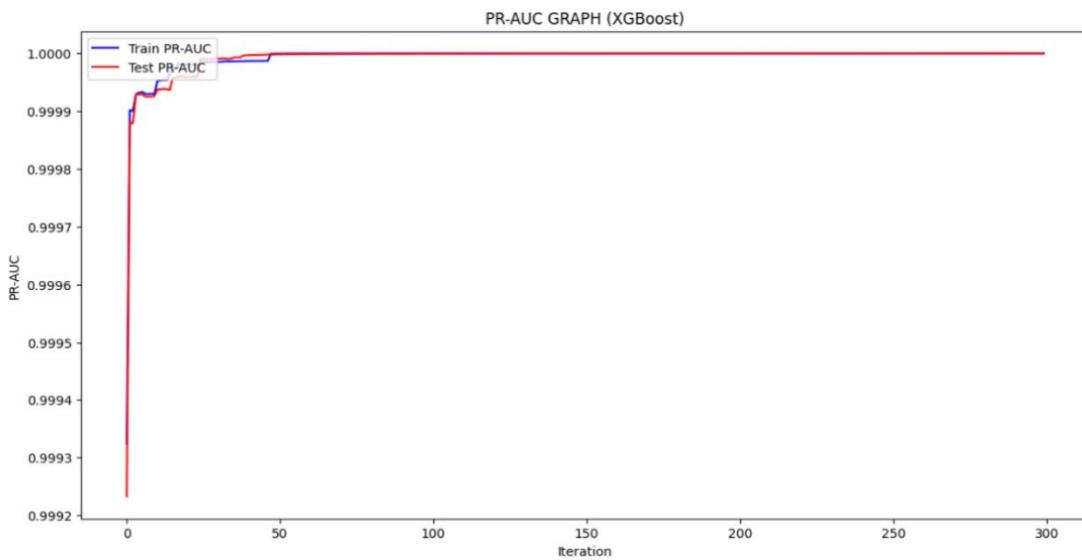
Sau khi load dữ liệu train/test đặc trưng số đã lưu sẵn trước đó nhóm tiến hành huấn luyện mô hình XGBoost và tính toán kết quả huấn luyện:



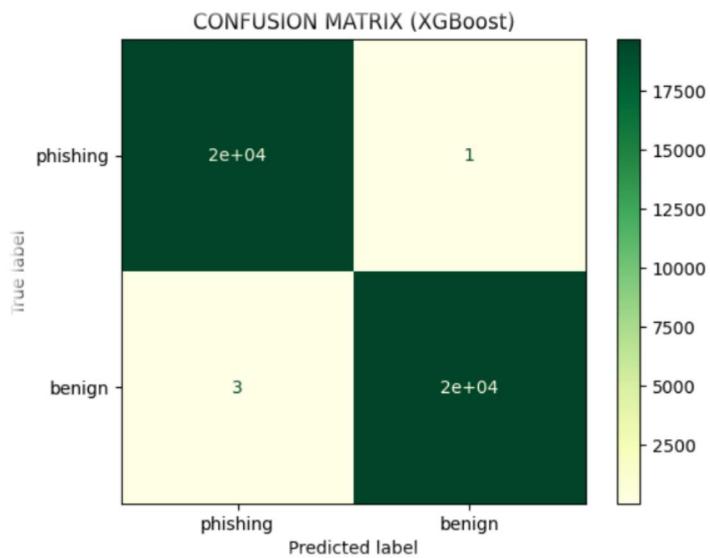
Hình 2.10 Accuracy Graph của model XGBoost numerical



Hình 2.11 Loss graph của model XGBoost numerical



Hình 2.12 PR-AUC graph của model XGBoost numerical



Hình 2.13 Confusion matrix của model XGBoost numerical

Đánh giá mô hình XGBoost numerical với các tiêu chí precision, recall:

```
Classification Report:
  precision    recall   f1-score   support
  phishing      1.00      1.00      1.00     19699
  benign        1.00      1.00      1.00     19699

  accuracy                           1.00     39398
  macro avg       1.00      1.00      1.00     39398
  weighted avg    1.00      1.00      1.00     39398

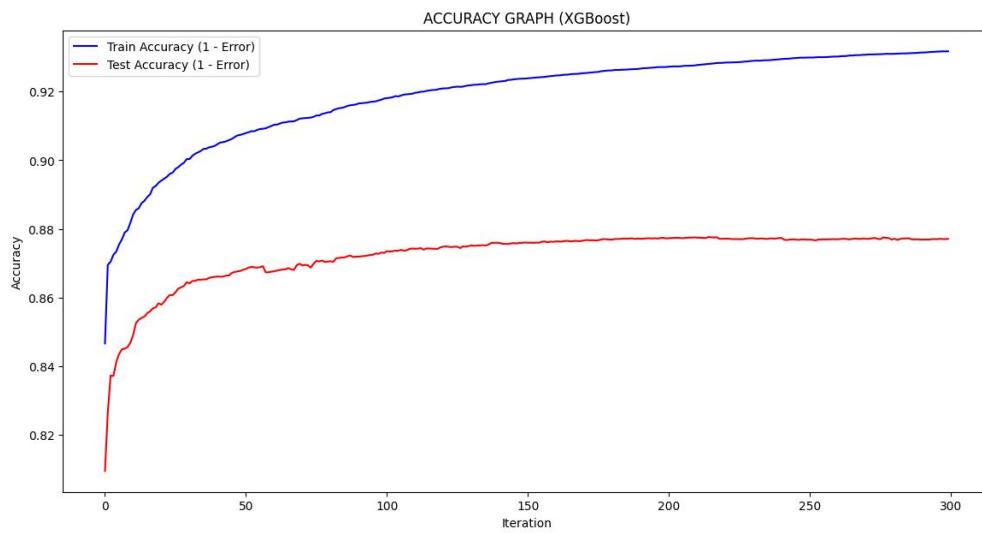
  PR-AUC (Test): 1.0000
  ROC-AUC (Test): 1.0000
```

Hình 2.14 Đánh giá mô hình XGBoost numerical

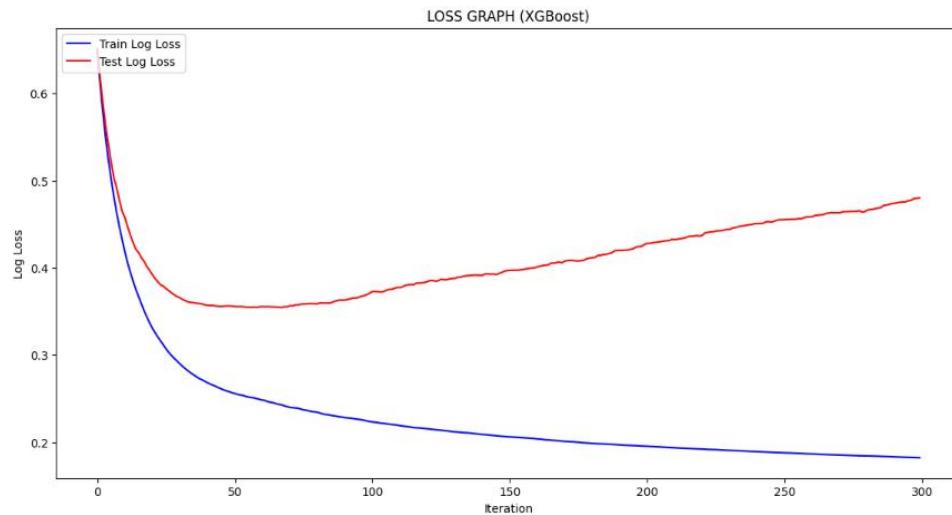
Huấn luyện mô hình với đầu vào là URL embedding:

```
eval_set = [(X_train2, Y_train2), (X_test2, Y_test2)]
xgb.fit(
    X_train2, Y_train2,
    eval_set=eval_set,
    verbose=50)
```

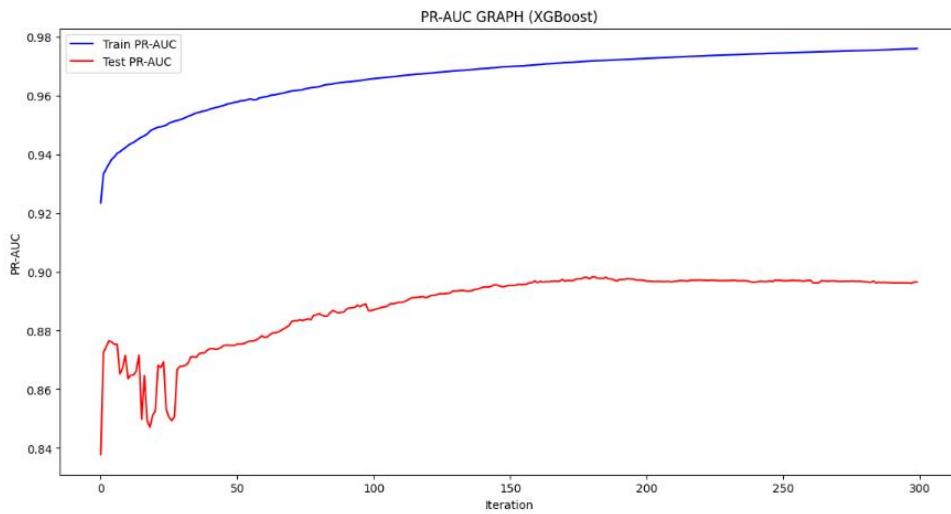
Sau khi load dữ liệu train/test URL Embedding đã lưu sẵn trước đó ta tiến hành huấn luyện mô hình XGBoost và tính toán kết quả huấn luyện.



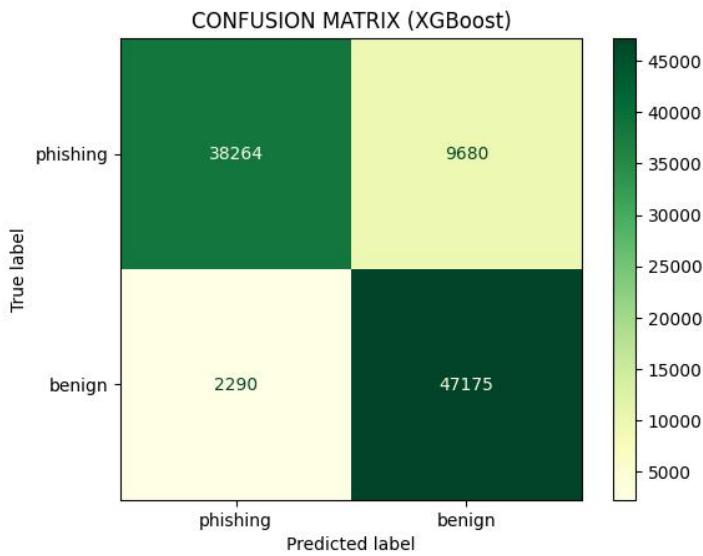
Hình 2.15 Accuracy graph của model XGBoost embedding



Hình 2.16: Loss graph của model XGBoost embedding



Hình 2.17 PR-AUC của model XGBoost embedding



Hình 2.18 Confusion matrix của model XGBoost embedding

Đánh giá model XGBoost embedding với các tiêu chí precision, recall, f1-score:

Classification Report:				
	precision	recall	f1-score	support
phishing	0.94	0.80	0.86	47944
benign	0.83	0.95	0.89	49465
accuracy			0.88	97409
macro avg	0.89	0.88	0.88	97409
weighted avg	0.89	0.88	0.88	97409

PR-AUC (Test): 0.9015
ROC-AUC (Test): 0.9232

Hình 2.19 Đánh giá model XGBoost embedding

2.3.1.2 Random Forest

Cấu hình:

```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(
    n_estimators=100,
    max_depth=15,
    min_samples_split=5,
    min_samples_leaf=1,
    max_features='sqrt',
    bootstrap=True,
    n_jobs=-1,
    random_state=42)
```

Trong đó:

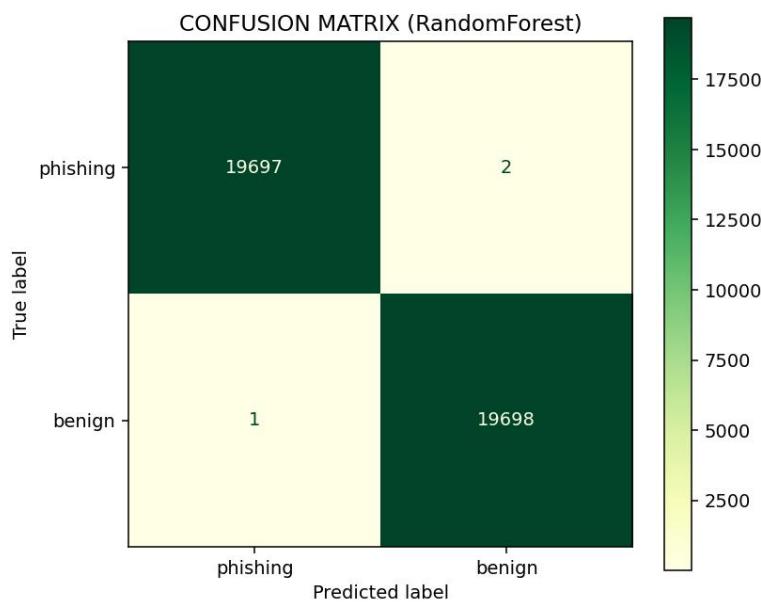
- n_estimators=100: Số lượng cây quyết định trong rừng. Nhiều cây hơn thường giúp mô hình ổn định và chính xác hơn, nhưng tốn nhiều thời gian huấn luyện hơn.
- max_depth=15: Độ sâu tối đa của mỗi cây. Giới hạn này giúp kiểm soát độ phức tạp của từng cây — cây sâu có thể học tốt hơn nhưng dễ bị overfitting nếu quá lớn.

- min_samples_split=5: Số mẫu tối thiểu cần có để chia một nút. Giá trị lớn hơn giúp cây bớt phân tách quá sâu, giúp giảm overfitting.
- min_samples_leaf=1: Số mẫu tối thiểu trong mỗi lá. Giá trị nhỏ cho phép mô hình học chi tiết hơn, nhưng có thể dẫn đến overfitting nếu dữ liệu nhiễu.
- max_features='sqrt': Số lượng đặc trưng được chọn ngẫu nhiên khi xem xét tách một nút.
- 'sqrt' là lựa chọn mặc định cho phân loại, giúp giảm tương quan giữa các cây, tăng tính đa dạng của mô hình.
- bootstrap=True: Cho phép lấy mẫu ngẫu nhiên có hoán lại để huấn luyện từng cây, tức mỗi cây học trên một tập con khác nhau của dữ liệu. Điều này giúp mô hình tổng quát hóa tốt hơn.
- n_jobs=-1: Sử dụng tất cả các lõi CPU khả dụng để huấn luyện song song, giúp tăng tốc đáng kể quá trình huấn luyện.
- random_state=42: Có định seed ngẫu nhiên để đảm bảo tính tái lập của kết quả.

Huấn luyện mô hình với dữ liệu đầu vào là đặc trưng số:

```
RF.fit(X_train1, Y_train1)
```

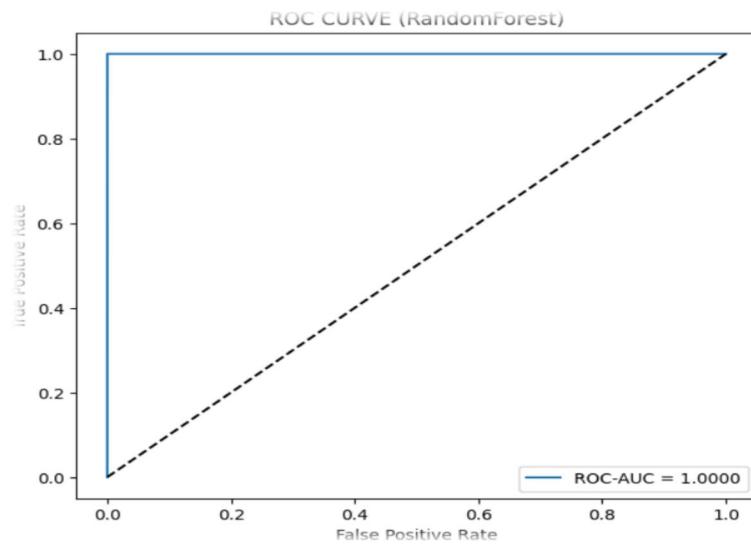
Sau khi load dữ liệu train/test đặc trưng số đã lưu sẵn trước đó ta tiến hành huấn luyện mô hình Random Forest và tính toán kết quả huấn luyện

*Hình 2.20 Confusion matrix của model Random Forest numerical*

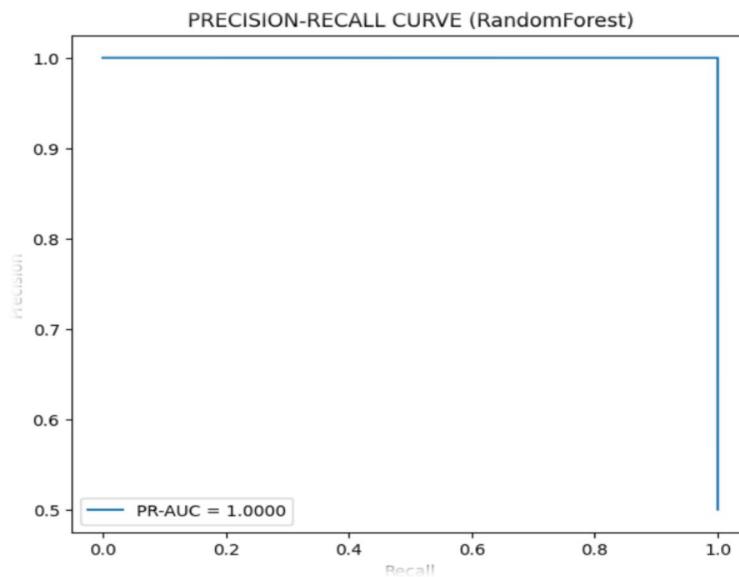
Đánh giá model Random Forest numerical với các tiêu chí precision, f1-score, recall:

Classification Report:					
	precision	recall	f1-score	support	
phishing	0.9999	0.9999	0.9999	19699	
benign	0.9999	0.9999	0.9999	19699	
				0.9999	39398
accuracy					39398
macro avg	0.9999	0.9999	0.9999	39398	
weighted avg	0.9999	0.9999	0.9999	39398	

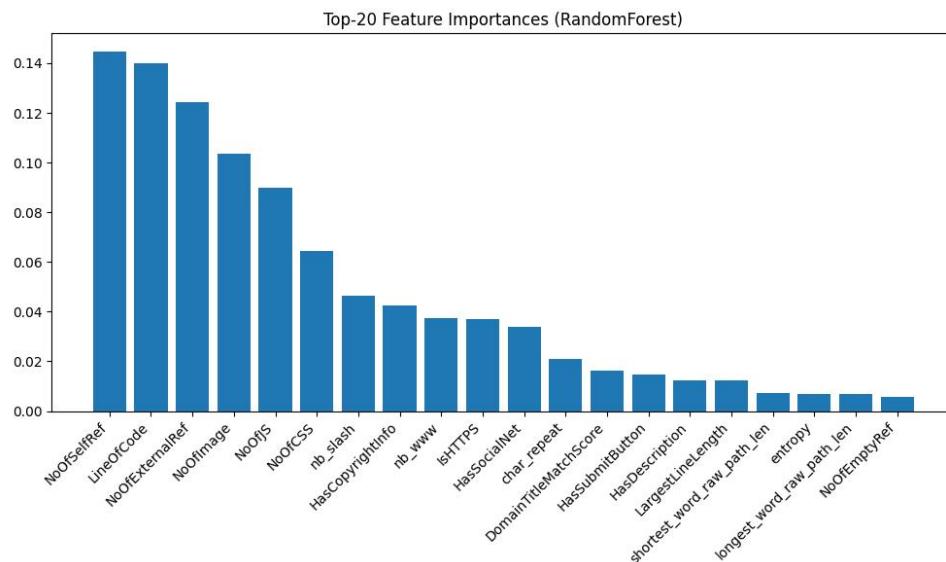
Hình 2.21 Đánh giá models Random Forest numerical



Hình 2.22 ROC-Curve của model Random Forest numerical



Hình 2.23 Precision/Recall Curve của model Random Forest numerical

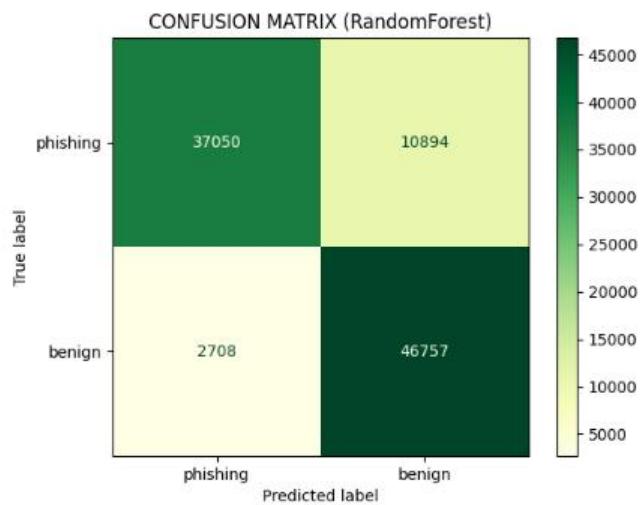


Hình 2.24 Top 20 đặc trưng quan trọng trong huấn luyện Random Forest model

Huấn luyện mô hình với dữ liệu đầu vào là URL Embedding:

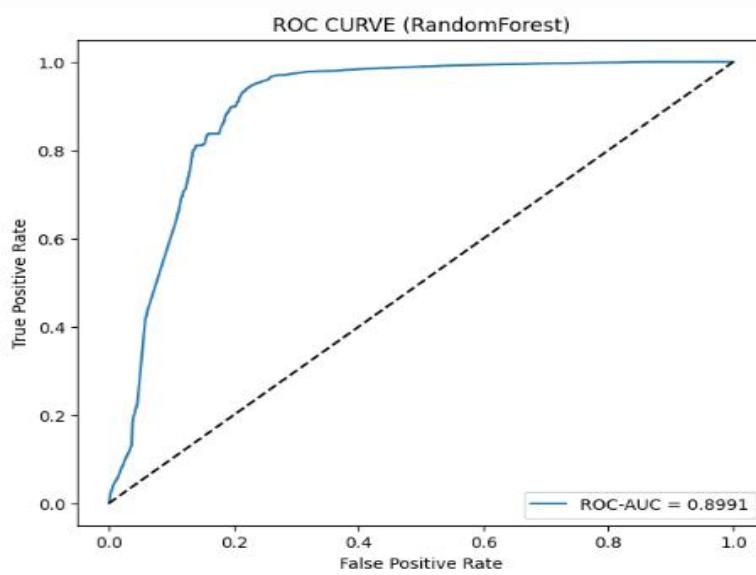
```
RF.fit(X_train2, Y_train2)
```

Sau khi load dữ liệu train/test URL Embedding đã lưu sẵn trước đó ta tiến hành huấn luyện mô hình Random Forest và tính toán kết quả huấn luyện

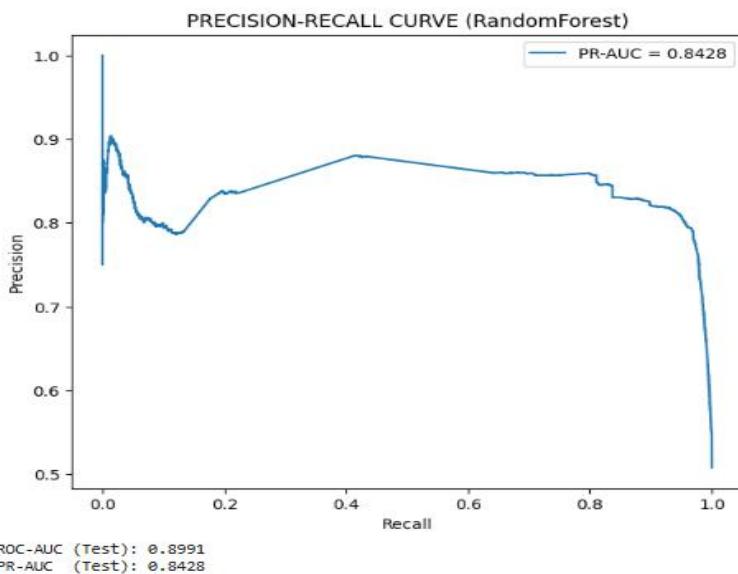
*Hình 2.25 Confusion matrix của model Random Forest embedding*

Classification Report:					
	precision	recall	f1-score	support	
phishing	0.93	0.77	0.84	47944	
benign	0.81	0.95	0.87	49465	
accuracy			0.86	97409	
macro avg	0.87	0.86	0.86	97409	
weighted avg	0.87	0.86	0.86	97409	
Accuracy (Test): 0.8604					

Hình 2.26 Đánh giá model RF với nhiều tiêu chí khác nhau



Hình 2.27 ROC Curve của model Random Forest embedding



Hình 2.28 Preciso/Recal Curve của model Random Forest embedding

2.3.2 Deep learning

2.3.2.1 WordCNN

Cấu hình:

```
from keras import layers, Sequential
def WordCNN(input_size):
    model = Sequential()
    model.add(layers.Input(input_size))
    model.add(layers.Conv1D(32, kernel_size=3, activation='relu', padding='same'))
    model.add(layers.Dropout(0.2))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling1D(pool_size=2, padding='same'))
    model.add(layers.Conv1D(64, kernel_size=3, activation='relu', padding='same'))
    model.add(layers.Dropout(0.2))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling1D(pool_size=2, padding='same'))
    model.add(layers.Conv1D(128, kernel_size=3, activation='relu', padding='same'))
    model.add(layers.Dropout(0.2))
    model.add(layers.BatchNormalization())
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model
```

Trong đó:

- layers.Input((n, 1)):
- Đầu vào gồm n đặc trưng, mỗi đặc trưng có 1 giá trị.
- Định nghĩa kích thước đầu vào cho mô hình.
- Conv1D(32, kernel_size=3, activation='relu', padding='same'):
- Lớp tích chập 1 chiều 1D Convolution với 32 bộ lọc và kernel = 3.
- Giúp mô hình học mẫu cục bộ (local patterns) trong chuỗi dữ liệu.

- relu giúp loại bỏ giá trị âm: tăng khả năng học phi tuyến.
- padding='same' đảm bảo kích thước đầu ra bằng kích thước đầu vào.
- Dropout(0.2):
 - Ngẫu nhiên bỏ 20% neuron trong quá trình huấn luyện.
 - Mục tiêu: giảm overfitting, tăng tính tổng quát.
- BatchNormalization():
 - Chuẩn hóa giá trị đầu ra của layer trước theo phân phối chuẩn.
 - Giúp mô hình hội tụ nhanh hơn và ổn định hơn khi huấn luyện.
- MaxPooling1D(pool_size=2, padding='same'):
 - Giảm kích thước đặc trưng bằng cách lấy giá trị lớn nhất trong mỗi nhóm 2 phần tử.
 - Giúp giảm số lượng tham số, tăng khả năng khai quật hóa.
- Conv1D(64, kernel_size=3, activation='relu', padding='same'):
 - Lặp lại khối WordCNN nhưng với 64 filters: học được mẫu phúc tạp hơn từ dữ liệu đã trích xuất ở tầng trước.
- Dropout(0.2) + BatchNormalization() + MaxPooling1D(pool_size=2):
 - Tiếp tục giảm overfitting và giảm chiều dữ liệu.
- Conv1D(128, kernel_size=3, activation='relu', padding='same'):
 - Tầng tích chập thứ ba với 128 filters, cho phép mô hình học đặc trưng trừu tượng và sâu hơn.
- Dropout(0.2) + BatchNormalization():
 - Tiếp tục ổn định và làm mượt mô hình.
- Flatten():
 - Biến đổi đầu ra 3D thành vector phẳng để đưa vào fully connected layers.
- Dense(128, activation='relu'):
 - Lớp kết nối đầy đủ (fully connected layer) với 128 neuron, giúp học mối quan hệ phi tuyến giữa các đặc trưng đã trích xuất.
- Dropout(0.3):
 - Loại bỏ 30% neuron ngẫu nhiên ở tầng dense, tăng khả năng tổng quát hóa.
- Dense(1, activation='sigmoid'):
 - Lớp đầu ra cho bài toán phân loại nhị phân.
 - sigmoid đưa đầu ra về khoảng (0,1) → biểu diễn xác suất.

```
WordCNN_model1 = WordCNN((81,1))
```

```
WordCNN_model1.summary()
```

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 81, 32)	128
dropout_2 (Dropout)	(None, 81, 32)	0
batch_normalization (BatchNormalization)	(None, 81, 32)	128
max_pooling1d (MaxPooling1D)	(None, 41, 32)	0
conv1d_1 (Conv1D)	(None, 41, 64)	6,208
dropout_3 (Dropout)	(None, 41, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 41, 64)	256
max_pooling1d_1 (MaxPooling1D)	(None, 21, 64)	0
conv1d_2 (Conv1D)	(None, 21, 128)	24,704
dropout_4 (Dropout)	(None, 21, 128)	0
batch_normalization_2 (BatchNormalization)	(None, 21, 128)	512
flatten (Flatten)	(None, 2688)	0
dense_4 (Dense)	(None, 128)	344,192
dropout_5 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 1)	129

Total params: 376,257 (1.44 MB)

Trainable params: 375,809 (1.43 MB)

Non-trainable params: 448 (1.75 KB)

Hình 2.29 Kiến trúc model WordCNN numerical

```
WordCNN_model2 = WordCNN((50,1))
```

```
WordCNN_model2.summary()
```

Layer (type)	Output Shape	Param #
conv1d_3 (Conv1D)	(None, 50, 32)	128
dropout_6 (Dropout)	(None, 50, 32)	0
batch_normalization_3 (BatchNormalization)	(None, 50, 32)	128
max_pooling1d_2 (MaxPooling1D)	(None, 25, 32)	0
conv1d_4 (Conv1D)	(None, 25, 64)	6,208
dropout_7 (Dropout)	(None, 25, 64)	0
batch_normalization_4 (BatchNormalization)	(None, 25, 64)	256
max_pooling1d_3 (MaxPooling1D)	(None, 13, 64)	0
conv1d_5 (Conv1D)	(None, 13, 128)	24,704
dropout_8 (Dropout)	(None, 13, 128)	0
batch_normalization_5 (BatchNormalization)	(None, 13, 128)	512
flatten_1 (Flatten)	(None, 1664)	0
dense_6 (Dense)	(None, 128)	213,120
dropout_9 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 1)	129

Total params: 245,185 (957.75 KB)

Trainable params: 244,737 (956.00 KB)

Non-trainable params: 448 (1.75 KB)

Hình 2.30 Kiến trúc model WordCNN embedding

Huấn luyện mô hình với dữ liệu đầu vào là đặc trưng số:

```
WordCNN_model1.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name="precision"),
        tf.keras.metrics.Recall(name="recall"),
        tf.keras.metrics.AUC(curve='PR', name="pr_auc")])
```

Trong đó:

- optimizer='adam':
- Bộ tối ưu hóa: giúp mô hình cập nhật trọng số trong quá trình học.
- Adam là thuật toán tối ưu phổ biến, kết hợp ưu điểm của AdaGrad và RMSProp, có khả năng thích ứng tốc độ học cho từng tham số.
- Tốc độ hội tụ nhanh, ổn định, phù hợp cho hầu hết các mô hình deep learning.
- loss='binary_crossentropy':
- Hàm mất mát: cho bài toán phân loại nhị phân.
- Công thức đo độ lệch giữa xác suất dự đoán và nhãn thật.
- Mô hình cố gắng giảm giá trị loss qua mỗi epoch để cải thiện độ chính xác.
- metrics:
- 'accuracy': Tỷ lệ mẫu được dự đoán đúng.
- `tf.keras.metrics.Precision(name="precision")`: Precision = TP / (TP + FP)
- `tf.keras.metrics.Recall(name="recall")`: Recall = TP / (TP + FN).
- `tf.keras.metrics.AUC(curve='PR', name="pr_auc")`: Diện tích dưới đường cong Precision-Recall.

```
callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        'PhiUSIIL_CNN_Numerical.keras',
        verbose=1,
        save_best_only=True),
    tf.keras.callbacks.EarlyStopping()
```

```
monitor='val_loss',
min_delta=0.001,
patience=15,
verbose=1)]
```

Trong đó:

- ModelCheckpoint
- Tự động lưu lại mô hình tốt nhất trong quá trình huấn luyện.
- save_best_only=True → chỉ lưu khi val_loss giảm so với mô hình trước đó, tránh lưu trùng lặp.
- verbose=1 → in thông báo mỗi khi mô hình được lưu.
- EarlyStopping
- Dừng quá trình huấn luyện sớm nếu mô hình không còn cải thiện.
- monitor='val_loss' → theo dõi loss của tập validation.
- min_delta=0.001 → cần ít nhất giảm 0.001 mới được coi là “cải thiện”.
- patience=15 → nếu 15 epoch liên tiếp không cải thiện → tự động dừng.

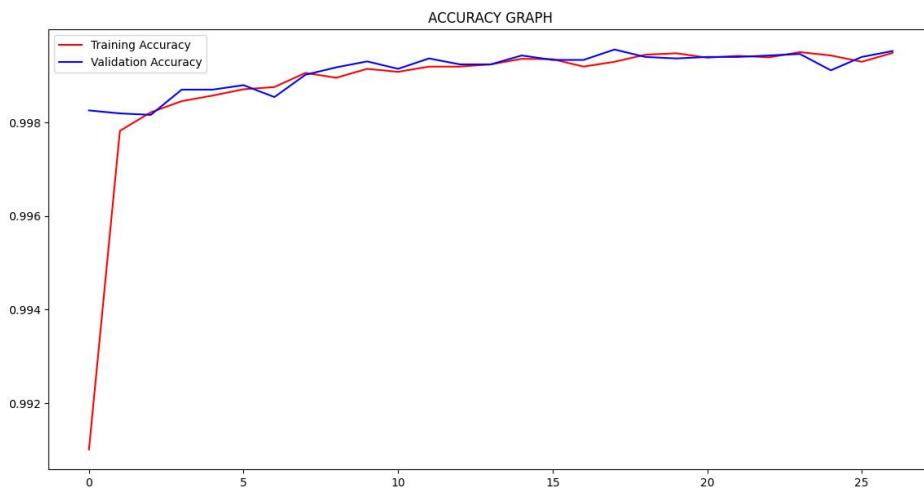
```
X_train3 = np.expand_dims(X_train3, axis=-1)
```

```
X_test3 = np.expand_dims(X_test3, axis=-1)
```

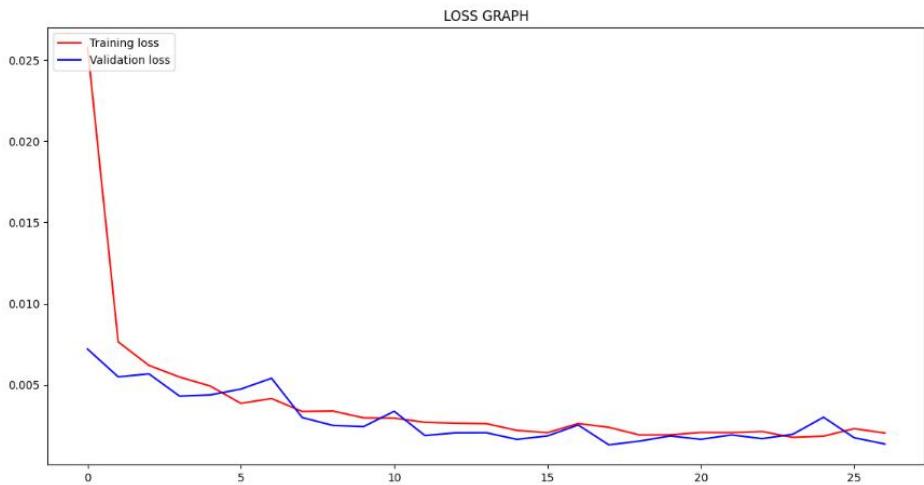
```
WordCNN_results_1 = WordCNN_model1.fit(
    X_train3, Y_train3,
    validation_split=0.2,
    batch_size=128,
    epochs=200,
    callbacks=callbacks)
```

- validation_split=0.2: Tự động tách 20% dữ liệu huấn luyện làm tập validation
- batch_size=128: Số mẫu xử lý trong mỗi bước cập nhật trọng số
- epochs=200: Số vòng lặp tối đa qua toàn bộ dữ liệu
- callbacks=callbacks: Gọi ModelCheckpoint và EarlyStopping trong quá trình huấn luyện

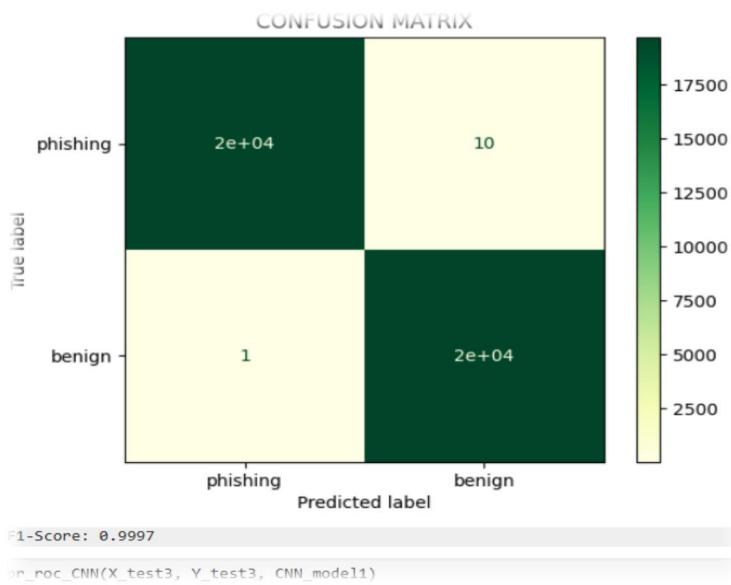
Sau khi quá trình huấn luyện kết thúc ta tiến hành tính toán kết quả huấn luyện



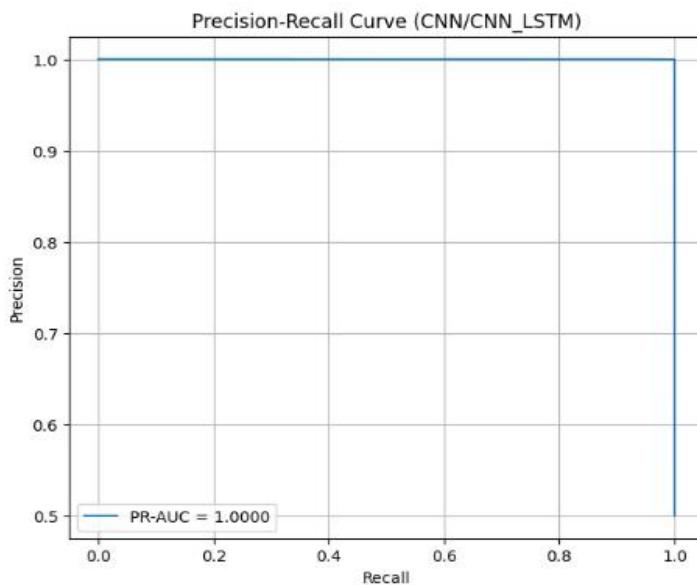
Hình 2.31 Accuracy graph của model WordCNN numerical



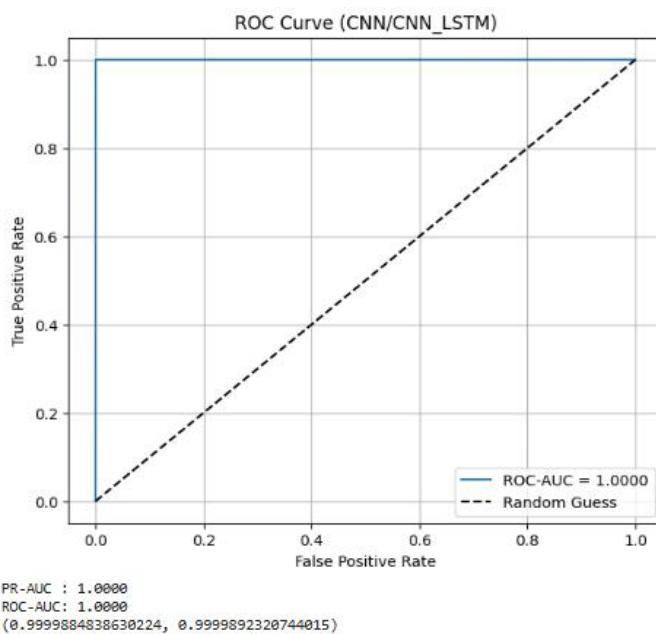
Hình 2.32 Loss graph của model WordCNN numerical



Hình 2.33 Confusion matrix của model WordCNN numerical



Hình 2.34 Precision/Recall Curve của model WordCNN numerical

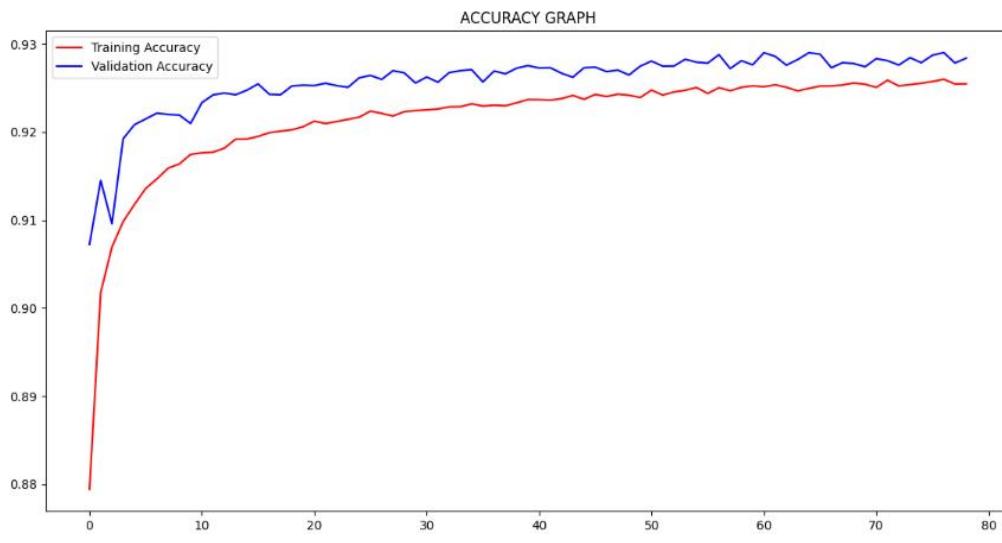
*Hình 2.35 ROC Curve của model WordCNN numerical*

Huấn luyện mô hình với dữ liệu đầu vào là URL Embedding:

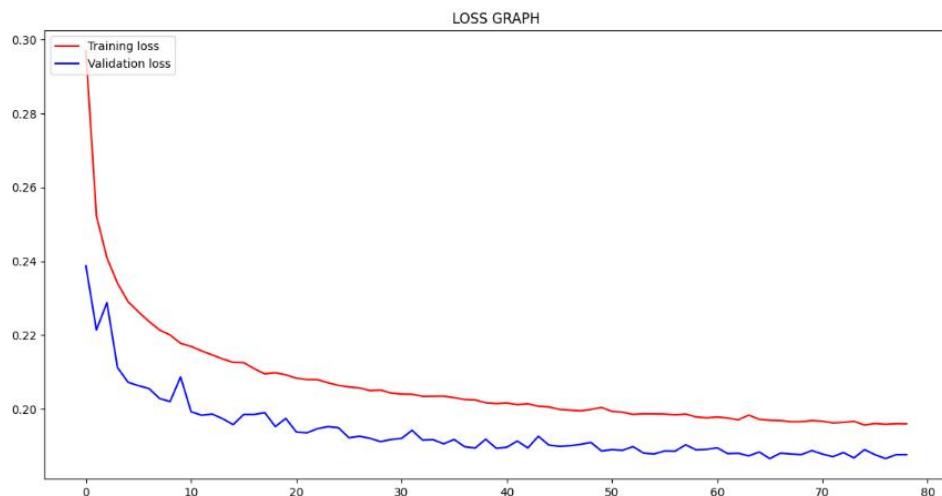
Cấu hình và tham số huấn luyện của WordCNN_model2 tương tự như WordCNN_model1

```
X_train4 = joblib.load("/content/drive/MyDrive/DeAnTotNghiệp/X_train_embed.pkl")
X_test4 = joblib.load("/content/drive/MyDrive/DeAnTotNghiệp/X_test_embed.pkl")
Y_train4 = joblib.load("/content/drive/MyDrive/DeAnTotNghiệp/Y_train_embed.pkl")
Y_test4 = joblib.load("/content/drive/MyDrive/DeAnTotNghiệp/Y_test_embed.pkl")
X_train4 = np.expand_dims(X_train4, axis=-1)
X_test4 = np.expand_dims(X_test4, axis=-1)
WordCNN_results_2 = WordCNN_model2.fit(
    X_train4, Y_train4,
    validation_split=0.2,
    batch_size=128,
    epochs=200,
    callbacks=callbacks)
```

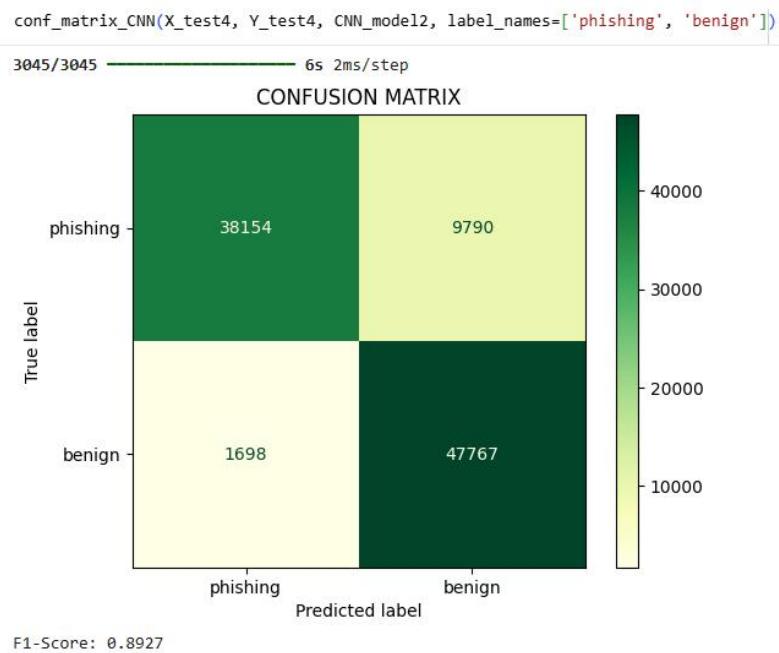
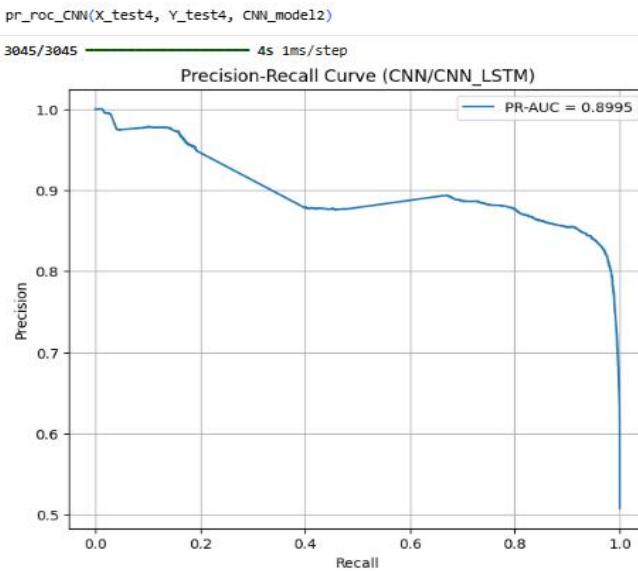
Sau khi quá trình huấn luyện kết thúc ta tiến hành tính toán kết quả huấn luyện

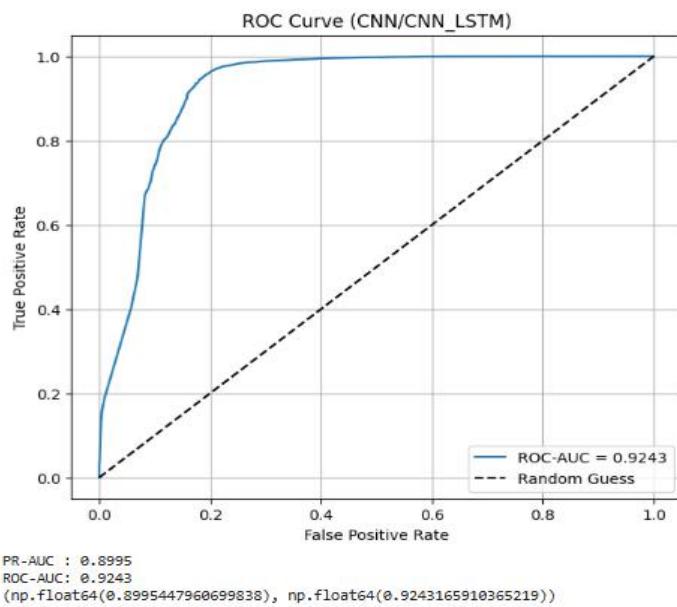


Hình 2.36 Accuracy graph của model WordCNN embedding



Hình 2.37 Loss graph của model WordCNN embedding

*Hình 2.38 Confusion matrix của model WordCNN embedding**Hình 2.39 Precisio/Recall Curve của model WordCNN embedding*

*Hình 2.40 ROC Curve của model WordCNN embedding*

2.3.2.2 CNN-LSTM

Cấu hình:

```
from keras import layers, Sequential
def CNN_LSTM(input_size):
    model = Sequential()
    model.add(layers.Input(input_size))
    model.add(layers.Conv1D(32, kernel_size=3, activation='relu', padding='same'))
    model.add(layers.Dropout(0.2))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling1D(pool_size=2, padding='same'))
    model.add(layers.Conv1D(64, kernel_size=3, activation='relu', padding='same'))
    model.add(layers.Dropout(0.2))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling1D(pool_size=2, padding='same'))
    model.add(layers.Conv1D(128, kernel_size=3, activation='relu', padding='same'))
    model.add(layers.Dropout(0.2))
```

```

model.add(layers.BatchNormalization())
model.add(layers.LSTM(64, return_sequences=False))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(1, activation='sigmoid'))
return model

```

Trong đó:

- layers.Input((n, 1)):
 - Đầu vào gồm n đặc trưng, mỗi đặc trưng có 1 giá trị.
 - Định nghĩa kích thước đầu vào cho mô hình.
- Conv1D(32, kernel_size=3, activation='relu', padding='same'):
 - Lớp tích chập 1 chiều (1D Convolution) với 32 bộ lọc và kernel = 3.
 - Giúp mô hình học mẫu cục bộ trong chuỗi dữ liệu.
 - relu giúp loại bỏ giá trị âm và tăng khả năng học phi tuyến.
 - padding='same' đảm bảo kích thước đầu ra bằng kích thước đầu vào.
- Dropout(0.2):
 - Ngẫu nhiên bỏ 20% neuron trong quá trình huấn luyện.
 - Mục tiêu: giảm overfitting, tăng tính tổng quát.
- BatchNormalization():
 - Chuẩn hóa giá trị đầu ra của layer trước theo phân phối chuẩn.
 - Giúp mô hình hội tụ nhanh hơn và ổn định hơn khi huấn luyện.
- Conv1D(64, kernel_size=3, activation='relu', padding='same'):
 - Lặp lại khôi CNN nhưng với 64 filters sẽ học được mẫu phức tạp hơn từ dữ liệu đã trích xuất ở tầng trước.
- Dropout(0.2) + BatchNormalization() + MaxPooling1D(pool_size=2):
 - Tiếp tục giảm overfitting và giảm chiều dữ liệu.
- Conv1D(128, kernel_size=3, activation='relu', padding='same'):
 - Tầng tích chập thứ ba với 128 filters, cho phép mô hình học đặc trưng trừu tượng và sâu hơn.
- Dropout(0.2) + BatchNormalization():
 - Tiếp tục ổn định và làm mượt mô hình.
- LSTM(64, return_sequences=False):
 - Đầu ra là một vector duy nhất.

- Học mối quan hệ theo thời gian giữa các đặc trưng do CNN trích xuất.
return_sequences=False chỉ lấy vector cuối cùng.
 - Dense(128, activation='relu'):
- Lớp kết nối dày đủ với 128 neuron, giúp học mối quan hệ phi tuyến giữa các đặc trưng đã trích xuất.
- Dropout(0.3):
- Loại bỏ 30% neuron ngẫu nhiên ở tầng dense, tăng khả năng tổng quát hóa.
- Dense(1, activation='sigmoid'):
- Lớp đầu ra cho bài toán phân loại nhị phân.
- sigmoid đưa đầu ra về khoảng (0,1) → biểu diễn xác suất.

```
CNN_LSTM_model1 = CNN_LSTM((81,1))
```

```
CNN_LSTM_model1.summary()
```

Layer (type)	Output Shape	Param #
conv1d_6 (Conv1D)	(None, 81, 32)	128
dropout_10 (Dropout)	(None, 81, 32)	0
batch_normalization_6 (BatchNormalization)	(None, 81, 32)	128
max_pooling1d_4 (MaxPooling1D)	(None, 41, 32)	0
conv1d_7 (Conv1D)	(None, 41, 64)	6,208
dropout_11 (Dropout)	(None, 41, 64)	0
batch_normalization_7 (BatchNormalization)	(None, 41, 64)	256
max_pooling1d_5 (MaxPooling1D)	(None, 21, 64)	0
conv1d_8 (Conv1D)	(None, 21, 128)	24,704
dropout_12 (Dropout)	(None, 21, 128)	0
batch_normalization_8 (BatchNormalization)	(None, 21, 128)	512
lstm_4 (LSTM)	(None, 64)	49,408
dense_8 (Dense)	(None, 128)	8,320
dropout_13 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 1)	129

Total params: 89,793 (350.75 KB)

Trainable params: 89,345 (349.00 KB)

Non-trainable params: 448 (1.75 KB)

Hình 2.41 Kiến trúc mô hình của CNN-LSTM numerical

```
CNN_LSTM_model2 = CNN_LSTM((50,1))
```

```
CNN_LSTM_model2.summary()
```

Layer (type)	Output Shape	Param #
conv1d_9 (Conv1D)	(None, 50, 32)	128
dropout_14 (Dropout)	(None, 50, 32)	0
batch_normalization_9 (BatchNormalization)	(None, 50, 32)	128
max_pooling1d_6 (MaxPooling1D)	(None, 25, 32)	0
conv1d_10 (Conv1D)	(None, 25, 64)	6,208
dropout_15 (Dropout)	(None, 25, 64)	0
batch_normalization_10 (BatchNormalization)	(None, 25, 64)	256
max_pooling1d_7 (MaxPooling1D)	(None, 13, 64)	0
conv1d_11 (Conv1D)	(None, 13, 128)	24,704
dropout_16 (Dropout)	(None, 13, 128)	0
batch_normalization_11 (BatchNormalization)	(None, 13, 128)	512
lstm_5 (LSTM)	(None, 64)	49,408
dense_10 (Dense)	(None, 128)	8,320
dropout_17 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 1)	129

Total params: 89,793 (350.75 KB)

Trainable params: 89,345 (349.00 KB)

Non-trainable params: 448 (1.75 KB)

Hình 2.42 Kiến trúc mô hình CNN-LSTM embedding

Huấn luyện mô hình với dữ liệu đầu vào là đặc trưng số:

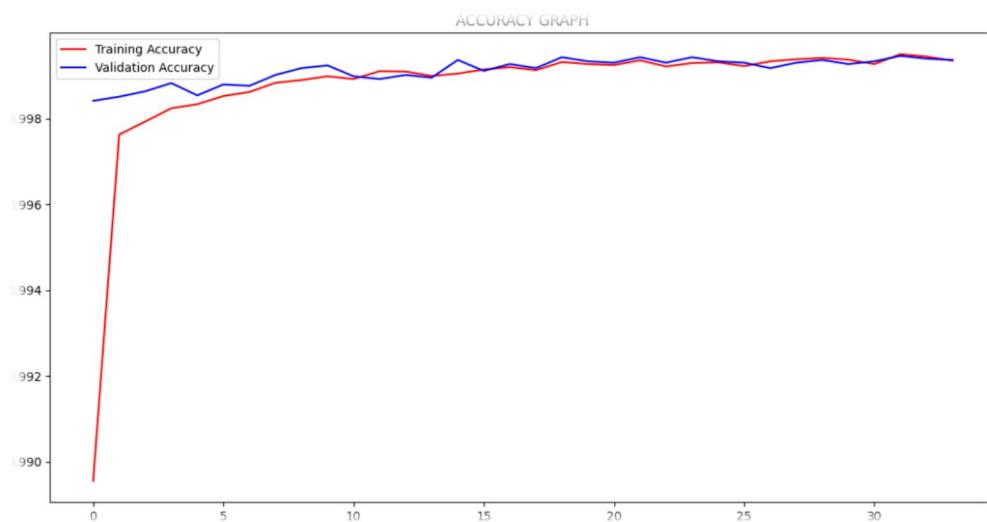
Cấu hình và tham số huấn luyện tương tự như CNN-LSTM:

```
X_train3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/X_train3.npy")
```

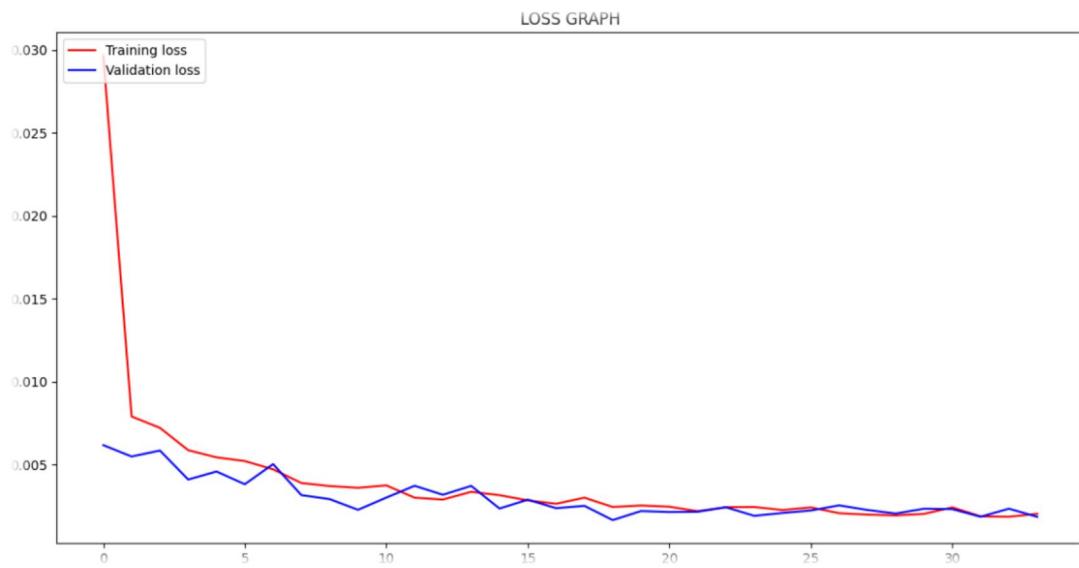
```
X_test3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/X_test3.npy")
```

```
Y_train3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/Y_train3.npy")
Y_test3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/Y_test3.npy")
CNN_LSTM_results_1 = CNN_LSTM_model1.fit(
    X_train3, Y_train3,
    validation_split=0.2,
    batch_size=128,
    epochs=200,
    callbacks=callbacks)
```

Sau khi hoàn thành quá trình huấn luyện ta tiến hành tính toán kết quả và đánh giá huấn luyện



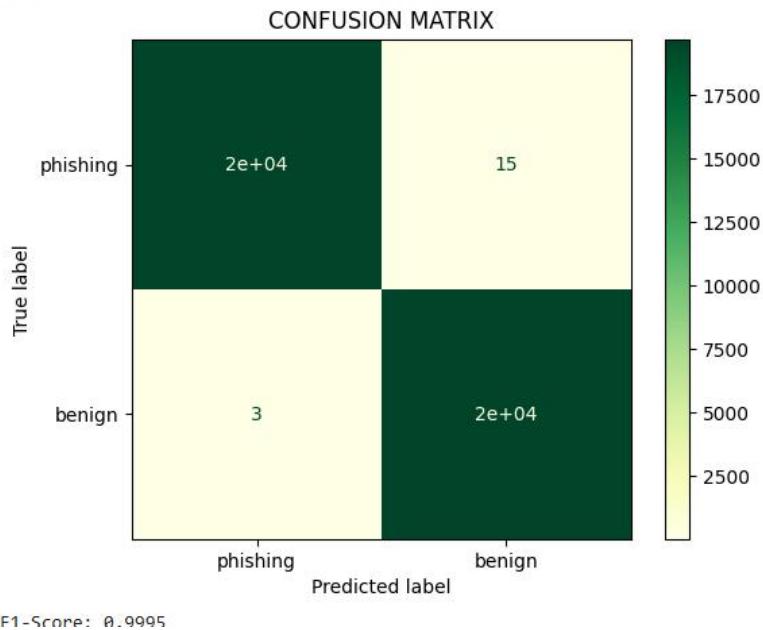
Hình 2.43 Accuracy graph của model CNN-LSTM numerical



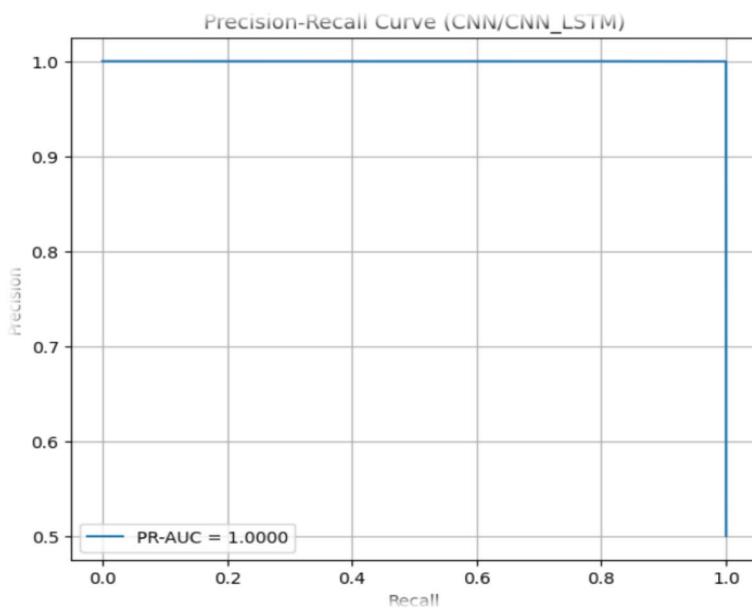
Hình 2.44 Loss graph của model CNN-LSTM numerical

```
conf_matrix_CNN(X_test3, Y_test3, CNN_LSTM_model1, label_names=['phishing', 'benign'])
```

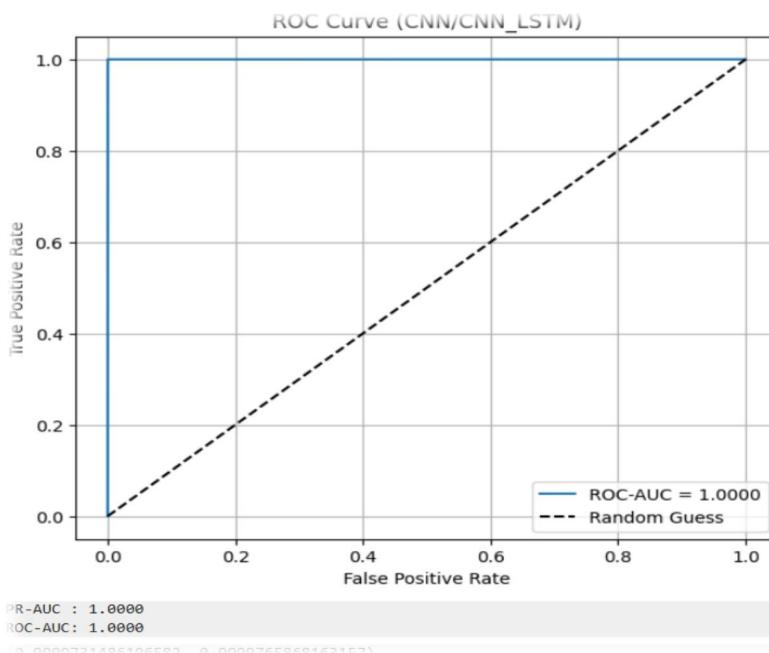
1232/1232 —————— 3s 2ms/step



Hình 2.45 Confusion matrix của model CNN-LSTM numerical



Hình 2.46 Precision/Recall Curve CNN-LSTM numerical



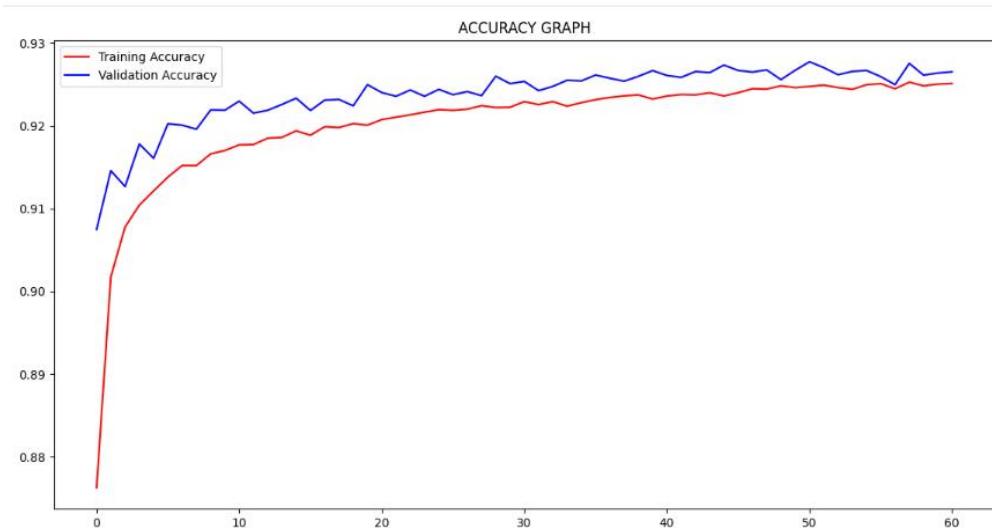
Hình 2.47 ROC Curve của model CNN-LSTM numerical

Huấn luyện mô hình với dữ liệu đầu vào là URL Embedding:

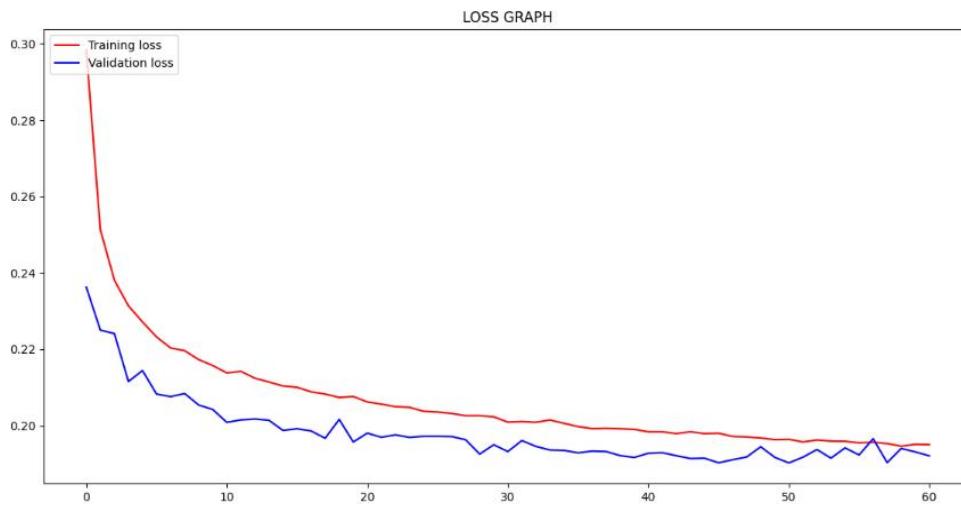
Cấu hình và tham số huấn luyện tương tự như CNN-LSTM:

```
X_train4 = np.load("/content/drive/MyDrive/DeAnTotNghiep/X_train4.npy")
X_test4 = np.load("/content/drive/MyDrive/DeAnTotNghiep/X_test4.npy")
Y_train4 = np.load("/content/drive/MyDrive/DeAnTotNghiep/Y_train4.npy")
Y_test4 = np.load("/content/drive/MyDrive/DeAnTotNghiep/Y_test4.npy")
CNN_LSTM_results_2 = CNN_LSTM_model2.fit(
    X_train4, Y_train4,
    validation_split=0.2,
    batch_size=128,
    epochs=200,
    callbacks=callbacks)
```

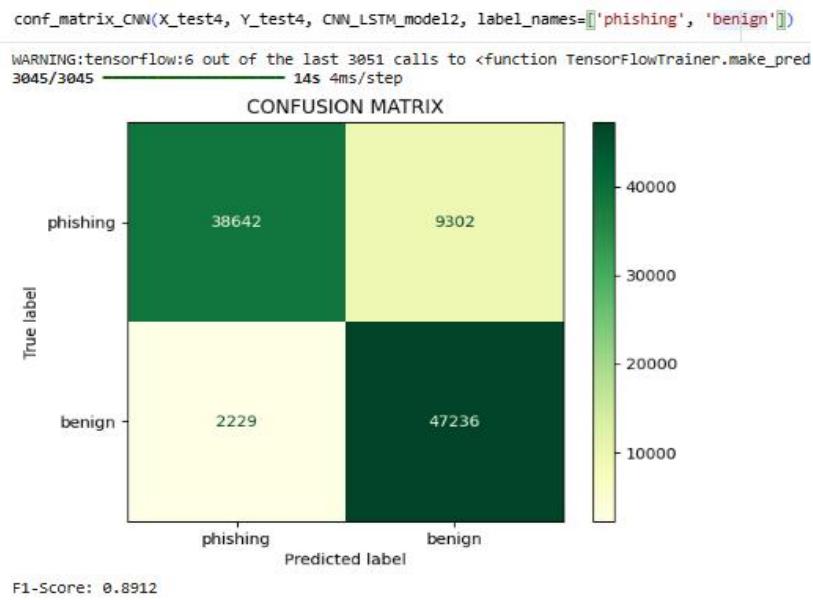
Sau khi hoàn thành quá trình huấn luyện ta tiến hành tính toán kết quả và đánh giá huấn luyện



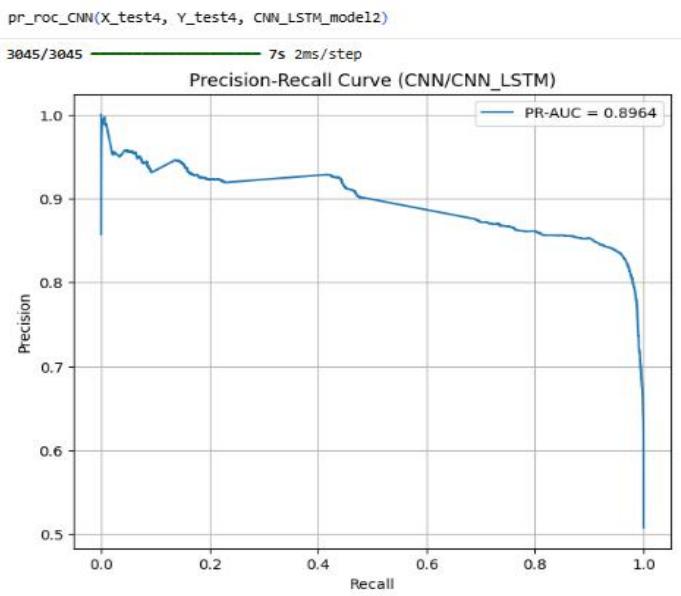
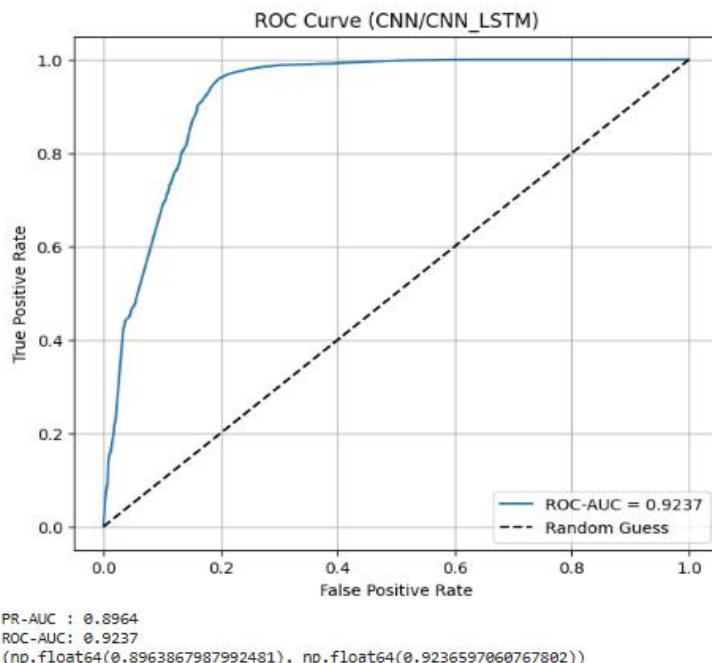
Hình 2.48 Accuracy graph của model CNN-LSTM embedding



Hình 2.49 Loss graph của model CNN-LSTM embedding



Hình 2.50 Confusion matrix của model CNN-LSTM embedding

*Hình 2.51 Precision/Recall curve của model CNN-LSTM embedding**Hình 2.52 ROC Curve của model CNN-LSTM embedding*

2.3.2.3 CharCNN

Cấu hình:

```
import numpy as np
import string
import joblib
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense, Dropout

char_tokenizer =
joblib.load("/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_Char_Tokenizer.joblib")
vocab_size = len(char_tokenizer.word_index) + 1
MAX_LEN = 200
EMBED_DIM = 64

def CharCNN(vocab_size, max_len=MAX_LEN, embed_dim=EMBED_DIM):
    model = Sequential([
        Embedding(input_dim=vocab_size, output_dim=embed_dim, input_length=max_len),
        Conv1D(256, kernel_size=5, activation='relu', padding='same'),
        MaxPooling1D(pool_size=2),
        Conv1D(128, kernel_size=3, activation='relu', padding='same'),
        MaxPooling1D(pool_size=2),
        Conv1D(64, kernel_size=3, activation='relu', padding='same'),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])

```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
return model
```

```
CharCNN_model = CharCNN(vocab_size)
CharCNN_model.build(X_train7.shape)
CharCNN_model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: /
  warnings.warn(
Model: "sequential_6"



| Layer (type)                    | Output Shape       | Param # |
|---------------------------------|--------------------|---------|
| embedding (Embedding)           | (389635, 200, 64)  | 5,824   |
| conv1d_12 (Conv1D)              | (389635, 200, 256) | 82,176  |
| max_pooling1d_8 (MaxPooling1D)  | (389635, 100, 256) | 0       |
| conv1d_13 (Conv1D)              | (389635, 100, 128) | 98,432  |
| max_pooling1d_9 (MaxPooling1D)  | (389635, 50, 128)  | 0       |
| conv1d_14 (Conv1D)              | (389635, 50, 64)   | 24,640  |
| max_pooling1d_10 (MaxPooling1D) | (389635, 25, 64)   | 0       |
| flatten_2 (Flatten)             | (389635, 1600)     | 0       |
| dense_12 (Dense)                | (389635, 128)      | 204,928 |
| dropout_18 (Dropout)            | (389635, 128)      | 0       |
| dense_13 (Dense)                | (389635, 1)        | 129     |



Total params: 416,129 (1.59 MB)
Trainable params: 416,129 (1.59 MB)
Non-trainable params: 0 (0.00 B)
```

Hình 2.53 Kiến trúc mô hình CharCNN

Huấn luyện mô hình với dữ liệu đầu vào là URL embedding:

Cấu hình và tham số huấn luyện CharCNN

```
tokenizer = Tokenizer(
    char_level=True,      # Kích hoạt chế độ ký tự
    lower=True,           # Chuyển tất cả về lowercase
    filters="")           # Không loại bỏ ký tự nào
```

```
    oov_token='<UNK>'      # Token cho ký tự không nằm trong vocab
)
tokenizer.fit_on_texts(X_train_urls)

vocab_size = len(tokenizer.word_index) + 1
print(f"Vocabulary size (char-level): {vocab_size}")

X_train_seq = tokenizer.texts_to_sequences(X_train_urls)
X_test_seq = tokenizer.texts_to_sequences(X_test_urls)

X_train7 = pad_sequences(X_train_seq, maxlen=200, padding='post', truncating='post')
X_test7 = pad_sequences(X_test_seq, maxlen=200, padding='post', truncating='post')

Y_train7 = np.array(Y_train)
Y_test7 = np.array(Y_test)

import tensorflow as tf
CharCNN_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name="precision"),
        tf.keras.metrics.Recall(name="recall"),
        tf.keras.metrics.AUC(curve='PR', name="pr_auc")
    ]
)
callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        'PhiUSIIL_CharCNN_URL.keras',
        verbose=1,
        save_best_only=True
),
```

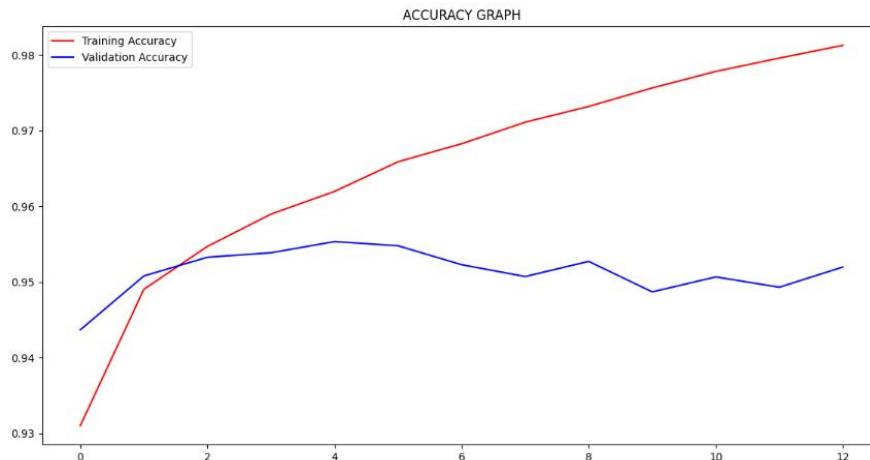
```

tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.001,
    patience=10,
    verbose=1
)
]

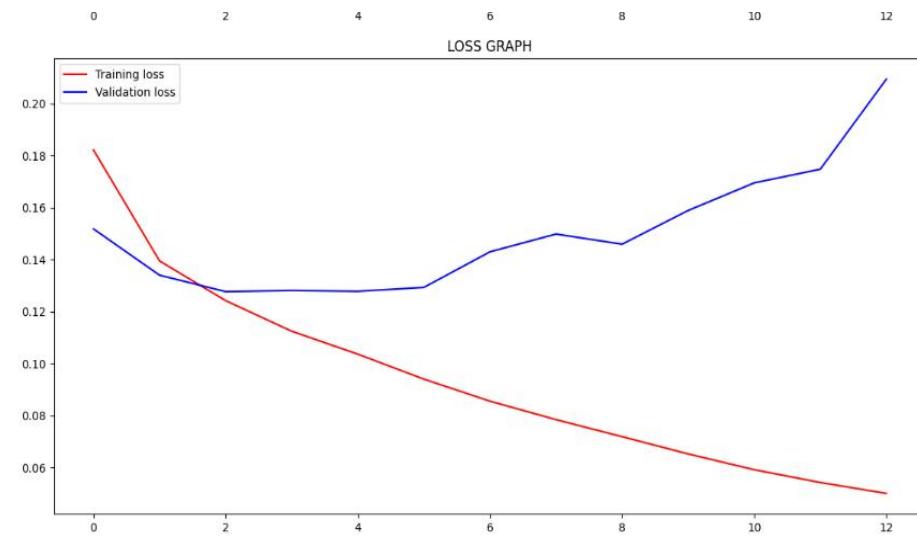
CharCNN_results = CharCNN_model.fit(
    X_train7, Y_train7,
    validation_split=0.2,
    batch_size=64,
    epochs=200,
    callbacks=callbacks
)

```

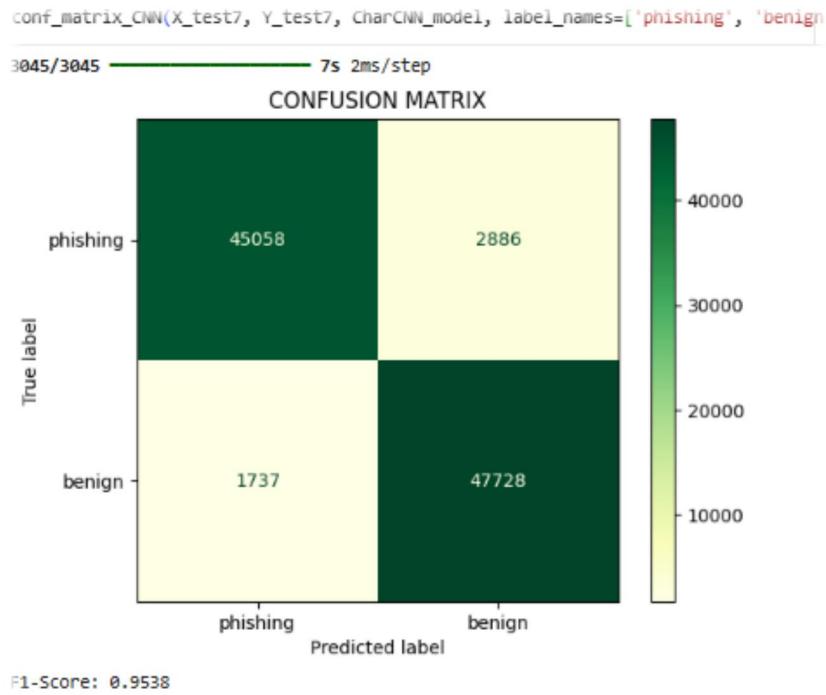
Sau khi hoàn thành quá trình huấn luyện ta tiến hành tính toán kết quả và đánh giá huấn luyện



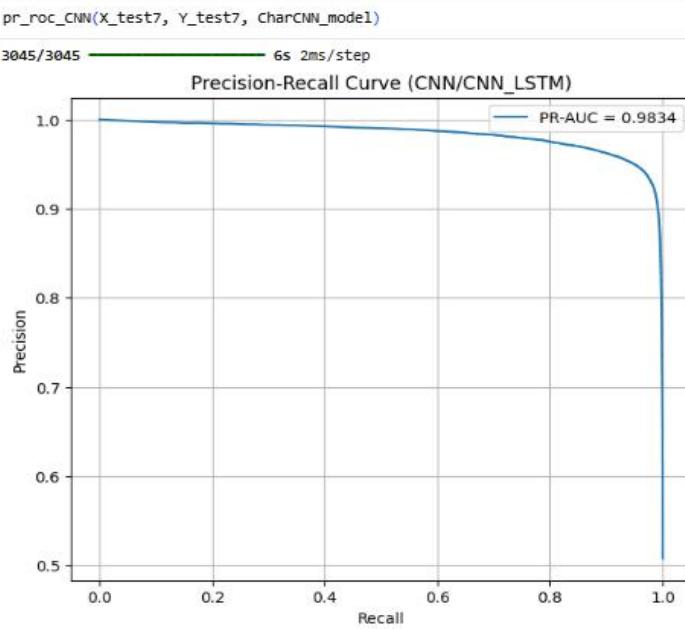
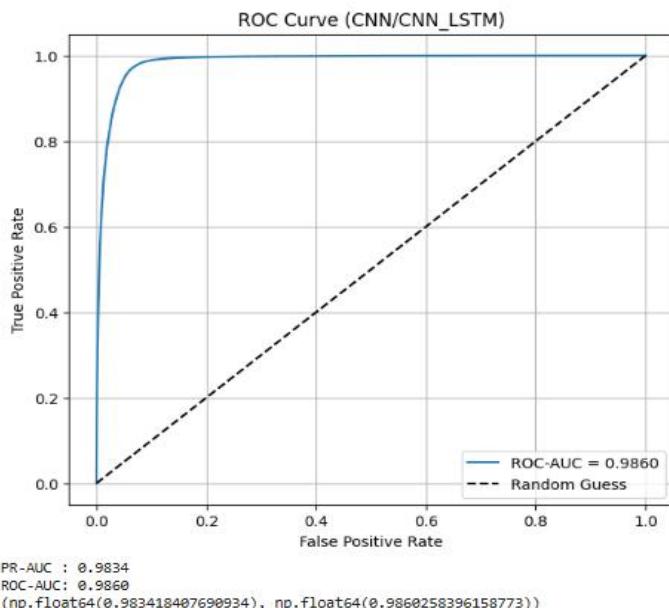
Hình 2.54 Accuracy graph của model CharCNN



Hình 2.55 Loss graph của model CharCNN



Hình 2.56 Confusion matrix của model CharCNN

*Hình 2.57 Precision/Recall Curve CharCNN**Hình 2.58 ROC Curve của model CharCNN*

2.3.2.4 CharCNN-LSTM

Cấu hình:

```
import numpy as np
import string
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from tensorflow.keras.layers import LSTM, Bidirectional

char_tokenizer = joblib.load("/kaggle/working/PhiUSIIL_Char_Tokenizer.joblib")
vocab_size = len(char_tokenizer.word_index) + 1
MAX_LEN = 200
EMBED_DIM = 64

def CharCNN_LSTM(vocab_size, max_len=MAX_LEN, embed_dim=EMBED_DIM):
    model = Sequential([
        Embedding(input_dim=vocab_size, output_dim=embed_dim, input_length=max_len),
        # --- CNN feature extractor ---
        Conv1D(256, kernel_size=5, activation='relu', padding='same'),
        MaxPooling1D(pool_size=2),
        Conv1D(128, kernel_size=3, activation='relu', padding='same'),
        MaxPooling1D(pool_size=2),
        # --- LSTM sequence encoder ---
        Bidirectional(LSTM(64, dropout=0.2, return_sequences=False)),
        # --- Dense layers ---
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
return model
```

```
CharCNN_LSTM_model = CharCNN_LSTM(vocab_size)
CharCNN_LSTM_model.build(X_train7.shape)
CharCNN_LSTM_model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(392807, 200, 64)	6,784
conv1d_3 (Conv1D)	(392807, 200, 256)	82,176
max_pooling1d_3 (MaxPooling1D)	(392807, 100, 256)	0
conv1d_4 (Conv1D)	(392807, 100, 128)	98,432
max_pooling1d_4 (MaxPooling1D)	(392807, 50, 128)	0
bidirectional (Bidirectional)	(392807, 128)	98,816
dense_2 (Dense)	(392807, 128)	16,512
dropout_1 (Dropout)	(392807, 128)	0
dense_3 (Dense)	(392807, 1)	129

Total params: 302,849 (1.16 MB)
Trainable params: 302,849 (1.16 MB)
Non-trainable params: 0 (0.00 B)

Hình 2.59 Kiến trúc mô hình CharCNN-LSTM

Huấn luyện mô hình với dữ liệu đầu vào là URL embedding:

Cáu hình và tham số huấn luyện CharCNN-LSTM:

```

tokenizer = Tokenizer(
    char_level=True,          # Kích hoạt chế độ ký tự
    lower=True,               # Chuyển tất cả về lowercase
    filters='',              # Không loại bỏ ký tự nào
    oov_token='<UNK>'       # Token cho ký tự không nằm trong vocab
)
tokenizer.fit_on_texts(X_train_urls)

vocab_size = len(tokenizer.word_index) + 1
print(f"Vocabulary size (char-level): {vocab_size}")

X_train_seq = tokenizer.texts_to_sequences(X_train_urls)
X_test_seq = tokenizer.texts_to_sequences(X_test_urls)

X_train7 = pad_sequences(X_train_seq, maxlen=200, padding='post', truncating='post')
X_test7 = pad_sequences(X_test_seq, maxlen=200, padding='post', truncating='post')

Y_train7 = np.array(Y_train)
Y_test7 = np.array(Y_test)

import tensorflow as tf

CharCNN_LSTM_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name="precision"),
        tf.keras.metrics.Recall(name="recall"),
        tf.keras.metrics.AUC(curve='PR', name="pr_auc")
    ]
)

callbacks = [

```

```

tf.keras.callbacks.ModelCheckpoint(
    'PhiUSIIL_CharCNN_LSTM_URL.keras',
    verbose=1,
    save_best_only=True
),
tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.001,
    patience=10,
    verbose=1
)
]

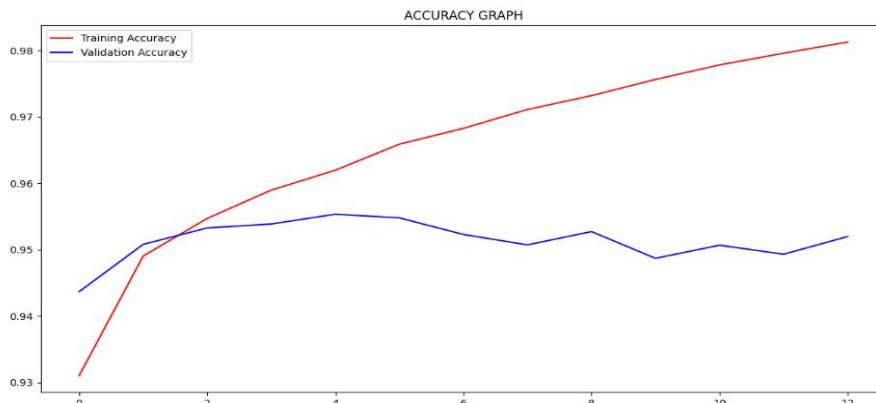
```

```

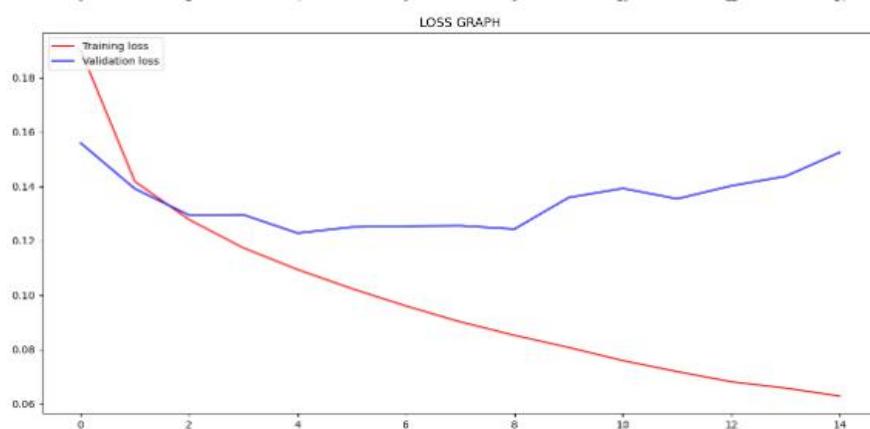
CharCNN_LSTM_results = CharCNN_LSTM_model.fit(
    X_train7, Y_train7,
    validation_split=0.2,
    batch_size=64,
    epochs=200,
    callbacks=callbacks
)

```

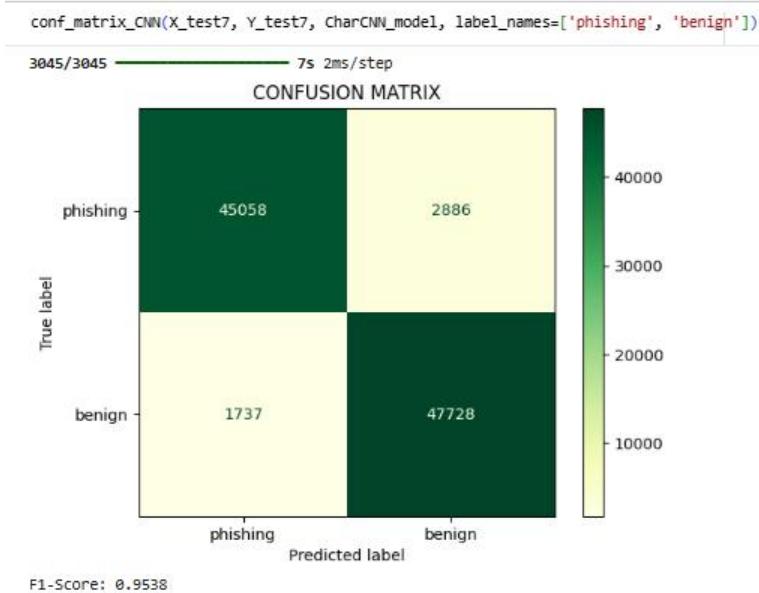
Sau khi hoàn thành quá trình huấn luyện ta tiến hành tính toán kết quả và đánh giá huấn luyện



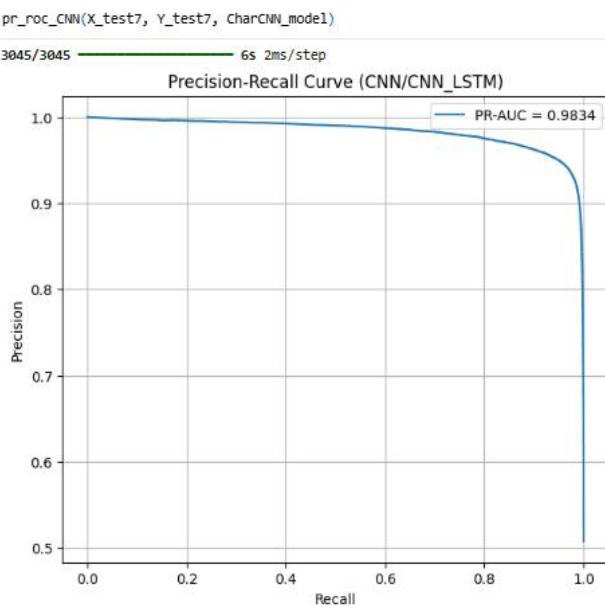
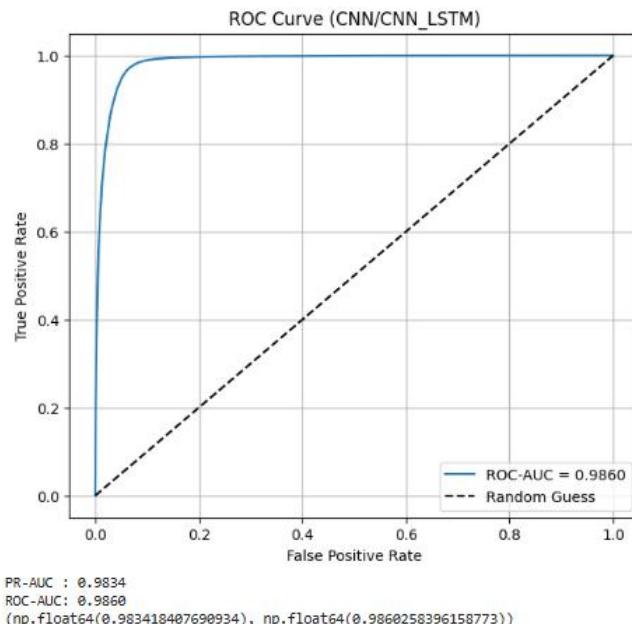
Hình 2.60 Accuracy graph của model CharCNN-LSTM



Hình 2.61 Loss graph của model CharCNN-LSTM



Hình 2.62 Confusion matrix của model CharCNN-LSTM

*Hình 2.63 Precision/Recall Curve CharCNN-LSTM**Hình 2.64 ROC Curve của model CharCNN-LSTM*

2.3.2.5 CNN Hybrid

Cấu hình:

```
def PhiUSIIL_CNN_Hybrid(word_embedding_dim=50, char_embedding_dim=32,
char_max_len=200, vocab_size=106,
    num_filters=256, filter_sizes=[3, 4, 5, 6], dropout_rate=0.5, fc_units=[512, 256, 128]):
# ===== Character-Level =====
char_input = layers.Input(shape=(char_max_len,), name='char_input')
# Char Embedding Layer
char_embedding = layers.Embedding(input_dim=vocab_size,
output_dim=char_embedding_dim,
    input_length=char_max_len, name='char_embedding',
    embeddings_regularizer=keras.regularizers.l2(1e-6))(char_input)
# Parallel Convolutional Filters
char_conv_blocks = []
for filter_size in filter_sizes:
    conv = layers.Conv1D(filters=num_filters, kernel_size=filter_size, activation='relu',
        padding='same', kernel_regularizer=keras.regularizers.l2(1e-6),
        name=f'char_conv_{filter_size}')(char_embedding)
    # Batch Normalization
    conv = layers.BatchNormalization(name=f'char_bn_{filter_size}')(conv)
    # Global Max Pooling
    pool = layers.GlobalMaxPooling1D(name=f'char_pool_{filter_size}')(conv)
    char_conv_blocks.append(pool)
# Concatenate all pooled features
char_features = layers.concatenate(name='char_concat')(char_conv_blocks)
char_features = layers.Dropout(dropout_rate, name='char_dropout')(char_features)
char_features = layers.Dense(fc_units[0], activation='relu',
    kernel_regularizer=keras.regularizers.l2(1e-6),
name='char_fc')(char_features)
char_features = layers.BatchNormalization(name='char_fc_bn')(char_features)
# ===== Word-Level =====
word_input = layers.Input(shape=(word_embedding_dim,), name='word_input')
```

```

# Dense layers
word_features = layers.Dense(512, activation='relu',
                             kernel_regularizer=keras.regularizers.l2(1e-6),
                             name='word_fc1')(word_input)
word_features = layers.BatchNormalization(name='word_bn1')(word_features)
word_features = layers.Dropout(dropout_rate, name='word_dropout1')(word_features)

word_features = layers.Dense(fc_units[0], activation='relu',
                             kernel_regularizer=keras.regularizers.l2(1e-6),
                             name='word_fc2')(word_features)
word_features = layers.BatchNormalization(name='word_bn2')(word_features)
word_features = layers.Dropout(dropout_rate, name='word_dropout2')(word_features)
# ===== Concatenate Char &
Word =====
combined = layers.concatenate(name='combined_features')[char_features, word_features])
# ===== Classification Head
=====
x = combined
for i, units in enumerate (fc_units[1:], 1):
    x = layers.Dense(units, activation='relu', kernel_regularizer=keras.regularizers.l2(1e-2),
                     name=f'fc_{i}')(x)
    x = layers.BatchNormalization(name=f'fc_bn_{i}')(x)
    x = layers.Dropout(dropout_rate, name=f'dropout_{i}')(x)
# ===== Output layer
=====
=
output = layers.Dense(1, activation='sigmoid', name='output')(x)
# ===== Build Model
=====
=
model = Model(inputs=[char_input, word_input], outputs=output,
name='PhiUSIIL_CNN_Hybrid')

return model

```

Huấn luyện mô hình với dữ liệu đầu vào là URL embedding:

Cấu hình và tham số huấn luyện CNN-Hybrid:

```
print("\n Building PhiUSIIL_CNN_Hybrid model...")
model = PhiUSIIL_CNN_Hybrid(
    word_embedding_dim=50, char_embedding_dim=32, char_max_len=200,
    vocab_size=vocab_size,
    num_filters=256, filter_sizes=[3, 4, 5, 6], dropout_rate=0.5, fc_units=[512, 256, 128] )

model.compile(
    optimizer=Adam(learning_rate=0.001, clipnorm=1.0),
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        keras.metrics.AUC(name='auc'),
        keras.metrics.Precision(name='precision'),
        keras.metrics.Recall(name='recall')
    ]
)

print("\n Model Architecture:")
model.summary()
total_params = model.count_params()
print(f"\n Total Parameters: {total_params:,}")

callbacks = [
    ModelCheckpoint(
        filepath='/kaggle/working/PhiUSIIL_CNN_Hybrid_Checkpoints.keras',
        save_best_only=True,
        verbose=1
    ),
    EarlyStopping(
        monitor='val_loss',
```

```
        mode='max',
        patience=10,
        verbose=1,
        restore_best_weights=True
),
ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=3,
    verbose=1,
    min_lr=1e-7
)
]
print("\n" + "=" * 80)
print(" TRAINING PhiUSIIL-CNN-Hybrid Model")
print("=" * 80)
```

```
result = model.fit(
    [X_train_c2v, X_train_w2v],
    Y_train_w2v,
    batch_size=512,
    epochs=100,
    validation_split=0.15,
    callbacks=callbacks,
    class_weight=class_weights,
    verbose=1
)
```

Sau khi hoàn thành quá trình huấn luyện ta tiến hành tính toán kết quả và đánh giá huấn luyện:

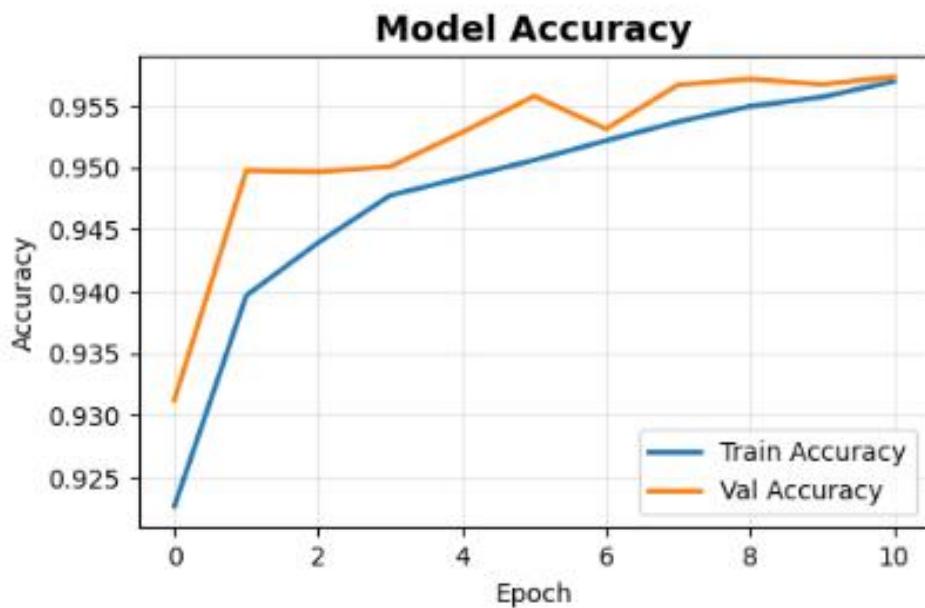
```
=====
[+] DETAILED PERFORMANCE METRICS
=====

[+] Classification Report:
      precision    recall    f1-score   support

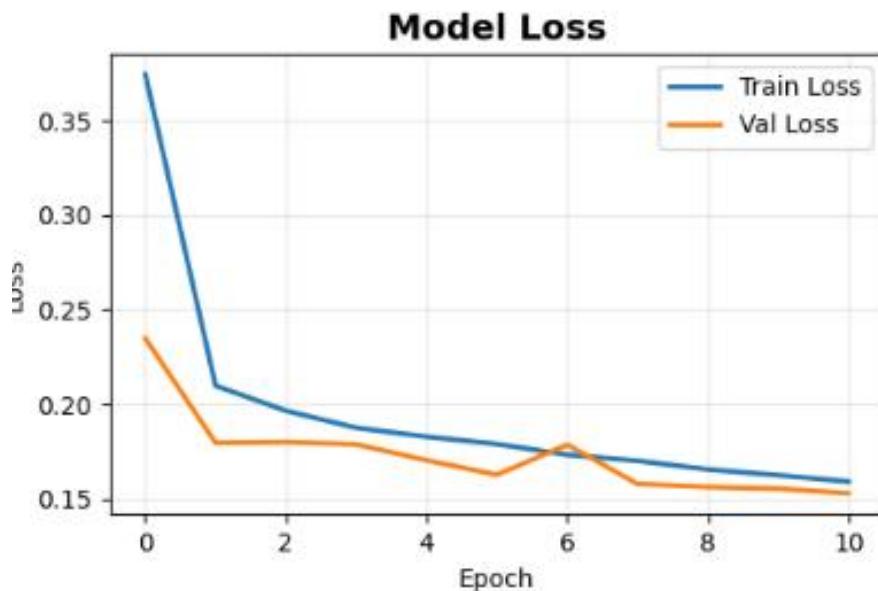
Phishing (0)      0.9646    0.8713    0.9156    47944
Benign (1)        0.8860    0.9690    0.9257    49465

accuracy          -         -         -         97409
macro avg         0.9253    0.9202    0.9206    97409
weighted avg      0.9247    0.9210    0.9207    97409
```

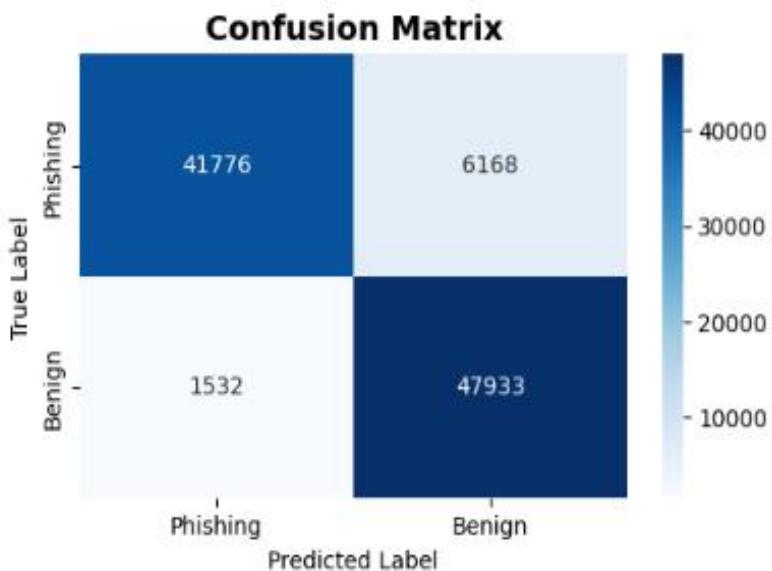
Hình 2.65 Đánh giá tổng quan model CNN-Hybrid được huấn luyện



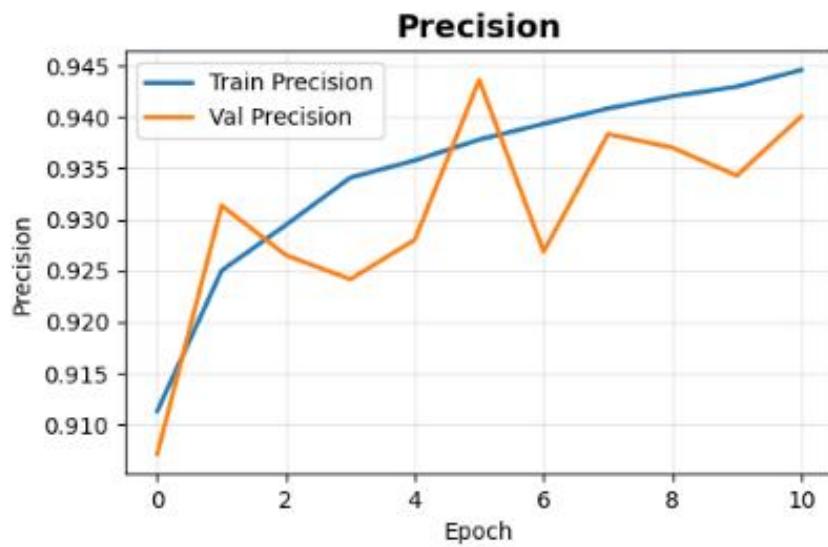
Hình 2.66 Accuracy graph của model CNN-Hybrid



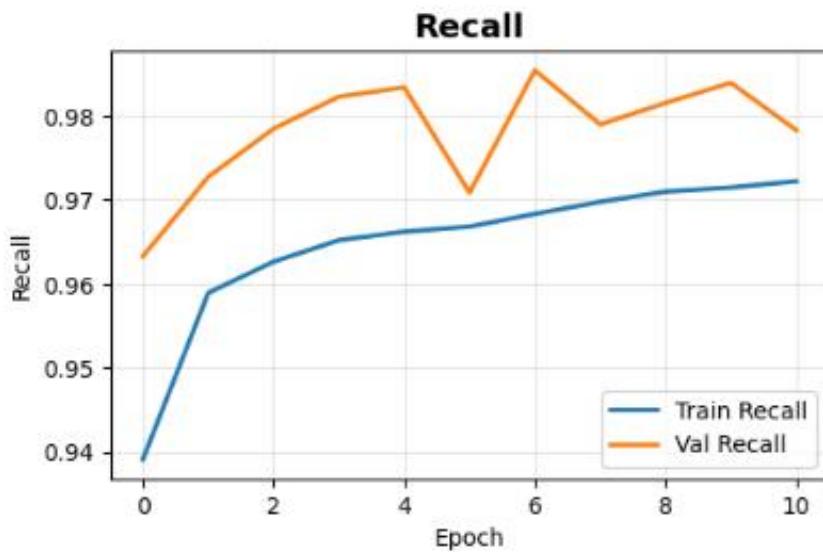
Hình 2.67 Loss graph của model CNN-Hybrid



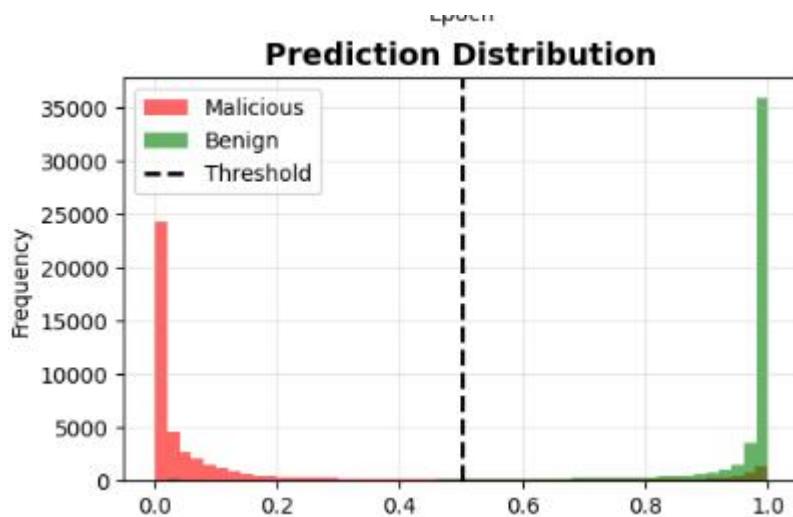
Hình 2.68 Confusion matrix của model CNN-Hybrid



Hình 2.69 Precision Curve CNN-Hybrid



Hình 2.70 Recall Curve của model CNN-Hybrid



Hình 2.71 Prediction distribution của model CNN-Hybrid

Metric	Predicted Probability
Accuracy	0.9210
AUC	0.9732
Precision	0.8860
Recall	0.9690
F1-Score	0.9257
FPR	0.1287
FNR	0.0310

Hình 2.72 Đánh giá tổng quan model CNN-Hybrid với nhiều chỉ số khác nhau

2.3.3 SLM/LLM

2.3.3.1 ALBERT

Cấu hình:

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
tokenizer = AutoTokenizer.from_pretrained("google/mobilebert-uncased", use_fast=True)
albert_model = AutoModelForSequenceClassification.from_pretrained(
    "albert-base-v2",
    num_labels=2)
```

Trong đó:

- AutoModelForSequenceClassification: Dùng để tạo mô hình ALBERT cho bài toán phân loại chuỗi.
- Tokenizer: Token hóa text thành input_ids/attention_mask
- from_pretrained("albert-base-v2"): Tải mô hình ALBERT/tokenizer đã được huấn luyện trước trên dữ liệu ngôn ngữ tự nhiên.
- num_labels=2: Đầu ra có 2 lớp.

```
def compute_metrics(pred):
    labels = pred.label_ids
    preds = np.argmax(pred.predictions, axis=1)
    probs = torch.nn.functional.softmax(torch.tensor(pred.predictions), dim=1)[:, 1].numpy()
    #Dùng softmax để chuyển logits thành xác suất thực tế cho mỗi lớp.
    p, r, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
    try:
        auc = roc_auc_score(labels, probs)
    except:
        auc = 0.0
    try:
        pr_auc = average_precision_score(labels, probs)
    except:
        pr_auc = 0.0
    acc = (preds == labels).mean()
```

```

return {
    "accuracy": acc,
    "precision": p,
    "recall": r,
    "f1": f1,
    "roc_auc": auc,
    "pr_auc": pr_auc }

```

Trong đó:

- pred.label_ids: nhãn thật.
- pred.predictions: logits đầu ra của mô hình.
- Precision: Dự đoán đúng trong số mẫu dự đoán dương tính.
- Recall: Tỷ lệ mẫu dương tính thực tế được mô hình phát hiện đúng.
- F1-score: Trung bình điều hòa giữa Precision và Recall.
- ROC-AUC: Diện tích dưới đường cong ROC, phản ánh khả năng phân biệt giữa 2 lớp.
- PR-AUC: Diện tích dưới đường cong Precision-Recall rất quan trọng trong dữ liệu mất cân bằng.

MAX_LEN = 128

NUM_EPOCHS = 3

LR = 3e-5

BATCH_SIZE = 64

ACCUM_STEPS = 1

training_args = TrainingArguments(

output_dir="/kaggle/working/PhiUSIIL_ALBERT_Checkpoints",

num_train_epochs=NUM_EPOCHS,

per_device_train_batch_size=BATCH_SIZE,

per_device_eval_batch_size=BATCH_SIZE * 2,

gradient_accumulation_steps=ACCUM_STEPS,

learning_rate=LR,

#Đánh giá & lưu mỗi 1000 steps.

eval_strategy="steps",

eval_steps=1000,

save_strategy="steps",

```

save_steps=1000,
load_best_model_at_end=True,
metric_for_best_model="pr_auc",
greater_is_better=True,
save_total_limit=3,
fp16=True,
fp16_full_eval=True,
dataloader_num_workers=4,
dataloader_pin_memory=True,
dataloader_prefetch_factor=2,
gradient_checkpointing=False,
auto_find_batch_size=False,
optim="adamw_torch_fused",
warmup_ratio=0.1,
weight_decay=0.01,
max_grad_norm=1.0,
lr_scheduler_type="cosine",
#Ghi log mỗi 100 bước, giúp theo dõi quá trình học trên console
logging_strategy="steps",
logging_steps=100,
logging_first_step=True,
report_to="none",
disable_tqdm=False,
seed=42,
remove_unused_columns=True,
label_smoothing_factor=0.0,)
```

Trong đó

- num_train_epochs=3: huấn luyện qua 3 epochs.
 - batch_size=64: số mẫu trong mỗi bước huấn luyện.
 - gradient_accumulation_steps=1: tích lũy gradient mỗi 1 batch.
 - learning_rate=3e-5: tốc độ học nhỏ, phù hợp cho fine-tuning mô hình pretrained.
 - load_best_model_at_end=True: tự động nạp mô hình có PR-AUC cao nhất.
 - save_total_limit=3: chỉ giữ lại 3 checkpoint tốt nhất để tiết kiệm bộ nhớ.
-

- fp16=True: bật Mixed Precision Training, giúp huấn luyện nhanh hơn.
- dataloader_*: tối ưu hiệu năng khi đọc dữ liệu.
- optim="adamw_torch_fused": bản tối ưu hóa của AdamW.
- warmup_ratio=0.1: 10% đầu tiên dùng tăng dần learning rate, giúp ổn định huấn luyện.
- weight_decay=0.01: regularization nhẹ để tránh overfitting.
- max_grad_norm=1.0: giới hạn độ lớn gradient.
- lr_scheduler_type="cosine": Cosine Decay Scheduler, giúp learning rate giảm dần theo đường cong cosine, thường hội tụ mượt và ổn định hơn.
- report_to="none": không gửi log lên WandB hay TensorBoard.
- seed=42: cố định random seed để kết quả tái lập.
- remove_unused_columns=True: tự động loại bỏ cột dữ liệu không dùng.
- label_smoothing_factor=0.0: có thể bật 0.1 nếu bị overfitting.

```
data_collator = DataCollatorWithPadding(
    tokenizer=tokenizer,
    padding='longest',)
```

Trong đó:

- tokenizer: dùng chính tokenizer của ALBERT để đảm bảo token hóa thống nhất.
- padding='longest': mỗi batch chỉ pad đến độ dài chuỗi dài nhất trong batch đó, thay vì pad toàn bộ dataset đến MAX_LEN.

```
early_stopping = EarlyStoppingCallback(
    early_stopping_patience=5,
    early_stopping_threshold=0.0005,)
```

Trong đó:

- early_stopping_patience=5: nếu sau 5 lần đánh giá mà metric không cải thiện thì dừng.
- early_stopping_threshold=0.0005: chỉ coi là “cải thiện” nếu metric tốt hơn ít nhất 0.0005.

```
from datasets import load_from_disk
X_train5 = load_from_disk("/content/drive/MyDrive/DeAnTotNghiep/X_train5")
X_test5 = load_from_disk("/content/drive/MyDrive/DeAnTotNghiep/X_test5")
X_train5.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
X_test5.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
```

Trong đó:

- input_ids: mã hóa từng token trong câu.
- attention_mask: mặt nạ chú ý, 1 cho token thật, 0 cho token padding.
- label: nhãn thực là 0 hoặc 1 trong bài toán nhị phân.

trainer = Trainer(

```
model=albert_model,
args=training_args,
train_dataset=X_train5,
eval_dataset=X_test5,
processing_class=tokenizer,
data_collator=data_collator,
compute_metrics=compute_metrics,
callbacks=[early_stopping],)
```

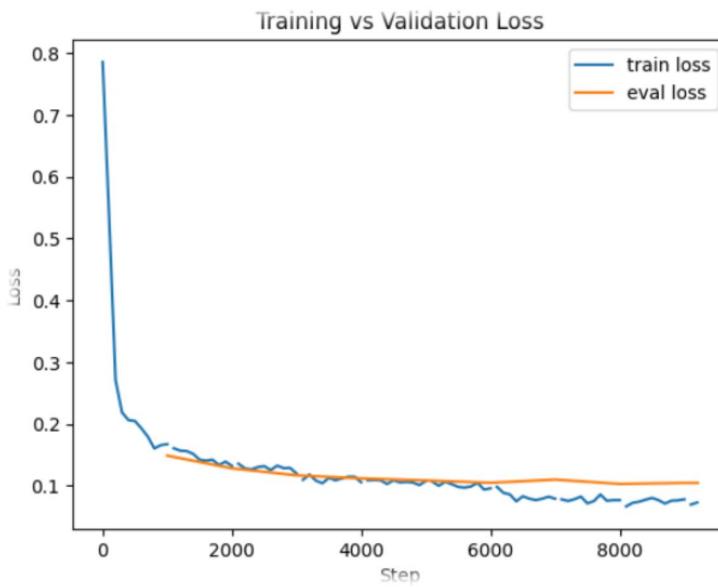
Trong đó:

- model=albert_model: Mô hình ALBERT đã load sẵn cấu trúc + trọng số pretrained.
- args=training_args: Cấu hình huấn luyện định nghĩa trước đó
- train_dataset=X_train5: Dataset huấn luyện.
- eval_dataset=X_test5: Dataset đánh giá.
- processing_class=tokenizer: Tokenizer xử lý input text: ID để mô hình hiểu.
- data_collator=data_collator: Collator giúp tạo batch hợp lệ, pad động mỗi batch.
- compute_metrics=compute_metrics: Hàm tính toán các metric định nghĩa trước đó.
- callbacks=[early_stopping]: Thêm callback dừng sớm khi model ngừng cải thiện.

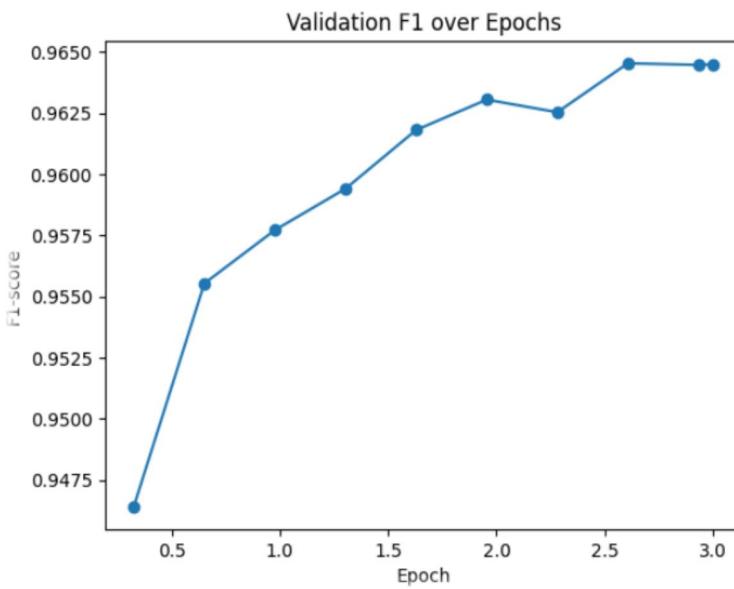
Sau khi quá trình huấn luyện kết thúc ta sẽ tính toán và đánh giá kết quả huấn luyện

```
FINAL EVALUATION ON TEST SET
=====
/usr/local/lib/python3.11/dist-packages/torch/nn/parallel/_functions.py:70: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
warnings.warn(
epoch      : 3.0000
eval_accuracy : 0.9640
eval_f1       : 0.9645
eval_loss     : 0.1049
eval_pr_auc   : 0.9918
eval_precision: 0.9554
eval_recall   : 0.9738
eval_roc_auc   : 0.9927
eval_runtime   : 104.8982
eval_samples_per_second: 936.1650
```

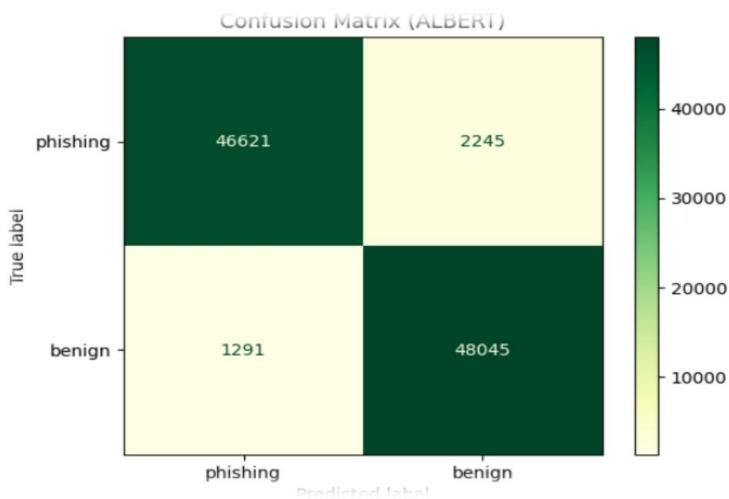
Hình 2.73 Đánh giá bộ test mô hình ALBERT



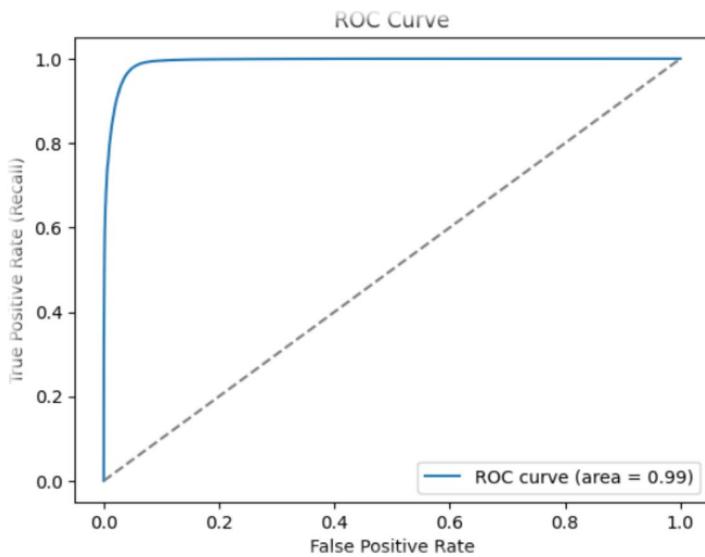
Hình 2.74 Training/validation Loss graph của model ALBERT



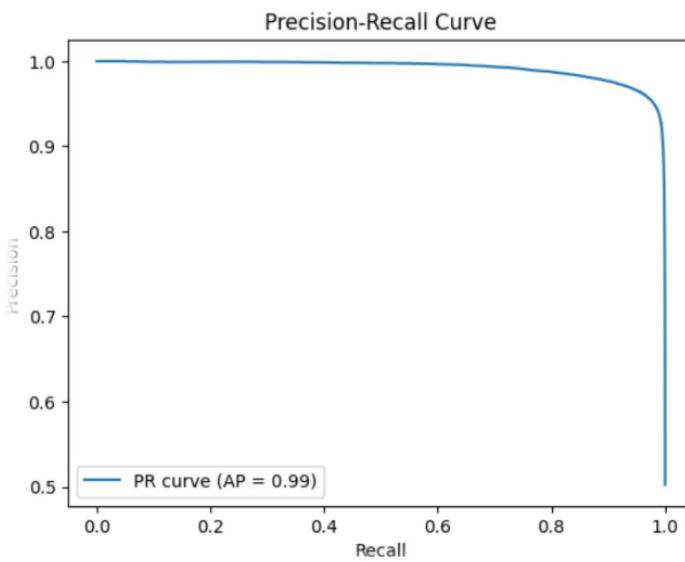
Hình 2.75 Validation F1-score/epoch của model ALBERT



Hình 2.76 Confusion matrix cua model ALBERT



Hình 2.77 ROC Curve cua model ALBERT



Hình 2.78 Precision/Recall Curve của model ALBERT

2.3.3.2 MobileBERT

MobileBERT ngoài mô hình huấn luyện và tokenizer ra thì cấu hình và tham số huấn luyện tương tự như ALBERT

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
mobile_bert_model = AutoModelForSequenceClassification.from_pretrained(
    "google/mobilebert-uncased",
    num_labels=2)
tokenizer = AutoTokenizer.from_pretrained("google/mobilebert-uncased", use_fast=True)
```

Vì có tokenizer riêng nên ta cũng sẽ cần chuẩn bị một bộ dữ liệu train/test riêng cho MobileBERT

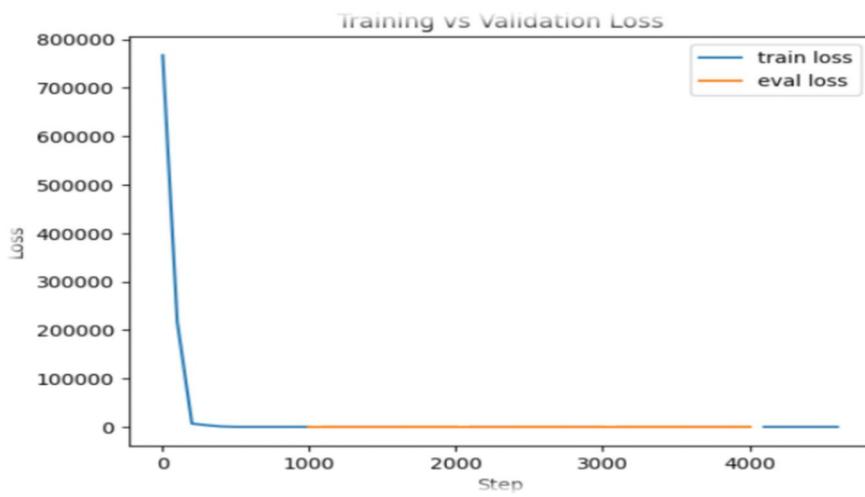
```
from datasets import load_from_disk
X_train6 = load_from_disk("/content/drive/MyDrive/DeAnTotNghiệp/X_train6")
X_test6 = load_from_disk("/content/drive/MyDrive/DeAnTotNghiệp/X_test6")
X_train6.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
X_test6.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
```

```
trainer = Trainer(  
    model=mobile_bert_model,  
    args=training_args,  
    train_dataset=X_train6,  
    eval_dataset=X_test6,  
    processing_class=tokenizer,  
    data_collator=data_collator,  
    compute_metrics=compute_metrics,  
    callbacks=[early_stopping],)
```

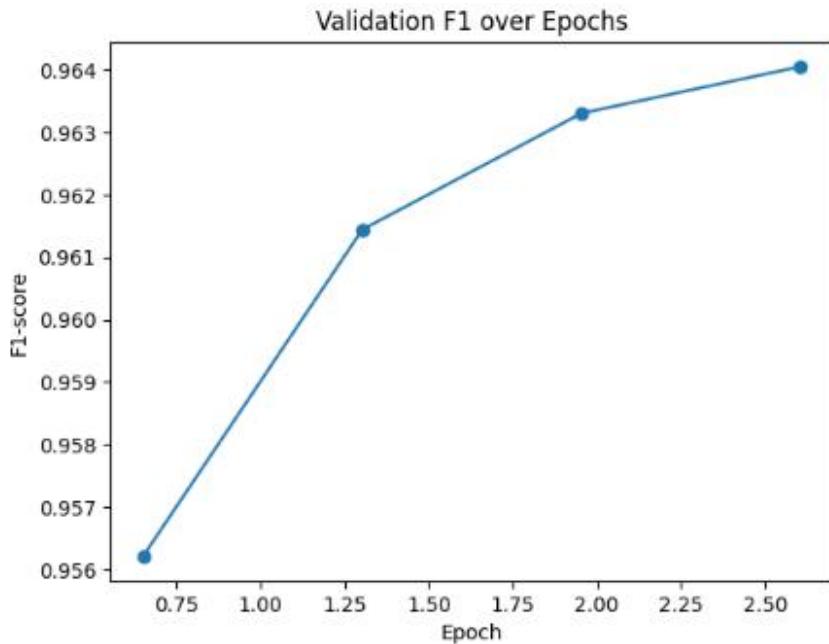
Sau khi quá trình huấn luyện kết thúc ta sẽ tính toán và đánh giá kết quả huấn luyện

```
=====
FINAL EVALUATION ON TEST SET
=====
/usr/local/lib/python3.11/dist-packages/torch/nn/parallel/_functions.py:70: UserWarning: Was asked to gather along dimension 0, but all input
ensors were scalars; will instead unsqueeze and return a vector.
warnings.warn(
epoch : 3.0000
eval_accuracy : 0.9632
eval_f1 : 0.9640
eval_loss : nan
eval_pr_auc : 0.0000
eval_precision : 0.9482
eval_recall : 0.9804
eval_roc_auc : 0.0000
eval_runtime : 69.3610
eval_samples_per_second: 1415.8110
eval_steps_per_second: 2.7680
```

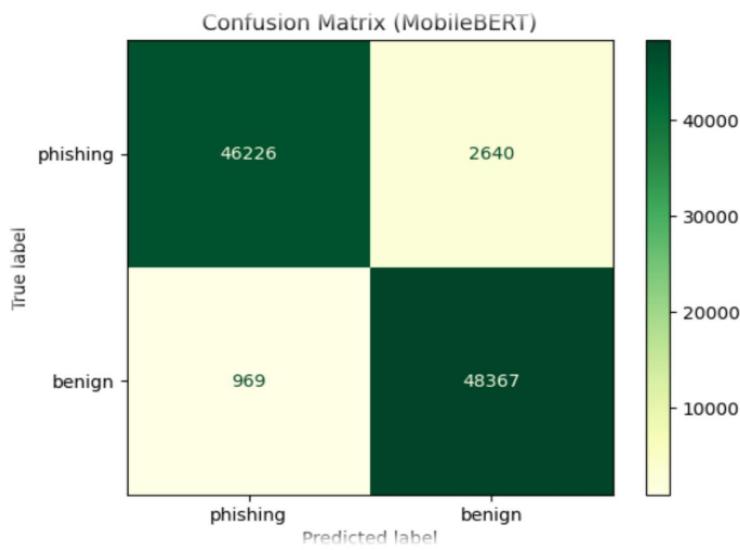
Hình 2.79 Đánh giá model MobileBERT với nhiều tiêu chí đánh giá khác nhau



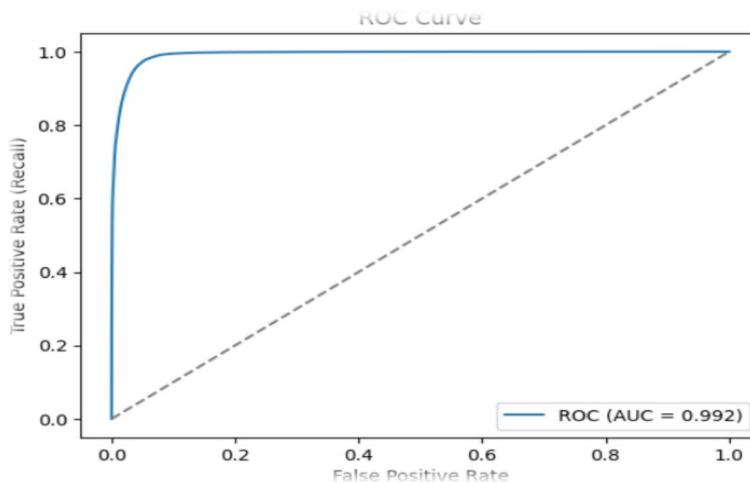
Hình 2.80 Training/Validation Loss graph của model MobileBERT



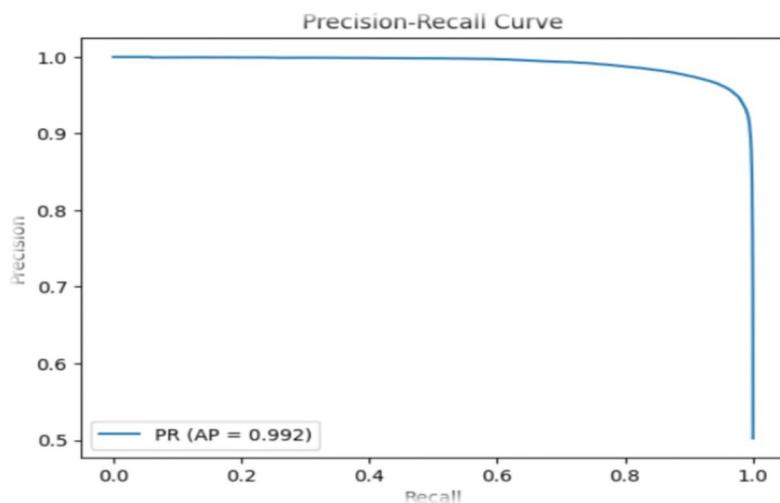
Hình 2.81 Validation F1 over Epoch model MobileBERT



Hình 2.82 Confusion matrix của model MobileBERT



Hình 2.83 ROC Curve của model MobileBERT

*Hình 2.84 Precision/Recall Curve của model MobileBERT*

2.3.3.3 TinyLlama

Tokenizer và cấu hình model TinyLlama

```
model_name = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
```

```
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token # Quan trọng cho padding
```

```
# Load model với 8-bit quantization
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    load_in_8bit=True,
    device_map="auto", # Tự động phân bổ lên 2 GPU
    torch_dtype=torch.float16
)
```

```
# Chuẩn bị model cho k-bit training
model = prepare_model_for_kbit_training(model)
# Cấu hình LoRA - Tối ưu hơn
```

```

lora_config = LoraConfig(
    r=16,                      # Tăng từ 8 -> 16 cho chất lượng tốt hơn
    lora_alpha=32,              # = 2*r
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj"], # Thêm k_proj, o_proj
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)
model = get_peft_model(model, lora_config)
model.print_trainable_parameters()

```

Vì có tokenizer riêng nên ta cũng sẽ cần chuẩn bị một bộ dữ liệu train/test riêng cho TinyLlama

```

df = pd.read_csv("/kaggle/working/PhiUSIIL_final_dataset.csv", sep=";")

# Giảm dataset xuống 50% để chạy nhanh hơn (có thể điều chỉnh)
# Nếu muốn dùng full data, comment dòng dưới
#df = df.sample(frac=0.5, random_state=42).reset_index(drop=True)

df = df.sample(frac=1, random_state=42).reset_index(drop=True)
df["label_text"] = df["label"].map({0: "phishing", 1: "legitimate"})

dataset = Dataset.from_pandas(df)
train_test = dataset.train_test_split(test_size=0.1, seed=42)
train_data, val_data = train_test["train"], train_test["test"]

print(f" Training samples: {len(train_data)}")
print(f" Validation samples: {len(val_data)}")

def format_example(example):
    messages = [
        {"role": "system", "content": "You are a cybersecurity assistant that classifies URLs as
phishing or legitimate."},
        {"role": "user", "content": f"Classify the following URL:\n{example['url']}"},
    ]

```

```

        {"role": "assistant", "content": example["label_text"]},
    ]
text = tokenizer.apply_chat_template(messages, tokenize=False,
add_generation_prompt=False)
return {"text": text}

train_data = train_data.map(format_example, num_proc=4) # Dùng 4 cores để xử lý song song
val_data = val_data.map(format_example, num_proc=4)

training_args = TrainingArguments(
    output_dir=CHECKPOINT_DIR,
    # Batch size & Gradient Accumulation
    per_device_train_batch_size=16,      # Tăng từ 8 -> 16
    per_device_eval_batch_size=32,
    gradient_accumulation_steps=2,      # Giảm từ 4 -> 2
    # => Effective batch = 16 * 2 * 2 GPU = 64

    # Learning rate & Epochs
    learning_rate=3e-4,                 # Tăng LR để học nhanh hơn
    num_train_epochs=1,
    warmup_ratio=0.03,                  # Warmup 3% steps

    # Optimizer
    optim="adamw_torch_fused",          # Optimizer nhanh hơn

    # Precision
    fp16=True,
    bf16=False,

    # Logging & Checkpointing
    logging_steps=100,                  # Log thường xuyên hơn
    save_steps=500,                     # Lưu checkpoint mỗi 500 steps

```

```

save_total_limit=3,           # Chỉ giữ 3 checkpoint gần nhất
# Evaluation
eval_strategy="no",          # Tắt eval
load_best_model_at_end=False, # Tắt để nhanh hơn

# Performance
dataloader_num_workers=4,     # Dùng 4 workers
dataloader_pin_memory=True,
gradient_checkpointing=True,   # Giảm memory, tăng tốc

# Multi-GPU
ddp_find_unused_parameters=False,

# Misc
report_to="none",
remove_unused_columns=True,

# Resume
resume_from_checkpoint=find_latest_checkpoint(CHECKPOINT_DIR) if
RESUME_FROM_CHECKPOINT else None,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=X_train,
    eval_dataset=X_val,
    data_collator=DataCollatorForLanguageModeling(tokenizer, mlm=False),
)

print("\n" + "*50)
print(" BẮT ĐẦU TRAINING")
print("*50)
print(f" Số GPU được sử dụng: {torch.cuda.device_count()}")

```

```

print(f"Trainable params: {model.print_trainable_parameters()}")
# Kiểm tra nếu có checkpoint
latest_checkpoint = find_latest_checkpoint(CHECKPOINT_DIR)
if RESUME_FROM_CHECKPOINT and latest_checkpoint:
    print(f"Tiếp tục từ checkpoint: {latest_checkpoint}")
    trainer.train(resume_from_checkpoint=latest_checkpoint)
else:
    print("Bắt đầu training mới")
    trainer.train()

```

Sau khi quá trình huấn luyện kết thúc ta sẽ tính toán và đánh giá kết quả huấn luyện

```

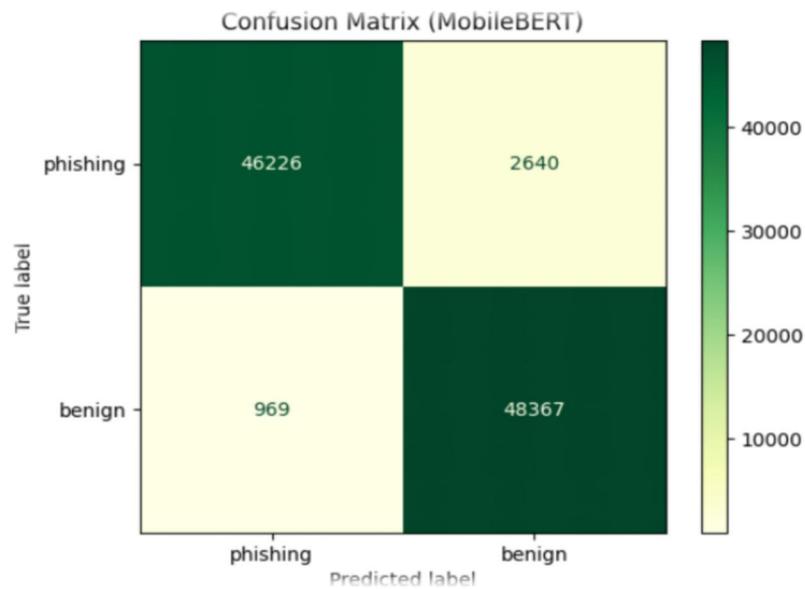
=====
[ FINAL EVALUATION ON TEST SET
=====
usr/local/lib/python3.11/dist-packages/torch/nn/parallel/_functions.py:70: UserWarning: Was asked to gather along dimension 0, but all input
ensors were scalars; will instead unsqueeze and return a vector.
warnings.warn(
epoch : 3.0000
eval_accuracy : 0.9632
eval_f1 : 0.9640
eval_loss : nan
eval_pr_auc : 0.0000
eval_precision : 0.9482
eval_recall : 0.9804
eval_roc_auc : 0.0000
eval_runtime : 69.3610
eval_samples_per_second: 1415.8110
eval_steps_per_second: 2.7680

```

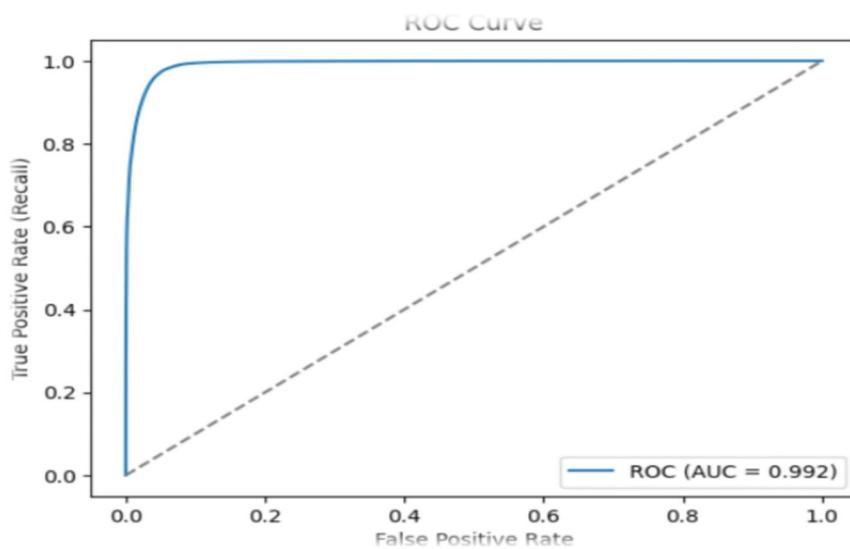
Hình 2.85 Đánh giá model TinyLlama với nhiều tiêu chí đánh giá khác nhau



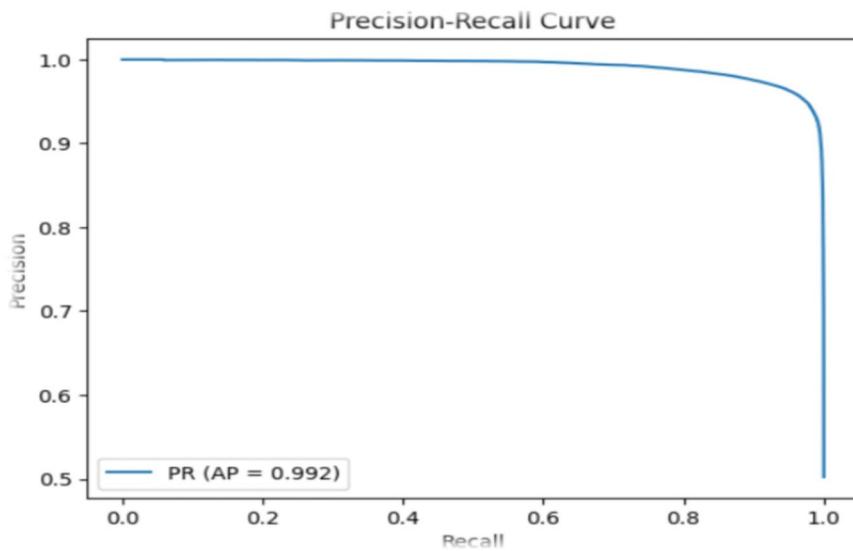
Hình 2.86 Training/Validation Loss graph của model TinyLlama



Hình 2.87 Confusion matrix của model TinyLlama



Hình 2.88 ROC Curve của model TinyLlama



Hình 2.89 Precision/Recall Curve của model TinyLlama

2.4 Giải thích mô hình với SHAP/LIME

2.3.1 XGBoost

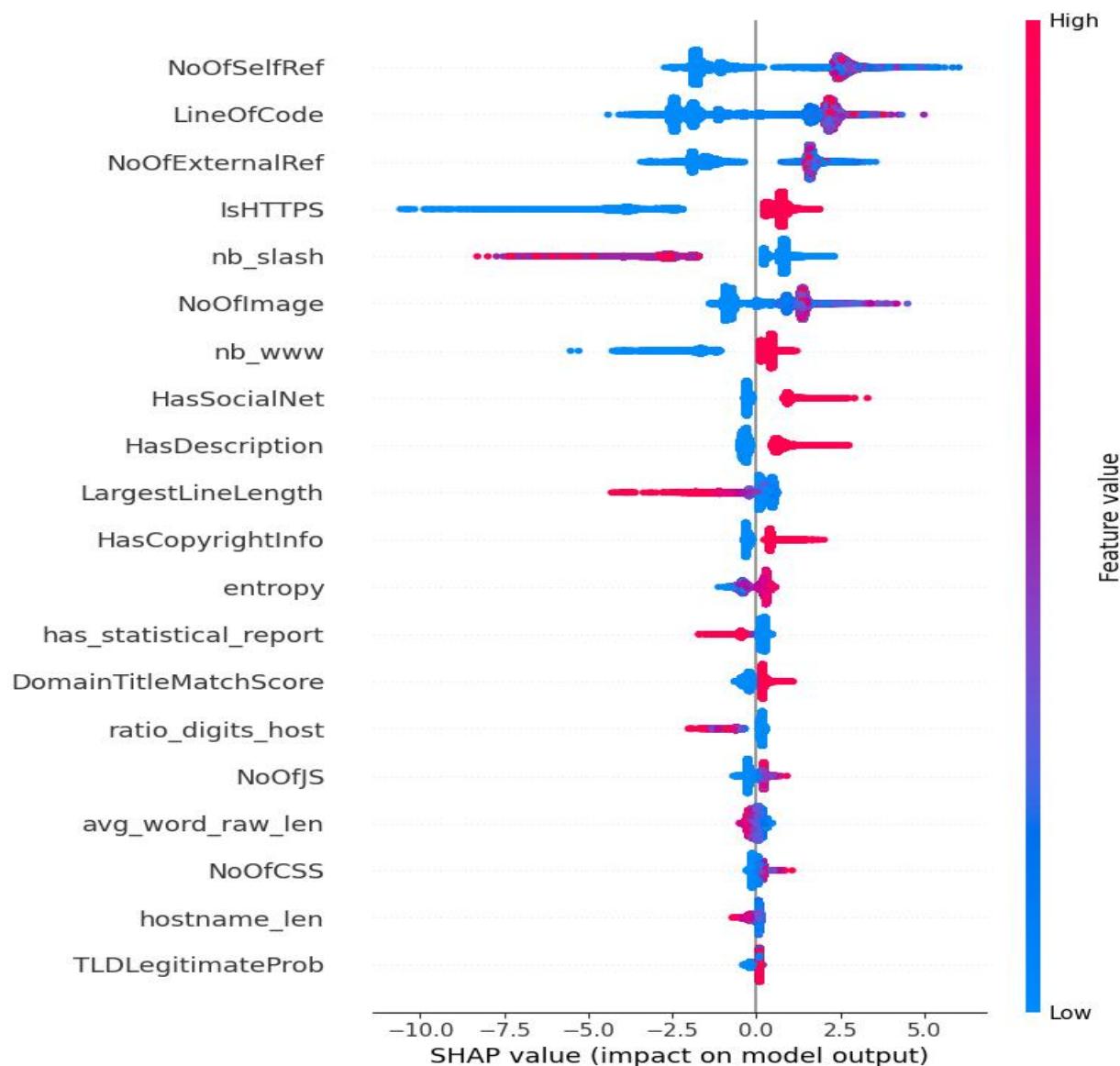
Giải thích mô hình XGBoost với đặc trưng số:

- Chuẩn bị dữ liệu:

```
import shap
xgb = xgb.XGBClassifier()
X_train1 = pd.read_csv("/content/drive/MyDrive/DeAnTotNghiep/X_train_81.csv")
X_test1 = pd.read_csv("/content/drive/MyDrive/DeAnTotNghiep/X_test_81.csv")
Y_train1 = pd.read_csv("/content/drive/MyDrive/DeAnTotNghiep/Y_train_81.csv")
Y_test1 = pd.read_csv("/content/drive/MyDrive/DeAnTotNghiep/Y_test_81.csv")
xgb.load_model('/content/drive/MyDrive/DeAnTotNghiep/PhiUSIIL_XGB_Numerical.json')
explainer_xgb = shap.TreeExplainer(xgb)
shap_values_xgb = explainer_xgb.shap_values(X_test1)
```

- Giải thích mô hình XGB numerical với SHAP:

```
shap.summary_plot(shap_values_xgb, X_test1)
```



Hình 2.90 Tính SHAP value những đặc trưng tác động đến model XGBoost

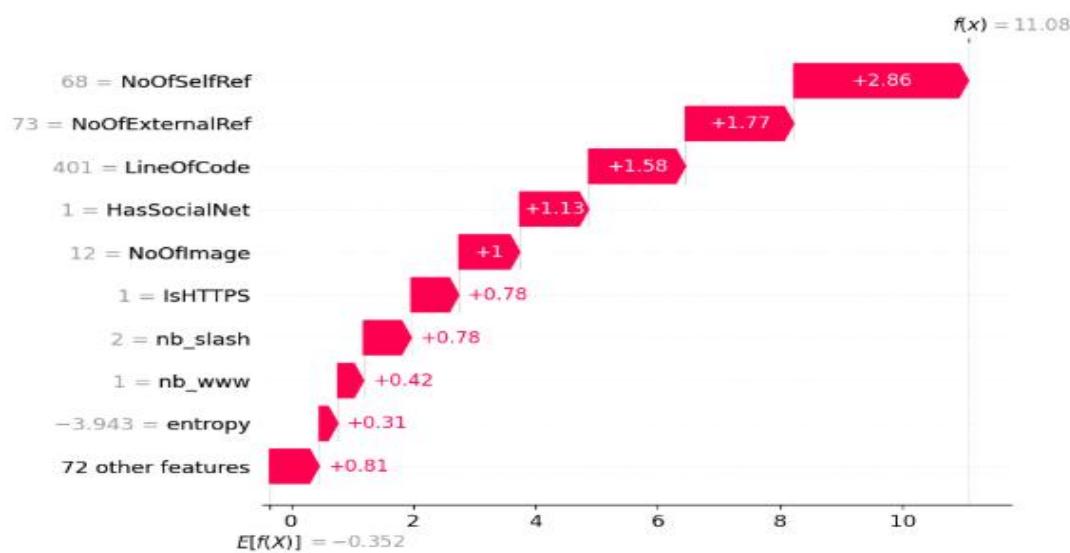
```
shap.initjs()
```

```
shap.force_plot(explainer_xgb.expected_value, shap_values_xgb[57,:], X_test1.iloc[57,:])
```



Hình 2.91 Vẽ `shap.force_plot shap_values_xgb[57,:]` XGB model

```
shap.plots.waterfall(  
    shap.Explanation(  
        values=shap_values_xgb[57],  
        base_values=explainer_xgb.expected_value,  
        data=X_test1.iloc[57],  
        feature_names=X_test1.columns  
    )  
)
```



Hình 2.92 Vẽ `shap.plots.waterfall XGB model`

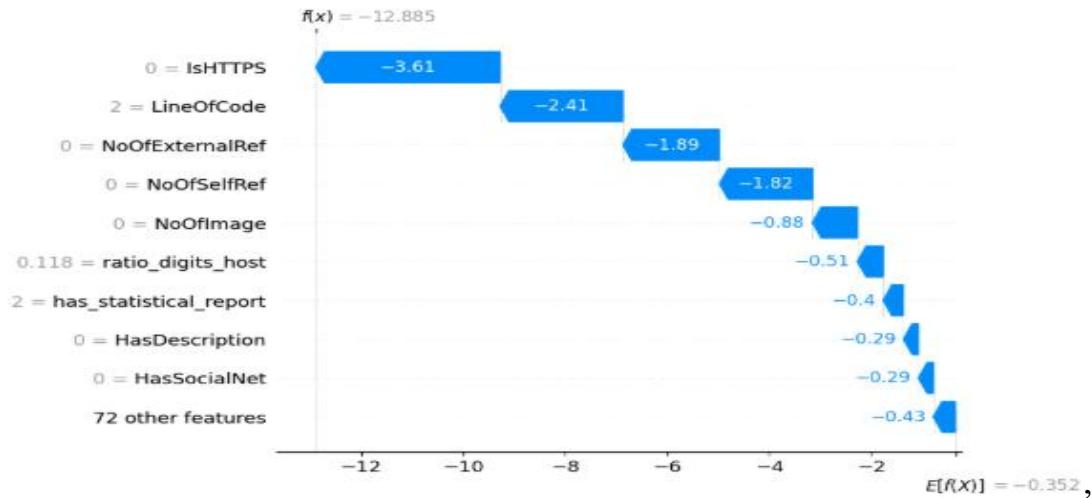
```
shap.initjs()
```

```
shap.force_plot(explainer_xgb.expected_value, shap_values_xgb[38,:], X_test1.iloc[57,:])
```



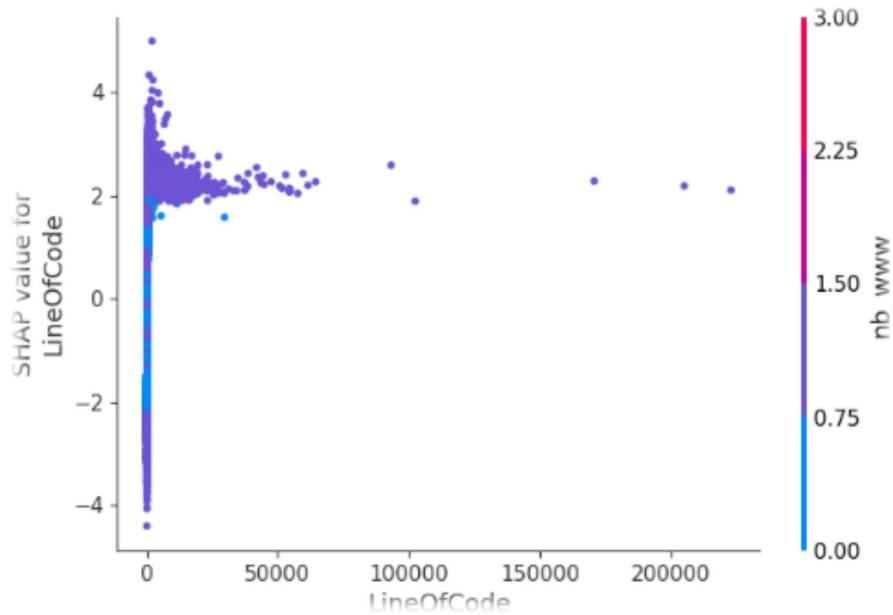
Hình 2.93 Vẽ `shap.force_plot shap_values_xgb[38,:]` XGBoost model

```
shap.plots.waterfall(  
    shap.Explanation(  
        values=shap_values_xgb[38],  
        base_values=explainer_xgb.expected_value,  
        data=X_test1.iloc[38],  
        feature_names=X_test1.columns  
    )  
)
```



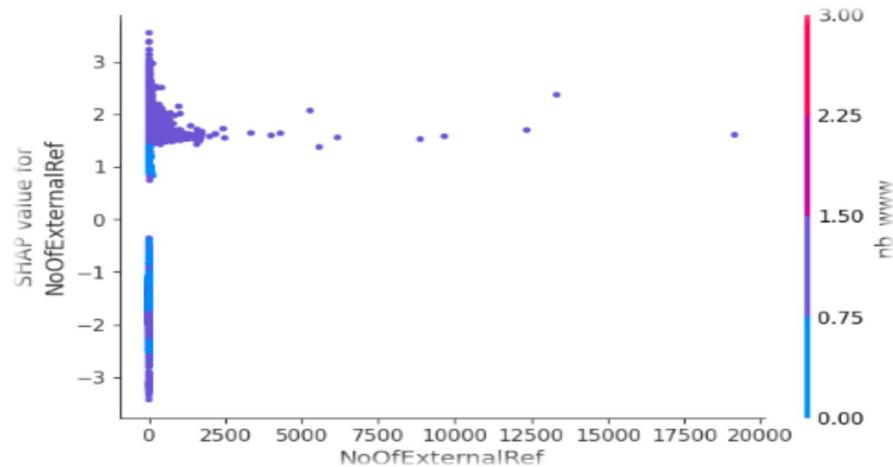
Hình 2.94 Vẽ `shap.plots.waterfall` XGBoost model

```
shap.dependence_plot("LineOfCode", shap_values_xgb, X_test1)
```



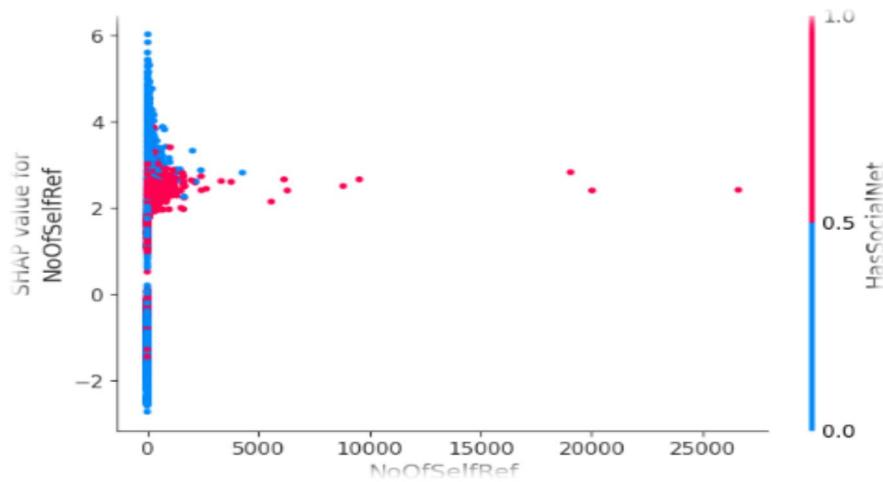
Hình 2.95 Vẽ `shap.dependence_plot` đặc trưng `LineOfCode` XGBoost model

```
shap.dependence_plot("NoOfExternalRef", shap_values_xgb, X_test1)
```



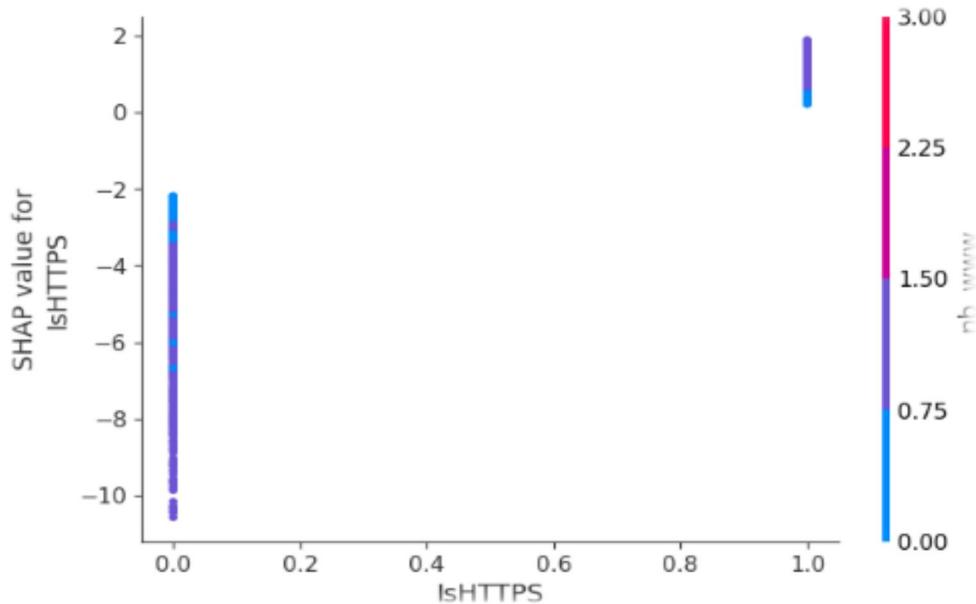
Hình 2.96 Vẽ `shap.dependence_plot` đặc trưng `NoOfExternalRef` XGBoost model

```
shap.dependence_plot("NoOfSelfRef", shap_values_xgb, X_test1)
```



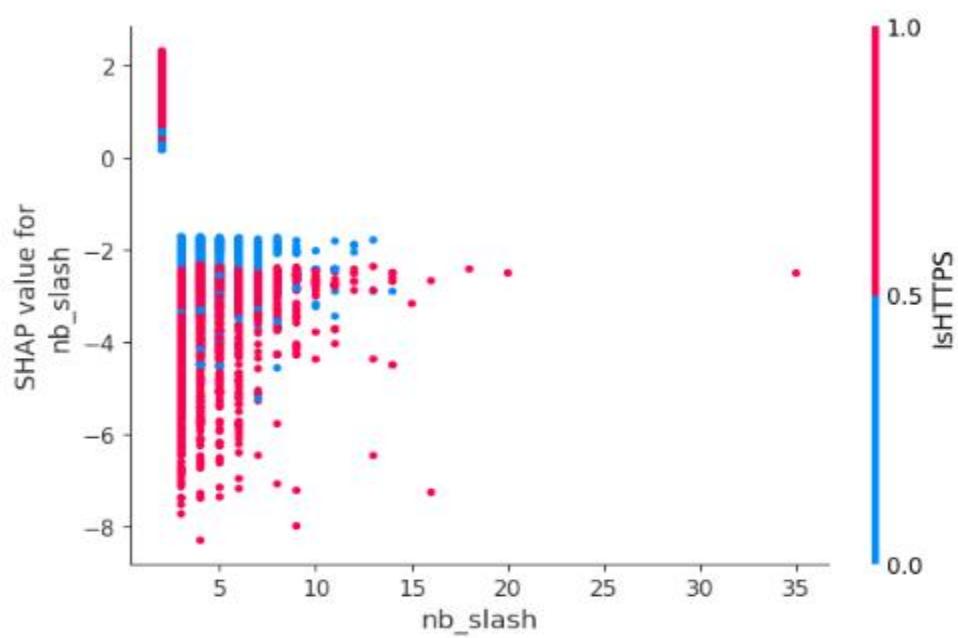
Hình 2.97 Vẽ shap.dependence_plot đặc trưng NoOfSelfRef XGBoost model

```
shap.dependence_plot("IsHTTPS", shap_values_xgb, X_test1)
```



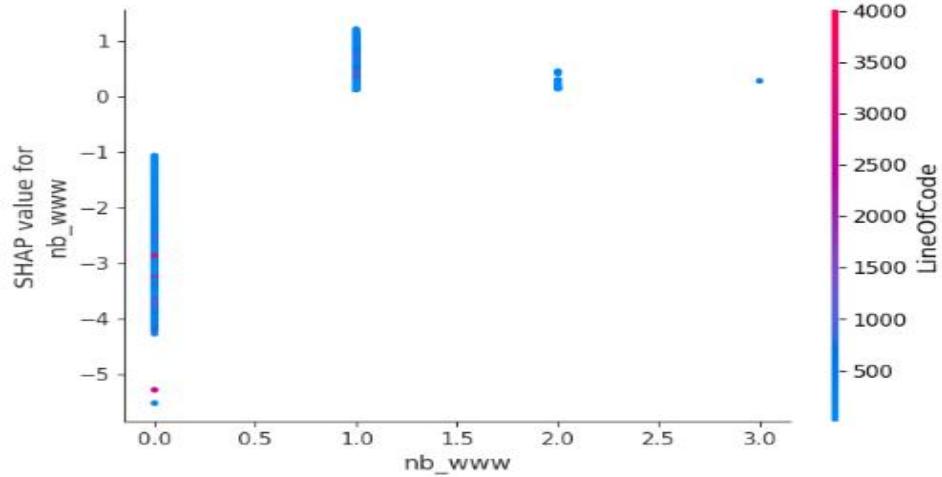
Hình 2.98 Vẽ shap.dependence_plot IsHTTPS XGBoost model

```
shap.dependence_plot("nb_slash", shap_values_xgb, X_test1)
```



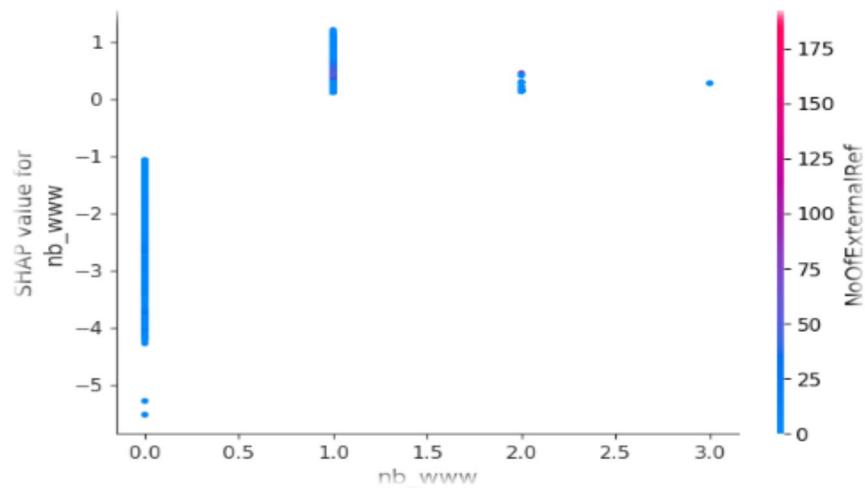
Hình 2.99 Vẽ shap.dependence_plot đặc trưng nb_slash XGBoost model

```
shap.dependence_plot("nb_www", shap_values_xgb, X_test1, interaction_index="LineOfCode")
```



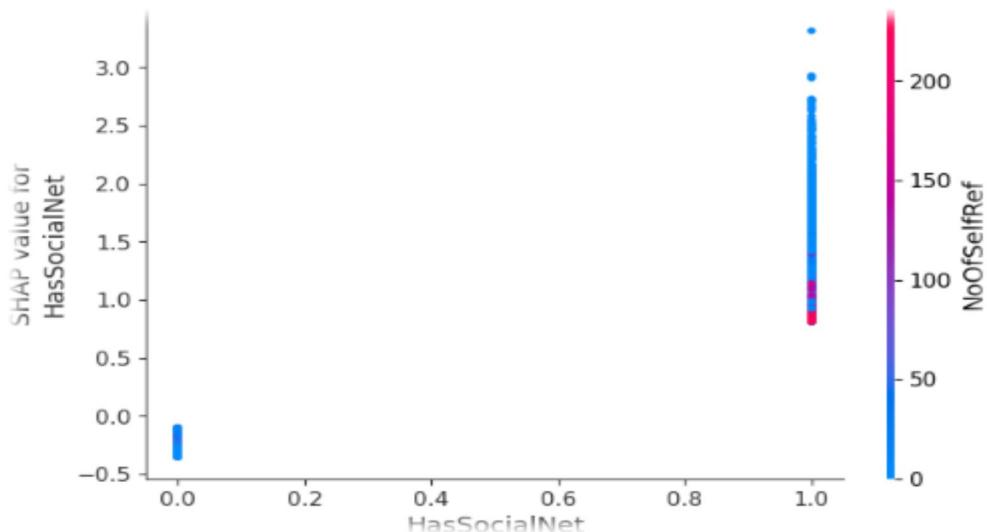
Hình 2.100 Vẽ shap.dependence_plot đặc trưng nb_www x LineOfCode XGBoost model

```
shap.dependence_plot("nb_www", shap_values_xgb, X_test1,
interaction_index="NoOfExternalRef")
```



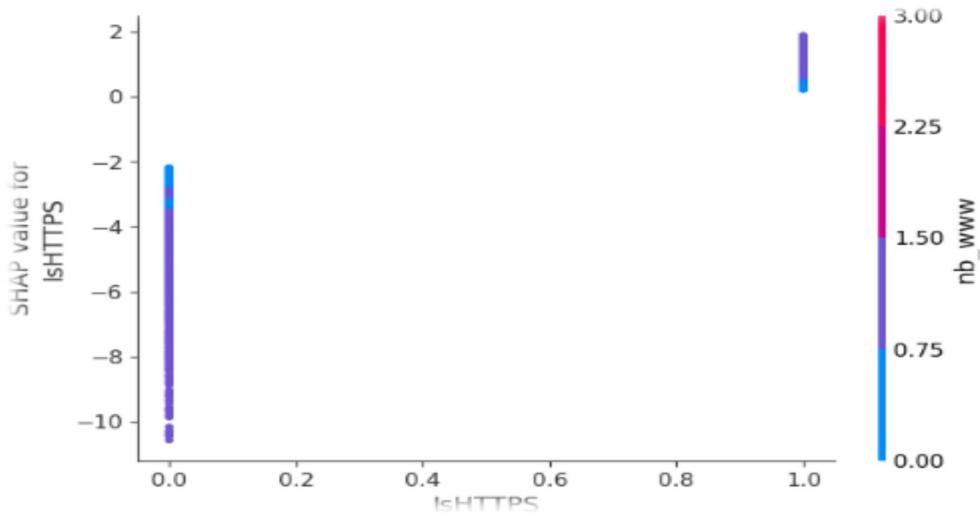
Hình 2.101 Vẽ shap.dependence_plot đặc trưng nb_www x NoOfExternalRef XGBoost model

```
shap.dependence_plot("HasSocialNet", shap_values_xgb, X_test1,
interaction_index="NoOfSelfRef")
```



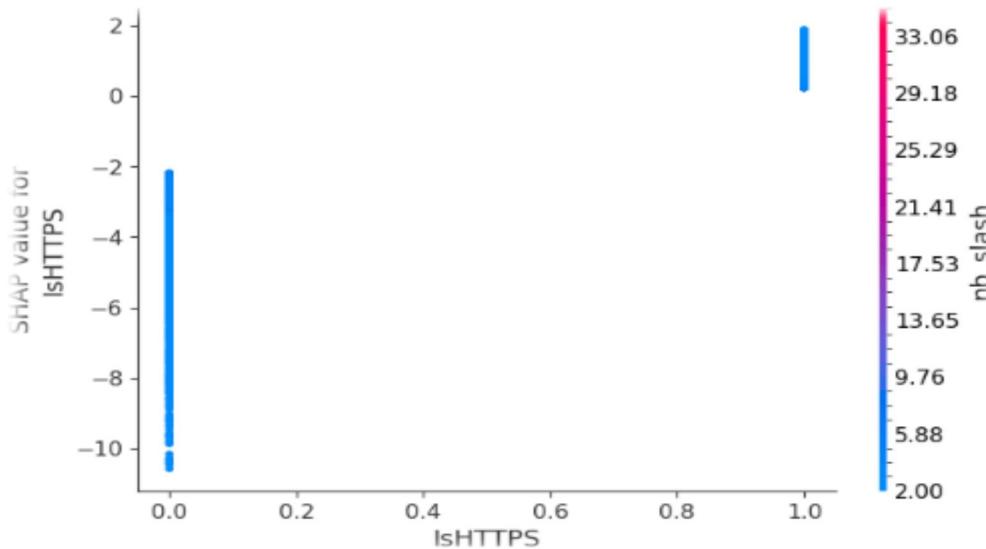
Hình 2.102 Vẽ shap.dependence_plot đặc trưng HasSocialNet x NoOfExternalRef XGBoost model

```
shap.dependence_plot("IsHTTPS", shap_values_xgb, X_test1, interaction_index="nb_www")
```



Hình 2.103 Vẽ shap.dependence_plot IsHTTPS x nb_www XGBoost model

```
shap.dependence_plot("IsHTTPS", shap_values_xgb, X_test1, interaction_index="nb_slash")
```



Hình 2.104 Vẽ shap.dependence_plot đặc trưng IsHTTPS x nb_slash XGBoost model

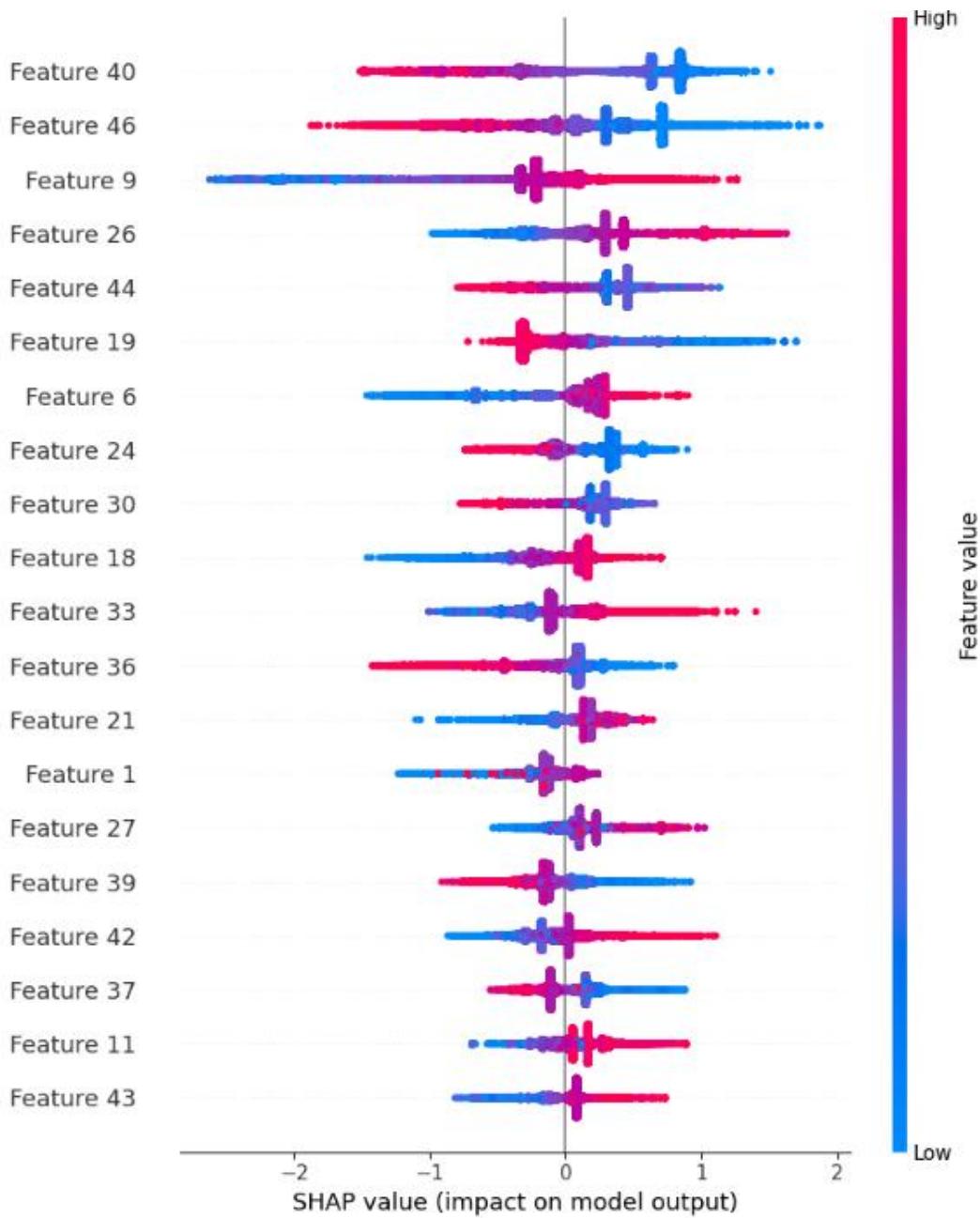
Giải thích mô hình XGBoost URL embedding:

- Chuẩn bị dữ liệu:

```
import shap
import xgboost as xgb
xgb = xgb.XGBClassifier()
xgb.load_model('/content/drive/MyDrive/DeAnTotNghiep/PhiUSIIL_XGB_URL_EMBEDDING.js
on')
explainer_xgb = shap.TreeExplainer(xgb)
shap_values_xgb = explainer_xgb.shap_values(X_test2)
```

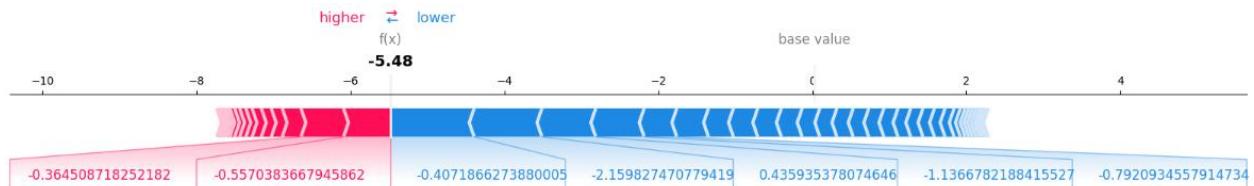
- Giải thích mô hình XGBoost embedding với SHAP:

```
shap.summary_plot(shap_values_xgb, X_test2)
```



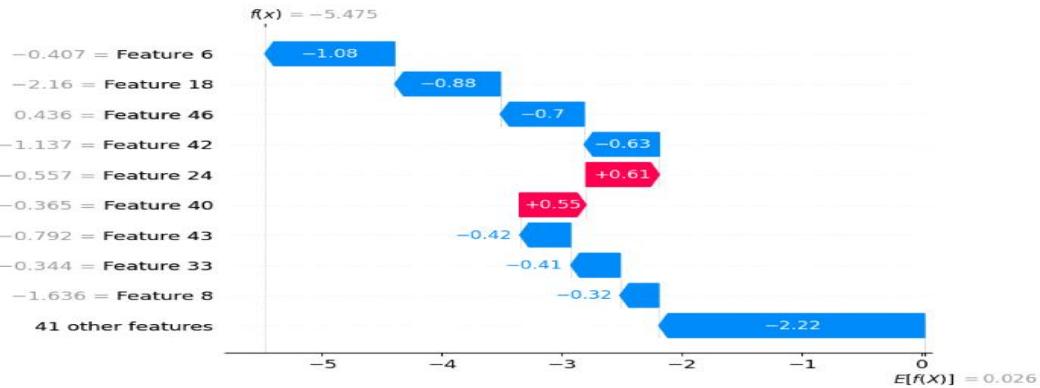
Hình 2.105 Tính shap value XGBoost embedding model

```
shap.initjs()
shap.force_plot(
    explainer_xgb.expected_value,
    shap_values_xgb[68, :].astype(float),
    X_test2[68, :].astype(float),
    matplotlib=True
)
```



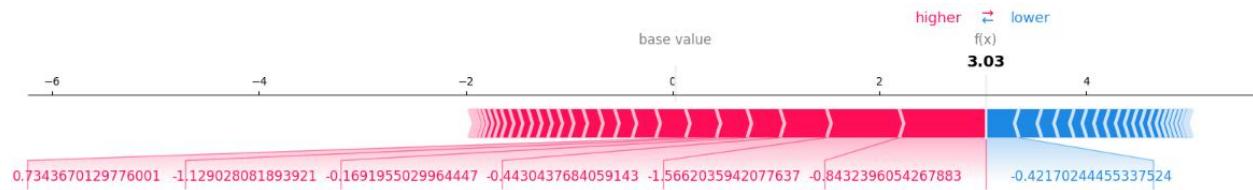
Hình 2.106 Vẽ shap.force_plot XGBoost embedding

```
shap.plots.waterfall(
    shap.Explanation(
        values = shap_values_xgb[68],
        base_values = explainer_xgb.expected_value,
        data = X_test2[68],
        feature_names = [f'Feature {i}' for i in range(X_test2.shape[1])]
    )
)
```



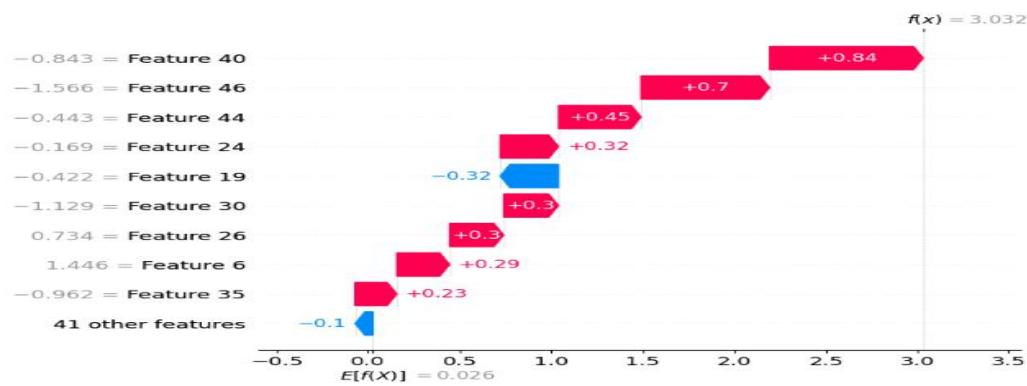
Hình 2.107 Vẽ shap.plots.waterfall XGB embedding mode

```
shap.initjs()
shap.force_plot(
    explainer_xgb.expected_value,
    shap_values_xgb[150, :].astype(float),
    X_test2[150, :].astype(float),
    matplotlib=True
)
```



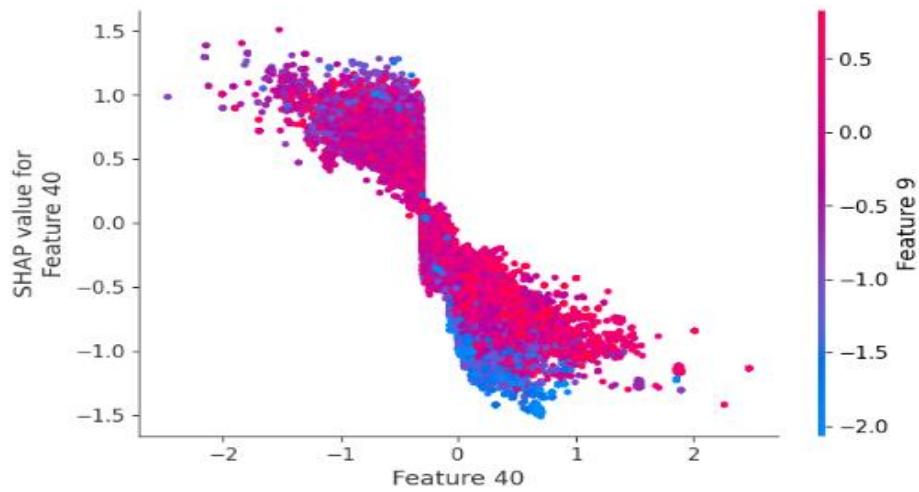
Hình 2.108 Vẽ shap.force_plot XGBoost embedding

```
shap.plots.waterfall(
    shap.Explanation(
        values = shap_values_xgb[150],
        base_values = explainer_xgb.expected_value,
        data = X_test2[150],
        feature_names = [f'Feature {i}' for i in range(X_test2.shape[1])]
    )
)
```



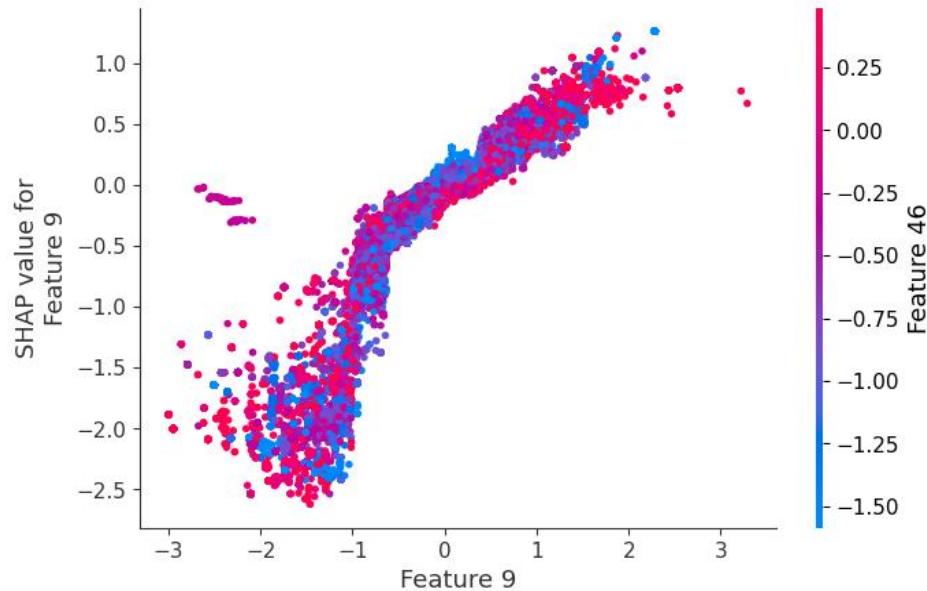
Hình 2.109 Vẽ shap.plots.waterfall XGB embedding model

```
shap.dependence_plot("Feature 40", shap_values_xgb, X_test2)
```



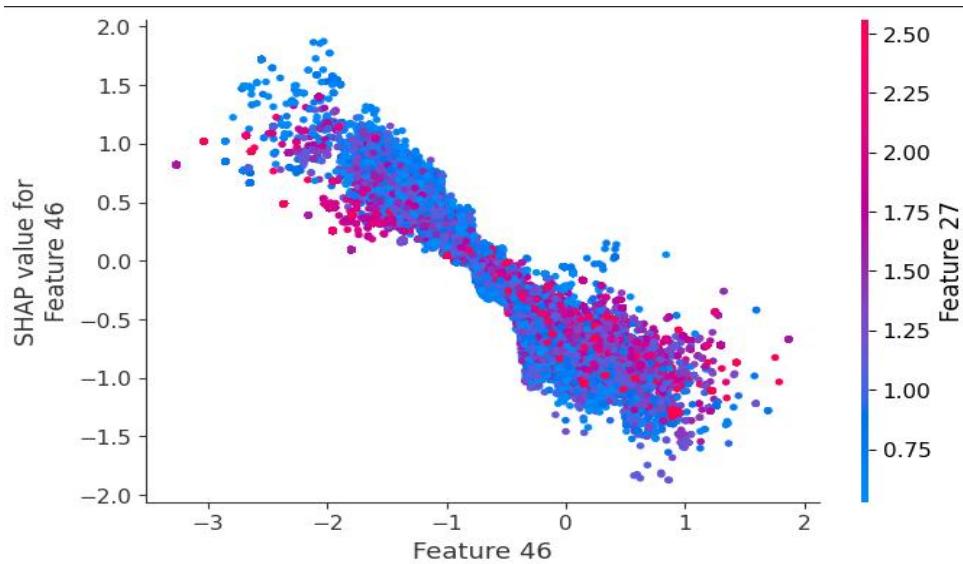
Hình 2.110 Vẽ shap.dependence_plot Feature 40 XGB embedding model

```
shap.dependence_plot("Feature 9", shap_values_xgb, X_test2)
```



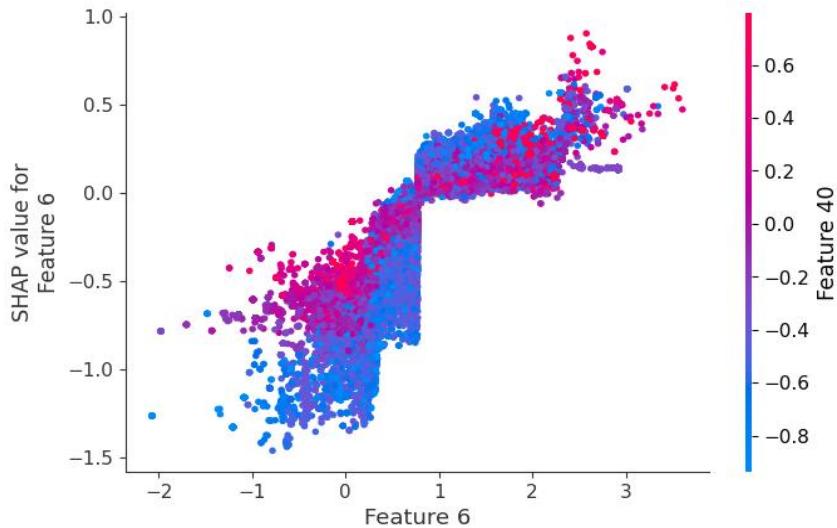
Hình 2.111 Vẽ shap.dependence_plot Feature 9 XGB embedding model

```
shap.dependence_plot("Feature 46", shap_values_xgb, X_test2)
```



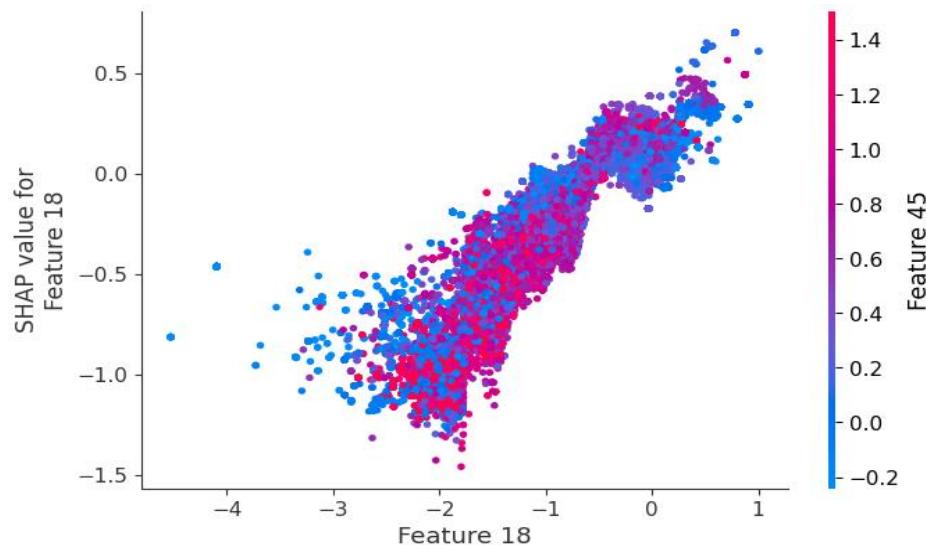
Hình 2.112 Vẽ shap.dependence_plot Feature 46 XGBoost embedding model

```
shap.dependence_plot("Feature 6", shap_values_xgb, X_test2)
```



Hình 2.113 Vẽ shap.dependence_plot Feature 6 XGB embedding model

```
shap.dependence_plot("Feature 18", shap_values_xgb, X_test2)
```



Hình 2.114 Vẽ shap.dependence_plot Feature 18 XGBoost embedding model

2.3.2 RandomForest

Giải thích mô hình RandomForest với đặc trưng số

- Load model và chuẩn bị dữ liệu:

```
import joblib
RF = joblib.load('/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_RF_NUMERICAL.pkl')
X_train1 = pd.read_csv("/content/drive/MyDrive/DeAnTotNghiệp/X_train1.csv")
X_test1 = pd.read_csv("/content/drive/MyDrive/DeAnTotNghiệp/X_test1.csv")
Y_train1 = pd.read_csv("/content/drive/MyDrive/DeAnTotNghiệp/Y_train1.csv")
Y_test1 = pd.read_csv("/content/drive/MyDrive/DeAnTotNghiệp/Y_test1.csv")

import shap
explainer_rf = shap.TreeExplainer(
    RF,
    X_test1,
```

```

feature_perturbation="interventional",
model_output="probability"
)

shap_values_rf = explainer_rf.shap_values(X_test1)

import shap
import joblib
rf = joblib.load('/content/drive/MyDrive/DeAnTotNghiep/PhiUSIIL_RF_URL_EMBEDDING.pkl')
shap_values_rf =
np.load("/content/drive/MyDrive/DeAnTotNghiep/PhiUSIIL_RF_URL_EMBEDDING_SHAP/
shap_values_rf.npy")
explainer_rf = shap.TreeExplainer(
    rf,
    X_test2,
    feature_perturbation="interventional",
    model_output="probability"
)

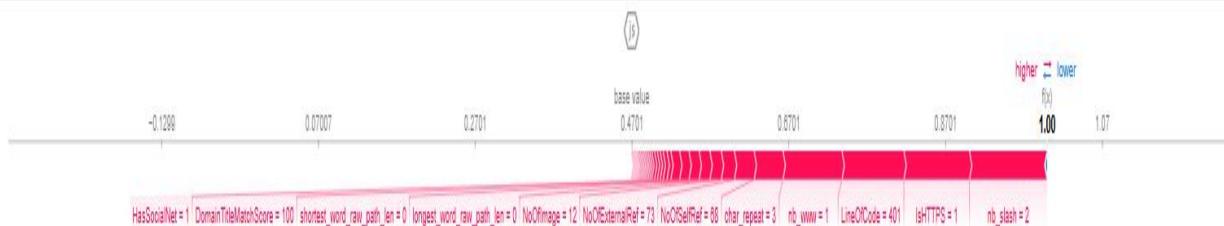
```

- Giải thích mô hình RF với SHAP:

```

shap.initjs()
shap.force_plot(explainer_rf.expected_value[1], # base value cho class 1
                shap_values_rf[57, :, 1], # SHAP values cho sample 57, class 1
                X_test1.iloc[57])

```

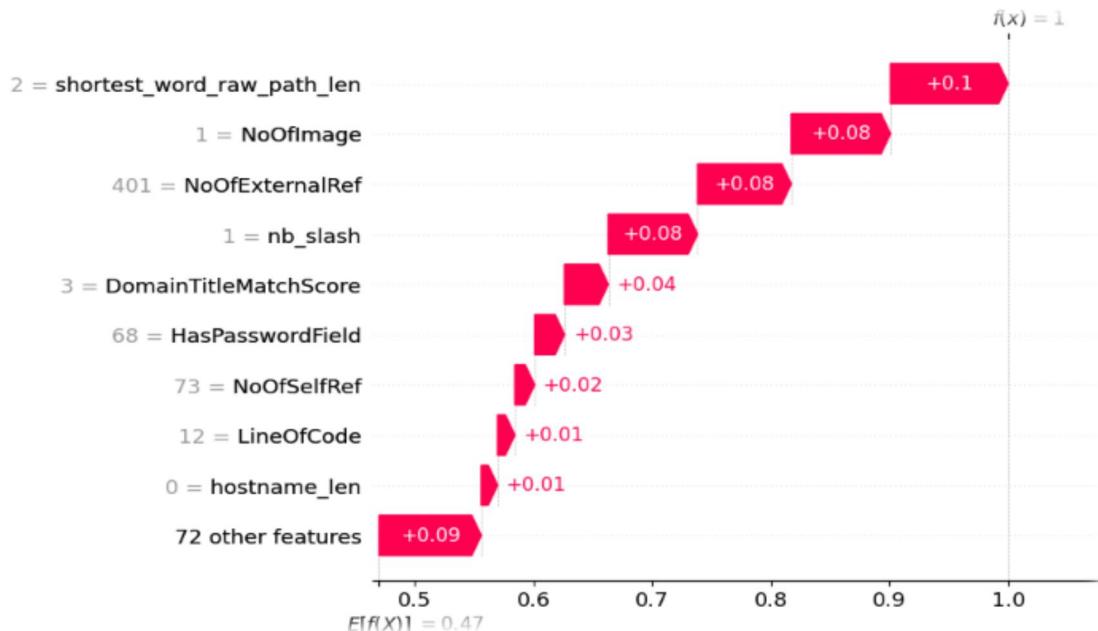


Hình 2 115 Vẽ shap.force_plot RF model

```

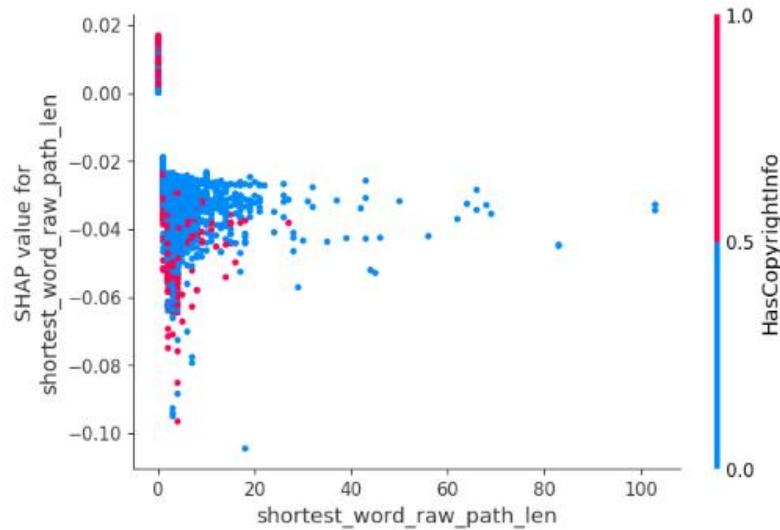
shap.plots.waterfall(
    shap.Explanation(
        values = shap_values_rf[57, :, 1],
        base_values = explainer_rf.expected_value[1],
        data = X_test1.iloc[57],
        feature_names = feature_names
    )
)

```



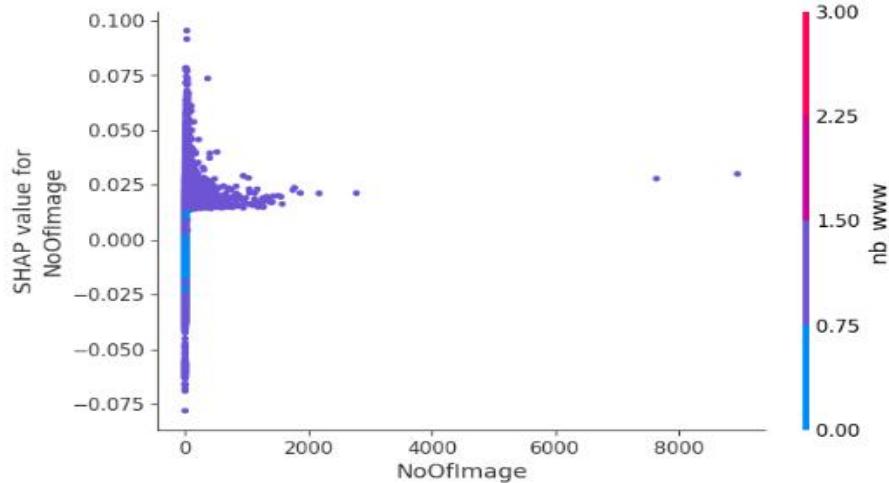
Hình 2.116 Vẽ shap.plots.waterfall Random Foreset numerical model

```
shap_values_rf_class1 = shap_values_rf[:, :, 1]
shap.dependence_plot("shortest_word_raw_path_len", shap_values_rf_class1, X_test1)
```



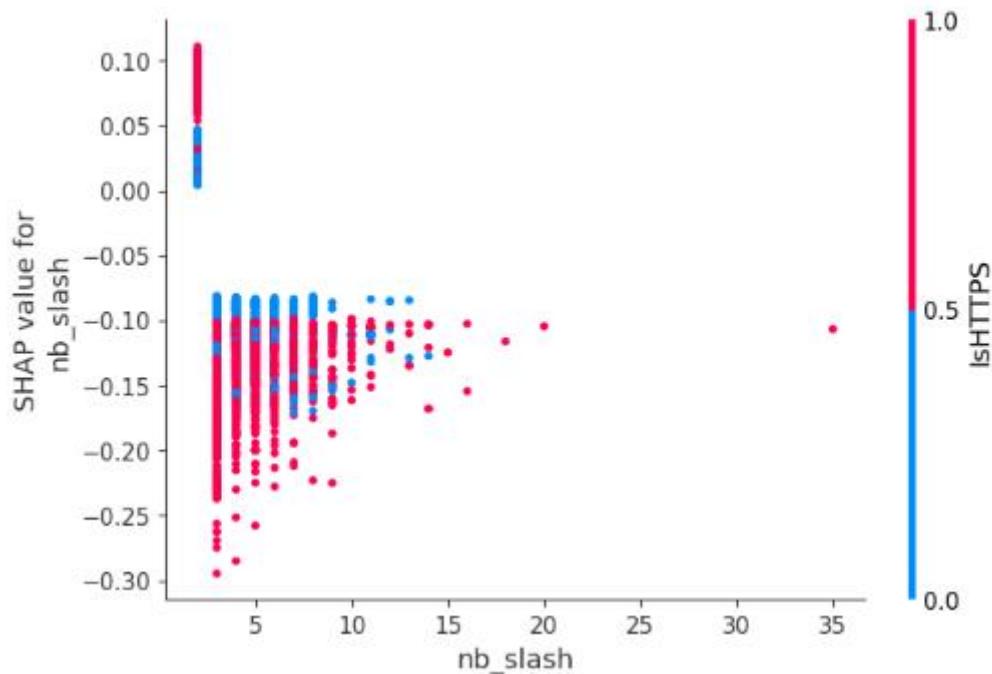
Hình 2.117 Vẽ shap.dependence_plot shortest_word_raw_path_len RF model

```
shap.dependence_plot("NoOfImage", shap_values_rf_class1, X_test1)
```



Hình 2.118 Vẽ shap.dependence_plot NoOfImage RF model

```
shap.dependence_plot("nb_slash", shap_values_rf_class1, X_test1)
```



Hình 2.119 Vẽ `shap.dependence_plot` `nb_slash` RF model

```
shap.initjs()
shap.force_plot(explainer_rf.expected_value[0],
                shap_values_rf[225, :, 1],
                X_test1.iloc[225])
```

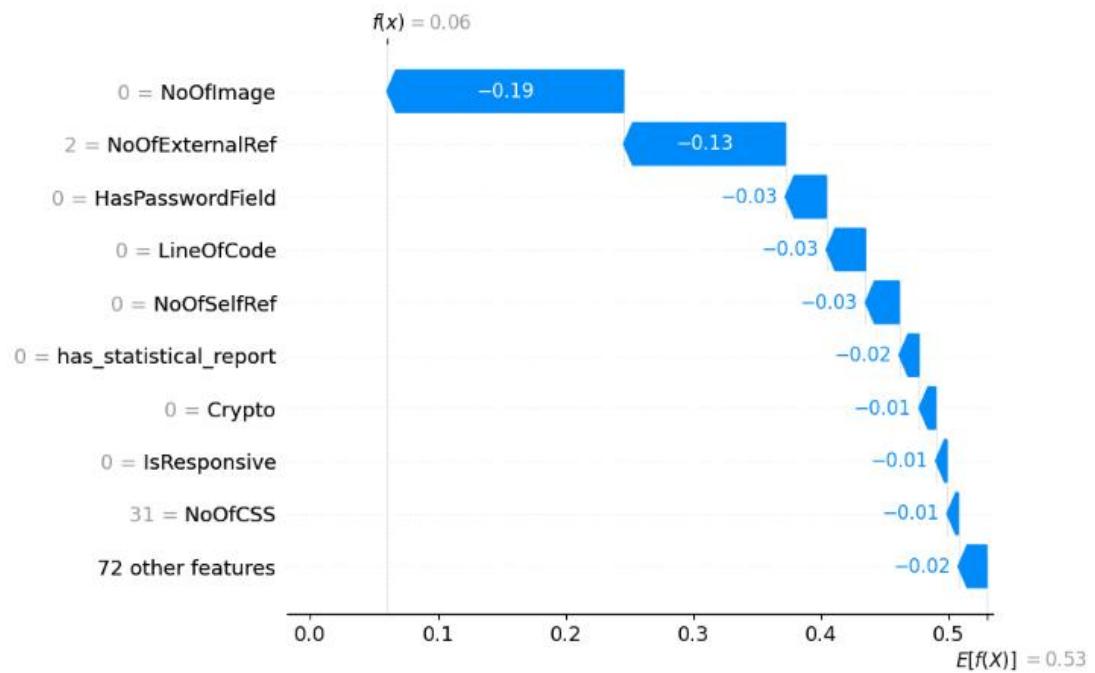


Hình 2.120 Vẽ `shap.force_plot` RF model

```

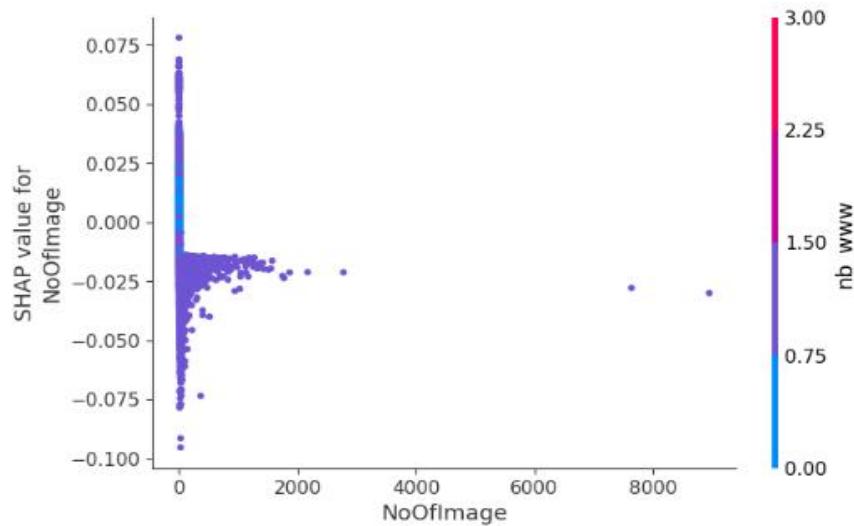
shap.plots.waterfall(
    shap.Explanation(
        values = shap_values_rf[225, :, 1],
        base_values = explainer_rf.expected_value[0],
        data = X_test1.iloc[225],
        feature_names = feature_names
    )
)

```



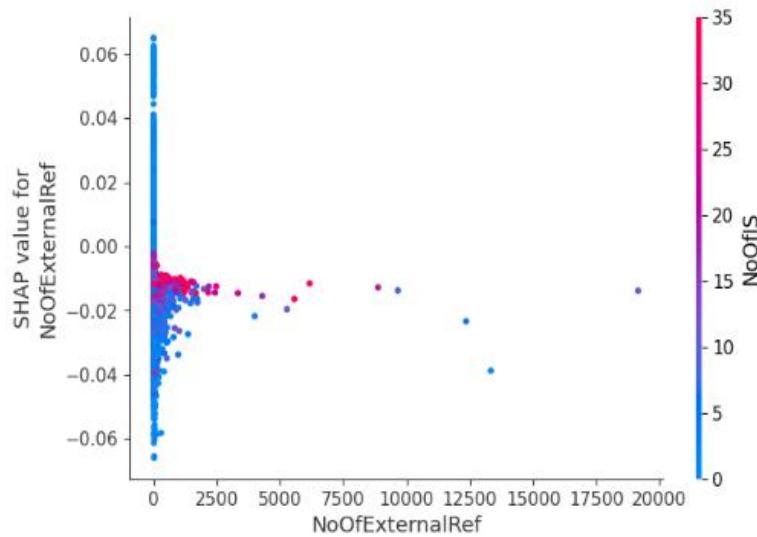
Hình 2.121 Vẽ shap.plots.waterfall RandomForest model

```
shap_values_rf_class0 = shap_values_rf[:, :, 0]
shap.dependence_plot("NoOfImage", shap_values_rf_class0, X_test1)
```



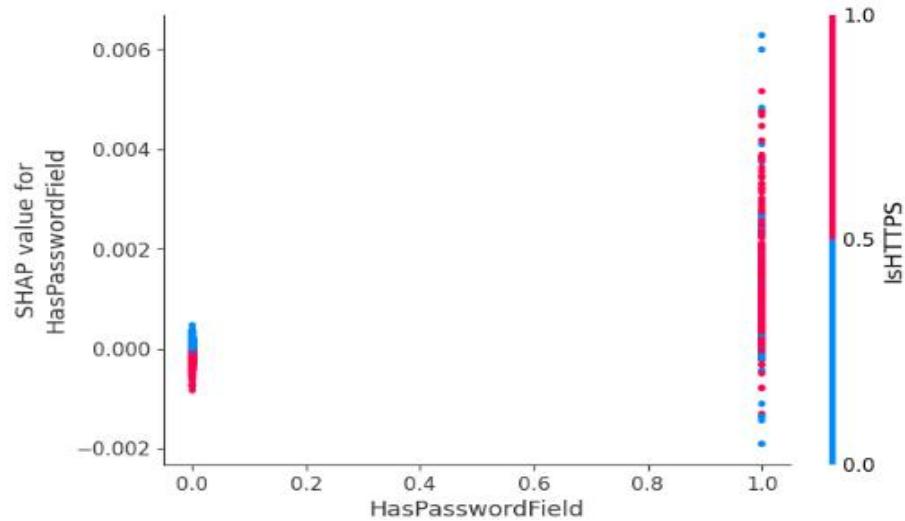
Hình 2.122 Vẽ shap.dependence_plot NoOfImage RandomForest model

```
shap.dependence_plot("NoOfExternalRef", shap_values_rf_class0, X_test1)
```



Hình 2.123 Vẽ shap.dependence_plot NoOfExternalRef

```
shap.dependence_plot("HasPasswordField", shap_values_rf_class0, X_test1)
```



Hình 2.124 Vẽ shap.dependence_plot HasPasswordField RF model

Giai thích mô hình RandomForest URL embedding:

- Load model và chuẩn bị dữ liệu:

```
import shap
import joblib
rf = joblib.load('/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_RF_URL_EMBEDDING.pkl')
explainer_rf = shap.TreeExplainer(
    rf,
    X_test2,
    feature_perturbation="interventional",
    model_output="probability"
)
shap_values_rf = explainer_rf.shap_values(X_test2)

from gensim.models import Word2Vec
import numpy as np
w2v = Word2Vec.load("/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_URL2Vec.model")
tokens = list(w2v.wv.index_to_key) # list token trong vocab
```

```

vectors = w2v.wv.vectors # shape = (vocab_size, embedding_dim)

def top_tokens_for_dim(dim, topn=10):
    """
    Trả về top token có giá trị embedding lớn nhất và nhỏ nhất tại chiều `dim`.
    """
    values = vectors[:, dim]
    sorted_idx = np.argsort(values)

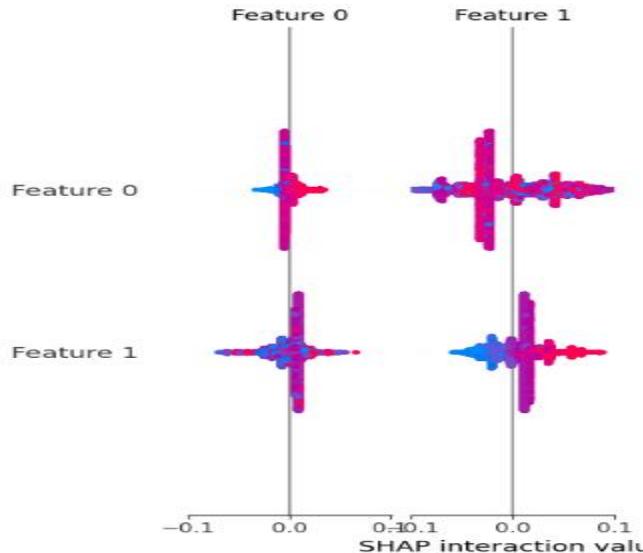
    # top âm (nhỏ nhất)
    low_tokens = [(tokens[i], values[i]) for i in sorted_idx[:topn]]
    # top dương (lớn nhất)
    high_tokens = [(tokens[i], values[i]) for i in sorted_idx[-topn:]]

    return low_tokens, high_tokens

```

- Giải thích mô hình:

```
shap.summary_plot(shap_values_rf, X_test2)
```

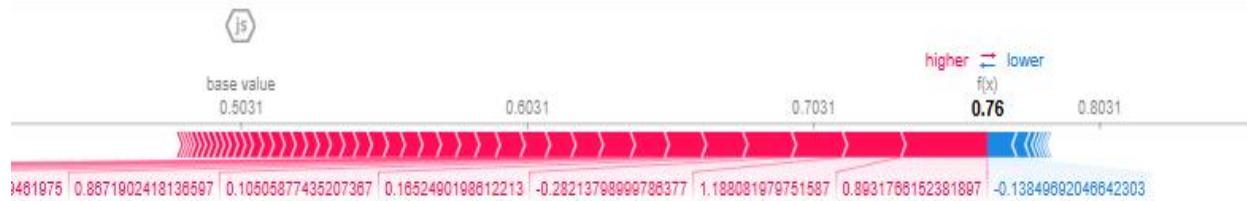


Hình 2.125 Vẽ shap.summary_plot RF embedding model

```

shap.initjs()
shap.force_plot(
    float(explainer_rf.expected_value[1]),
    shap_values_rf[256, :, 1].astype('float64'),
    X_test2[256].astype('float64')
)

```

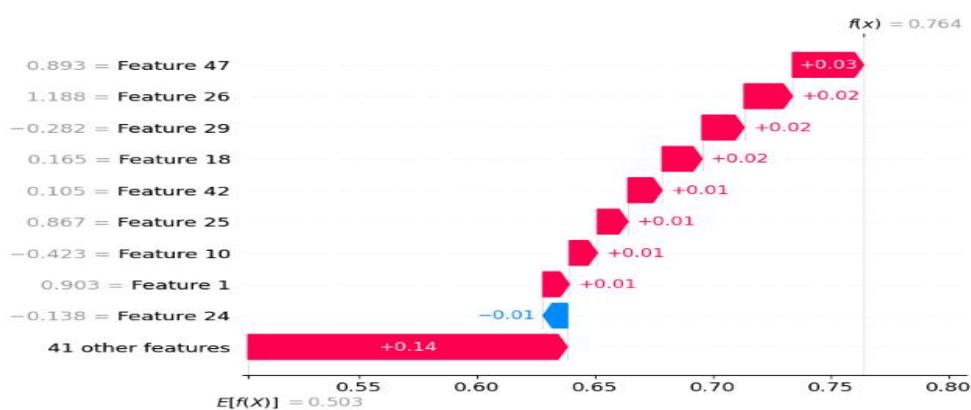


Hình 2.126 Vẽ shap.force_plot RandomForest embedding model

```

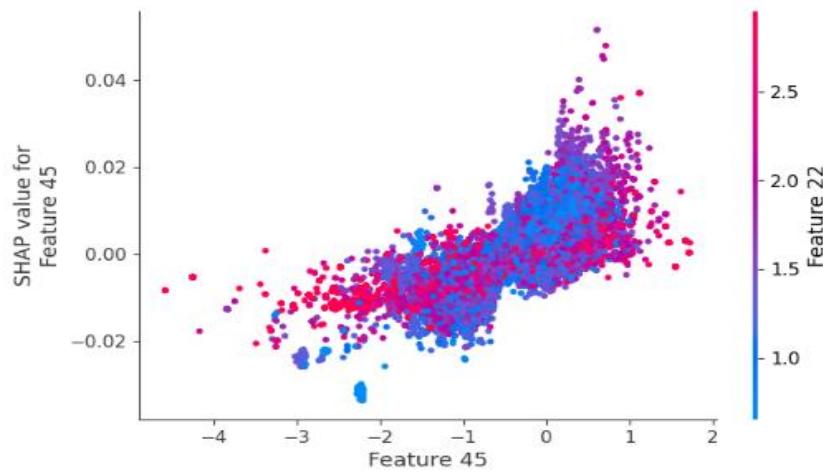
shap.plots.waterfall(
    shap.Explanation(
        values = shap_values_rf[256, :, 1].astype('float64'),
        base_values = float(explainer_rf.expected_value[1]),
        data = X_test2[256].astype('float64'),
        feature_names = [f'Feature {i}' for i in range(X_test2.shape[1])]
    )
)

```



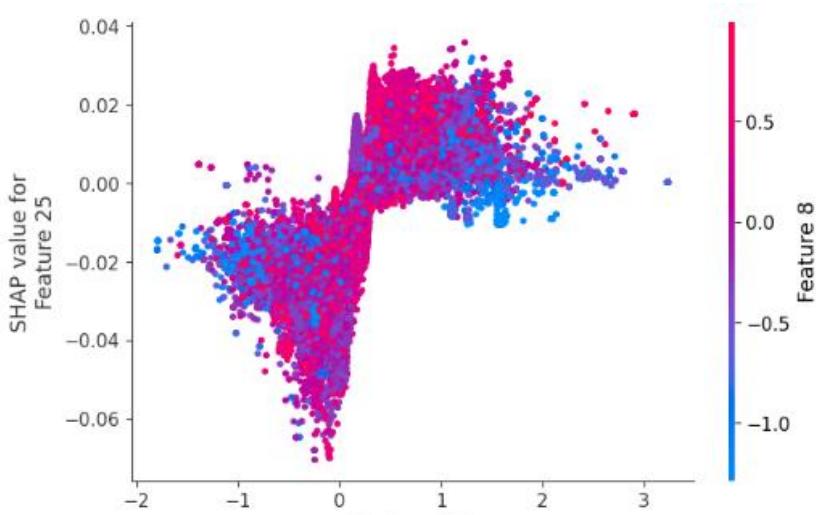
Hình 2.127 Vẽ shap.plots.waterfall RandomForest embedding model

```
shap_values_rf_class1 = shap_values_rf[:, :, 1]  
shap.dependence_plot("Feature 45", shap_values_rf_class1, X_test2)
```



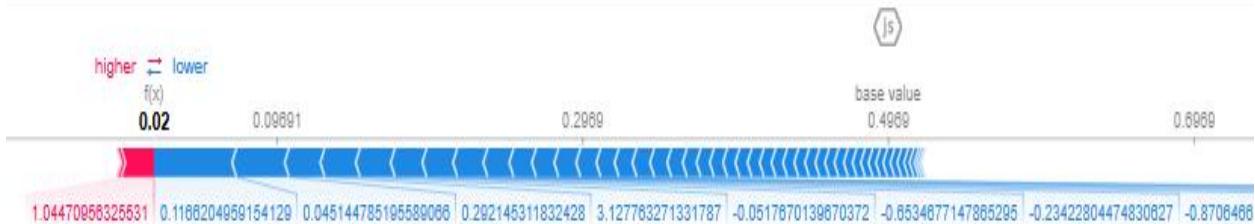
Hình 2.128 Vẽ shap_values_rf_class1 Feature 45 RF embedding model

```
shap.dependence_plot("Feature 25", shap_values_rf_class1, X_test2)
```



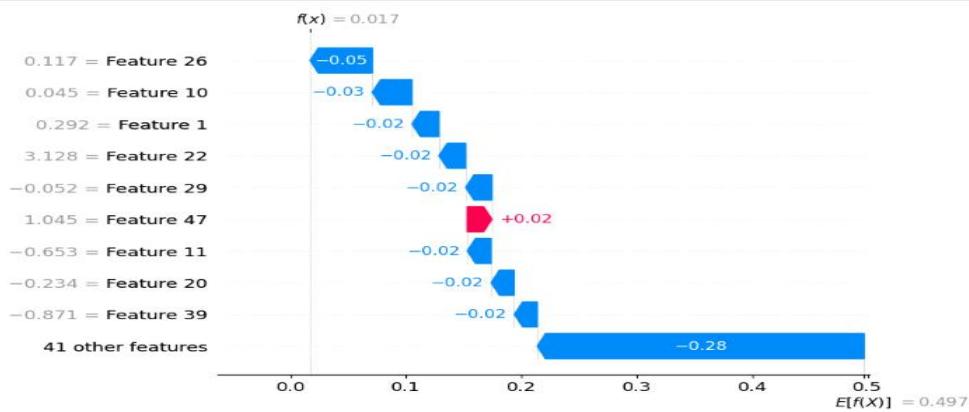
Hình 2.129 Vẽ shap.dependence_plot Feature 25 RF embedding model

```
shap.initjs()
shap.force_plot(
    float(explainer_rf.expected_value[0]),
    shap_values_rf[44, :, 1].astype('float64'),
    X_test2[44].astype('float64')
)
```



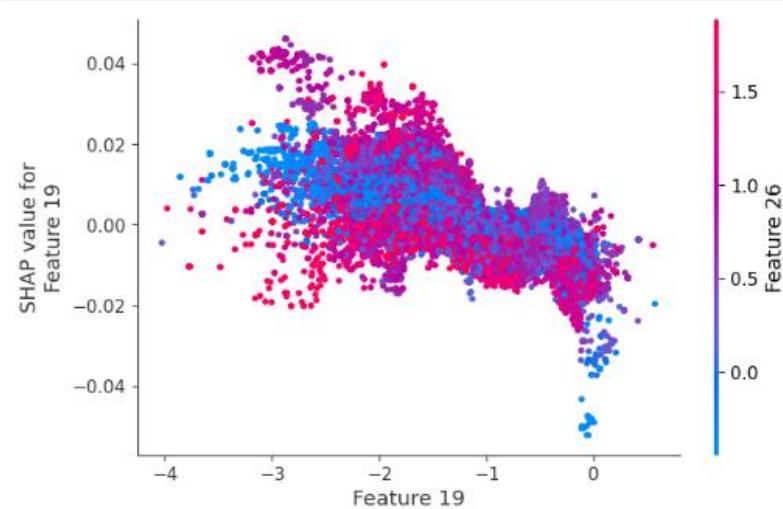
Hình 2.130 Vẽ shap.force_plot RandomForest embedding model

```
shap.plots.waterfall(
    shap.Explanation(
        values = shap_values_rf[44, :, 1].astype('float64'),
        base_values = float(explainer_rf.expected_value[0]),
        data = X_test2[44].astype('float64'),
        feature_names = [f'Feature {i}' for i in range(X_test2.shape[1])]
    )
)
```



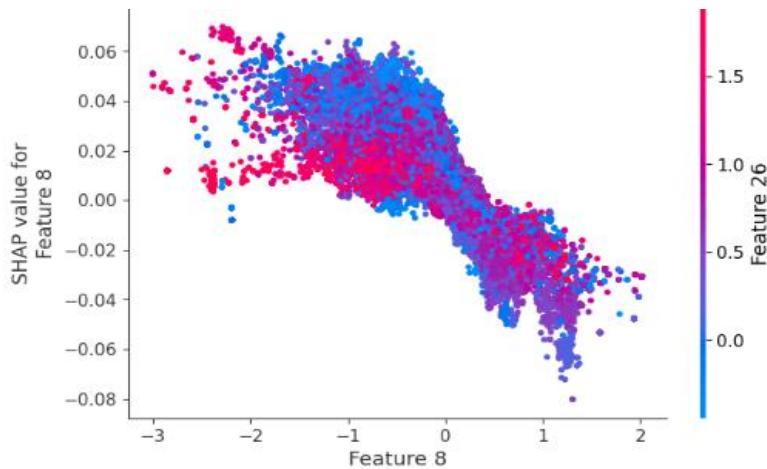
Hình 2.131 Vẽ shap.plots.waterfall RandomForest embedding model

```
shap_values_rf_class0 = shap_values_rf[:, :, 0]  
shap.dependence_plot("Feature 19", shap_values_rf_class0, X_test2)
```



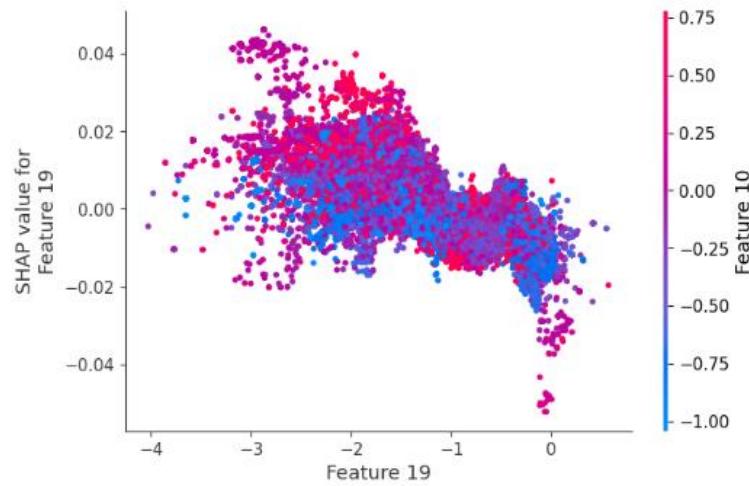
Hình 2.132 Vẽ shap.dependence_plot Feature 19 RandomForest embedding model

```
shap.dependence_plot("Feature 8", shap_values_rf_class0, X_test2)
```



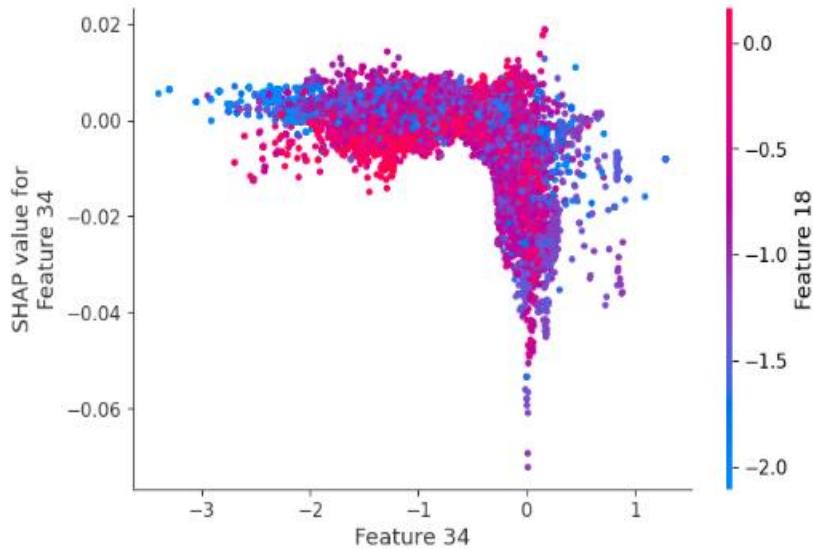
Hình 2.133 Vẽ shap.dependence_plot Feature 8 RandomForest embedding model

```
shap_values_rf_class0 = shap_values_rf[:, :, 0]
shap.dependence_plot("Feature 19", shap_values_rf_class0, X_test2)
```



Hình 2.134 Vẽ shap.dependence_plot Feature 19 RandomForest embedding model

```
shap.dependence_plot("Feature 34", shap_values_rf_class0, X_test2)
```



Hình 2.135 Vẽ shap.dependence_plot Feature 34 RF embedding model

2.3.3 WordCNN

Giải thích mô hình WordCNN đặc trưng số:

- Load model và chuẩn bị dữ liệu:

```
import shap
import tensorflow as tf
import numpy as np
X_train3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/X_train3.npy")
X_test3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/X_test3.npy")
Y_train3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/Y_train3.npy")
Y_test3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/Y_test3.npy")
WordCNN_model1 =
tf.keras.models.load_model("/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_WordCNN_N
umerical.keras")

# Tạo background set cân bằng (5000 legitimate + 5000 phish)
legitimate_indices_train = np.where(Y_train3 == 1)[0]
phish_indices_train = np.where(Y_train3 == 0)[0]

# Lấy ngẫu nhiên 5000 mẫu từ mỗi class
np.random.seed(42) # Để có thể tái tạo kết quả
legitimate_sample_train = np.random.choice(legitimate_indices_train, size=5000, replace=False)
phish_sample_train = np.random.choice(phish_indices_train, size=5000, replace=False)

# Kết hợp và tạo background
background_indices = np.concatenate([legitimate_sample_train, phish_sample_train])
np.random.shuffle(background_indices) # Trộn ngẫu nhiên
background = X_train3[background_indices]

print(f"Background shape: {background.shape}")
print(f"Background labels distribution: Legitimate={np.sum(Y_train3[background_indices] ==
1)}, Phish={np.sum(Y_train3[background_indices] == 0)}")
```

```
# Tạo test set cân bằng (2500 legitimate + 2500 phish)
legitimate_indices_test = np.where(Y_test3 == 1)[0]
phish_indices_test = np.where(Y_test3 == 0)[0]

legitimate_sample_test = np.random.choice(legitimate_indices_test, size=2500, replace=False)
phish_sample_test = np.random.choice(phish_indices_test, size=2500, replace=False)

# Kết hợp và tạo test set
test_indices = np.concatenate([legitimate_sample_test, phish_sample_test])
np.random.shuffle(test_indices) # Trộn ngẫu nhiên
X_test_balanced = X_test3[test_indices]
Y_test_balanced = Y_test3[test_indices]

print(f"Test set shape: {X_test_balanced.shape}")
print(f"Test labels distribution: Legitimate={np.sum(Y_test_balanced == 1)}, Phish={np.sum(Y_test_balanced == 0)}")

explainer = shap.DeepExplainer(WordCNN_model1, background)
shap_values = explainer.shap_values(X_test_balanced)

print("Output classes:", len(shap_values))
print("Shape SHAP values:", shap_values[0].shape)

if isinstance(shap_values, list):
    shap_values = shap_values[0]

print("Raw shap_values shape:", np.array(shap_values).shape)

np.save("/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_WordCNN_NUMERICAL_SHAP/shap_values.npy", shap_values)
np.save("/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_WordCNN_NUMERICAL_SHAP/X_test2d.npy", X_test2d)

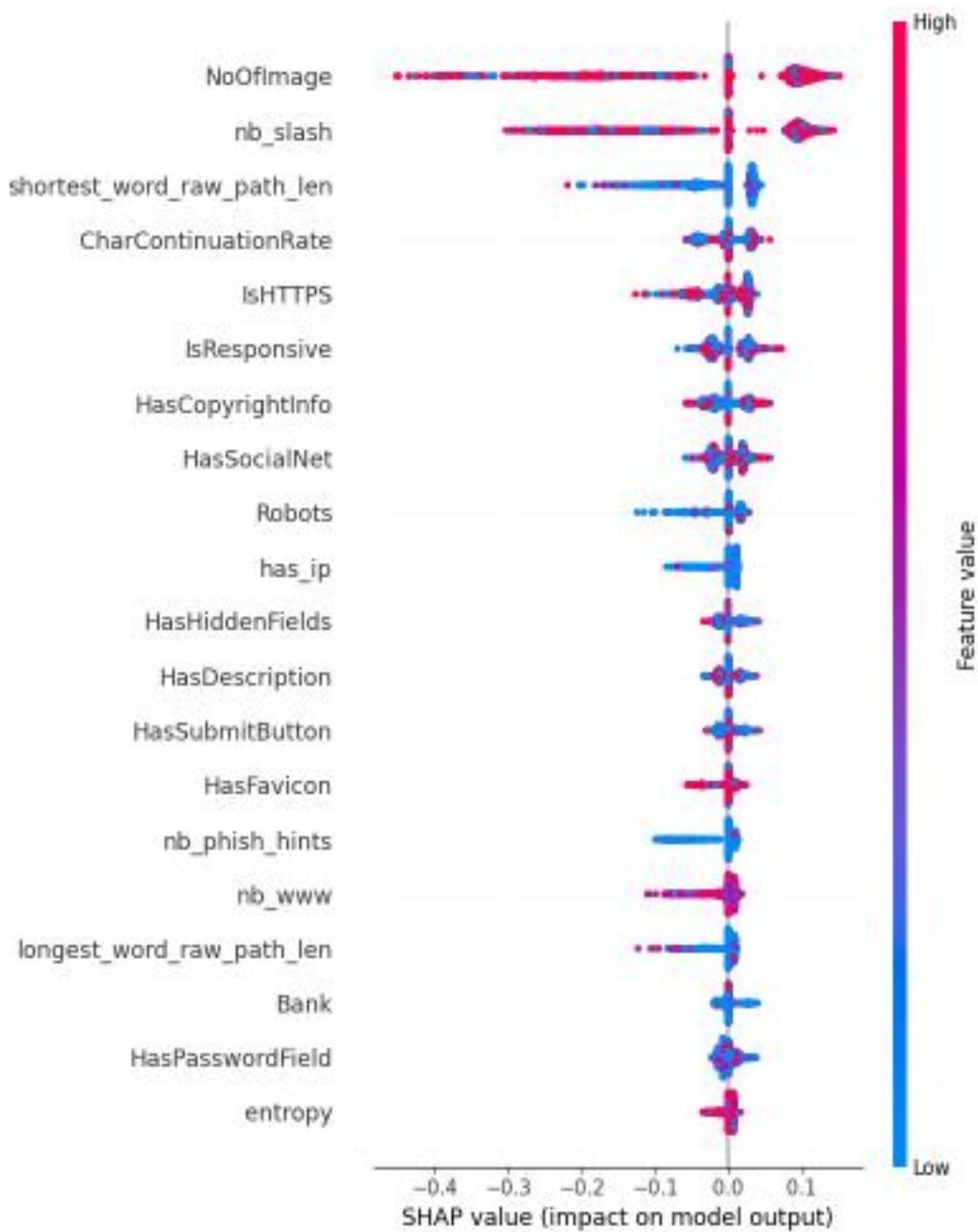
shap_values =
```

```
np.load("/content/drive/MyDrive/DeAnTotNghiep/PhiUSIIL_WordCNN_NUMERICAL_SHAP/shap_values.npy", allow_pickle=True)
X_test2d =
np.load("/content/drive/MyDrive/DeAnTotNghiep/PhiUSIIL_WordCNN_NUMERICAL_SHAP/X_test2d.npy")
feature_names = np.load("/content/drive/MyDrive/DeAnTotNghiep/feature_names.npy",
allow_pickle=True).tolist()

shap_values_2d = shap_values.reshape(shap_values.shape[0], shap_values.shape[1])
#X_test2d = X_test3[:5000].reshape(5000, 81) #Bỏ comment trong trường hợp chạy lỗi
explainer = shap.DeepExplainer(WordCNN_model1, background)
```

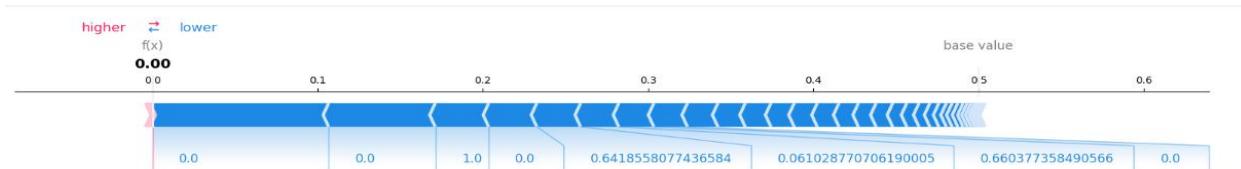
- Giải thích mô hình với SHAP:

```
shap.summary_plot(
    shap_values_2d,
    X_test2d,
    feature_names=feature_names,
    max_display=20
)
```



Hình 2.136 Tính SHAP value ảnh hưởng đến output WordCNN model

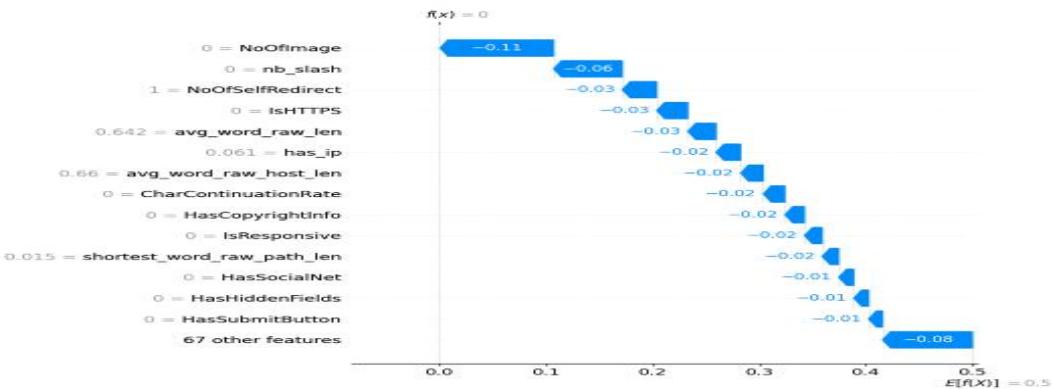
```
shap.force_plot(
    base_value,
    shap_values_reshaped[sample_idx],
    X_test_flat[sample_idx],
    matplotlib=True
)
```



Hình 2.137 Vẽ Shap.force_plot WordCNN model

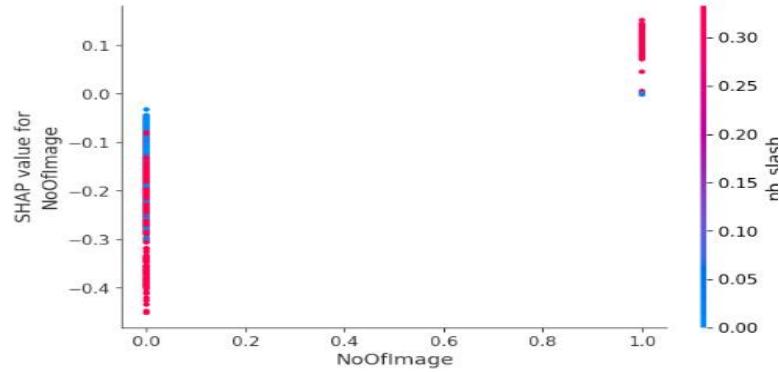
```
explanation = shap.Explanation(
    values=shap_values_reshaped[sample_idx],
    base_values=base_value,
    data=X_test_flat[sample_idx],
    feature_names=real_feature_names
)
```

```
shap.plots.waterfall(explanation, max_display=15)
```



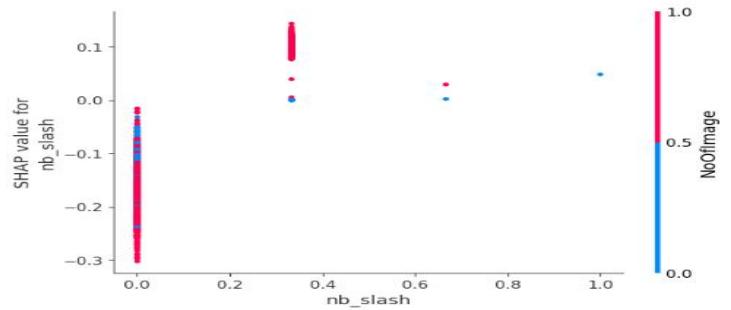
Hình 2.138 Vẽ shap.plots.waterfall WordCNN model

```
shap.dependence_plot(
    top_features[0],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```



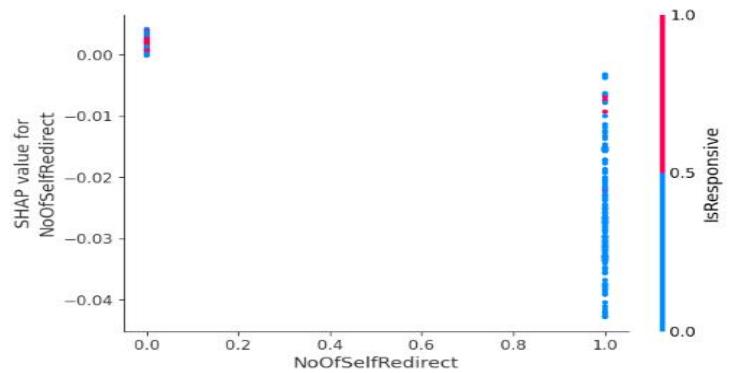
Hình 2.139 Vẽ shap.dependence_plot Feature NoOfimage WordCNN model

```
shap.dependence_plot(
    top_features[1],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```



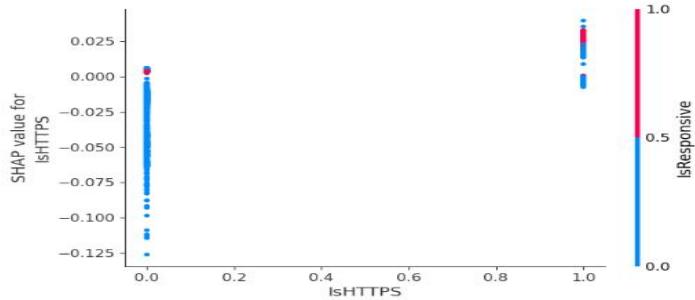
Hình 2.140 Vẽ shap.dependence_plot Feature nb_slash WordCNN model

```
shap.dependence_plot(
    top_features[2],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```



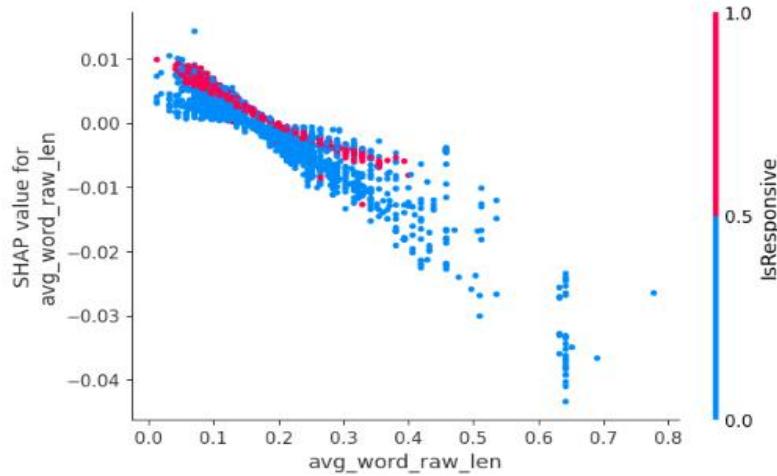
Hình 2.141 Vẽ shap.dependence_plot Feature NoOfSelfRedirect WordCNN model

```
shap.dependence_plot(
    top_features[3],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```



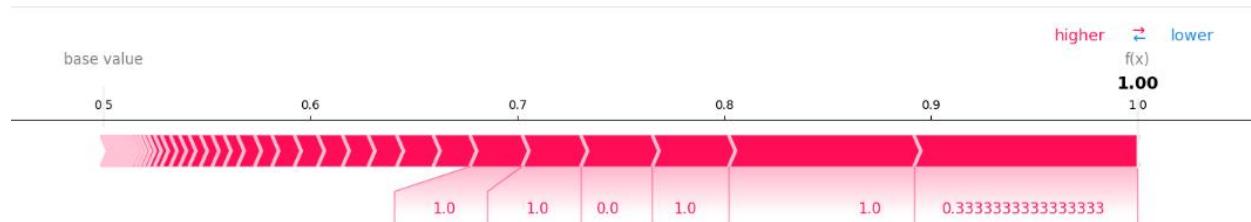
Hình 2.142 Vẽ shap.dependence_plot IsHTTPS WordCNN model

```
shap.dependence_plot(
    top_features[4],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```



Hình 2.143 Vẽ shap.dependence_plot avg_word_raw_len WordCNN model

```
shap.force_plot(
    base_value,
    shap_values_reshaped[sample_idx],
    X_test_flat[sample_idx],
    matplotlib=True
)
```



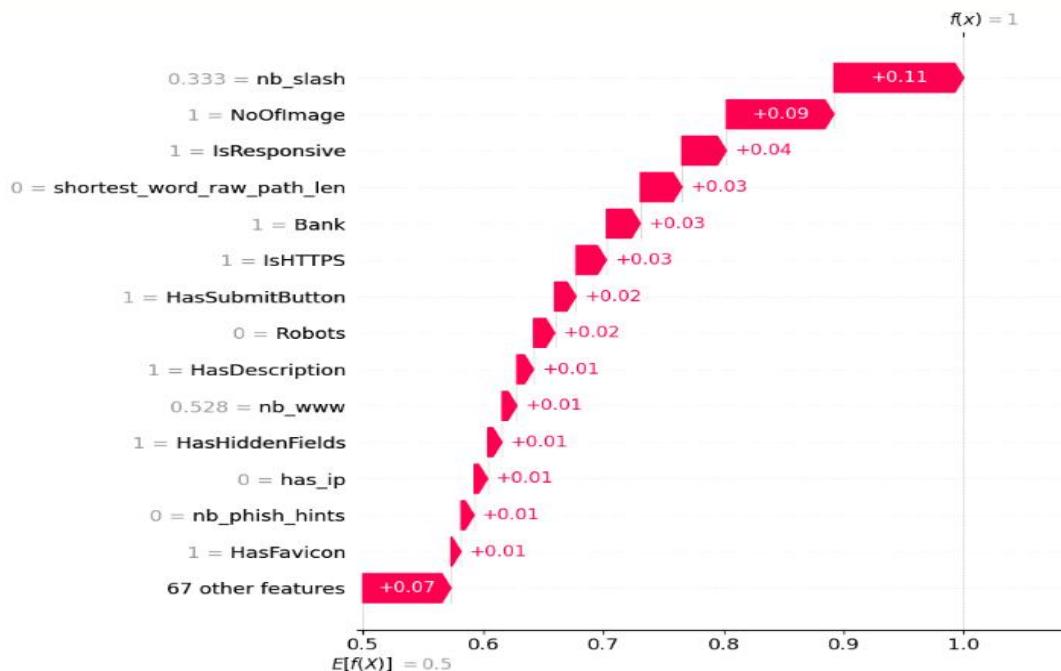
Hình 2.144 Vẽ shap.force_plot WordCNN model

```

explanation = shap.Explanation(
    values=shap_values_reshaped[sample_idx],
    base_value=base_value,
    data=X_test_flat[sample_idx],
    feature_names=real_feature_names
)

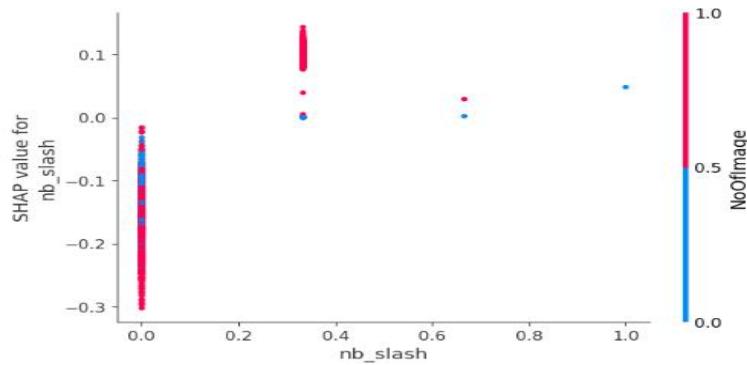
```

```
shap.plots.waterfall(explanation, max_display=15)
```



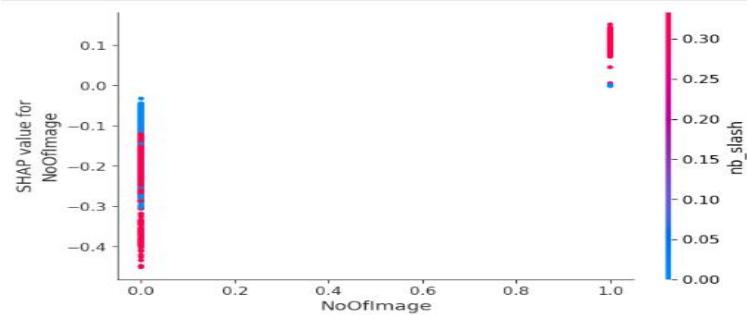
Hình 2.145 Vẽ shap.plots.waterfall WordCNN model

```
shap.dependence_plot(
    top_features[0],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```



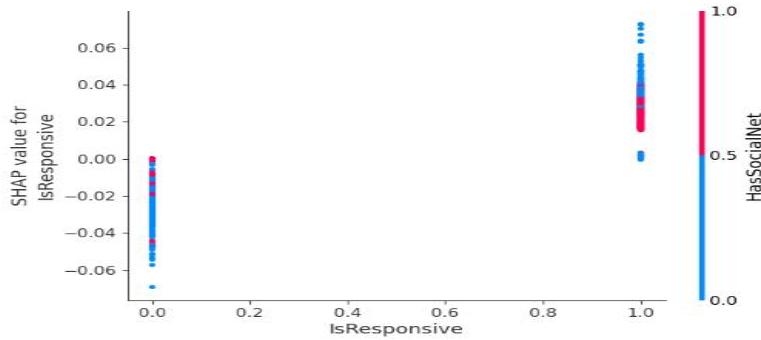
Hình 2.146 Vẽ shap.dependence_plot nb_slash WordCNN model

```
shap.dependence_plot(
    top_features[1],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```

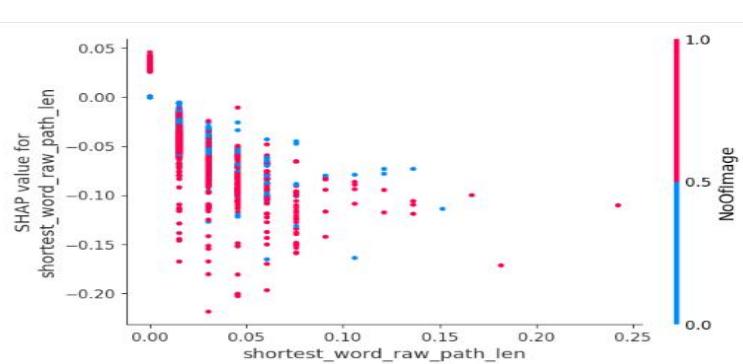


Hình 2.147 Vẽ shap.dependence_plot NoOfImage WordCNN model

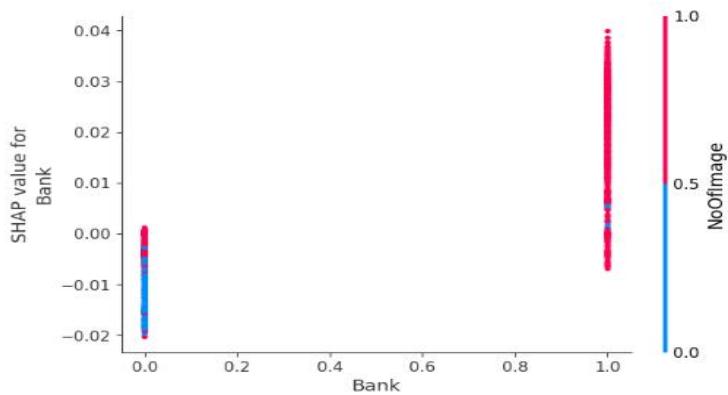
```
shap.dependence_plot(
    top_features[2],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```

*Hình 2.148 Vẽ shap.dependence_plot isResponsive WordCNN model*

```
shap.dependence_plot(
    top_features[3],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```

*Hình 2.149 Vẽ shap.dependence_plot shortest_word_raw_path_len WordCNN model*

```
shap.dependence_plot(
    top_features[4],
    shap_values_reshaped,
    X_test_flat,
    feature_names=feature_names,
    interaction_index='auto'
)
```



Hình 2.150 Vẽ shap.dependence_plot Feature Bank WordCNN model

Giải thích mô hình WordCNN đặc URL embedding:

- Load model và chuẩn bị dữ liệu:

```
import shap
import tensorflow as tf
import numpy as np
import joblib
X_train4 = joblib.load("/content/drive/MyDrive/DeAnTotNghiep/X_train_embed.pkl")
X_test4 = joblib.load("/content/drive/MyDrive/DeAnTotNghiep/X_test_embed.pkl")
Y_train4 = joblib.load("/content/drive/MyDrive/DeAnTotNghiep/Y_train_embed.pkl")
Y_test4 = joblib.load("/content/drive/MyDrive/DeAnTotNghiep/Y_test_embed.pkl")

# Kiểm tra kiểu dữ liệu
print(f"Y_train4 type: {type(Y_train4)}")
print(f"Y_test4 type: {type(Y_test4)}")
```

```

# Chuyển đổi sang numpy array
if hasattr(Y_train4, 'values'):
    Y_train4 = Y_train4.values
if hasattr(Y_test4, 'values'):
    Y_test4 = Y_test4.values

# Đảm bảo là numpy array 1D
Y_train4 = np.array(Y_train4).flatten()
Y_test4 = np.array(Y_test4).flatten()

# Reset index về 0, 1, 2, ...
Y_train4 = np.array([Y_train4[i] for i in range(len(Y_train4))])
Y_test4 = np.array([Y_test4[i] for i in range(len(Y_test4))])

X_train4 = np.expand_dims(X_train4, axis=-1)
X_test4 = np.expand_dims(X_test4, axis=-1)

print(f"\nAfter conversion:")
print(f"Y_train4 shape: {Y_train4.shape}")
print(f"Y_test4 shape: {Y_test4.shape}")
print(f"Y_train4 type: {type(Y_train4)}")

CNN_model2 =
tf.keras.models.load_model("/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_CNN_URL_E
mbedding.keras")

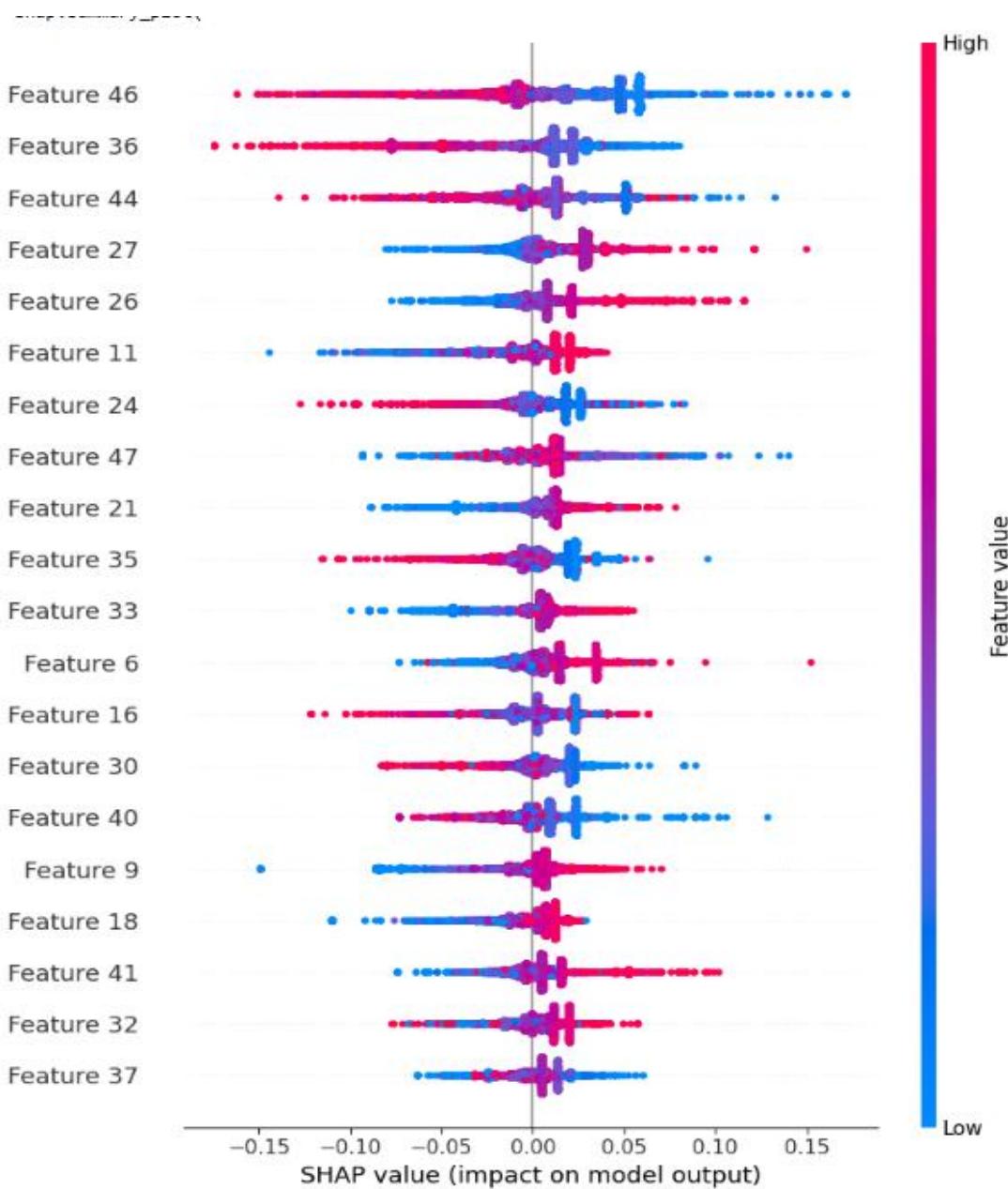
```

- Giải thích mô hình với SHAP:

```

shap.summary_plot(
    shap_to_plot_2d,
    X_test_2d,
    feature_names=None,
    max_display=20
)

```



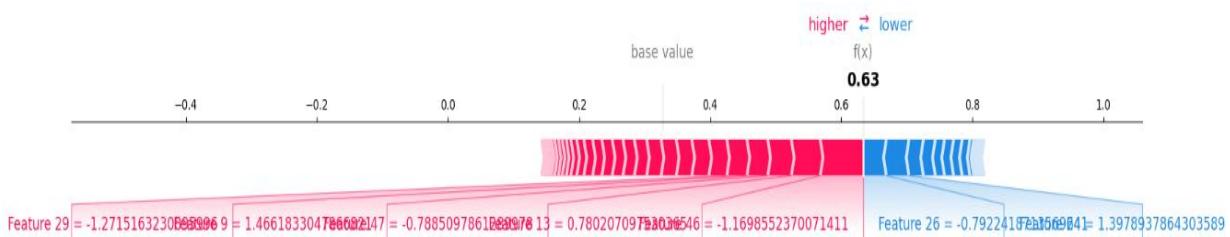
Hình 2.151 Vẽ shap value impact output of WordCNN embedding model

```
shap.initjs()
```

```
if isinstance(explainer.expected_value, (list, np.ndarray)):
    base_value = float(explainer.expected_value[0])
else:
    base_value = float(explainer.expected_value)

sample_idx = 57
feature_names = [f"Feature {i}" for i in range(X_test_2d.shape[1])]

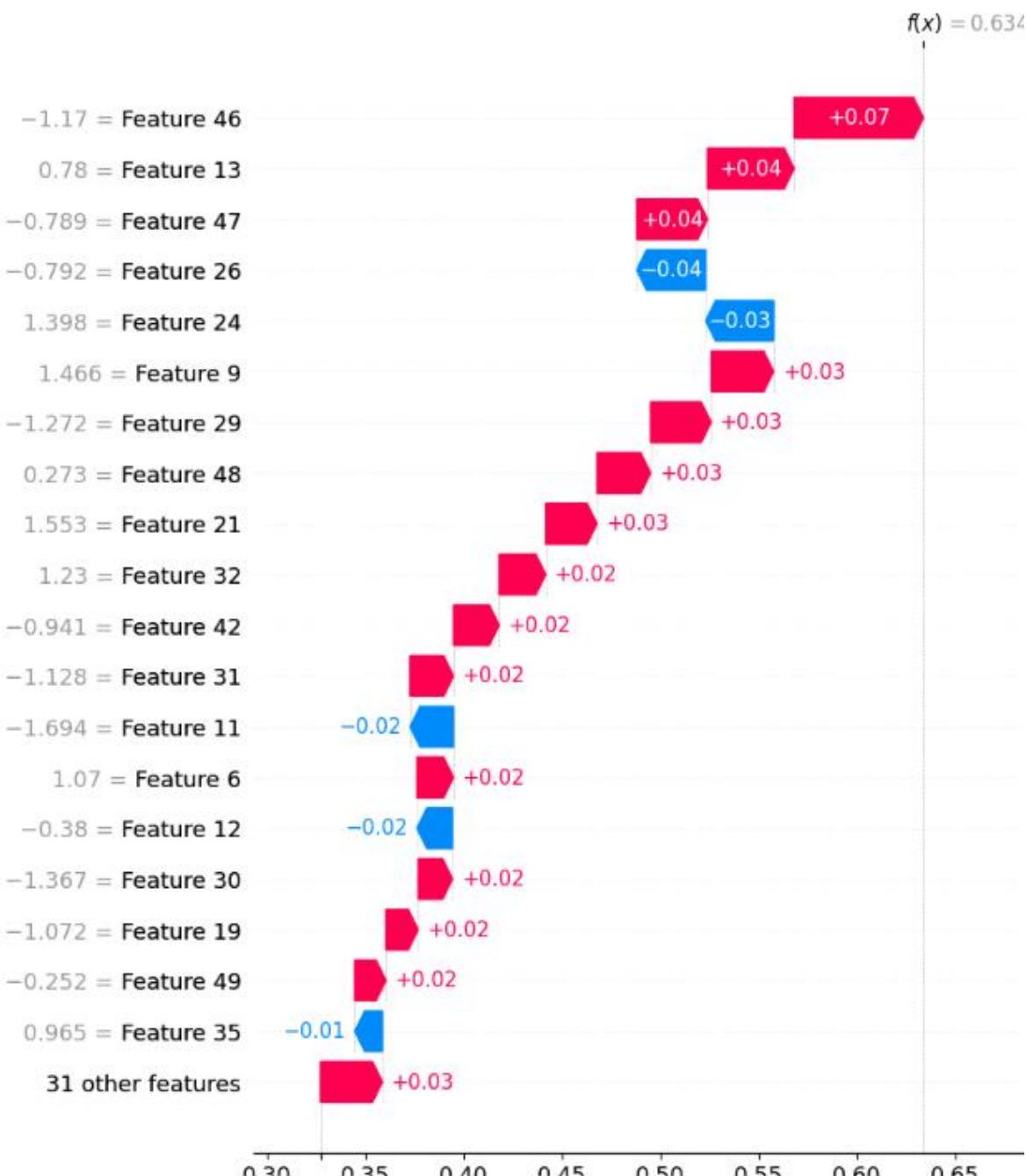
shap.force_plot(
    base_value,
    shap_to_plot_2d[sample_idx],
    X_test_2d[sample_idx],
    feature_names=feature_names,
    matplotlib=True
)
```



Hình 2.152 Vẽ shap.force_plot sample 57 WordCNN embedding model

```
explanation = shap.Explanation(
    values=shap_to_plot_2d[sample_idx],
    base_values=base_value,
    data=X_test_2d[sample_idx],
    feature_names=feature_names
)
```

```
shap.plots.waterfall(explanation, max_display=20)
```



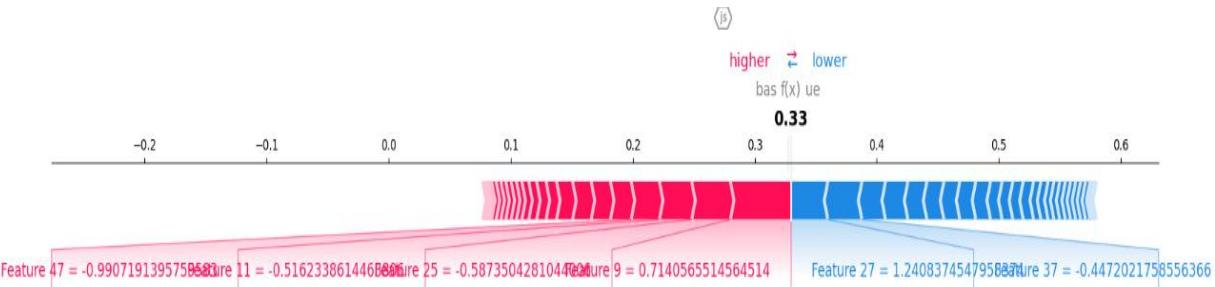
Hình 2.153 Vẽ shap.plots.waterfall WordCNN embedding model

```
shap.initjs()
```

```
if isinstance(explainer.expected_value, (list, np.ndarray)):
    base_value = float(explainer.expected_value[0])
else:
    base_value = float(explainer.expected_value)
```

```
sample_idx = 33
```

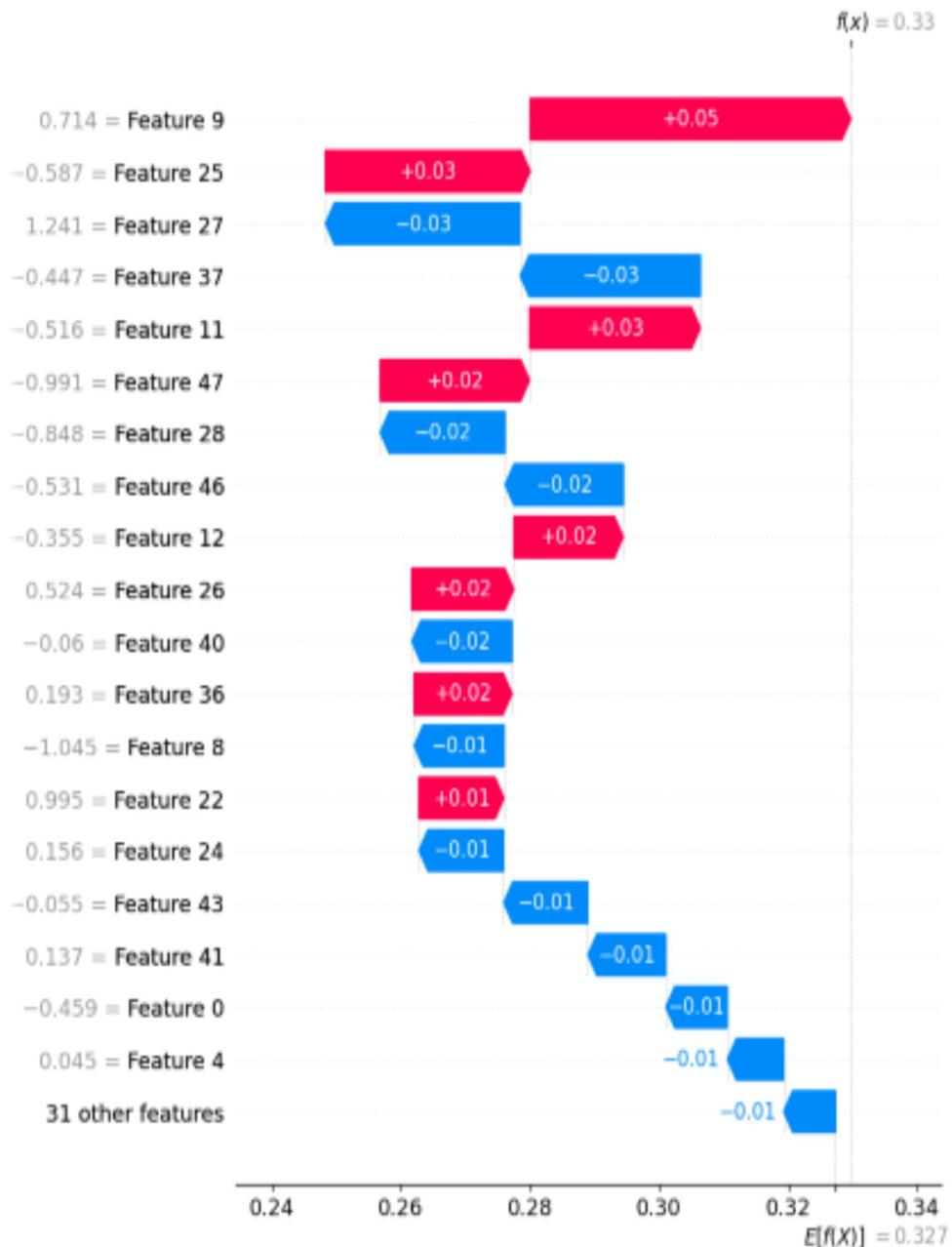
```
shap.force_plot(
    base_value,
    shap_to_plot_2d[sample_idx],
    X_test_2d[sample_idx],
    feature_names=feature_names,
    matplotlib=True
)
```



Hình 2.154 Vẽ shap.force_plot sample 33 WordCNN embedding model

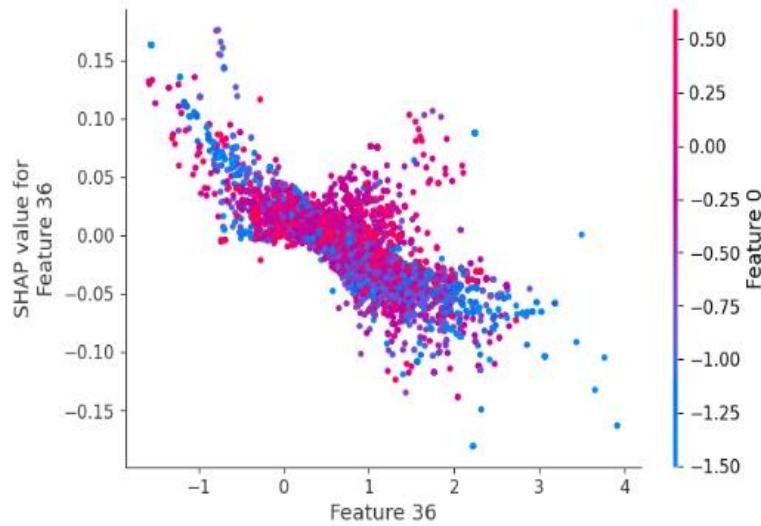
```
explanation = shap.Explanation(
    values=shap_to_plot_2d[sample_idx],
    base_values=base_value,
    data=X_test_2d[sample_idx],
    feature_names=feature_names
)
```

```
shap.plots.waterfall(explanation, max_display=20)
```



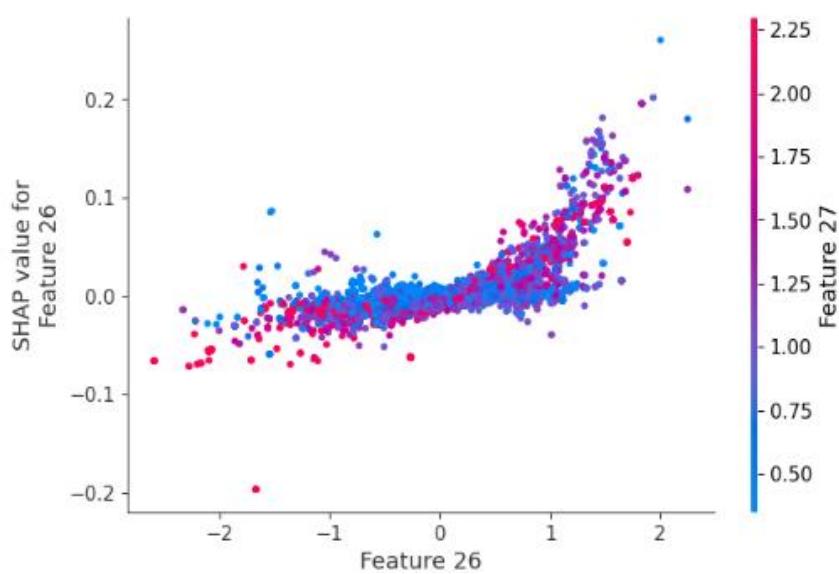
Hình 2.155 Vẽ shap.plots.waterfall WordCNN embedding model

```
shap.dependence_plot("Feature 36", shap_to_plot_2d, X_test_2d)
```



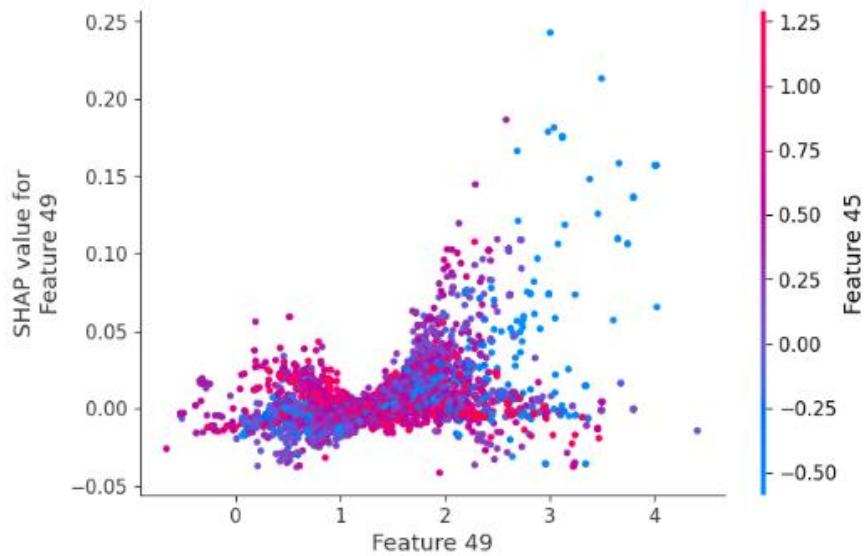
Hình 2.156 Vẽ shap.dependence_plot Feature 36 WordCNN embedding model

```
shap.dependence_plot("Feature 26", shap_to_plot_2d, X_test_2d)
```



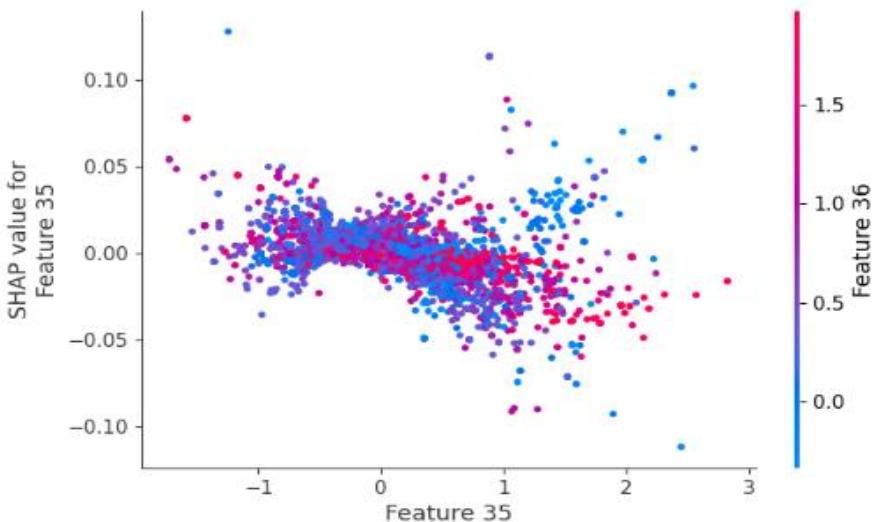
Hình 2.157 Vẽ shap.dependence_plot Feature 26 WordCNN embedding model

```
shap.dependence_plot("Feature 49", shap_to_plot_2d, X_test_2d)
```



Hình 2.158 Vẽ shap.dependence_plot Feature 49 WordCNN embedding model

```
shap.dependence_plot("Feature 35", shap_to_plot_2d, X_test_2d)
```



Hình 2.159 Vẽ shap.dependence_plot Feature 35 WordCNN embedding model

2.3.4 CNN-LSTM

Giải thích mô hình CNN-LSTM đặc trưng số:

- Load model và chuẩn bị dữ liệu

```
from lime import lime_tabular
import tensorflow as tf
import numpy as np
X_train3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/X_train3.npy")
X_test3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/X_test3.npy")
Y_test3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/Y_test3.npy")
Y_train3 = np.load("/content/drive/MyDrive/DeAnTotNghiệp/Y_train3.npy")
feature_names = np.load("/content/drive/MyDrive/DeAnTotNghiệp/feature_names.npy",
allow_pickle=True).tolist()
CNN_LSTM_model1 =
tf.keras.models.load_model("/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_CNN_LSTM_
Numerical.keras")
# Flatten input từ (81, 1) -> (81,)
X_train_flat = X_train3.reshape(X_train3.shape[0], -1)
X_test_flat = X_test3.reshape(X_test3.shape[0], -1)

def predict_proba(x):
    x_reshaped = x.reshape(x.shape[0], 81, 1)
    p_legitimate = CNN_LSTM_model1.predict(x_reshaped, verbose=0)
    p_phish = 1 - p_legitimate
    return np.concatenate([p_phish, p_legitimate], axis=1)

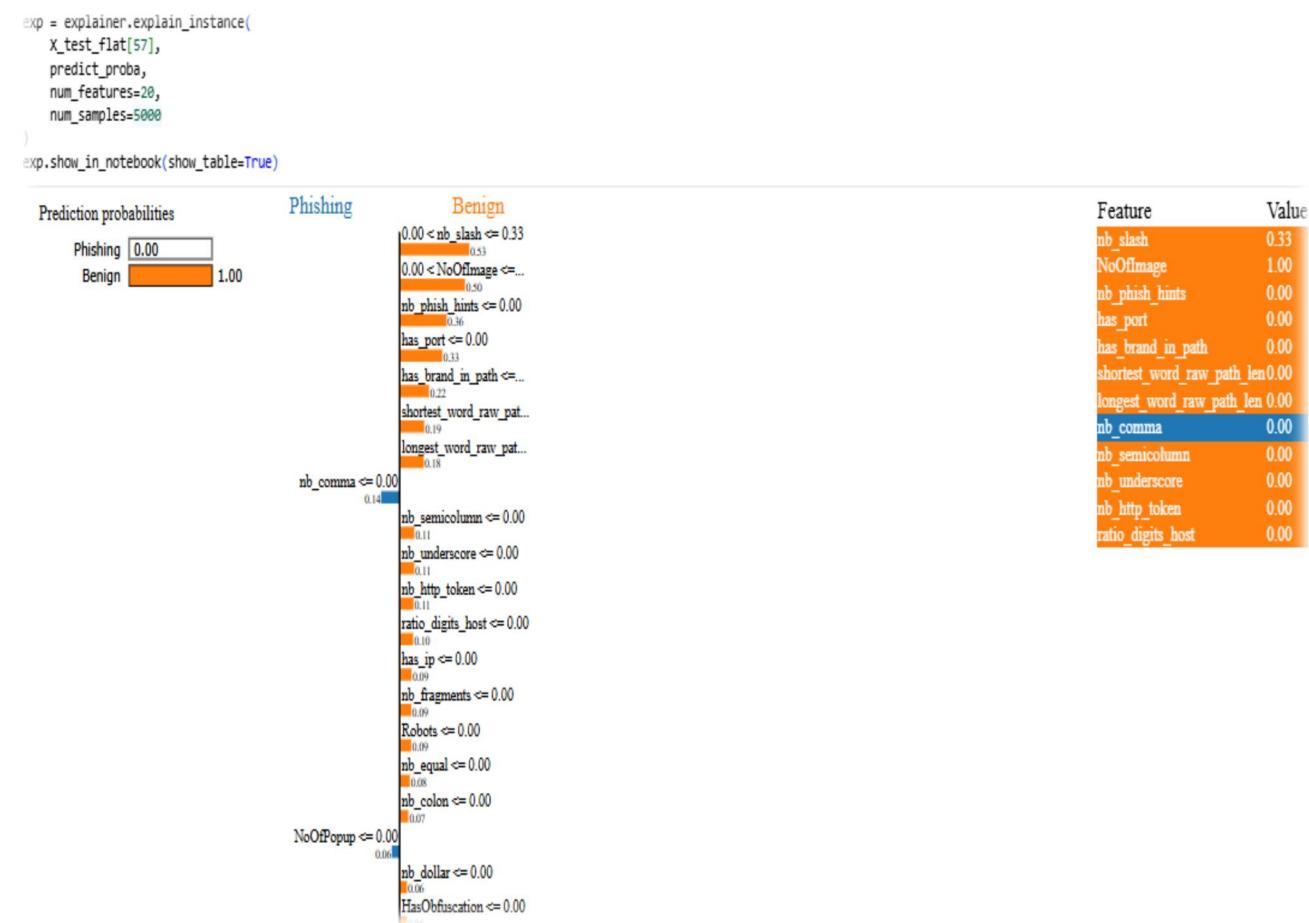
explainer = lime_tabular.LimeTabularExplainer(
    X_train_flat,
    feature_names=feature_names,
    class_names=['Phishing', 'Legitimate'],
    mode='classification'
)
```

- Giải thích mô hình với LIME:

```

exp = explainer.explain_instance(
    X_test_flat[57],
    predict_proba,
    num_features=20,
    num_samples=5000
)
exp.show_in_notebook(show_table=True)

```

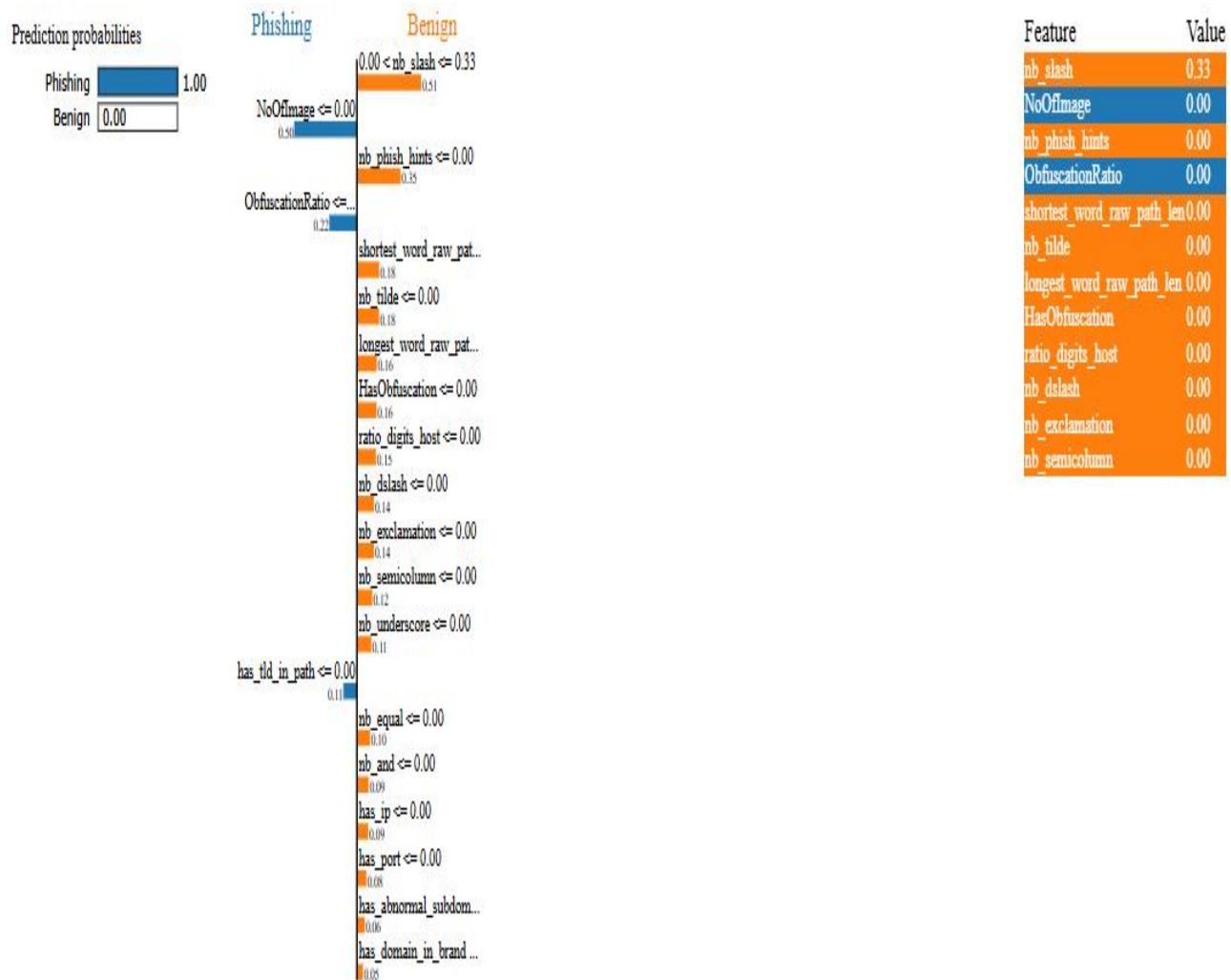


Hình 2.160 LIEM hiển thị 20 đặc trưng quan trọng nhất mẫu 58 của CNN-LSTM model

```

exp = explainer.explain_instance(
    X_test_flat[11],
    predict_proba,
    num_features=20,
    num_samples=5000
)
exp.show_in_notebook(show_table=True)

```



Hình 2161 LIEM hiển thị 20 đặc trưng quan trọng nhất mẫu 12 của CNN-LSTM model

Giai thích mô hình CNN-LSTM URL embedding:

- Load model và chuẩn bị dữ liệu:

```
# 1) Load embedding 2D cho LIME
if os.path.exists(os.path.join(BASE, "X_train_embed.pkl")):
    X_train4 = joblib.load(os.path.join(BASE, "X_train_embed.pkl"))
    X_test4 = joblib.load(os.path.join(BASE, "X_test_embed.pkl"))
    Y_test4 = joblib.load(os.path.join(BASE, "Y_test_embed.pkl"))
else:
    X_train4 = ensure_2d(np.load(os.path.join(BASE, "X_train4.npy")))
    X_test4 = ensure_2d(np.load(os.path.join(BASE, "X_test4.npy")))
    Y_test4 = np.load(os.path.join(BASE, "Y_test4.npy"))

X_train4 = np.asarray(X_train4, dtype=np.float32) # (N,50)
X_test4 = np.asarray(X_test4, dtype=np.float32) # (N,50)

cnn_lstm_emb = None
for p in CAND_CNNLSTM:
    if os.path.exists(p):
        try:
            cnn_lstm_emb = keras.saving.load_model(p, compile=False, safe_mode=False)
            print("[OK] Loaded CNN-LSTM embedding:", p)
            break
        except Exception as e:
            print("[WARN] load_model failed:", p, e)

if cnn_lstm_emb is None:
    # Fallback:
    assert 'CNN_LSTM_model2' in globals(), \
        "Không tìm thấy file .keras của CNN-LSTM embedding và cũng không có biến \
        CNN_LSTM_model2 trong session."
    cnn_lstm_emb = CNN_LSTM_model2
    print("[OK] Using in-memory CNN_LSTM_model2")
```

```

# Warm-up để build (numpy, không wrap tf.keras)
_=cnn_lstm_emb(np.zeros((1, 50, 1), dtype=np.float32))
print("[OK] Warmed up CNN-LSTM embedding.")

#Hàm dự đoán cho LIME: nhận (N,50) -> (N,50,1)
def predict_proba_cnnlstm_emb(x2d):
    x3d = np.expand_dims(x2d.astype(np.float32), axis=-1) # (N,100)->(N,100,1)
    y = cnn_lstm_emb(x3d, training=False)
    try:
        y = y.numpy()
    except Exception:
        pass
    p1 = y.reshape(-1)
    p0 = 1.0 - p1
    return np.stack([p0, p1], axis=1)

#LIME explainer
expl_cnnlstm_emb = LimeTabularExplainer(
    training_data=X_train4,
    feature_names=[f"emb{i}" for i in range(X_train4.shape[1])],
    class_names=CLASS_NAMES,
    discretize_continuous=True,
    mode='classification'
)

np.random.seed(SEED)
N = X_test4.shape[0]
take = min(600, N)
idx = np.random.choice(np.arange(N), size=take, replace=False)

F = X_train4.shape[1]
contrib = np.zeros((take, F), dtype=np.float32)

for j, i in enumerate(idx):

```

```

exp = expl_cnnlstm_emb.explain_instance(
    X_test4[i], predict_proba_cnnlstm_emb,
    num_features=15, top_labels=2
)
for rule, w in exp.as_list(label=1): # label=1 legitimate
    tok = rule.split()[0] # "embK"
    if tok.startswith("emb"):
        try:
            k = int(tok[3:])
            if 0 <= k < F:
                contrib[j, k] += abs(float(w))
        except Exception:
            pass

imp = contrib.mean(axis=0) # (100,
pd.DataFrame({
    "feature": [f"emb{i}" for i in range(F)],
    "importance_abs_mean": imp
}).to_csv(OUT_CSV, index=False)
print("[Saved]", OUT_CSV)

top = np.argsort(imp)[::-1][:TOPK]
feats = [f"emb{i}" for i in top][::-1]
vals = imp[top][::-1]

plt.figure(figsize=(8,5))
plt.barh(feats, vals) # KHÔNG set màu theo yêu cầu
plt.xlabel("|LIME| mean contribution")
plt.title("CNN-LSTM embedding – Top-15 by |LIME|")
plt.tight_layout(); plt.savefig(OUT_PNG, dpi=150); plt.close()
print("[Saved]", OUT_PNG)

def lime_barplot(exp, label=1, title="CNN-LSTM-EMB – LIME per-sample", savepath=None):
    pairs = exp.as_list(label=label) # [(rule, weight), ...]

```

```

if not pairs:
    print("[WARN] Không có rule để vẽ."); return

# sắp xếp theo |weight|
rules, weights = zip(*pairs)
order = np.argsort(np.abs(weights))[:-1]
rules = [rules[i] for i in order]
weights = [weights[i] for i in order]

plt.figure(figsize=(8,5))
y = list(range(len(rules)))[::-1]
w = list(weights)[::-1]
r = list(rules)[::-1]
plt.barh(y, w)
plt.yticks(y, r)
plt.xlabel("LIME contribution (\u00b1)")
plt.title(title)
plt.tight_layout()

if savepath:
    plt.savefig(savepath, dpi=150)
    plt.close()
    print("[Saved]", savepath)
else:
    plt.show()

for i in [0, 1, 2]:
    exp = expl_cnnlstm_emb.explain_instance(
        X_test4[i], predict_proba_cnnlstm_emb,
        num_features=15, top_labels=2
    )
    png_path = os.path.join(OUT_DIR_SAMPLES, f'LIME_CNNLSTM_EMB_sample_{i}.png')
    lime_barplot(
        exp, label=1,

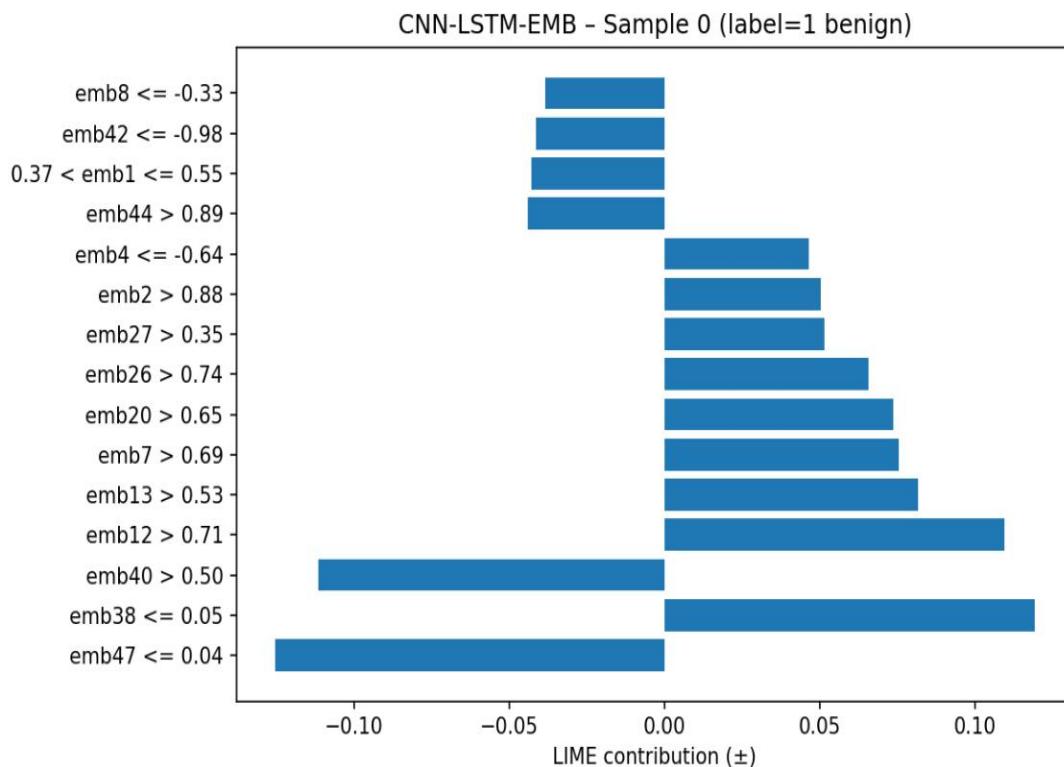
```

```

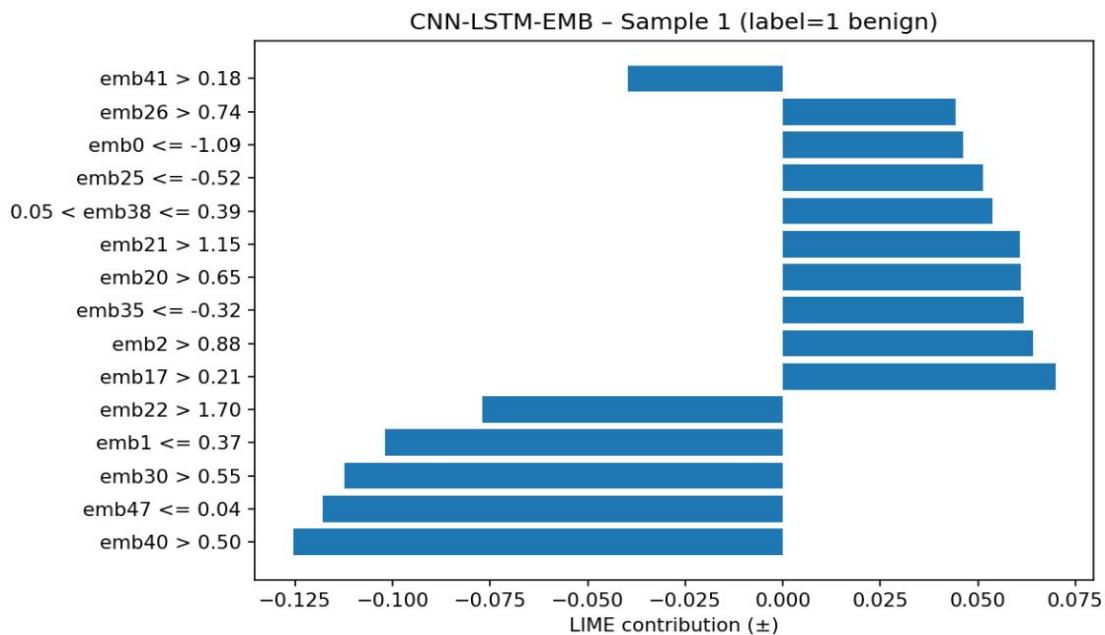
    title=f"CNN-LSTM-EMB – Sample {i} (label=1 legitimate)",
    savepath=png_path
)
print("Hoàn tất CNN-LSTM Explain Using LIME embedding.")

```

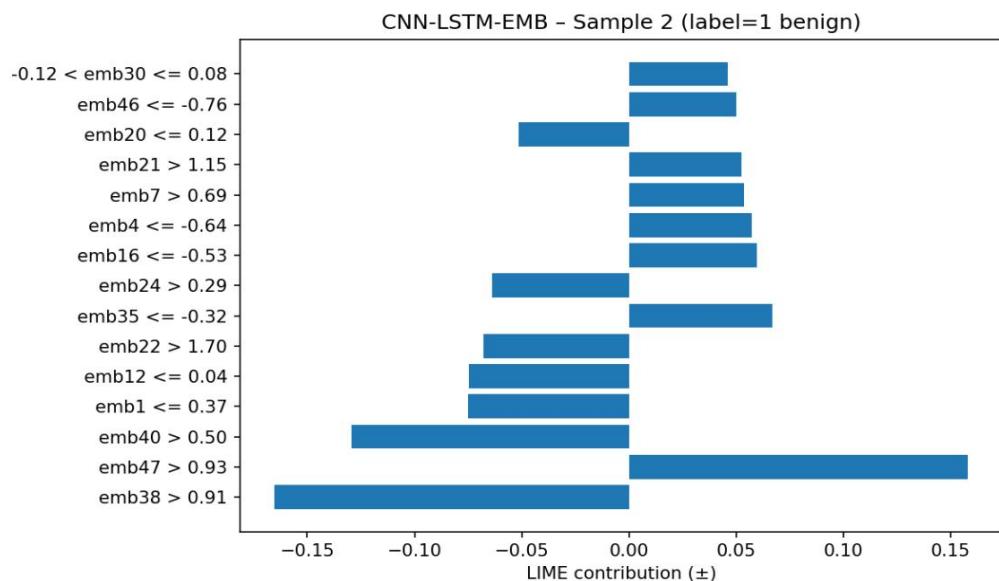
- Giải thích mô hình:



Hình 2.162 LIME contribution sample 0 CNN-LSTM model



Hình 2.163 LIME contribution sample 1 CNN-LSTM model



Hình 2.164 LIME contribution sample 2 CNN-LSTM model

2.3.5 ALBERT

Giải thích mô hình ALBERT:

- Load model và chuẩn bị dữ liệu

```
MODEL_PATH =  
"/content/drive/MyDrive/DeAnTotNghiệp/Trained_Models/PhiUSIIL_ALBERT_Tranning_Results/PhiUSIIL_ALBERT_Best_Model/"  
model = AutoModelForSequenceClassification.from_pretrained(MODEL_PATH)  
tokenizer = AutoTokenizer.from_pretrained("albert-base-v2", use_fast=True)  
  
import pandas as pd  
import numpy as np  
from sklearn.model_selection import train_test_split  
  
# Load lại dataset gốc  
df = pd.read_csv('/content/drive/MyDrive/DeAnTotNghiệp/PhiUSIIL_final_dataset.csv', sep=',')  
  
# Giữ nguyên chia train/test như ban đầu  
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['label'], random_state=42)  
  
# Lấy ngẫu nhiên background và sample text  
background_size = 3000  
sample_size = 10000  
  
background_texts = train_df['url'].sample(background_size, random_state=42).tolist()  
sample_texts = test_df['url'].sample(sample_size, random_state=42).tolist()  
  
print(f' Background: {len(background_texts)} | Samples: {len(sample_texts)}')  
  
masker = shap.maskers.Text(tokenizer)  
explainer = shap.Explainer(predict_proba, masker, output_names=["class_0", "class_1"])
```

```

batch_size = 2500
all_shap_values = []
print("☒ Bắt đầu tính SHAP values...\n")

for i in tqdm(range(0, len(sample_texts), batch_size)):
    batch = sample_texts[i:i+batch_size]

    # Tính SHAP cho batch
    shap_values = explainer(batch)

    all_shap_values.extend(shap_values)
    torch.cuda.empty_cache()
    gc.collect()

    print(f"❖ Done {i+len(batch)}/{len(sample_texts)}")

with open("/kaggle/working/PhiUSIIL_ALBERT_SHAP_values.pkl", "wb") as f:
    pickle.dump(all_shap_values, f)

summary_data = {
    "text": sample_texts,
    "pred_class": [np.argmax(v.values.mean(0)) for v in all_shap_values],
    "mean_abs_shap": [np.mean(np.abs(v.values)) for v in all_shap_values],
}
pd.DataFrame(summary_data).to_csv("/kaggle/working/PhiUSIIL_ALBERT_SHAP_summary.csv", index=False)

print("\n❖ Lưu thành công shap values & summary!")

def prepare_shap_with_aggregated_tokens(shap_values_list, top_k_tokens=50):
    """
    Aggregate SHAP values theo các tokens phổ biến nhất
    Tạo features có tên có ý nghĩa thay vì "token_0", "token_1"
    """

```

```

# Bước 1: Tìm top_k tokens xuất hiện nhiều nhất
token_counts = {}
for shap_val in shap_values_list:
    for token in shap_val.data:
        token = str(token).strip()
        if token and token not in ['[CLS]', '[SEP]', '[PAD]']:
            token_counts[token] = token_counts.get(token, 0) + 1

# Lấy top_k tokens
top_tokens = sorted(token_counts.items(), key=lambda x: x[1], reverse=True)[:top_k_tokens]
top_token_list = [t[0] for t in top_tokens]
token_to_idx = {token: idx for idx, token in enumerate(top_token_list)}

print(f"Top {top_k_tokens} most frequent tokens:")
for i, (token, count) in enumerate(top_tokens[:10], 1):
    print(f" {i:2d}. '{token}' ({count} occurrences)")

# Bước 2: Tạo ma trận SHAP values cho các tokens này
n_samples = len(shap_values_list)
shap_matrix = np.zeros((n_samples, top_k_tokens))
data_matrix = np.zeros((n_samples, top_k_tokens)) # 1 nếu token xuất hiện, 0 nếu không

for i, shap_val in enumerate(shap_values_list):
    values = shap_val.values[:, 1] if len(shap_val.values.shape) > 1 else shap_val.values
    tokens = shap_val.data

    for token, value in zip(tokens, values):
        token = str(token).strip()
        if token in token_to_idx:
            idx = token_to_idx[token]
            shap_matrix[i, idx] += value # Cộng dồn nếu token xuất hiện nhiều lần
            data_matrix[i, idx] = 1 # Đánh dấu token có xuất hiện

return shap_matrix, data_matrix, top_token_list

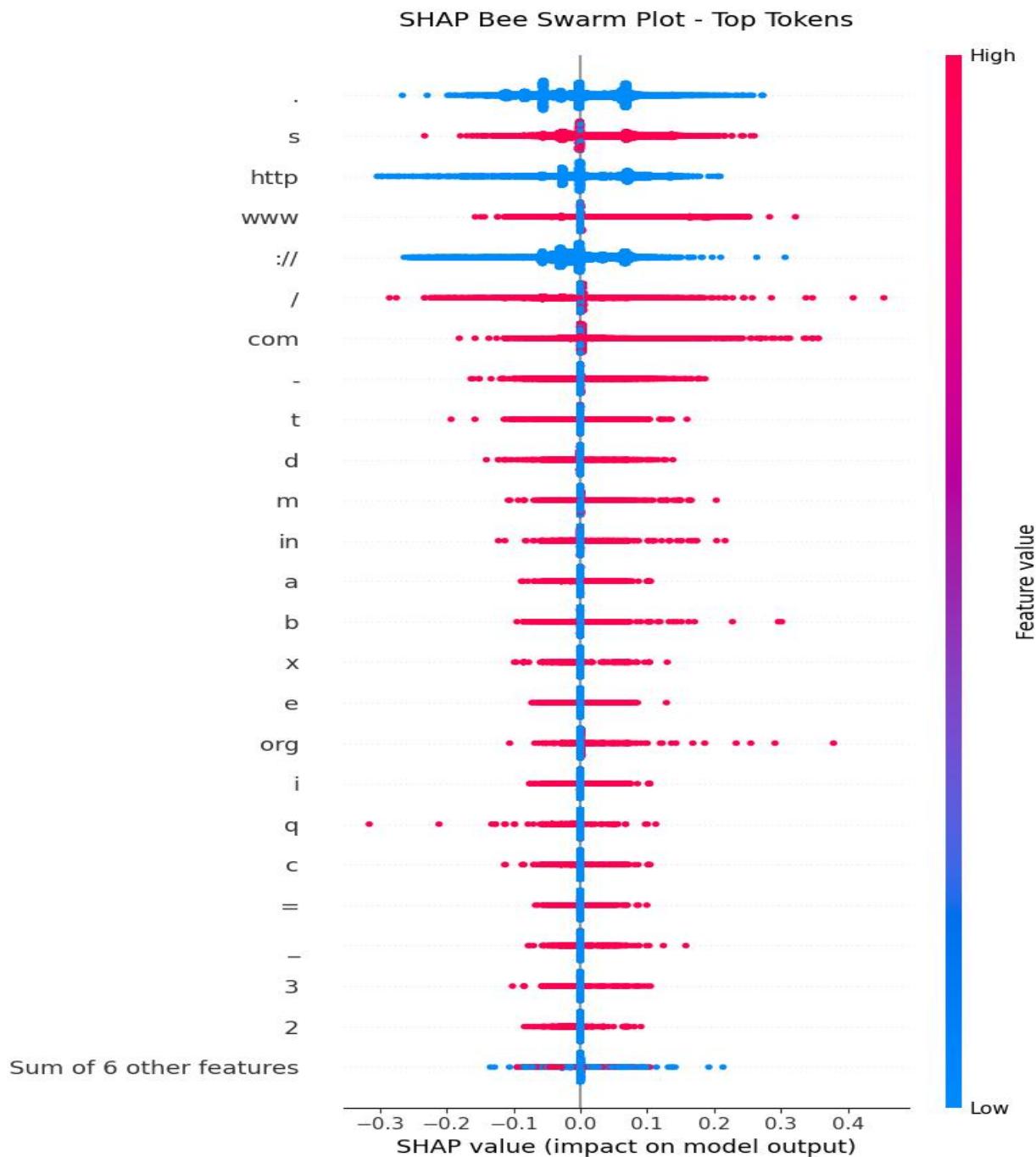
```

```
print("\n Đang tạo bee swarm plot với token names...")
shap_matrix_agg, data_matrix_agg, feature_names = prepare_shap_with_aggregated_tokens(
    all_shap_values,
    top_k_tokens=30
)

shap_explanation_agg = shap.Explanation(
    values=shap_matrix_agg,
    data=data_matrix_agg,
    feature_names=feature_names
)
```

- Giải thích mô hình với SHAP:

```
plt.figure(figsize=(10, 10))
shap.plots.beeswarm(shap_explanation_agg, max_display=25, show=False)
plt.title("SHAP Bee Swarm Plot - Top Tokens", fontsize=14, pad=20)
plt.tight_layout()
plt.savefig("/content/drive/MyDrive/DeAnTotNghiep/Explain_Models/PhiUSIIL_ALBERT_SH
AP/shap_beeswarm_tokens.png", dpi=300, bbox_inches='tight')
plt.show()
```



Hình 2.165 Tính SHAP value ALBERT model

```

import numpy as np
import shap
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display, HTML

sample_idx = 42
sample_text = sample_texts[sample_idx]
sample_shap = all_shap_values[sample_idx]

sample_probs = predict_proba_auto([sample_text])[0]
print(f"Class 0 (Phishing): {sample_probs[0]:.6f}") #~0.000031
print(f"Class 1 (Legitimate): {sample_probs[1]:.6f}") #~0.999969

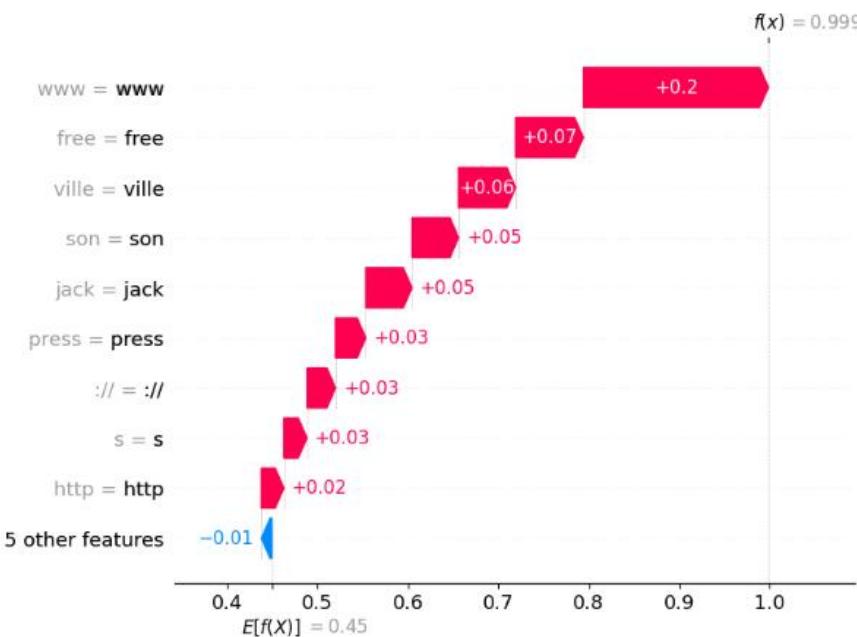
shap.plots.text(sample_shap)

```



Hình 2.166 Vẽ SHAP plot [42:] ALBERT model

```
shap.plots.waterfall(sample_shap[:, 1])
```



Hình 2.167 Vẽ `shap.plots.waterfall(sample_shap[:, 1])` ALBERT model

```
def plot_top_tokens_bar(shap_values, data, class_idx=1, top_n=15):
    """Vẽ bar plot cho top tokens có impact lớn nhất"""
    values = shap_values[:, class_idx]
    tokens = data

    # Lấy top tokens (both positive and negative)
    abs_values = np.abs(values)
    top_indices = np.argsort(abs_values)[-top_n:][::-1]

    top_tokens = [tokens[i] for i in top_indices]
    top_values = [values[i] for i in top_indices]

    # Vẽ
    fig, ax = plt.subplots(figsize=(10, 6))
    colors = ['red' if v < 0 else 'blue' for v in top_values]
    bars = ax.barh(range(len(top_tokens)), top_values, color=colors, alpha=0.7)
```

```
ax.set_yticks(range(len(top_tokens)))
ax.set_yticklabels([f'{t}' for t in top_tokens])
ax.set_xlabel('SHAP Value (Impact on Model Output)', fontsize=11)
ax.set_title(f'Top {top_n} Tokens by Impact - Sample #{sample_idx}',
            fontsize=13, fontweight='bold')
ax.axvline(x=0, color='black', linestyle='--', linewidth=0.8)
ax.invert_yaxis()

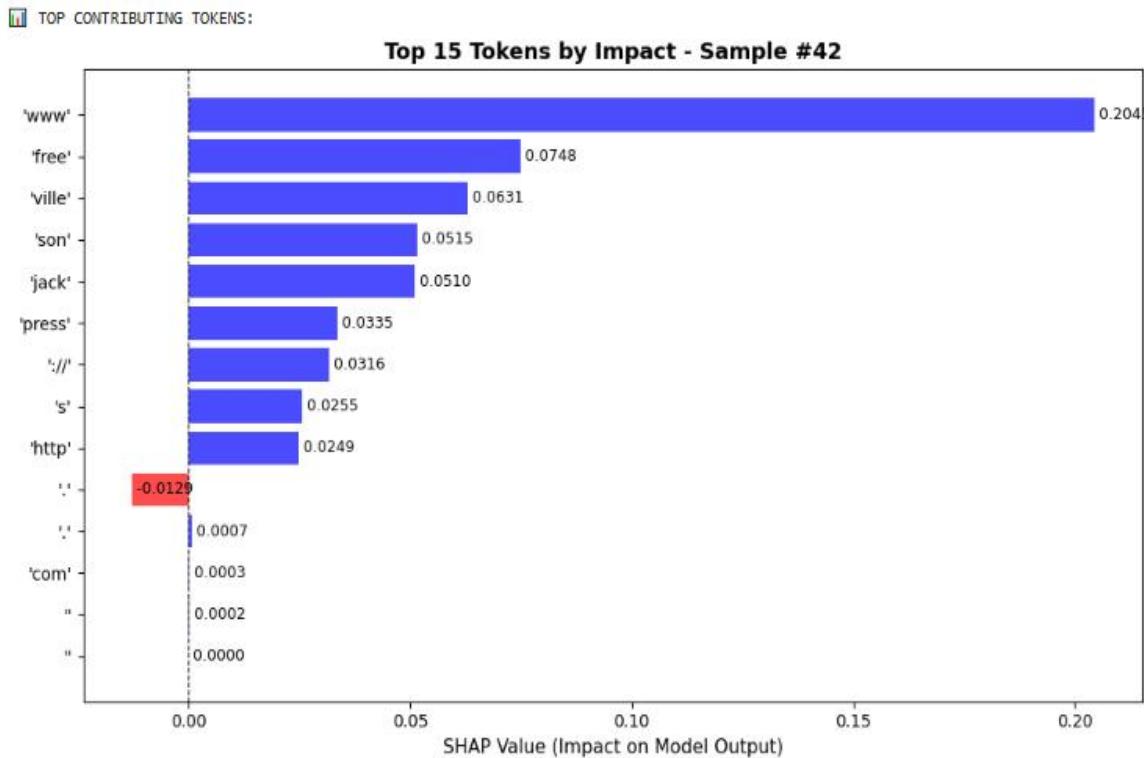
# Add value labels
for i, (bar, val) in enumerate(zip(bars, top_values)):
    ax.text(val, i, f'{val:.4f}', va='center', fontsize=9)

plt.tight_layout()

plt.savefig(f'/content/drive/MyDrive/DeAnTotNghiệp/Explain_Models/PhiUSIIL_ALBERT_SH
AP/sample_{sample_idx}_bar.png", dpi=300, bbox_inches='tight')
plt.show()

return top_tokens, top_values

print("\nTOP CONTRIBUTING TOKENS:")
top_tokens, top_values = plot_top_tokens_bar(sample_shap.values, sample_shap.data,
                                              class_idx=1, top_n=15)
```



Hình 2.168 Top 15 tokens impact sample 42 ALBERT model

```
def create_token_analysis_table(shap_values, data, class_idx=1):
    """Tạo bảng phân tích chi tiết từng token"""
    import pandas as pd

    values = shap_values[:, class_idx]
    tokens = data

    # Tạo DataFrame
    analysis_data = []
    for i, (token, value) in enumerate(zip(tokens, values)):
        if token.strip() and token not in ['[PAD]', '[CLS]', '[SEP]']:
            analysis_data.append({
                'Position': i,
```

```

'Token': token,
'SHAP Value': value,
'Abs SHAP': abs(value),
'Impact': 'Push to Legitimate' if class_idx == 1 else 'Push to Phishing',
'Strength': ' Strong' if abs(value) > 0.05 else
    ' Medium' if abs(value) > 0.02 else ' Weak'
})

df = pd.DataFrame(analysis_data)
df = df.sort_values('Abs SHAP', ascending=False)

return df

print("\n DETAILED TOKEN ANALYSIS TABLE:")
token_df = create_token_analysis_table(sample_shap.values, sample_shap.data, class_idx=1)
print(token_df.head(20).to_string(index=False))

# Lưu full table
token_df.to_csv(f"/content/drive/MyDrive/DeAnTotNghiep/Explain_Models/PhiUSIIL_ALBERT_SHAP/sample_{sample_idx}_token_analysis.csv", index=False)

```

DETAILED TOKEN ANALYSIS TABLE:						
Position	Token	SHAP Value	Abs SHAP	Impact	Strength	
4	www	0.204548	0.204548	Push to Benign	Strong	● Strong
9	free	0.074774	0.074774	Push to Benign	Strong	● Strong
8	ville	0.063056	0.063056	Push to Benign	Strong	● Strong
7	son	0.051504	0.051504	Push to Benign	Strong	● Strong
6	jack	0.050990	0.050990	Push to Benign	Strong	● Strong
10	press	0.033465	0.033465	Push to Benign	Medium	○ Medium
3	::/	0.031590	0.031590	Push to Benign	Medium	○ Medium
2	s	0.025497	0.025497	Push to Benign	Medium	○ Medium
1	http	0.024933	0.024933	Push to Benign	Medium	○ Medium
5	.	-0.012888	0.012888	Push to Benign	Weak	● Weak
11	.	0.000670	0.000670	Push to Benign	Weak	● Weak
12	com	0.000291	0.000291	Push to Benign	Weak	● Weak

Hình 2.169 Phân tích chi tiết ảnh hưởng của các token đến model ALBERT

```

def plot_cumulative_impact(shap_values, data, class_idx=1):
    """Vẽ cumulative SHAP values theo position"""
    values = shap_values[:, class_idx]

    # Tính cumulative sum
    cumsum = np.cumsum(values)
    base_value = sample_shap.base_values[class_idx]

    fig, ax = plt.subplots(figsize=(14, 5))

    # Plot cumulative impact
    ax.plot(cumsum, linewidth=2, label='Cumulative SHAP')
    ax.axhline(y=0, color='black', linestyle='--', linewidth=0.8, alpha=0.5)
    ax.fill_between(range(len(cumsum)), cumsum, 0, alpha=0.3)

    # Highlight important tokens
    abs_values = np.abs(values)
    top_indices = np.argsort(abs_values)[-10:]

    for idx in top_indices:
        if data[idx].strip():
            ax.scatter(idx, cumsum[idx], s=100, c='red', zorder=5, alpha=0.7)
            ax.annotate(f'{data[idx]}',
                        xy=(idx, cumsum[idx]),
                        xytext=(0, 10),
                        textcoords='offset points',
                        ha='center',
                        fontsize=8,
                        bbox=dict(boxstyle='round,pad=0.3', facecolor='yellow', alpha=0.7))

    ax.set_xlabel('Token Position', fontsize=11)
    ax.set_ylabel('Cumulative SHAP Value', fontsize=11)
    ax.set_title(f'Cumulative Impact Over Token Sequence - Sample #{sample_idx}', 
                fontsize=13, fontweight='bold')

```

```

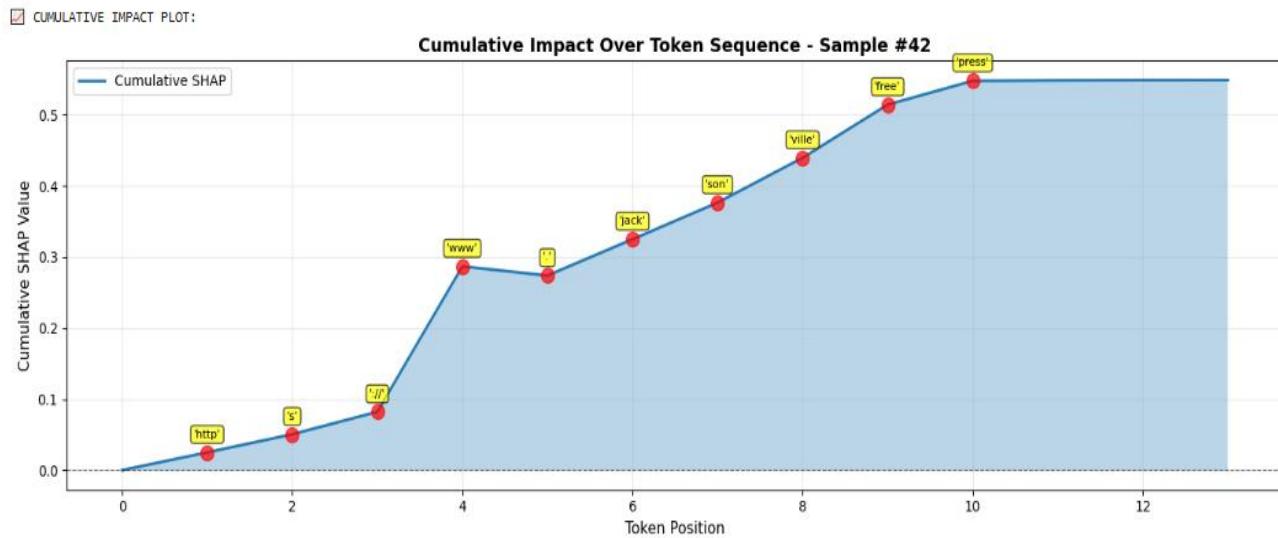
ax.legend()
ax.grid(True, alpha=0.3)

plt.tight_layout()

plt.savefig(f"/content/drive/MyDrive/DeAnTotNghiep/Explain_Models/PhiUSIL_ALBERT_SH
AP/sample_{sample_idx}_cumulative.png", dpi=300, bbox_inches='tight')
plt.show()

print("\n CUMULATIVE IMPACT PLOT:")
plot_cumulative_impact(sample_shap.values, sample_shap.data, class_idx=1)

```



Hình 2.170 Mức độ ảnh hưởng các token đến ALBERT model

```

def plot_shap_heatmap(shap_values, data, top_n=20):
    """Heatmap so sánh SHAP values giữa 2 classes"""
    # Lấy top tokens theo abs value trung bình
    mean_abs_shap = np.abs(shap_values).mean(axis=1)
    top_indices = np.argsort(mean_abs_shap)[-top_n:][:-1]

    # Tạo ma trận 2 cột (class 0, class 1)
    heatmap_data = shap_values[top_indices, :]

```

```
token_labels = [data[i] if data[i].strip() else f"[pos_{i}]" for i in top_indices]

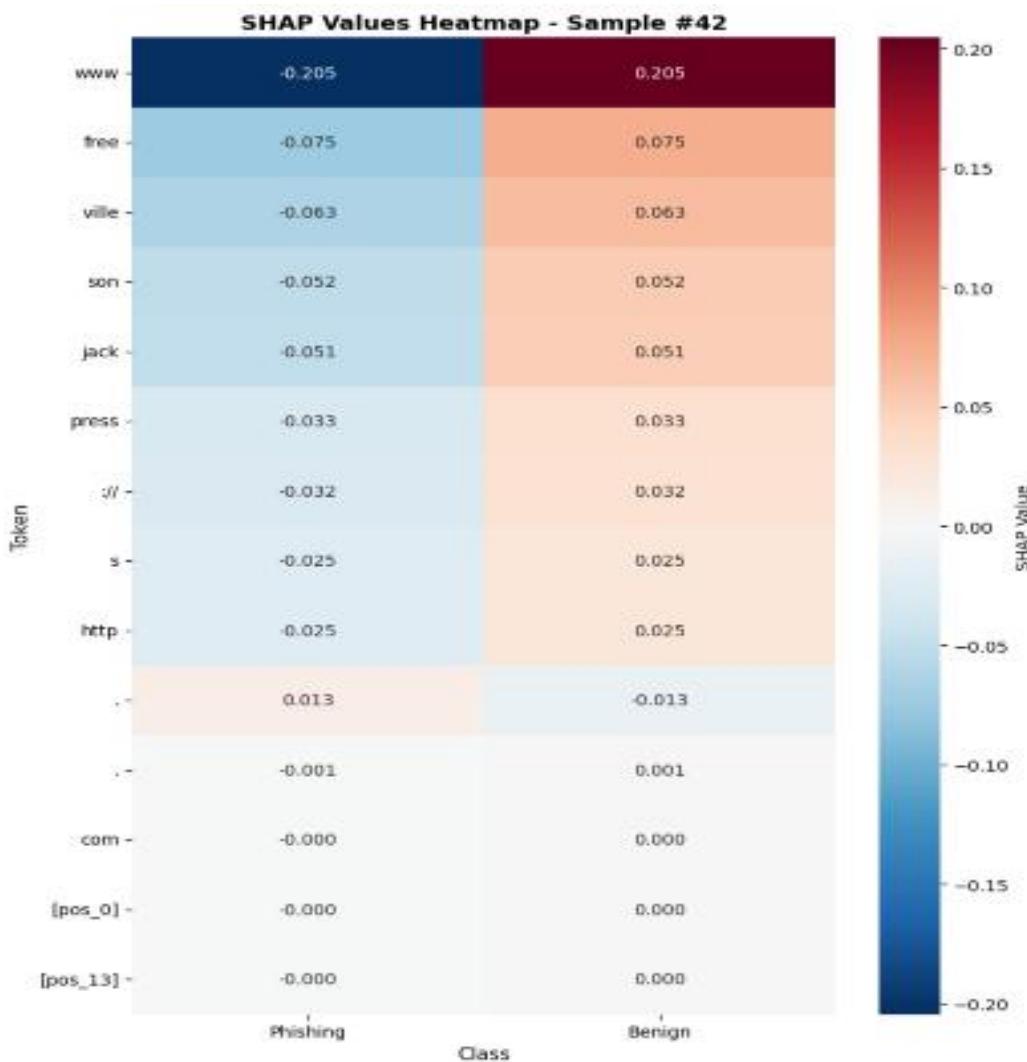
fig, ax = plt.subplots(figsize=(8, 10))
sns.heatmap(heatmap_data,
            annot=True,
            fmt='.3f',
            cmap='RdBu_r',
            center=0,
            cbar_kws={'label': 'SHAP Value'},
            yticklabels=token_labels,
            xticklabels=['Phishing', 'Legitimate'],
            ax=ax)

ax.set_title(f'SHAP Values Heatmap - Sample #{sample_idx}',
            fontsize=13, fontweight='bold')
ax.set_xlabel('Class', fontsize=11)
ax.set_ylabel('Token', fontsize=11)

plt.tight_layout()

plt.savefig(f'/content/drive/MyDrive/DeAnTotNghiệp/Explain_Models/PhiUSIIL_ALBERT_SH
AP/sample_{sample_idx}_heatmap.png", dpi=300, bbox_inches='tight')
plt.show()

print("\n HEATMAP - Both Classes Comparison:")
plot_shap_heatmap(sample_shap.values, sample_shap.data, top_n=20)
```

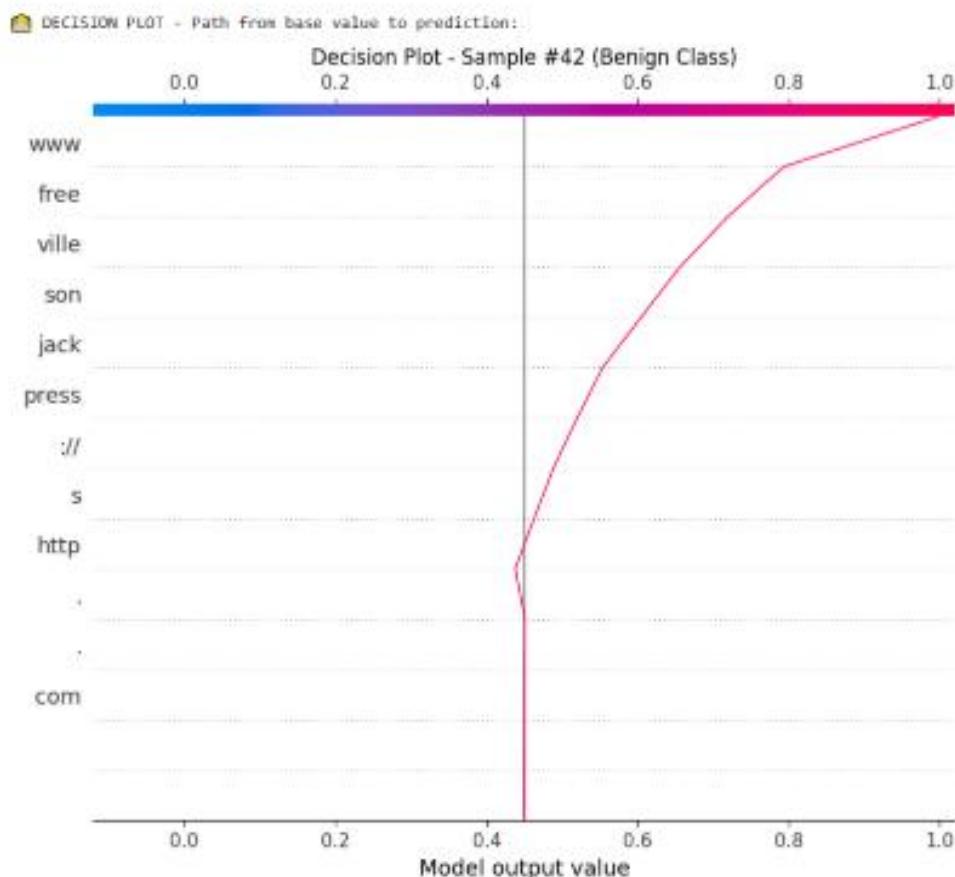


Hình 2.171 SHAP value heatmap sample 42 ALBERT model

```
print("\n DECISION PLOT - Path from base value to prediction:")
```

```
# Decision plot cho legitimate class (class 1)
shap.decision_plot(
    sample_shap.base_values[1],
    sample_shap.values[:, 1],
    sample_shap.data,
```

```
feature_display_range=slice(-1, -21, -1), # Top 20 features
show=False
)
plt.title(f'Decision Plot - Sample #{sample_idx} (Legitimate Class)", fontsize=12)
plt.tight_layout()
plt.savefig(f'/content/drive/MyDrive/DeAnTotNghiep/Explain_Models/PhiUSIIL_ALBERT_SH
AP/sample_{sample_idx}_decision.png", dpi=300, bbox_inches='tight')
plt.show()
```



Hình 2.172 Decision Plot - Sample 42 ALBERT model

```

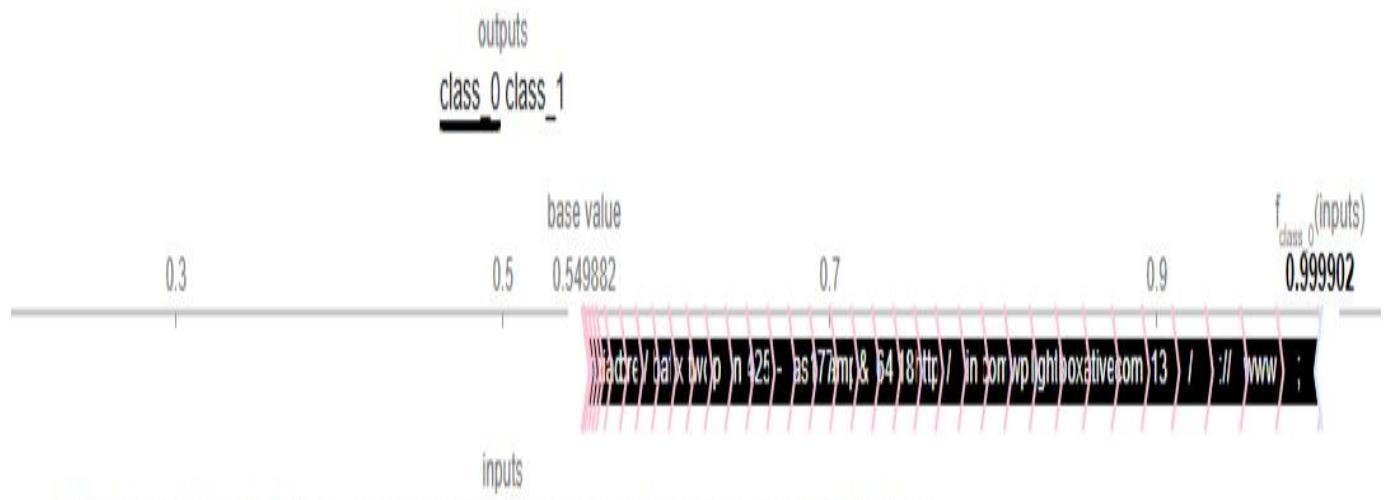
import numpy as np
import shap
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display, HTML

sample_idx = 125
sample_text = sample_texts[sample_idx]
sample_shap = all_shap_values[sample_idx]

sample_probs = predict_proba_auto([sample_text])[0]
print(f"Class 0 (Phishing): {sample_probs[0]:.6f}") #~0.000031
print(f"Class 1 (Legitimate): {sample_probs[1]:.6f}") #~0.999969

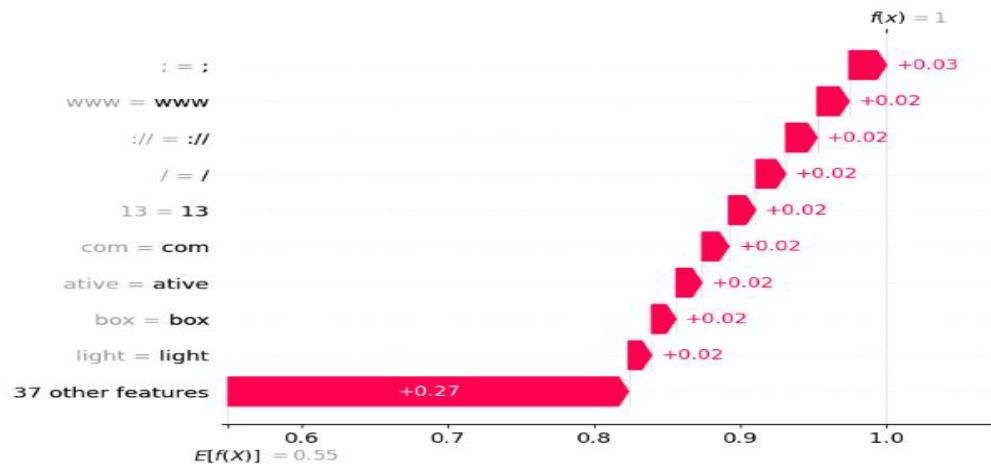
shap.plots.text(sample_shap)

```



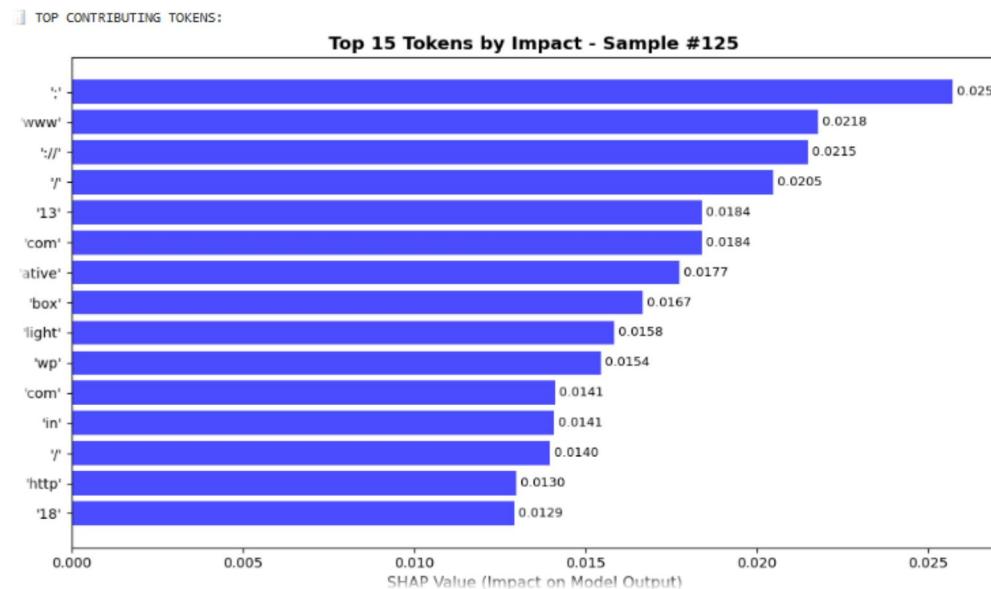
Hình 2.173 Phishing sample 125 SHAP ALBERT model

```
shap.plots.waterfall(sample_shap[:, 0])
```



Hình 2.174 Vẽ shap.plots.waterfall(sample_shap[:, 0]) ALBERT model

```
print("\n TOP CONTRIBUTING TOKENS:")
top_tokens, top_values = plot_top_tokens_bar(sample_shap.values, sample_shap.data,
                                              class_idx=0, top_n=15)
```



Hình 2.175 Top 15 tokens impact sample 125 ALBERT model

```

print("\nDETAILED TOKEN ANALYSIS TABLE:")
token_df = create_token_analysis_table(sample_shap.values, sample_shap.data, class_idx=0)
print(token_df.head(20).to_string(index=False))

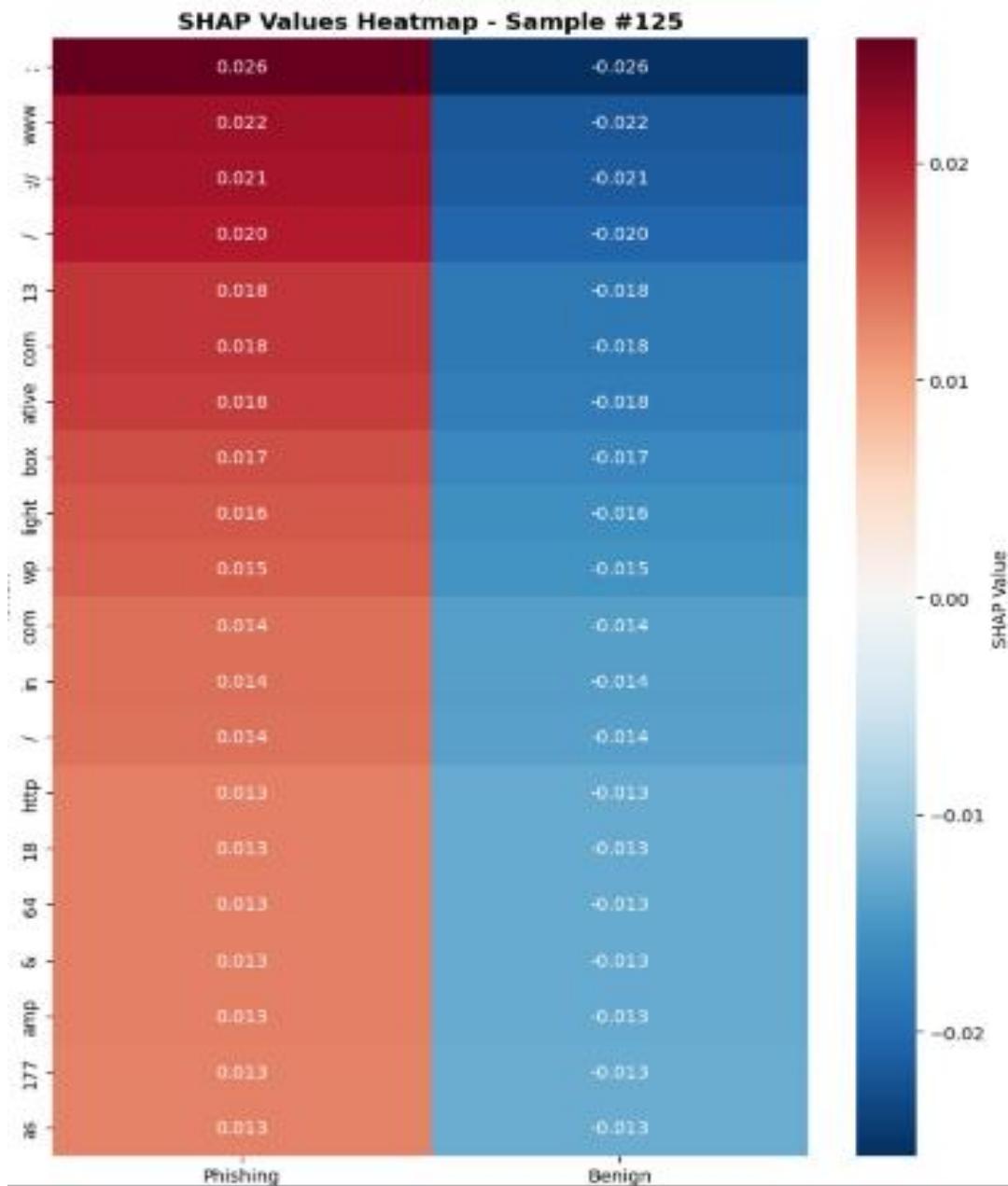
# Lưu full table
token_df.to_csv(f"/content/drive/MyDrive/DeAnTotNghiệp/Explain_Models/PhiUSIIL_ALBERT_T_SHAP/sample_{sample_idx}_token_analysis.csv", index=False)

```

DETAILED TOKEN ANALYSIS TABLE:						
Position	Token	SHAP Value	Abs SHAP	Impact	Strength	
44	;	0.025707	0.025707	Push to Phishing	Medium	
3	www	0.021775	0.021775	Push to Phishing	Medium	
2	//	0.021488	0.021488	Push to Phishing	Medium	
18	/	0.020466	0.020466	Push to Phishing	Medium	
29	13	0.018388	0.018388	Push to Phishing	Weak	
7	com	0.018384	0.018384	Push to Phishing	Weak	
6	ative	0.017720	0.017720	Push to Phishing	Weak	
31	box	0.016664	0.016664	Push to Phishing	Weak	
32	light	0.015825	0.015825	Push to Phishing	Weak	
12	wp	0.015446	0.015446	Push to Phishing	Weak	
10	com	0.014092	0.014092	Push to Phishing	Weak	
30	in	0.014061	0.014061	Push to Phishing	Weak	
11	/	0.013953	0.013953	Push to Phishing	Weak	
1	http	0.012971	0.012971	Push to Phishing	Weak	
41	18	0.012904	0.012904	Push to Phishing	Weak	
40	64	0.012900	0.012900	Push to Phishing	Weak	
42	&	0.012881	0.012881	Push to Phishing	Weak	
43	amp	0.012877	0.012877	Push to Phishing	Weak	
38	177	0.012845	0.012845	Push to Phishing	Weak	
33	as	0.012830	0.012830	Push to Phishing	Weak	

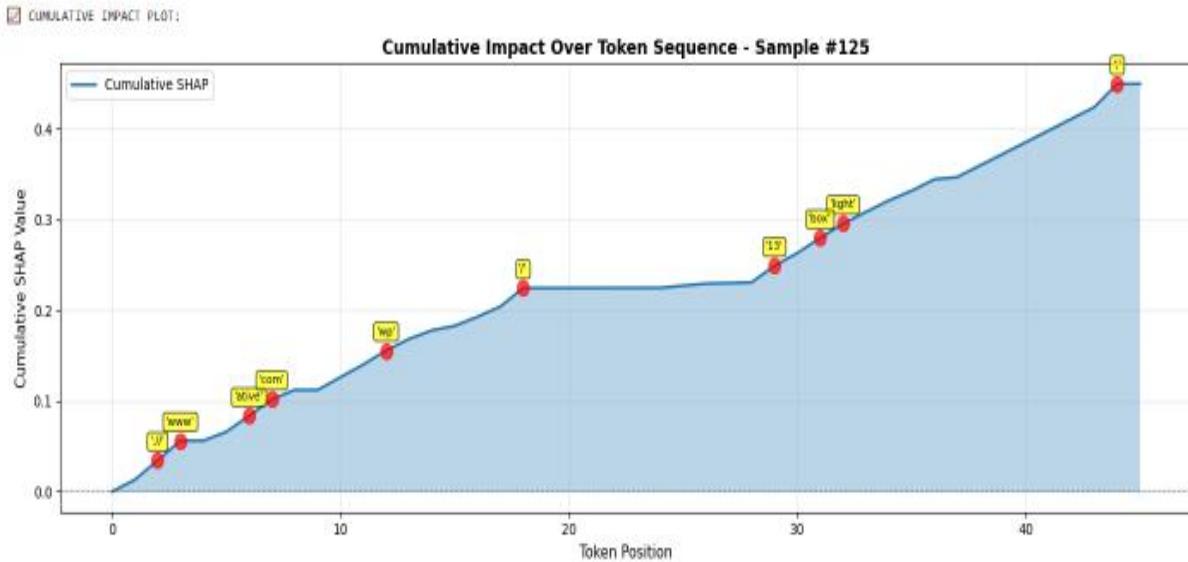
Hình 2.176 Phân tích chi tiết ảnh hưởng của các tokens đến model ALBERT

```
print("\n HEATMAP - Both Classes Comparison:")
plot_shap_heatmap(sample_shap.values, sample_shap.data, top_n=20)
```



Hình 2.177 SHAP value heatmap sample 125 ALBERT model

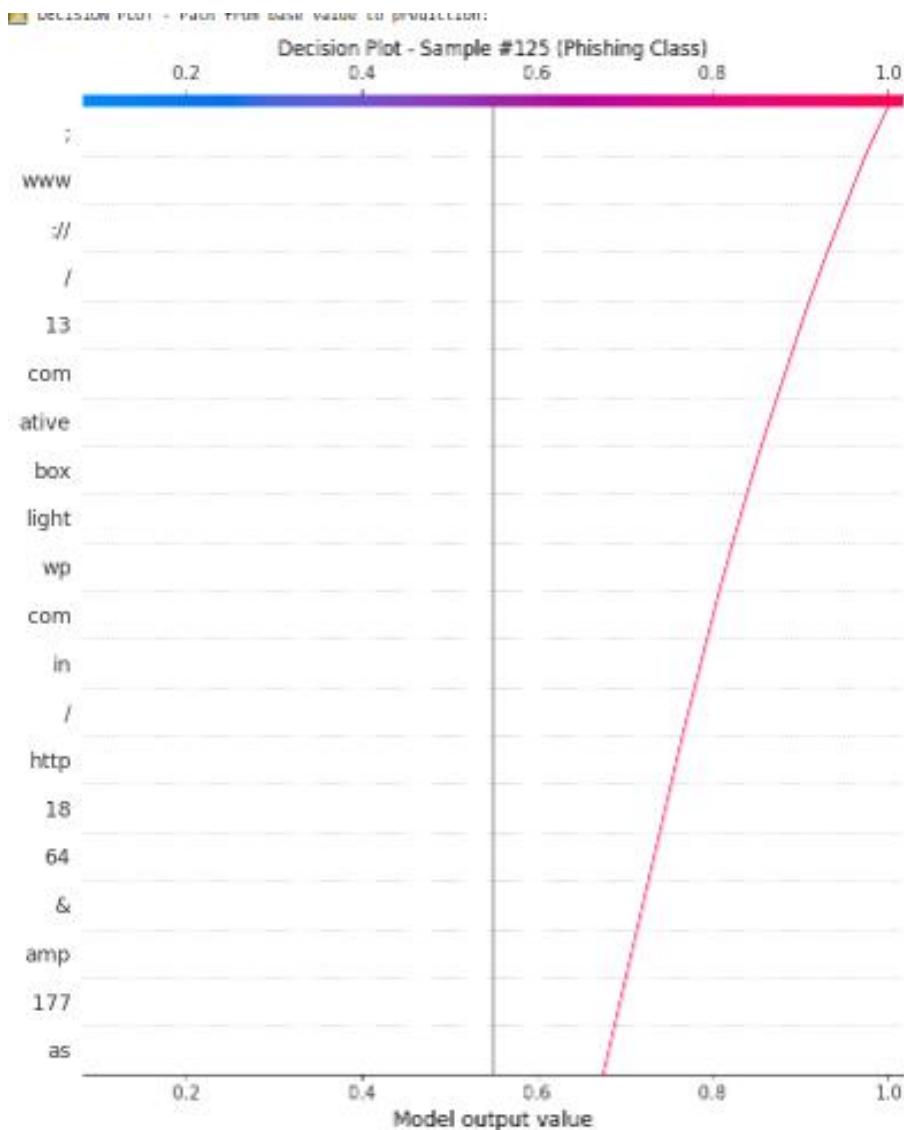
```
print("\n CUMULATIVE IMPACT PLOT:")
plot_cumulative_impact(sample_shap.values, sample_shap.data, class_idx=0)
```



Hình 2.178 Biểu đồ mức độ ảnh hưởng của các tokens đến model ALBERT

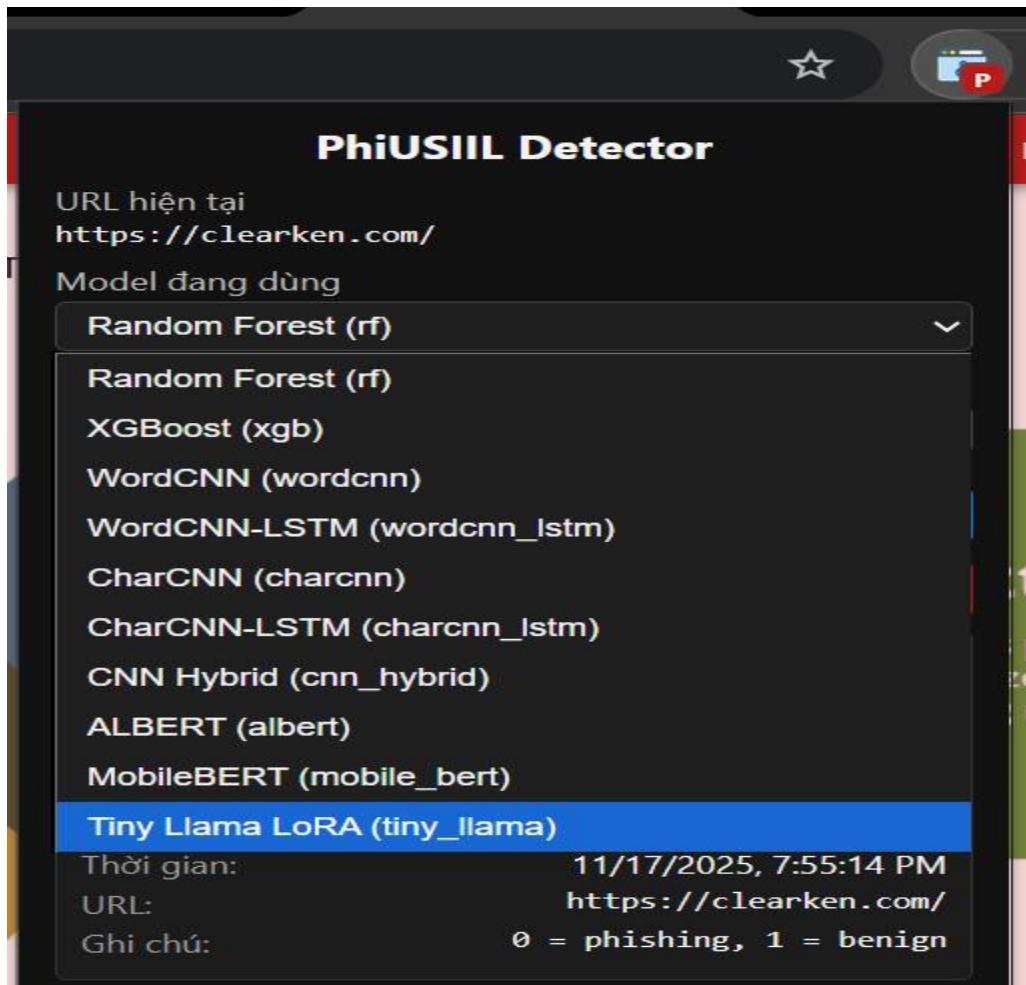
```
print("\n DECISION PLOT - Path from base value to prediction:")
```

```
# Decision plot cho legitimate class (class 0)
shap.decision_plot(
    sample_shap.base_values[0],
    sample_shap.values[:, 0],
    sample_shap.data,
    feature_display_range=slice(-1, -21, -1), # Top 20 features
    show=False
)
plt.title(f'Decision Plot - Sample #{sample_idx} (Phishing Class)', fontsize=12)
plt.tight_layout()
plt.savefig(f'/content/drive/MyDrive/DeAnTotNghiệp/Explain_Models/PhiUSIIL_ALBERT_SH
AP/sample_{sample_idx}_decision.png', dpi=300, bbox_inches='tight')
plt.show()
```

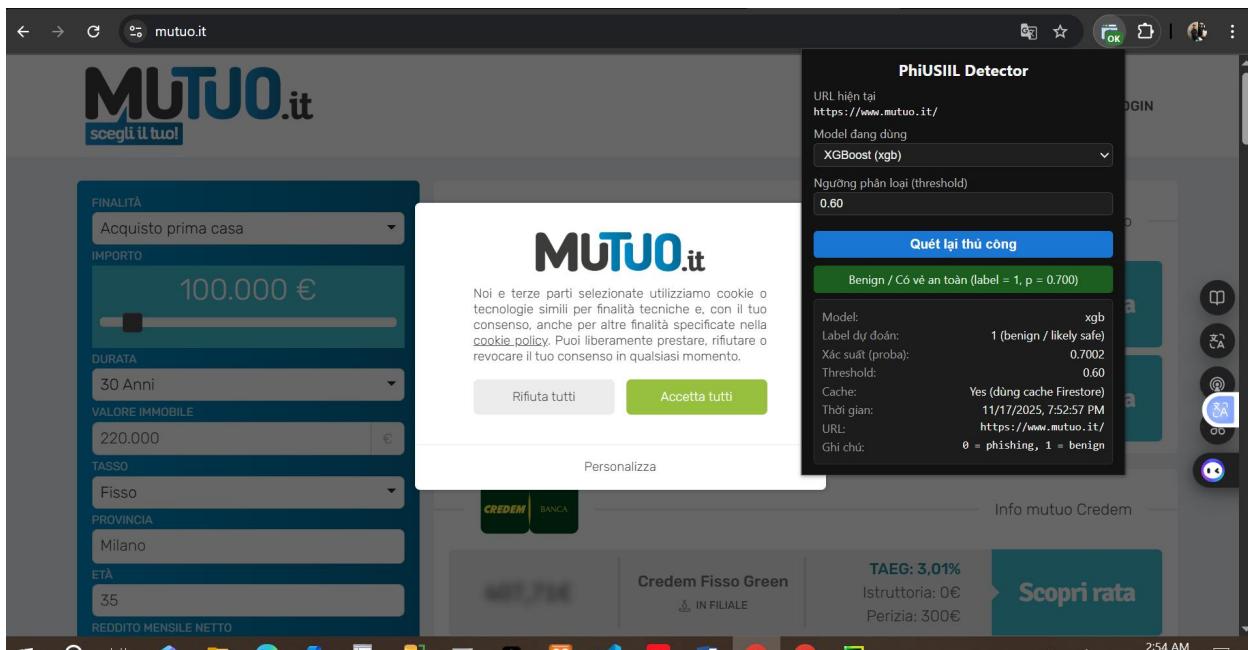


Hình 2.179 Decision Plot sample 125 Phishing Class ALBERT model

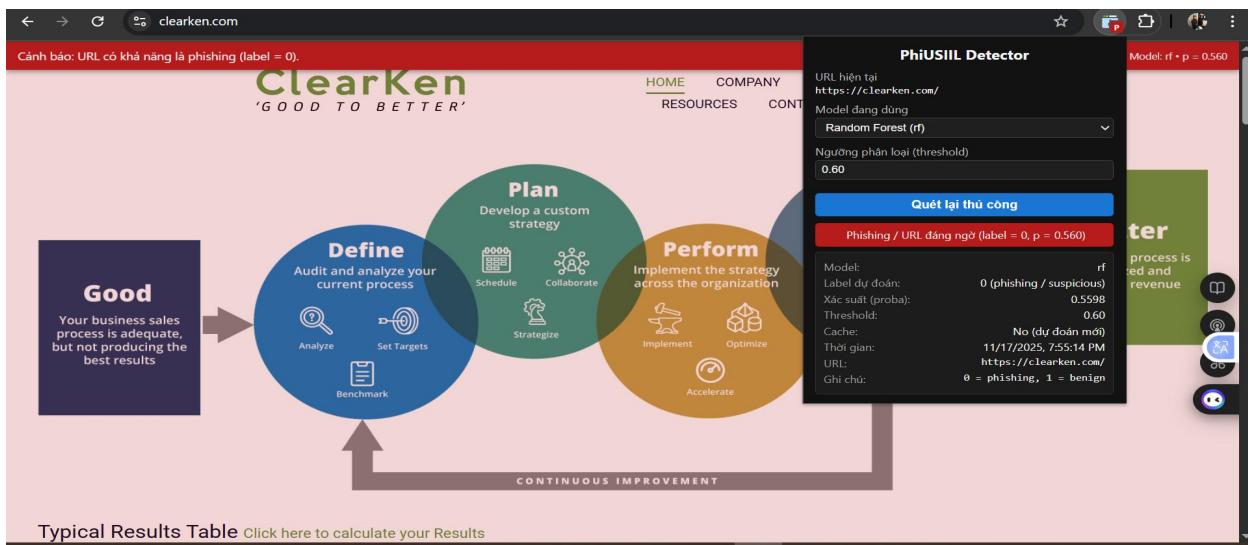
2.5 Kiểm thử dự đoán với các mô hình đã huấn luyện



Hình 2.180 Extension tích hợp trên trình duyệt web sử dụng những models đã huấn luyện để phát hiện URL phising real-time



Hình 2.181 Giao diện demo dự đoán với url legitimate



Hình 2.182 Giao diện demo dự đoán với url phishing

CHƯƠNG 3: KẾT LUẬN VÀ ĐÁNH GIÁ

3.1 Kết quả thực nghiệm

Mô hình	F1 Score	Precision	Recall	PR-AUC	ROC-AUC
XGBoost	0.88	0.89	0.88	0.90	0.92
RF	0.86	0.87	0.86	0.84	0.89
WordCNN	0.89	0.93	0.96	0.90	0.92
CNN-LSTM	0.89	0.92	0.95	0.89	0.92
CharCNN	0.95	0.94	0.96	0.98	0.99
CharCNN-LSTM	0.96	0.95	0.97	0.98	0.99
CNN-Hybrid	0.93	0.89	0.97	0.97	0.97
ALBERT	0.96	0.96	0.97	0.99	0.99
MobileBERT	0.96	0.95	0.98	0.99	0.99

Bảng 3.1 So sánh các mô hình đã huấn luyện

3.2 Thảo luận

Đánh giá so sánh: Mô hình học máy truyền thống so với những mô hình học sâu kết hợp và mô hình ngôn ngữ lớn:

Tiêu chí	Machine Learning (XGBoost, RF)	Deep Learning / LLM (CNN, ALBERT, Hybrid)
Độ chính xác	Cao và đạt hiệu quả tối ưu nhờ tập đặc trưng thủ công được thiết kế tốt.	Cao. Tuy nhiên, đôi khi bị nhiễu do có gắng học từ các chuỗi ký tự ngẫu nhiên trong URL mà không qua tiền xử lý kỹ.
Tốc độ huấn luyện	Rất nhanh. Phù hợp cho các ứng dụng thời gian thực yêu cầu độ trễ thấp.	Chậm hơn đáng kể. Đặc biệt là các mô hình LLM (ALBERT, TinyLlama) đòi hỏi tài nguyên GPU lớn và thời gian suy luận lâu hơn.
Khả năng khai quát hóa	Phụ thuộc hoàn toàn vào chất lượng của đặc trưng đầu vào. Dễ bị qua mặt nếu kẻ tấn công thay đổi cấu trúc URL để né tránh các quy tắc.	Tốt hơn về mặt ngữ nghĩa. Có khả năng phát hiện các biến thể tấn công dựa trên ngữ cảnh mà các quy tắc cứng nhắc của ML có thể bỏ sót.
Khả năng giải thích	Dễ dàng giải thích qua SHAP/LIME dựa trên các đặc trưng cụ thể.	Khó giải thích hơn do đặc trưng đã bị token hóa trước khi đưa dữ liệu vào. Việc giải thích dựa trên Embedding vectors thường trừu tượng và khó hiểu đối với người dùng cuối.

Bảng 3.2 Đánh giá ưu và nhược điểm của các mô hình đã huấn luyện

Ưu điểm của các phương pháp kết hợp:

- Khắc phục điểm mù của những đặc trưng trích xuất thủ công: Các đặc trưng thủ công như số dấu chấm, độ dài url... rất hiệu quả nhưng giới hạn. Việc huấn luyện mô hình LLM như ALBERT/MobileBERT giúp mô hình hiểu được ý nghĩa của các từ trong URL, ví dụ: phát hiện "g00gle" là giả mạo của "google" dựa trên sự tương đồng ngữ nghĩa thay vì chỉ so khớp ký tự.
- Tính bền vững trước tấn công mới - Zero-day Attacks: Mô hình Hybrid không chỉ dựa vào các quy tắc cố định. Khi xuất hiện một dạng URL phishing mới chưa từng có trong tập train, phần mạng nơ-ron như CNN/LSTM có khả năng nhận diện các mẫu bất thường tốt hơn so với cây quyết định.

- Tận dụng sức mạnh của Pre-trained Models: Việc sử dụng ALBERT/TinyLlama cho phép tận dụng tri thức từ hàng tỷ văn bản đã học trước đó, giúp trích xuất đặc trưng sâu sắc hơn mà không cần huấn luyện lại từ đầu nhờ kỹ thuật LoRA/QLoRA.

Nhược điểm và hạn chế tồn tại:

- Vấn đề đánh đổi tài nguyên và hiệu năng: Để đạt được sự cải thiện nhỏ về khả năng hiểu ngữ nghĩa, mô hình Hybrid tiêu tốn tài nguyên tính toán gấp nhiều lần so với XGBoost. Điều này gây khó khăn cho việc triển khai trên các thiết bị người dùng cuối hoặc Extension trình duyệt nếu không có Server mạnh hỗ trợ.
- Nguy cơ Overfitting: Kết quả thực nghiệm cho thấy một số mô hình đạt độ chính xác tiệm cận tuyệt đối gần 99.9%. Điều này cảnh báo nguy cơ mô hình đã bị overfitting và học vẹt các đặc điểm đặc thù của bộ dữ liệu PhiUSIIL thay vì học bản chất của hành vi Phishing.

Vấn đề về độ chính xác và tốc độ:

- Nhánh Machine Learning gồm XGBoost/RF có tốc độ phản hồi gần như tức thời, phù hợp cho xử lý thời gian thực lưu lượng lớn.
- Nhánh LLM: Mặc dù đã áp dụng kỹ thuật lượng tử hóa 4-bit - QLoRA để giảm tải, tốc độ suy luận vẫn chậm hơn đáng kể so với hai nhánh còn lại \ tùy thuộc vào độ dài prompt.
- Hệ quả việc tích hợp LLM vào luồng xử lý chính có thể gây tắc nghẽn nếu áp dụng cho hệ thống tường lửa yêu cầu độ trễ thấp.

Phân tích các trường hợp nhận diện sai:

- False Positives: Các website hợp pháp nhưng có cấu trúc URL lộn xộn, chứa nhiều tham số ngẫu nhiên hoặc sử dụng tên miền phụ dài, ví dụ: các link CDN, link tracking marketing. Mô hình DL thường nhầm lẫn các chuỗi ký tự ngẫu nhiên này là hành vi mã hóa của hacker.
 - False Negatives: Một số trường hợp hacker tấn công và chiếm quyền điều khiển các website uy tín, ví dụ: uni-hcm.edu.vn/wp-content/uploads/.... Vì tên miền chính có độ uy tín cao, nhánh ML dựa trên Whitelist/Blacklist hoặc Lexical features thường bỏ qua, dẫn đến việc phân loại sai là an toàn.
-

KẾT LUẬN VÀ KIẾN NGHỊ

4.1 Kết luận

Từ những kết quả đánh giá mô hình thì nhóm chọn ra mô hình tốt nhất ở mỗi loại cho việc phát hiện url giả mạo như sau

Mô hình	Precision	Recall	F1-Score	PR-AUC	ROC-AUC
ML (XGBoost)	0.89	9.88	0.88	0.90	0.92
DL (CharCNN-LSTM)	0.95	0.97	0.96	0.98	0.99
LLM (ALBERT)	0.96	0.97	0.96	0.99	0.99

Bảng 3.3 Bảng kết quả model tốt nhất

4.2 Hướng phát triển

Tiếp cận đa phương thức: Mở rộng phạm vi phát hiện không chỉ dựa vào chuỗi URL mà tích hợp thêm Computer Vision để phân tích hình ảnh giao diện website và NLP để quét nội dung văn bản, giúp phát hiện các trang giả mạo thương hiệu tinh vi.

Thêm cơ chế học tăng cường: Xây dựng hệ thống tự động cập nhật dữ liệu từ các nguồn tình báo mới đe dọa để mô hình liên tục học các mẫu tấn công mới Zero-day attacks mà không cần huấn luyện lại từ đầu.

Tăng cường khả năng chống tấn công: Nghiên cứu mô phỏng các mẫu URL được sinh ra bởi AI để tấn công thử nghiệm hệ thống, từ đó huấn luyện lại mô hình nhằm tăng độ bền vững trước các kỹ thuật qua mặt bộ lọc của hacker.

TÀI LIỆU THAM KHẢO

- [1] A. Prasad and S. Chandra, “PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning” 26 June 2023, pp 5-8. , doi: 10.1016/j.cose.2023.103545.
- [2] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique”, Journal Of Artificial Intelligence Research, Volume 16, pp. 321-357, 2002, doi: 10.48550/arXiv.1106.1813
- [3] G. Ramkumar and L. Prasanna P, "Enhancing User Safety Through URL-Based Phishing Detection with Random Forest And Decision Tree" 2025 International Conference on Emerging Technologies in Engineering Applications (ICETEA), Puducherry, India, 05-06 June 2025, pp. 8-9, doi: 10.1109/ICETEA64585.2025.11099651
- [4] X. D. Hoang, D. L. Minh and T. T. Trang Ninh “A CNN-Based Model for Detecting Malicious URLs” 2023 RIVF International Conference on Computing and Communication Technologies (RIVF), Hanoi, Vietnam, 2023, pp. 15-16, doi: 10.1109/RIVF60135.2023.10471782.
- [5] H. Le, Q. Pham, D. Sahoo and S. C. H Hoi, “URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection”, 9 February 2018, pp. 5-6, doi: 10.48550/arXiv.1802.03162
- [6] H. Yuan, Z. Yang, X. Chen, Y. Li and W. Liu, “URL2Vec: URL Modeling with Character Embeddings for Fast and Accurate Phishing Website Detection”, 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), Melbourne, VIC, Australia, 11-13 December 2018, pp. 11-13, doi: 10.1109/BDCLOUD.2018.00050
- [7] W. Zhang, M. Ji, T. Li, P. Song and Y. He, “Sentiment Text Spatio-Temporal Feature Analysis based on CharCNN and BiLSTM”, 2023 4th International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE), Nanjing, China, 25-27 August 2023, pp. 20-22, doi: 10.1109/ICBASE59196.2023.10303214
- [8] F. Tajaddodianfar, J. W. Stokes and A. Gururajan, “Texception: A Character/Word-Level Deep Learning Model for Phishing URL Detection”, ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 20-08 May 2020, pp. 12-14, doi: 10.1109/ICASSP40776.2020.9053670

- [9] C. Patra, D. Giri, T. Maitra and B. Kundu, “A Comparative Study on Detecting Phishing URLs Leveraging Pre-trained BERT Variants”, 2024 6th International Conference on Computational Intelligence and Networks (CINE), Bhubaneswar, India, 29-21 December 2024, pp. 15-17, doi: 10.1109/CINE63708.2024.10881521
- [10] F. Raashid, B. Doyle and S. Senevirante, “FeUDA-BERT: Federated URL Domain-Aware BERT for Phishing URL Detection”, 2025 IEEE 50th Conference on Local Computer Networks (LCN), Sydney, Australia, 13-16 October 2025, pp. 5-8, doi: 10.1109/LCN65610.2025.11146376
- [11] W. Chang, F. Du and Y. Wang, “Research on Malicious URL Detection Technology Based on BERT Model”, 2021 IEEE 9th International Conference on Information, Communication and Networks (ICICN), Xi'an, China, 25-28 November 2021, pp. 7-9, doi: 10.1109/ICICN52636.2021.9673860
- [12] J. Lee, P. Lim, B. Hooi and D. M. Divakaran, “Multimodal Large Language Models for Phishing Webpage Detection and Identification”, 2024 APWG Symposium on Electronic Crime Research (eCrime), 24-26 September 2024, Boston, MA, USA, pp. 24-27, doi: 10.1109/eCrime66200.2024.00007
- [13] Bhupathi Vishva Pavani, Desham Mahitha, B Uma Maheswari, “Enhancing Online Safety: Phishing URL Detection Using Machine Learning and Explainable AI”, 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), 24-28 June 2024, Kamand, India, pp. 8-11, doi: 10.1109/ICCCNT61001.2024.10723976
- [14] Md Robiul Islam, Md Mahamodul Islam, Mst. Suraiya Afrin, Anika Antara, Nujhat Tabassum, “PhishGuard: A Convolutional Neural Network-Based Model for Detecting Phishing URLs with Explainability Analysis”, 2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIoT), 03-04 May 2024, Vellore, India, pp. 67-70, doi: 10.1109/AIoT58432.2024.10574688
- [15] Maolin Wang; Xiangyu Zhao; Ruocheng Guo; Junhui Wang, “MetaLoRA: Tensor-Enhanced Adaptive Low-Rank Fine-Tuning”, 2025 IEEE 41st International Conference on Data Engineering (ICDE), 19-23 May 2025, Hong Kong, Hong Kong, pp. 42-45, doi: 10.1109/ICDE65448.2025.00376