Evaluating the Out-of-Domain Generalization in Source Code Vulnerability Detection

Nin Ho Le Viet
School of Computer Science, Duy Tan University, Danang, Vietnam | Faculty of Computer Science and Engineering, Thuyloi University, Tay Son, Hanoi, Vietnam
holvietnin@dtu.edu.vn

Tuan Nguyen Kim
Phenikaa School of Computing, Phenikaa University, Duong Noi, Hanoi, Vietnam
tuan.nguyenkim@phenikaa-uni.edu.vn (corresponding author)

Chieu Ta Quang
Faculty of Computer Science and Engineering, Thuyloi University, Tay Son, Hanoi, Vietnam
quangchieu.ta@tlu.edu.vn

Received: 21 February 2026 | Revised: 22 March 2026 | Accepted: 1 April 2026

 DOI: https://doi.org/10.48084/etasr.18302

ABSTRACT

Machine learning-based vulnerability detection is increasingly used in practice, yet most studies evaluate models only under in-domain settings. In real deployments, detectors often face code from different projects or data sources, leading to Out-of-Domain (OOD) performance degradation. This study proposes a systematic framework to evaluate OOD generalization in source code vulnerability detection. Experiments were conducted on widely used benchmarks, including Devign, Juliet Test Suite, Big-Vul, and National Vulnerability Database (NVD), with OOD test data stratified into Low, Medium, and High levels. The results indicate that accuracy and F1-score declined by 10-25% under OOD conditions, with the largest decrease reported at the High OOD level. Deep learning models, such as CodeBERT and GrapCodeBERT, are more efficient than traditional methods, including Support Vector Machine (SVM) and XGBoost; however, none of them remains stable across all domains. The proposed framework provides a practical basis for analyzing and improving OOD robustness.

Keywords-machine learning; out-of-domain; cross-domain evaluation; CodeBERT; GraphCodeBERT

I. INTRODUCTION

Machine learning and deep learning are widely employed for source code vulnerability detection, with models, such as CodeBERT and GraphCodeBERT, alongside traditional methods, such as Support Vector Machine (SVM) and XGBoost, achieving strong in-domain performance. However, most studies evaluate models under similar training and test conditions, whereas real deployments often involve domain shifts that lead to noticeable performance degradation. Although Out-of-Domain (OOD) generalization [1] has been extensively studied in natural language processing, it remains underexplored in vulnerability detection, and no standardized framework exists to quantify performance loss across domains. To address this gap, the present study proposes a systematic OOD evaluation pipeline based on domain partitioning and applies it to four widely used datasets: Devign [2], Juliet Test Suite [3], Big-Vul [4], and National Vulnerability Database (NVD) [5].

The experimental results show severe domain-shift effects: CodeBERT loses over 18% F1-score when transferring from Devign to Juliet, and XGBoost declines nearly 25% from Big-Vul to NVD. These findings highlight the need for systematic OOD evaluation and more robust vulnerability detection methods.

Authors in [6] proposed MNCRI using contrastive learning and instance re-weighting for cross-domain alignment, while authors in [7] introduced a cluster-based contrastive approach that improves OOD detection across multiple metrics. Graph-based adaptation methods such as VulGDA [8] enable zero-shot or semi-supervised detection, and pre-trained models such as Vul-BERTa [9] show strong semantic representations but are largely evaluated in-domain. Other works, including MVD [10] and CLaSCoRe [11], extend detection to cross-lingual settings via zero-shot transfer. Despite these advances, a unified framework for systematically comparing traditional and deep models under well-defined cross-domain scenarios is still lacking, motivating the multi-model OOD evaluation framework.

II. PROPOSED OOD EVALUATION PIPELINE

A. Objectives and Overview

The proposed pipeline evaluates model performance under OOD conditions in source code vulnerability detection, focusing on performance degradation and practical generalization rather than in-domain accuracy. It follows a unified workflow that includes domain partitioning, feature representation using TF-IDF [12] and CountVector for traditional models, CodeBERT-based and GraphCodeBERT-based embeddings for deep learning models, cross-domain training and testing, evaluation using F1-score, ROC-AUC, and relative performance drop. By standardizing this process, the pipeline enables systematic and quantitative comparison of heterogeneous models under domain shift, addressing a key limitation in existing vulnerability detection studies.

B. Domain Split Design

To ensure meaningful OOD evaluation, domains are defined by differences in project origin, coding style, or dataset construction, with training and testing data strictly separated to reflect realistic deployment scenarios:

- Scenario 1: Devign to Juliet (cross-project): models are trained on real-world C/C++ code from Devign and evaluated on the synthetic, CWE-structured Juliet Test Suite, assessing transfer from a noisy to a highly structured code.
- Scenario 2: Big-Vul to NVD (cross-style): training uses the GitHub-based Big-Vul dataset, while testing uses the NVD-linked code with cleaner and more standardized styles, evaluating robustness to structural and stylistic variation.
- Scenario 3: Juliet to Devign (reverse OOD): models are trained on synthetic Juliet samples and tested on the real-world Devign code, examining generalization from structured to noisy environments.

In all scenarios, training and test sets are strictly non-overlapping, with no shared source code samples or projects, ensuring a fair assessment of OOD generalization.

C. Feature Representation

Feature representation is crucial to model learning and generalization under OOD conditions. This study considers two representation groups aligned with model types: Frequency-based features for traditional models (SVM, XGBoost) and pre-trained code embeddings for deep models. Traditional representations use TF-IDF and CountVector to encode token importance and frequency, complemented by basic handcrafted statistics to provide limited syntactic cues. Deep learning models rely on pre-trained embeddings, where CodeBERT [13] captures contextual semantics, and GraphCodeBERT [14] further integrates structural information through data-flow and Abstract Syntax Tree (AST) relations, offering greater robustness to domain shift. In addition, Out-of-Vocabulary (OOV) effects and domain divergence are analyzed to assess how different feature representations influence OOD generalization.

D. Models and Train-Test Procedure

This study evaluates both traditional and deep learning models to examine vulnerability detection performance under OOD conditions. SVM and XGBoost are used as baseline methods with simple feature representations, while CodeBERT and GraphCodeBERT leverage pre-trained embeddings, with GraphCodeBERT additionally capturing structural information. Both of these pre-trained models have demonstrated strong performance in detecting source code vulnerabilities [15].

For each OOD scenario, models are trained on one domain and tested on a different, unseen domain, ensuring strict separation between training and evaluation data. All models follow the same pipeline and domain splits, enabling a fair comparison of OOD robustness.

E. Evaluation Metrics and Measurement Criteria

To assess model effectiveness under OOD conditions, this study combines standard classification metrics with an additional indicator designed to capture generalization capability under domain shift.

- Model performance metrics: precision and recall measure detection accuracy and coverage, respectively, while the F1-score, used as the primary metric, balances both aspects. ROC-AUC evaluates class separability, and AUPR [16] is particularly suitable for the imbalanced nature of vulnerability datasets, providing a more reliable assessment of positive-class performance under OOD settings.
- Performance degradation across domains: beyond absolute scores, the analysis focuses on how performance changes when moving from in-domain evaluation (Train A - Test A) to OOD evaluation (Train A - Test B). The resulting performance drop reflects a model's sensitivity to domain shift.
- Proposed metric: to quantify this effect, the OOD-Gap metric is introduced and defined as $OOD - Gap = F1_{in-domain} - F1_{OOD}$, where $F1_{in-domain}$ denotes performance on the same training domain and $F1_{OOD}$ denotes performance on a different target domain. This metric provides a direct and comparable measure of robustness across models and feature representations.
- Overall evaluation framework: within the proposed pipeline, domain splits, diverse feature representations, and both traditional and deep models are evaluated consistently using precision, recall, F1-score, ROC-AUC, AUPR, and OOD-Gap, offering a systematic basis for analyzing generalization under OOD conditions.

III. EXPERIMENTAL DEPLOYMENT

A. Data and Training Models

1) Datasets

This study uses four widely adopted datasets: Devign [2], Juliet Test Suite [3], Big-Vul [4], and NVD [5] to evaluate model generalization under OOD conditions. Although all

datasets are based on C/C++ code, they differ substantially in data source, coding style, and labeling methodology, enabling the construction of clearly separated OOD scenarios: (i) Devign [2] consists of real-world C functions collected from GitHub and labeled at the function level; (ii) The Juliet Test Suite [3], curated by NIST, is a synthetic benchmark containing systematically generated C/C++ code snippets organized by CWE categories; (iii) Big-Vul [4] is derived from CVE-linked GitHub commits and provides paired vulnerable and patched code samples, representing realistic vulnerability patterns; (iv) NVD [5] is an official repository maintained by NIST. In this study, NVD-based code samples are extracted from vulnerability descriptions following established preprocessing pipelines. Despite using the same programming language (C/C++), these datasets differ in structure and origin, enabling cross-domain evaluation and realistic assessment of model effectiveness under domain shift.

2) Training Model Configuration

a) Configuration for Traditional Models

Traditional models (SVM, XGBoost) are trained on features extracted using TF-IDF, CountVectorizer, and hand-crafted statistics. SVM is implemented with a linear kernel (C = 1.0) and class-weight balancing, while XGBoost uses 100 trees with max depth 6, learning rate 0.1, and a binary logistic objective, adjusting class imbalance via scale_pos_weight. These settings provide efficient and reproducible baselines with limited hardware requirements, and with hyperparameters tuned using cross-validation rather than epoch-based training.

b) Configuration for Deep Learning Models

Deep learning models, including CodeBERT and GraphCodeBERT, are fine-tuned from pre-trained embeddings on large-scale code corpora. Training uses a batch size of 16, learning rate 2×10^{-5}, 5 epochs, and the AdamW optimizer with weight decay. Binary cross-entropy with Logits is applied as the loss function, together with a linear warmup-decay scheduler and a maximum sequence length of 256 tokens. Fine-tuning is sufficient due to extensive pre-training on CodeSearchNet, allowing the models to adapt with limited updates while preserving learned semantics. Graph-CodeBERT additionally processes structural information (AST and AFG) to enhance semantic understanding. Both models are initialized from official checkpoints and trained using the HuggingFace and PyTorch frameworks under practical GPU constraints.

Algorithm 1: Non-overlapping data split by domain/project

Input:

- Dataset D containing labeled source code samples;
- Domain/project information M (e.g., Devign, Juliet, Big Vul, NVD);
- Split mode: Cross-dataset or Cross-project.

Output:

- Training set D_{train} and D_{test} with no overlap.

Steps:

1. Define Training and Test Domains

- If Cross-dataset mode: select one dataset as the training domain (e.g., Devign); use the remaining datasets (Juliet, NVD, Big-Vul) for testing;
- If cross-project mode within a single dataset: Group samples by project/repository ID; choose project set A for training and project set B (disjoint from A) for testing.

2. Remove Duplicate Samples

- Normalize code: remove whitespace, tabs, and non-semantic comments to ensure equivalent code segments are treated consistently.
- Compute hash (SHA 1 or MD5): generate a unique hash value for each normalized sample;
- Match and filter: compare hashes between training and test sets; remove duplicates from the test set to eliminate data leakage and ensure accurate generalization evaluation.

3. Generate Final Output

- Return D_{train} and D_{test} ensuring $D_{train} \cap D_{test} = \emptyset$;
- Use these sets for training and testing in OOD.

This procedure ensures strict separation between training and testing data, making it suitable for evaluating OOD generalization across different domain shift scenarios.

B. OOD-Level Testing: Objectives and Design Rationale

1) Objectives and Rationale for OOD Stratification

a) Constructing Test Sets by OOD Levels

In contrast to discrete domain splits, this section stratifies test data into low-to-high OOD levels to enable controlled analysis of domain shift and its impact on model performance. This design supports systematic robustness evaluation and computation of the OOD-Gap metric, allowing OOD severity to be measured on a graded scale rather than treated as a binary condition.

b) OOD Level Design

To examine the impact of increasing domain gaps, test data are stratified into three OOD levels reflecting growing semantic, syntactic, and feature divergence: (i) Low OOD: training and test data come from the same dataset but different sub-projects (e.g., Devign to Devign), with similar code style and minimal OOV, serving as a near-domain baseline; (ii) Medium OOD: test data originate from different sources or styles within the same language (e.g., Big-Vul to Devign), introducing moderate shifts in naming, structure, and length; (iii) High OOD: test data differ substantially in construction and labeling (e.g., Juliet to Devign or Big-Vul to Juliet), with

high OOV rates and embedding drift, representing the most challenging generalization setting.

c) OOD Stratification Criteria

To ensure objective and measurable OOD stratification, two quantitative indicators are used to characterize the domain shift between training and test data: (i) OOV rate, defined as the proportion of test tokens absent from the training vocabulary, reflecting surface-level differences in naming, style, and token usage, particularly relevant for token-based features such as TF-IDF and CountVector; (ii) embedding divergence [17], which captures deeper semantic shifts by comparing embedding distributions from models such as CodeBERT or GraphCodeBERT, measured using cosine distance between mean embeddings [18] or KL divergence between distributions [19]; (iii) together, these metrics provide a quantitative basis for assigning OOD levels (Low-Medium-High) and for interpreting performance degradation, including the observed OOD-Gap, across different models and representations.

d) Significance of the Stratified Design

Stratifying test data by OOD levels provides both practical and analytical value beyond discrete domain evaluation. This design enables: (i) clearer insights into how increasing domain gaps drive performance drops (e.g., F1-score, AUPR), helping explain causes rather than merely reporting degradation; (ii) quantitative comparison of robustness between traditional and deep models across controlled OOD levels, reflecting true generalization ability; (iii) a principled basis for defining and computing the OOD-Gap metric, linking performance loss to domain distance.

Overall, OOD stratification offers a systematic framework for analyzing generalization and guiding subsequent evaluation and model improvement.

2) OOD Level Test Set Construction Method

To construct Low, Medium, and High OOD test sets, the current study uses a semi-automated approach that combines OOV rate with embedding-based semantic distance. This allows quantitative stratification of test samples according to their divergence from the training domain.

- Step 1 (Data preparation): training and test data are preprocessed using consistent representations (TF-IDF or CodeBERT/GraphCodeBERT embeddings). The full test set is denoted as T_{full}.
- Step 2 (Metric computation): for each test sample $x \in T_{full}$, two measures are computed: (i) OOV rate, defined as the proportion of tokens not appearing in the training vocabulary V_{train}; (ii) embedding distance, computed as the cosine distance between the sample embedding and the training embedding centroid μ_{train}.
- Step 3 (Stratification): based on predefined thresholds for OOV rate and embedding distance, samples are assigned to Low, Medium, or High OOD levels, corresponding to increasing domain divergence.

Algorithm 2: OOD Level Test Stratification

Input

- T_full: Full test set; V_train: Training vocabulary;
- E_train: Training embeddings;
- Thresholds: θ_oov_low, θ_oov_high; θ_emb_low, θ_emb_high.

Output

- T_low, T_mid, T_high: OOD-stratified test subsets.

Steps

1. Compute embedding centroid: μ_train ← mean(E_train).
2. Initialize subsets: T_low ← ∅; T_mid ← ∅; T_high ← ∅.
3. For each sample x in T_full:
   a. oov_x ← proportion of tokens(x) ∉ V_train
   b. emb_x ← Embedding(x) # (using CodeBERT)
   c. d_x ← cosine_distance(emb_x, μ_train)
   d. If (oov_x ≤ θ_oov_low) and (d_x ≤ θ_emb_low): T_low ← T_low ∪ {x}
      Else: if (oov_x ≥ θ_oov_high) or (d_x ≥ θ_emb_high): T_high ← T_high ∪ {x}
      else: T_mid ← T_mid ∪ {x}
4. Return: T_low, T_mid, T_high.

This procedure captures both lexical and semantic shifts, providing a controlled basis for analyzing model robustness under increasing OOD severity.

IV. EVALUATION OF EXPERIMENTAL RESULTS

The OOD evaluation focuses on model efficiency under domain shift and varying OOD severity. The results are organized by OOD scenario and model type to highlight generalization limits in vulnerability detection.

A. Aggregate Results Across OOD Scenarios

Three representative OOD settings are considered: cross-project (Devign to Juliet), cross-style (Big-Vul to NVD), and reverse OOD (Juliet to Devign). Table I presents precision, recall, F1-score, ROC-AUC, AUPR, and OOD-Gap values for all models, with in-domain performance shown as a baseline.

Overall, deep models (CodeBERT, GraphCodeBERT) are more stable under OOD conditions compared to traditional methods, which exhibit larger OOD-Gaps (often > 0.15). GraphCodeBERT consistently achieves the smallest degradation (≈ 0.09-0.18), while Juliet to Devign emerges as the most challenging scenario, yielding the lowest F1-scores across models. These results underscore the importance of semantic representations in mitigating OOV effects and style-related domain shifts.

TABLE I. MODEL PERFORMANCE ACROSS OOD SCENARIOS

Model	Scenario	Precision	Recall	F1-score	ROC-AUC	AUPR	OOD-Gap
SV-M	In-domain (Devign)	0.70	0.68	0.69	0.76	0.72	—
	Devign to Juliet	0.61	0.53	0.56	0.68	0.59	0.13
	Big-Vul to NVD	0.59	0.55	0.57	0.66	0.54	0.12
	Juliet to Devign	0.51	0.45	0.48	0.61	0.49	0.21
XG-Boost	In-domain (Big-Vul)	0.77	0.81	0.79	0.86	0.83	—
	Devign to Juliet	0.64	0.56	0.60	0.70	0.62	0.19
	Big-Vul to NVD	0.58	0.50	0.54	0.62	0.50	0.25
	Juliet to Devign	0.53	0.48	0.50	0.63	0.52	0.29
Cod-eBERT	In-domain (Devign)	0.83	0.81	0.79	0.88	0.85	—
	Devign to Juliet	0.68	0.62	0.65	0.75	0.70	0.14
	Big-Vul to NVD	0.70	0.66	0.68	0.78	0.72	0.11
	Juliet to Devign	0.63	0.59	0.61	0.75	0.69	0.18
GraphCodeBERT	In-domain (Devign)	0.85	0.82	0.81	0.89	0.87	—
	Devign to Juliet	0.74	0.69	0.71	0.82	0.76	0.10
	Big-Vul to NVD	0.71	0.68	0.69	0.80	0.74	0.12
	Juliet to Devign	0.65	0.61	0.63	0.76	0.71	0.18

The in-domain row provides a baseline, showing the original performance before applying OOD.

TABLE II. PERFORMANCE BY OOD LEVELS

OOD Level/Model	SVM (F1/AUPR)	XGBoost F1/AUPR	CodeBERT (F1/AUPR)	GraphCodeBERT (F1/AUPR)
Low OOD	0.68/0.71	0.73/0.75	0.78/0.82	0.81/0.85
Medium OOD	0.59/0.63	0.64/0.68	0.70/0.75	0.74/0.78
High OOD	0.48/0.51	0.54/0.58	0.65/0.70	0.69/0.73

B. Performance Across OOD Levels (Low-Medium-High)

Models were trained on Devign and evaluated across three stratified OOD levels: Low OOD (Devign, different sub-projects), Medium OOD (Juliet, same language with structural differences), and High OOD (Juliet, larger structural and labeling differences). Performance is measured using F1-score and AUPR, as summarized in Table II. Overall, performance declines as OOD severity increases, with traditional models showing the sharpest drops. Deep models are more robust, and GraphCodeBERT exhibits the smallest degradation from Low to High OOD. AUPR remains relatively stable for deep models, highlighting the benefit of semantic-rich representations under domain shift.

C. Analysis by OOV Rate and Embedding Divergence

To explain performance degradation under OOD conditions, the study examines two indicators: OOV rate and embedding divergence, which reflect surface-level mismatch and semantic drift between the training set (Devign) and OOD test tiers. As shown in Table III, both OOV rate and embedding divergence increase from Low to High OOD, corresponding to larger lexical and semantic shifts. These trends align with sharper F1-score drops, particularly for token-based methods, while deep models partially mitigate the effect through semantic embeddings. Overall, these factors account for much of the observed OOD performance loss.

D. Role of Feature Representation in Generalization

As presented in Table IV, feature representation plays a significant role in model generalization under OOD conditions. Traditional frequency-based representations (TF-IDF, CountVector) are highly sensitive to OOV tokens and syntactic variation, whereas semantic embeddings learned by CodeBERT and GraphCodeBERT provide greater robustness.

TABLE III. OOV AND EMBEDDING DRIFT ACROSS OOD LEVELS

OOD level	OOV rate (%)	Embedding divergence (cosine drift)
Low	7.2	0.09
Medium	15.5	0.18
High	25.3	0.27

TABLE IV. FEATURE REPRESENTATION METHODS

Method	Models used	Key characteristics
TF-IDF	SVM, XGBoost	Frequency-based; highly sensitive to OOV
CountVector	SVM, XGBoost	Uses raw token frequency; sensitive to syntax variations
CodeBERT	CodeBERT	Semantic representation; moderate OOV resilience
GraphCodeBERT	Graph-CodeBERT	Combines semantics and structure; best OOD robustness

To quantify OOD robustness, models were trained on Devign and evaluated across three OOD tiers (Low, Medium, High). Table V reports the F1-scores and the corresponding OOD-Gap (Low to High). Overall, semantic embeddings, especially Graph-CodeBERT, significantly reduce OOD-Gap compared to handcrafted features. Embedding visualizations further show tighter cross-domain clustering for GraphCodeBERT, indicating lower drift and more stable classification. Table V demonstrates that deep learning models maintain a higher F1-score as OOD increases, while traditional methods experience steep drops ($\approx$ 0.20), reinforcing the superior robustness of semantic embeddings.

E. Observations on Experimental Results

Two key observations emerge from the OOD evaluation. First, Graph-CodeBERT is the most stable model, achieving the highest F1-score and the lowest OOD-Gap across OOD

levels; CodeBERT performs well at Low and Medium OOD but degrades at High OOD, whereas traditional models (SVM, XGBoost with TF-IDF or CountVector) are highly sensitive to OOV and surface variations. Second, the Juliet to Devign scenario is the most challenging, as large differences in coding style, structure, and vocabulary lead to low F1-scores and high embedding divergence.

TABLE V. OOD-GAP COMPARISON ACROSS FEATURES

Representation	Low OOD (F1-score)	Medium OOD (F1-score)	High OOD (F1-score)	OOD -Gap
TF-IDF	0.68	0.59	0.48	0.20
CountVector	0.69	0.60	0.50	0.19
CodeBERT	0.79	0.72	0.62	0.17
GraphCodeBERT	0.81	0.75	0.69	0.12

V. DISCUSSION

A. Role of Feature Representations

Feature representation is crucial for OOD generalization. Frequency-based features (TF-IDF, CountVector) are sensitive to OOV and syntactic shifts, often yielding large OOD-Gaps (> 0.18), while deep semantic embeddings provide more stable cross-domain performance. As depicted in Table VI, Graph-CodeBERT achieves the lowest OOD-Gap (~ 0.12), outperforming TF-IDF (0.20) and CountVector (0.19), highlighting the importance of semantic-structural representations for robust vulnerability detection.

TABLE VI. OOD INDICATORS ACROSS SCENARIOS

OOD scenarios	OOV rate (%)	Embedding divergence (cosine)	OOD-Gap (F1-drop)
Devign to Juliet	8	0.25	0.12
Big-Vul to NVD	15	0.38	0.20
Juliet to Devign	12	0.33	0.18
Big-Vul to Juliet	20	0.45	0.25

B. OOV, Embedding Divergence, and OOD Performance

Performance degradation under OOD conditions is largely driven by distributional shifts between training and test data, captured by the OOV rate and embedding divergence. These indicators reflect token-level mismatch and semantic drift that hinder generalization. As displayed in Table VI, higher OOV rates and larger embedding divergence consistently correspond to larger OOD-Gaps, especially for traditional representations. Embedding-based models are more robust when divergence is lower, while scenarios with both high OOV and high divergence (e.g., Big-Vul to Juliet) exhibit the most severe performance degradation.

C. Impact of OOD Levels on Model Performance

Stratifying test data by OOD levels reveals robustness under increasing domain shift. As illustrated in Table VI, traditional models degrade sharply at higher OOD, with F1-score declining up to 0.20 due to OOV and syntactic variation. Deep models are more resilient; GraphCodeBERT maintains a stable OOD-Gap of ≈ 0.12 by leveraging semantic and structural information. Nonetheless, even CodeBERT shows noticeable degradation at High OOD, indicating that severe domain shift remains challenging.

VI. CONCLUSION

This study addresses Out-of-Domain (OOD) generalization in source code vulnerability detection, a setting often overlooked, as most studies focus on in-domain performance. To bridge this gap, the current study proposes a unified evaluation pipeline that systematically assesses model robustness under domain shift through domain partitioning and cross-domain testing. The framework is evaluated on four widely used datasets (Devign, Juliet, Big-Vul, and National Vulnerability Database (NVD)), employing both traditional machine learning and deep learning models. The results show that all models suffer noticeable performance degradation under OOD conditions, with declines of approximately 10-25%, especially in high OOD scenarios. Deep models, particularly GraphCodeBERT, demonstrate better robustness than traditional approaches.

Compared to prior work on specific models or limited settings, this study offers a systematic, reproducible evaluation across datasets and models, providing practical insights into real-world behavior and the need for robustness-aware evaluation. Future work will explore domain adaptation and OOD-aware training strategies to further improve generalization.

DECLARATION OF COMPETING INTERESTS

The authors declare no competing financial or intellectual property interests that could have influenced the work reported in this paper.

ACKNOWLEDGMENT

This work was supported by Phenikaa University. The authors would also like to thank their colleagues for their valuable comments and the supporting institutions for providing the necessary infrastructure and technical support.

DATA AVAILABILITY

This study uses four datasets: Devign [2], Juliet Test Suite [3], Big-Vul [4], and National Vulnerability Database (NVD) [5]. Additional details on data preprocessing are provided in the manuscript.

REFERENCES

[1] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain Generalization: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–20, 2022, https://doi.org/10.1109/TPAMI.2022.3195549.

[2] Y. Zhou, S. Liu, J. Siow, and Y. Liu, "Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Vancouver, BC, Canada, Dec. 2019, pp. 10197–10207.

[3] T. Boland and P. E. Black, "Juliet 1.1 C/C++ and Java Test Suite," *Computer*, vol. 45, no. 10, pp. 88–90, Oct. 2012, https://doi.org/10.1109/MC.2012.345.

[4] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries," in *Proceedings of the 17th International Conference on Mining Software Repositories*, Online

(Virtual), Jun. 2020, pp. 508–512, https://doi.org/10.1145/3379597.3387501.

[5] "National Vulnerability Database (NVD)," *National Institute of Standards and Technology, U.S. Dept. of Commerce*, Aug. 2024. https://nvd.nist.gov/.

[6] Q. Du, S. Zhou, X. Kuang, G. Zhao, and J. Zhai, "Joint Geometrical and Statistical Domain Adaptation for Cross-domain Code Vulnerability Detection," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore, 2023, pp. 12791–12800, https://doi.org/10.18653/v1/2023.emnlp-main.788.

[7] V. Nguyen, X. Yuan, T. Wu, S. Nepal, M. Grobler, and C. Rudolph, "Deep Learning-Based Out-of-Distribution Source Code Data Identification: How Far Have We Gone?" arXiv, 2024, https://doi.org/10.48550/ARXIV.2404.05964.

[8] X. Li, Y. Xin, H. Zhu, Y. Yang, and Y. Chen, "Cross-Domain Vulnerability Detection Using Graph Embedding and Domain Adaptation," *Computers & Security*, vol. 125, Feb. 2023, Art. no. 103017, https://doi.org/10.1016/j.cose.2022.103017.

[9] H. Hanif and S. Maffeis, "VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection," in *2022 International Joint Conference on Neural Networks*, Padua, Italy, Jul. 2022, pp. 1–8, https://doi.org/10.1109/IJCNN55064.2022.9892280.

[10] B. Zhang, T. H. M. Le, and M. A. Babar, "MVD: A Multi-Lingual Software Vulnerability Detection Framework." Jan. 2025, https://doi.org/10.32388/4AHQY3.

[11] S. Zaharia, T. Rebedea, and S. Trausan-Matu, "Detection of Software Security Weaknesses Using Cross-Language Source Code Representation (CLaSCoRe)," *Applied Sciences*, vol. 13, no. 13, Jul. 2023, Art. no. 7871, https://doi.org/10.3390/app13137871.

[12] G. Salton and C. Buckley, "Term-Weighting Approaches in Automatic Text Retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, Jan. 1988, https://doi.org/10.1016/0306-4573(88)90021-0.

[13] Z. Feng *et al.*, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages." arXiv, Sep. 18, 2020, https://doi.org/10.48550/arXiv.2002.08155.

[14] D. Guo *et al.*, "GraphCodeBERT: Pre-training Code Representations with Data Flow," in *Proceedings of the International Conference on Learning Representations*, 2020, pp. 1–18, https://doi.org/10.48550/ARXIV.2009.08366.

[15] B. B. Ammar and A. M. Alharbi, "SQL Injection Detection Using Fine-Tuned CodeBERT," *Engineering, Technology & Applied Science Research*, vol. 15, no. 5, pp. 27852–27857, Oct. 2025, https://doi.org/10.48084/etasr.13340.

[16] S. Riyanto, I. S. Sitanggang, T. Djatna, and T. D. Atikah, "Comparative Analysis Using Various Performance Metrics in Imbalanced Data for Multi-class Text Classification," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 6, 2023, https://doi.org/10.14569/IJACSA.2023.01406116.

[17] A. Ramesh Kashyap, D. Hazarika, M.-Y. Kan, and R. Zimmermann, "Domain Divergences: A Survey and Empirical Analysis," in Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Online (Virtual), 2021, pp. 1830–1849, https://doi.org/10.18653/v1/2021.naacl-main.147.

[18] A. Atghaei and M. Rahmati, "Domain Generalization via Geometric Adaptation Over Augmented Data," *Knowledge-Based Systems*, vol. 309, Jan. 2025, Art. no. 112765, https://doi.org/10.1016/j.knosys.2024.112765.

[19] A. Bulinski and D. Dimitrov, "Statistical Estimation of the Kullback–Leibler Divergence," *Mathematics*, vol. 9, no. 5, Mar. 2021, Art. no. 544, https://doi.org/10.3390/math9050544.