International Journal of AI, BigData, Computational and Management Studies
Noble Scholar Research Group | Volume 5, Issue 4, 182-188, 2024
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I4P120

Metric-Driven Automatic Software Vulnerability Detection Using Feature Extraction Pipelines and Interpretable ML

Ananya Nair[1], Rohit Menon[2], Pranav Kulkarni[3]
[1,3]Artificial Intelligence Department, Amrita Vishwa Vidyapeetham, Coimbatore, Tamil Nadu, India.
[2]Information Technology, Amrita Vishwa Vidyapeetham, Coimbatore, Tamil Nadu, India.

Abstract: *Software vulnerability detection has progressed from signature-driven static analysis toward learning-based detection over code, build, and operational signals. However, many machine-learning (ML) vulnerability detectors remain difficult to operationalize in real enterprise pipelines because they (i) rely on brittle or opaque feature representations, (ii) provide limited traceability from predictions to security-relevant metrics and engineering controls, and (iii) under-support governance requirements such as auditable decision pathways, risk thresholds, and continuous integration and delivery (CI/CD) gating. This manuscript proposes a metric-driven framework for automatic vulnerability detection that explicitly couples (a) a modular feature-extraction pipeline spanning program semantics and software delivery telemetry, (b) a unified risk metric schema grounded in vulnerability taxonomies and operational signals, and (c) interpretable ML for actionable explanations at multiple levels of granularity. The core idea is to treat vulnerability detection as a metric-aware learning problem: the detector outputs a probability of vulnerability and an aligned, human-auditable rationale mapped to metrics such as weakness category, severity tier, change-risk indicators, and runtime observability signals. The framework integrates transformer-based code representations and graph-based program semantics with explanation methods to produce transparent triage artifacts suitable for DevSecOps and compliance-constrained domains. A deployment-oriented methodology is presented, including feature engineering, model training, explanation generation, CI/CD integration, and evaluation protocols that emphasize decision utility in addition to predictive accuracy.*

Keywords: *Software Security, Vulnerability Detection, Interpretable Machine Learning, Feature Extraction Pipelines, Devsecops, Code Representation Learning, Program Graphs, Risk Metrics, CI/CD Governance.*

1. Introduction

The growth of publicly disclosed software vulnerabilities and the expansion of heterogeneous software stacks cloud services, micro services, container platforms, and enterprise integration layer shave intensified the need for scalable and actionable vulnerability detection. The National Vulnerability Database (NVD) provides structured vulnerability records and impact metadata that support standardized reporting and downstream tooling [1]. Complementary to vulnerability catalogs, the Common Weakness Enumeration (CWE) characterizes recurring classes of implementation errors and design flaws, enabling consistent labeling and remediation planning [2]. In practice, vulnerability detection must translate these structured security concepts into engineering actions such as build gating, code review prioritization, and patch planning.

Enterprise adoption of automated security analytics depends on the same operational qualities demanded of reliability engineering. Comparative evaluations of automated testing frameworks in Java enterprise systems emphasize repeatability, measurable outcomes, and workflow integration as prerequisites for sustained quality improvements [3]. Meanwhile, organizations continue to modernize legacy stacks by migrating from virtualized environments to container orchestration and cloud platforms. Architecture-led modernization introduces new dependency chains, configuration surfaces, and policy boundaries that change the security risk landscape [4]. In parallel, governance-centric software lifecycle approaches highlight that decisions about architecture, releases, and risk must be supported by metrics and transparent decision processes [5]. Observability-driven architectures for AI-enhanced root cause analysis (RCA) demonstrate that production telemetry can be systematically organized to explain anomalous behaviors in complex cloud deployments [6]. These trends jointly motivate vulnerability detection that is not isolated to source code analysis, but instead integrates code semantics with delivery and operational metrics.

Regulated domains further strengthen the requirement for transparent, auditable decision pathways. Secure microservices architectures for HIPAA-compliant processing emphasize controlled data flows, isolation, identity, and platform governance across AWS and OpenShift deployments [7]. Privacy-preserving governance patterns such as federated learning show that organizations can collaborate on analytics while limiting the exposure of sensitive data, provided that protocols and operational controls are explicit [8]. At the same time, the expanding role of ML models throughout software development increases both automation capacity and systemic risk, reinforcing the need for responsible and interpretable ML-enabled tooling [9]. In regulated analytics settings, explainable AI (XAI) has been used to produce auditable decision pathways aligned with validation and governance requirements [10]. Cloud-native frameworks that integrate OCR, ML inference, and microservices

orchestration illustrate how end-to-end pipelines can support operational workflows at scalean integration pattern directly relevant to security analytics services [11].

Despite progress in code representation learning and classification, many ML vulnerability detectors remain difficult to trust and operationalize because their predictions are not clearly tied to security metrics and actionable evidence. Model-agnostic explanation techniques were proposed to produce faithful local rationales for black-box predictions [12]. In adjacent software-quality tasks, comparative studies of ML models for defect prediction illustrate that feature design and model choice materially affect performance and operational viability [13]. SHAP provides a principled feature attribution framework grounded in Shapley values that supports consistent local and global interpretation [14]. These interpretability capabilities are essential when vulnerability detection results trigger CI/CD gates, require audit review, or guide remediation budgets.

This manuscript proposes MDFE-IML (Metric-Driven Feature Extraction with Interpretable ML), a framework that couples a modular feature extraction pipeline with a risk metric schema and interpretable modeling. The framework is designed to integrate into cloud-native enterprise pipelines, including performance and caching patterns commonly used in high-throughput operational systems [15], and CI/CD risk detection practices that use ML signals to guide deployment governance [16]. MDFE-IML incorporates widely used learned representations such as Code BERT embed dings [17] and complements them with classical ML baselines that remain competitive and interpretable in fault prediction settings [18]. Graph-based program semantics learning, exemplified by Devign, provides an additional branch for capturing richer vulnerability-relevant signals [19].

The remainder of the paper is organized as follows. Section II reviews background and related work. Section III defines the system model and problem formulation. Sections IV and V present the metric schema, feature extraction pipeline, and interpretable modeling layer. Section VI describes deployment architecture and governance integration. Section VII proposes an evaluation protocol and an illustrative metric-driven triage workflow. Section VIII discusses limitations and threats to validity, and Section IX concludes with future research directions.

2. Background and Related Work

Section II provides context for the three foundational pillars of MDFE-IML: (i) security knowledge bases and taxonomies, (ii) enterprise software modernization and governance, and (iii) interpretable ML with lifecycle integration.

2.1. Security Knowledge and Taxonomies

Structured vulnerability repositories enable consistent reporting and support the construction of labeled datasets for supervised learning [1]. However, vulnerability records are heterogeneous in quality and frequently lag behind code changes, emphasizing the value of weakness taxonomies that capture recurring error classes and can be mapped to code patterns [2]. MDFE-IML treats these structures not as passive labels, but as active elements of the model output: the detector emits a risk score accompanied by a weakness-aligned explanation.

2.2. Enterprise Reliability, Modernization, and Lifecycle Governance

Reliability engineering highlights the importance of repeatable test automation and integration with development workflows, particularly in Java enterprise systems where regression risk can be substantial [3]. Modernization frameworks for enterprise migration to OpenShift and AWS illustrate how platform transitions add configuration and dependency complexity, altering both threat surfaces and observability requirements [4]. Decision intelligence approaches to AI-driven agile governance emphasize measurable decisions and architecture-centered project management, motivating risk scoring that is compatible with lifecycle governance artifacts [5]. In cloud deployments, AI-enhanced observability and automated RCA provide a blueprint for converting telemetry into actionable explanations and operational insights [6].

2.3. Secure Microservices and Compliance-Aware Architectures

Secure microservices architectures for HIPAA-compliant processing underline the necessity of audit logs, identity boundaries, and platform controls across cloud and container environments [7]. In such settings, vulnerability detection must be compatible with compliance review, meaning that explanations should be traceable and stable across releases.

2.4. Privacy-Preserving Learning and Distributed Governance

Federated learning studies in regulated financial contexts highlight how analytics can be performed across institutions with operational governance controls and privacy-preserving protocols [8]. These patterns generalize to vulnerability detection when code or telemetry cannot be centralized, for example across subsidiaries or partner organizations.

2.5. Interpretable ML and Auditable Decision Pathways

Regulatory-grade XAI emphasizes auditable decision pathways and empirical validation, treating interpretability as an explicit system requirement [10]. Model-agnostic local explanation methods can provide actionable rationales even for

complex models by approximating the decision boundary locally [12]. SHAP extends this idea with a consistent attribution framework that supports both instance-level explanations and global monitoring of feature influence [14].

2.6. ML for Software Analytics and Representation Learning

The expanding role of ML models in the software lifecycle supports predictive and analytic decision tools but also introduces new governance demands [9]. Comparative studies of ML for defect prediction show that different feature sets and models vary in effectiveness, emphasizing the need for structured pipelines and careful evaluation [13]. Operational performance considerations are also central: cloud-native systems frequently use caching and predictive analytics to meet throughput needs, suggesting similar patterns for security analytics services [15]. CI/CD risk detection demonstrates the feasibility of integrating ML signals into deployment governance, but such integration benefits from transparent explanations and stable metrics [16]. CodeBERT provides a widely adopted representation learning baseline for combining programming language and natural language contexts [17]. Classical model comparisons for fault prediction further justify retaining interpretable baselines as strong reference points [18]. Finally, graph-based vulnerability identification illustrates how program semantics graphs can encode deep structural signals, providing complementary evidence to token-based representations [19].

3. System Model and Problem Formulation

This section formalizes metric-driven vulnerability detection as a supervised learning and decisioning problem integrated into DevSecOps.

3.1. Units of Analysis

Let U denote the set of analysis units. Depending on engineering workflow, U may represent functions, classes, files, or service endpoints. For each unit u in U, the pipeline extracts a set of feature groups F(u) partitioned by provenance: code features, repository/process features, and operational features.

3.2. Labels and Weakness Alignment

Each unit u is associated with a binary label y(u) indicating vulnerability presence, and optionally with a categorical weakness label c(u) derived from a weakness taxonomy mapping. In practice, c(u) may be multi-label when a single unit exhibits multiple weakness patterns. The framework supports weak supervision by allowing uncertain labels to be marked as review-needed rather than forcing a binary assignment.

3.3. Risk Metric Vector

Define a metric vector M(u) containing fields that support security decisioning, including weakness family indicators, severity tier proxies, change-risk signals, and operational anomaly scores. The key design goal is that M(u) is human-interpretable and policy-addressable; a CI/CD policy can specify rules over components of M(u) rather than only over opaque model scores.

3.4. Learning Objective

The model learns a function f that maps extracted features to a vulnerability probability $p_v(u) = f(F(u))$. A decision function g then combines $p_v(u)$ with metric fields to produce a deployable action a(u) in {allow, warn, require-review, block}. This separation supports governance: the model estimates risk, while the policy function encodes organizational appetite and compliance constraints.

3.5. Threat Model and Assumptions

MDFE-IML targets implementation vulnerabilities that manifest as insecure coding patterns, unsafe API usage, or missing validation controls, as well as configuration-related risk captured by pipeline metrics. The approach assumes access to source code and build telemetry, and optional access to aggregated operational metrics. The framework does not assume access to exploit proofs or attacker traces; instead, it uses observability signals as supportive evidence for risk calibration and prioritization.

4. Metric Schema and Feature Extraction Pipeline

4.1. Risk Metric Schema

The risk metric schema organizes signals into three layers: knowledge metrics, engineering process metrics, and operational metrics. Knowledge metrics capture weakness family and severity tier proxies. Engineering process metrics capture change and quality signals such as churn, dependency delta, and test adequacy. Operational metrics capture anomaly indicators and configuration drift.

4.2. Feature Extraction Pipeline Overview

MDFE-IML implements feature extraction as a modular pipeline with explicit provenance tracking. Each pipeline stage emits a typed feature group and attaches metadata (commit hash, repository, service, timestamp, pipeline run identifier). This enables reproducible scoring and auditing.

4.3. Code-Derived Feature Groups

- Lexical and Pattern Features: token n-grams, identifier morphology, and security-sensitive API patterns.
- Structural Features: AST statistics, control-flow complexity, and data-flow indicators.
- Representation Learning Features: contextual embeddings derived from transformer encoders and graph encoders.

4.4. Repository and Delivery Feature Groups

Repository and delivery features capture the process context of code changes, including churn and ownership metrics, dependency updates, build and test outcomes, security scan deltas, and release cadence signals.

4.5. Operational Feature Groups

Operational features are derived from service telemetry aggregates, including error-rate deviations, latency anomalies, unusual trace topologies, and configuration drift flags. These features are treated as optional inputs because they may be unavailable in some environments. When present, they enable a closed-loop risk calibration mechanism.

4.6. Data Quality and Normalization

Heterogeneous features require normalization and missingness handling. MDFE-IML uses consistent scaling per feature group and explicit missingness indicators so that the model can learn when absence of telemetry should reduce confidence in operational signals.

5. Interpretable Modeling and Explanation Generation

5.1. Model Families

MDFE-IML supports intrinsically interpretable models for governance-critical gating and stronger learners with post-hoc explanations for higher recall. Interpretable baselines include sparse logistic regression and calibrated tree ensembles constrained to metric-aligned features. Stronger learners include models that consume embeddings from transformer and graph modules.

5.2. Explanation Artifacts

For each unit u, the system emits an explanation bundle E(u) containing: (i) top contributing feature groups, (ii) evidence pointers to code regions or process events, and (iii) a mapping to metric fields such as weakness family cues and severity tier justifications.

5.3. Local Explanation Methods

Model-agnostic local explanation methods approximate model behavior in the neighborhood of a specific instance, enabling actionable rationales for individual flags. SHAP values provide consistent feature attributions suitable for both local explanations and global monitoring.

5.4. Metric-Mapped Rationales

A central novelty of MDFE-IML is the explicit mapping from explanatory features to risk metric fields. For example, an attribution to unsafe deserialization patterns is mapped to the corresponding weakness family indicator, while high churn and low test adequacy are mapped to process risk fields. This structure enables explanation stability and improves cross-team communication.

5.5. Calibration and Thresholding

Calibration aligns predicted probabilities with observed outcomes. MDFE-IML uses calibration curves on validation sets and supports per-service thresholds when risk profiles differ. Decision thresholds are documented as governance artifacts and reviewed periodically.

5.6. Human-in-the-Loop Review

Interpretability supports human-in-the-loop workflows. Reviewers can accept, reject, or relabel findings, and the pipeline records decisions for subsequent model improvement. To avoid feedback loops that bias labels, governance controls restrict how review outcomes are incorporated into training.

6. Devsecops Integration and Cloud-Native Deployment Architecture

6.1. CI/CD Integration Points

The pipeline integrates at pre-merge analysis (pull request checks), post-merge nightly scans, and release candidate gating. At each integration point, the policy function maps risk scores and metric fields to actions. For example, high-severity risk tiers may block release, while moderate tiers require security review.

6.2. Microservices Deployment Pattern

The vulnerability detection capability is deployed as a set of microservices: feature ingestion, feature store, model scoring, explanation service, and a policy adapter. Services expose versioned APIs, and all outputs are tagged with model and feature versions to support reproducibility.

6.3. Performance and Caching

To meet throughput requirements, the system caches intermediate artifacts such as embeddings and graph features keyed by commit hash. Explanation bundles are cached to support repeated review and audit. Cache invalidation policies are aligned with repository change events to avoid stale security decisions.

6.4. Observability and Feedback

The system emits operational metrics and traces for its own processing and can consume aggregated service telemetry to support risk calibration. This supports a closed-loop posture where anomalies and incident signals inform triage prioritization.

6.5. Privacy-Preserving and Federated Modes

In environments where code or telemetry cannot be centralized, MDFE-IML supports federated learning. Feature extraction and local scoring remain within each domain, and only model updates are shared under governance controls. Audit logs record participation and model versioning to support accountability.

7. Evaluation Protocol and Illustrative Workflow

7.1. Datasets and Splits

Evaluation should include projects with diverse languages and architectures where labels are derived from historical vulnerability fixes, scanner outputs mapped to weakness families, and disclosure-aligned records when available. Splits should be time-aware to reduce leakage: training on earlier commits and evaluating on later commits approximates deployment behavior.

7.2. Baselines

Baselines include (i) classical metric-based models, (ii) transformer-embedding classifiers, (iii) graph-based semantics classifiers, and (iv) heuristic gates based on CI/CD signals alone. Comparisons must control for feature availability to ensure fairness.

7.3. Predictive Metrics

Primary predictive metrics include AUPRC under class imbalance, AUROC, precision at top-k, and recall at fixed false-positive budgets. Because vulnerability review is costly, precision at actionable k (e.g., per release) is a key operational metric.

7.4. Decision Utility Metrics

Decision utility metrics evaluate how well the system supports engineering actions: tier-weighted recall, average review time per true positive, and gate correctness relative to downstream security outcomes. Explanation quality is assessed via fidelity and stability checks.

7.5. Ablation Studies

Ablation isolates the contribution of feature groups: code-only, code+process, code+process+operations. Additional ablations compare intrinsic interpretability vs post-hoc explanation workflows.

7.6. Illustrative Metric-Driven Triage Example

To demonstrate how metric alignment improves actionability, consider a release candidate that modifies eight functions across two services. The pipeline scores each function and emits metric-mapped explanations. Three functions are flagged as high risk because they combine (i) a code semantic indicator consistent with an input validation weakness family, (ii) elevated churn and dependency delta, and (iii) a correlated operational anomaly trend in the service endpoint. The policy engine blocks release for these changes while allowing the remaining five functions with low risk scores. For blocked items, the explanation bundle provides: the top feature drivers, code region pointers, and a weakness-aligned rationale. Reviewers can validate and remediate quickly because the output is already structured as an audit-ready artifact rather than an opaque probability.

7.7. Reproducibility and Audit Artifacts

All evaluation runs must store feature provenance, model versions, thresholds, and explanation snapshots. This enables post-hoc analysis of why a release was gated and supports compliance review when required.

8. Threats to Validity

8.1. Construct Validity

Severity proxies and weakness mappings may not perfectly represent real-world exploitability. Metric vectors should be calibrated and periodically reviewed, particularly after architectural modernization or platform changes.

8.2. Internal Validity

Label noise is a primary internal threat. Vulnerability labels derived from commit messages or tool outputs may contain false positives and false negatives. Time-aware evaluation and uncertainty handling reduce bias.

8.3. External Validity

Generalization may vary across languages, repositories, and organizational contexts. Operational feature availability varies widely, and privacy constraints may limit telemetry access. Federated modes address some constraints but may change convergence behavior.

8.4. Conclusion Validity

Comparisons must control for feature extraction, hyperparameter tuning, and evaluation splits. Interpretation metrics should be evaluated systematically rather than anecdotally.

9. Conclusion and Future Work

This manuscript presented MDFE-IML, a metric-driven framework for automatic software vulnerability detection that unifies feature extraction pipelines with interpretable ML and governance-aligned decisioning. The approach emphasizes metric alignment, modularity, and audit-ready explanation bundles to improve adoption in DevSecOps and compliance-constrained environments. Future work includes large-scale benchmarking across diverse codebases, richer weakness taxonomy mappings, robust probability calibration under evolving threat models, and federated deployments with formal governance protocols.

References:

1. Harold Booth, Doug Rike, and Gregory A. Witte, “The National Vulnerability Database (NVD): Overview,” ITL Bulletin, National Institute of Standards and Technology, Gaithersburg, MD, USA, Dec. 18, 2013. https://www.nist.gov/publications/national-vulnerability-database-nvd-overview?pub_id=915172.
2. Robert A. Martin, and Sean Barnum, “Common weakness enumeration (CWE) status update,” *ACM SIGAda Ada Letters*, vol. 28, no. 1, pp. 88–91, Apr. 2008. https://dl.acm.org/doi/abs/10.1145/1387830.1387835.
3. Srikanth Reddy Gudi, “Enhancing Reliability in Java Enterprise Systems Through Comparative Analysis of Automated Testing Frameworks,” *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. 4, no. 2, pp. 151–160, 2023. https://www.ijetcsit.org/index.php/ijetcsit/article/view/476.
4. Siva Kantha Rao Vanama, “Architecture Led Cloud Modernization: A Framework for Enterprise Migration from VMware to OpenShift and AWS,” International Journal of Emerging Research in Engineering and Technology, vol. 5, no. 1, pp. 117–125, Mar. 2024. https://ijeret.org/index.php/ijeret/article/view/393
5. Sai Krishna Gunda, et al., “Decision Intelligence Methodology for AI-Driven Agile Software Lifecycle Governance and Architecture-Centered Project Management,” *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, vol. 4, no. 1, pp. 102–108, Mar. 2023. https://ijaidsml.org/index.php/ijaidsml/article/view/301.
6. Indrasena Manga, Sai Dheeraj Sivva, and Vamsi Krishna Manga, “The Adaptive Intelligence in Cloud Systems: A Unified Architecture for AI Enhanced Observability and Automated Root Cause Analysis”, *IJAIDSML*, vol. 5, no. 1, pp. 160–166, Mar. 2024. https://ijaidsml.org/index.php/ijaidsml/article/view/366
7. Srikanth Reddy Gudi, “Design and Evaluation of Secure Microservices Architecture for HIPAA-Compliant Prescription Processing on AWS and OpenShift,” *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, vol. 5, no. 2,0 pp. 144–149, 2024. https://ijaidsml.org/index.php/ijaidsml/article/view/337
8. Rakesh Reddy Thalakanti, Sai Santhosh Goud Bandari, and Sai Dheeraj Sivva, “Federated Learning for Privacy Preserving Fraud Detection across Financial Institutions: Architecture Protocols and Operational Governance,” *International Journal of Emerging Research in Engineering and Technology*, vol. 5, no. 2, pp. 108–114, 2024. https://ijeret.org/index.php/ijeret/article/view/394
9. Sai Krishna Gunda, “The Future of Software Development and the Expanding Role of ML Models,” *International Journal of Emerging Research in Engineering and Technology*, vol. 4, no. 2, pp. 126–129, 2023. https://ijeret.org/index.php/ijeret/article/view/347.
10. Sai Santhosh Goud Bandari, Sai Dheeraj Sivva, and Rakesh Reddy Thalakanti, “Regulatory Grade Fraud Detection using Explainable Artificial Intelligence with Auditable Decision Pathways and Empirical Validation on Banking Data,” *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, vol. 5, no. 3, pp. 139–147, 2024. https://ijaidsml.org/index.php/ijaidsml/article/view/367

11. Srikanth Reddy Gudi, "AI-Driven Fax-to-Digital Prescription Automation: A Cloud-Native Framework Using OCR, Machine Learning, and Microservices for Pharmacy Operations," *International Journal of Emerging Research in Engineering and Technology*, vol. 5, no. 1, pp. 111–116, 2024. https://ijeret.org/index.php/ijeret/article/view/358
12. Macro Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," *in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD)*, San Francisco, CA, USA, pp. 1135–1144, Aug. 2016. https://dl.acm.org/doi/abs/10.1145/2939672.2939778.
13. Sai Krishna Gunda, "Comparative Analysis of Machine Learning Models for Software Defect Prediction," *in Proc. 2024 Int. Conf. Power, Energy, Control and Transmission Systems (ICPECTS)*, Chennai, India, pp. 1–6, 2024. https://ieeexplore.ieee.org/abstract/document/10780167
14. Scott M. Lundberg and Su-In Lee, "A Unified Approach to Interpreting Model Predictions," *in Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, pp. 4765–4774, 2017. https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.
15. Srikanth Reddy Gudi, "Leveraging Predictive Analytics and Redis-Backed Caching to Optimize Specialty Medication Fulfillment and Pharmacy Inventory Management," *International Journal of AI, BigData, Computational and Management Studies*, vol. 5, no. 3, pp. 155–160, 2024. https://ijaibdcms.org/index.php/ijaibdcms/article/view/327
16. Rakesh Reddy Thalakanti and Sai Santhosh Goud Bandari, "Intelligent Continuous Integration and Delivery for Banking Systems using Machine Learning Driven Risk Detection with Real World Deployment Evaluation," *International Journal of AI, BigData, Computational and Management Studies*, vol. 5, no. 4, pp. 168–175, 2024. https://ijaibdcms.org/index.php/ijaibdcms/article/view/335
17. Zhangyin Feng, et al., "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," *in Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536–1547, 2020. https://arxiv.org/abs/2002.08155.
18. Sai Krishna Gunda, "Fault Prediction Unveiled: Analyzing the Effectiveness of Random Forest, Logistic Regression, and KNeighbors," *in Proc. 2024 2nd Int. Conf. Self Sustainable Artificial Intelligence Systems (ICSSAS)*, Erode, India, pp. 107–113, 2024. https://ieeexplore.ieee.org/abstract/document/10760620
19. Yaqin Zhou, et al., "Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks," *in Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pp. 10197–10207, 2019. https://proceedings.neurips.cc/paper_files/paper/2019/hash/49265d2447bc3bbfe9e76306ce40a31f-Abstract.html.
20. Sai Dheeraj Sivva, et al., "AI-Driven Decision Intelligence for Agile Software Lifecycle Governance: An Architecture-Centered Framework Integrating Machine Learning Defect Prediction and Automated Testing," *IJETCSIT*, vol. 4, no. 4, pp. 167-72, 2023. https://www.ijetcsit.org/index.php/ijetcsit/article/view/554.