SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

AI-Driven Software Security Engineering In CI/CD Microservice Pipelines: A Leakage-Aware Shift-Left/Shift-Right Evidence Integration Framework

Yasmin Makki Mohialden[1], Israa M. Abdal Ameer Al-Khafaji[2], Tuqa Muneam Fakhir [3], Saba Abdulbaqi Salman[4], Qabas Abdal Zahraa Jabbar[5]

[1,3]Computer Science Department, College of Science, Mustansiriyah University, Baghdad, Iraq
[2,] College of Science, Mustansiriyah University, Baghdad, Iraq
[5]Department of Artificial Intelligence, College of Science, Mustansiriyah University, Baghdad, Iraq
[1] ymmiraq2009@uomustansiriyah.edu.iq
[2]ism@uomustansiriyah.edu.iq
[3]tuqa_m.f@uomustansiriyah.edu.iq
[5] qabas_a@uomustansiriyah.edu.iq

Abstract
DevSecOps requires security evidence to be continuously integrated throughout the software development lifecycle and across operational runtime stages. In this study, we propose a reproducible proof-of-concept framework that integrates Shift-Left pre-deployment security evidence with Shift-Right runtime anomaly detection. The framework employs four unsupervised anomaly detection methods: Isolation Forest (IF), Local Outlier Factor (LOF), One-Class Support Vector Machine (OCSVM), and an Autoencoder (AE). A controlled synthetic dataset comprising 4,600 sessions was generated, including 4,000 benign sessions, 300 sessions representing known attacks, and 300 sessions representing synthetic runtime-only anomalies. Each session contained 18 features, comprising eight Shift-Left pre-deployment security indicators and ten Shift-Right runtime telemetry features. The dataset was stratified into training (70%), validation (15%), and testing (15%) subsets. Only benign training samples were used for feature scaling and unsupervised detector training, while the labeled validation partition was used exclusively to calibrate fusion weights and decision thresholds. The test partition remained untouched until the final evaluation to reduce the risk of data leakage.In the primary run, validation-only selection reduced the four-component fusion to the Autoencoder alone, which achieved an F1-score of 0.9570, an MCC of 0.9510, a PR-AUC of 0.9982, a false-positive rate of 0.0117, a known-attack recall of 1.0000, and a runtime-only recall of 0.9778 on the locked test set. An exploratory primary-run ablation showed that the equal-weight Isolation Forest–Autoencoder configuration achieved a higher test F1-score of 0.9677; however, this configuration was not retrospectively selected, thereby preserving the independence of the locked test evaluation. To assess the robustness of the findings, the complete experimental pipeline, including synthetic data generation, data partitioning, model training, validation-based tuning, and locked-test evaluation, was independently repeated across 30 runs. Logistic Stacking achieved the highest mean F1-score (0.9704 ± 0.0163; 95% empirical percentile interval: 0.9362–0.9944), followed by LOF (0.9619 ± 0.0169), OCSVM (0.9584 ± 0.0154), and the validation-tuned four-component fusion (0.9568 ± 0.0175; 95% empirical percentile interval: 0.9210–0.9862). Paired Wilcoxon signed-rank tests with Holm correction showed that Logistic Stacking significantly outperformed the validation-tuned fusion, whereas no statistically significant differences were observed between the tuned fusion and the individual anomaly detectors.These findings highlight the importance of repeated experimentation, ablation analysis, leakage-aware evaluation, and validation-only calibration for obtaining reliable performance estimates. They further demonstrate that weighted evidence fusion does not consistently outperform strong standalone anomaly detectors and should therefore be selected based on validation evidence rather than the assumption that combining additional evidence sources necessarily improves predictive performance. Nevertheless, integrating Shift-Left and Shift-Right evidence provides an architectural mechanism for connecting security information across different stages of the software development lifecycle. Because all experiments were conducted using controlled synthetic data, the proposed framework should be interpreted as a methodological proof-of-concept rather than a production-ready security solution.

Keywords: DevSecOps; Shift-Left; Shift-Right; evidence fusion; runtime anomaly detection; microservices; Isolation Forest; Local Outlier Factor; autoencoder; logistic stacking,validation-only tuning; synthetic cybersecurity dataset.

1. Introduction

As integrated continuous deployment builds become more common, it is necessary for organizations to secure software at each stage of the software development lifecycle (SDLC). In the past, many companies would treat

software security as a final step for assurance at the end of the SDLC. However, DevSecOps helps solve this problem by integrating security controls into the SDLC along with the automated security analysis and individual responsibility for security (Ashenden & Ollis, 2020; Turner, 2021; Feio et al., 2024). In the contemporary microservice architecture, security evidence is seen at different stages of the SDLC. In the Shift-Left phase, security evidence is derived from a combination of SAST, SCA, IaC analysis, container scans, secret detection, SBOM analysis, DAST, and policy compliance assessment. Meanwhile, in the Shift-Right phase, operational security evidence is derived from a combination of runtime telemetry and host and process behavior along with network communications and service metrics.

Incorporating security tools into CI/CD processes detects vulnerabilities sooner in the SDLC, but threats to software integrity will continue to exist after software has been deployed. These threats include credential and privilege abuse, lateral movements within a network, runtime exploitation, and data exfiltration (Rajapakse et al., 2021; Pecka et al., 2022; Nagasundari et al., 2025). Detection of runtime anomalies is an important aspect of identifying new unexplained behaviors within a system, but there is often little connection to security evidence provided prior to deployment. This means that in order to ensure the success of DevSecOps practices, organizations need to focus on integrating pre-deployment and post-deployment evidence regardless of the assumption that multiple sources of evidence will yield better detection capabilities.

This study offers a novel, evidence-fusion framework—although, methodologically, more of a proof-of-concept than a deployable intrusion detection system. The framework delineates simulated Shift-Left security evidence from Shift-Right runtime telemetry, builds unsupervised anomaly detectors in a purely benign training data context, and executes feature scaling in a benign training data context. Iterative optimization of fusion weights and estimation of decision thresholds are performed on a dedicated validation data set. A completely isolated test data set is preserved to ensure no data leakage, and a robustness assessment consists of 30 repetitions of the overall experimental framework, an ablation study, and a series of statistical significance tests. As the Shift-Right runtime anomalies are generated in a statistically centered context and do not represent confirmed, real-world zero-day attacks, the authors refer to them as runtime-only synthetic anomalies in this context.

This study addresses the following research questions:

RQ1. How effectively can simulated Shift-Left security evidence distinguish signature-detectable attacks from runtime-only synthetic anomalies?

RQ2. Under a benign-only training paradigm, how do Isolation Forest, Local Outlier Factor (LOF), One-Class SVM (OCSVM), and Autoencoder models compare in detecting runtime anomalies?

RQ3. Does weighted security evidence fusion improve detection performance, measured by F1-score, MCC, false-positive rate, known-attack recall, and runtime-only recall, compared with individual anomaly detectors?

RQ4. Which evidence channels and detector combinations contribute most effectively according to an ablation study?

RQ5. Are the observed performance differences consistent across 30 independent experimental repetitions and statistically significant after correcting for multiple comparisons?

The principal contributions of this work are summarized as follows:

- A reproducible DevSecOps evidence-fusion framework that combines eight of the simulated Shift-Left security indicators and ten of the Shift-Right runtime telemetry features set within a singular CI/CD security architecture that integrates the elements of the Shift-Left and Shift-Right security within a DevSecOps context.
- A leakage-aware framework that combines a stratified 70/15/15 data partition, training, and scaling of benign data, optimization of the fusion weights and decision thresholds in a validation set, and a completely isolated set used for evaluation.
- A detailed assessment of Isolation Forest, Local Outlier Factor, One-Class SVM, Autoencoder, and validation-tuned weighted evidence fusion, with a Logistic Stacking as a supervised score-level baseline.
- An ablation study with 14 different configurations to establish the impact of each detector and combinations of detectors based on F1-score, MCC, PR-AUC, balanced accuracy, false-positive rate, known-attack recall, and recall for runtime-only scenarios.
- A thorough stability study including confidence intervals, paired Wilcoxon signed-rank tests, Holm's multiple-comparison corrections, and rank-biserial effect-size studies, conducted on 30 independent experiments.
- Evidence-based analysis showing that weighted evidence fusion is not the optimal solution in all scenarios. Based on the validation data, the model may comply with the validation data and justify the use of a single detector or a different fusion approach.

The structure of this paper is as follows. First, we provide a literature overview in Section 2 for DevSecOps, Shift-Left security, runtime anomaly detection, and software security and machine learning. Section 3 describes the problem and the evidence-fusion framework that we propose. In Section 4, we elaborate our dataset generation, experimental design, and explain feature engineering, model setup, and validation metrics and the leakage-aware validation framework. Section 5 reports the main results of the experiment, including the primary-run, ablation, stability testing, and statistical testing. In Section 6, we interpret the results, address the research questions, and consider their

implications. Section 7 identifies the research gaps and the study constraints. Section 8 provides a summary and research recommendations.

2. Related Work

2.1 DevSecOps and Shift-Left Security

DevSecOps extends conventional DevOps by integrating security controls, automated verification, and security responsibilities throughout the entire software development lifecycle rather than treating security as a separate post-development activity. Recent studies have shown that integrating security tools into CI/CD pipelines improves early vulnerability detection and software quality; however, practical adoption remains constrained by fragmented toolchains, organizational complexity, implementation cost, and the shortage of security expertise (Rajapakse et al., 2021; Cheenepalli et al., 2025; Sinan et al., 2025). Common Shift-Left practices include Static Application Security Testing (SAST), Software Composition Analysis (SCA), Infrastructure-as-Code (IaC) analysis, secret detection, container image scanning, Software Bill of Materials (SBOM) assessment, Dynamic Application Security Testing (DAST), and policy-as-code validation. Although these techniques provide valuable pre-deployment security evidence, they do not directly observe the operational behavior of deployed services and therefore cannot detect attacks that emerge only during runtime.

2.2 Runtime Protection and Microservice Security

Microservice-based systems substantially increase the attack surface through distributed services, APIs, containers, orchestration platforms, software dependencies, and service-to-service communication (Billawa et al., 2022; Alawneh & Abbadi, 2022). Consequently, runtime protection mechanisms such as Runtime Application Self-Protection (RASP), container monitoring, host telemetry, and behavior-based anomaly detection have become essential components of Shift-Right security strategies (Le-Thanh et al., 2023; Verderame et al., 2023; Nugraha & Irsan, 2025). Recent research has further demonstrated the value of post-deployment behavioral monitoring using temporal graph models, container telemetry, and runtime metrics for identifying sophisticated attacks that cannot be detected during static analysis (Yasar et al., 2024; Kwon et al., 2025). Nevertheless, these runtime approaches generally operate independently of pre-deployment security evidence, limiting traceability between detected anomalies and software engineering artifacts.

2.3 AI and Machine Learning in DevSecOps

Artificial intelligence and machine learning are increasingly being incorporated into cloud-native DevSecOps platforms to automate threat detection, security monitoring, and incident response (Yu et al., 2024; Yulianto & Ngo, 2024; Solaimalai, 2025). When labeled attack data are unavailable or incomplete, unsupervised anomaly detection techniques, including Isolation Forest, Local Outlier Factor, One-Class SVM, and Autoencoders, become attractive alternatives. However, comparisons reported in the literature are often affected by differences in datasets, threshold-selection strategies, class imbalance, evaluation protocols, and potential data leakage. Furthermore, studies based solely on synthetic datasets may overestimate detection performance if experimental protocols are not carefully controlled. Consequently, repeated experimentation, validation-only parameter tuning, statistical significance testing, and confidence interval reporting are essential for obtaining reliable and reproducible performance estimates.

2.4 Positioning of This Study

Existing DevSecOps studies typically emphasize either Shift-Left security practices or runtime anomaly detection, while only a limited number of studies investigate the integration of both evidence sources within a unified decision-making framework. Moreover, previous research rarely combines validation-only optimization, leakage-aware experimentation, repeated validation, ablation analysis, and statistical significance testing within a single reproducible framework.

The proposed framework addresses these limitations by integrating simulated Shift-Left security evidence with Shift-Right runtime anomaly scores through an evidence-aware score-fusion architecture. As illustrated in Figure 1, the framework combines pre-deployment evidence, runtime anomaly detectors, score normalization, validation-based fusion, logistic stacking, and statistical evaluation within a unified DevSecOps pipeline. Because the security findings and runtime telemetry are synthetically generated, the proposed framework should be regarded as a methodological proof-of-concept rather than a production-ready operational system

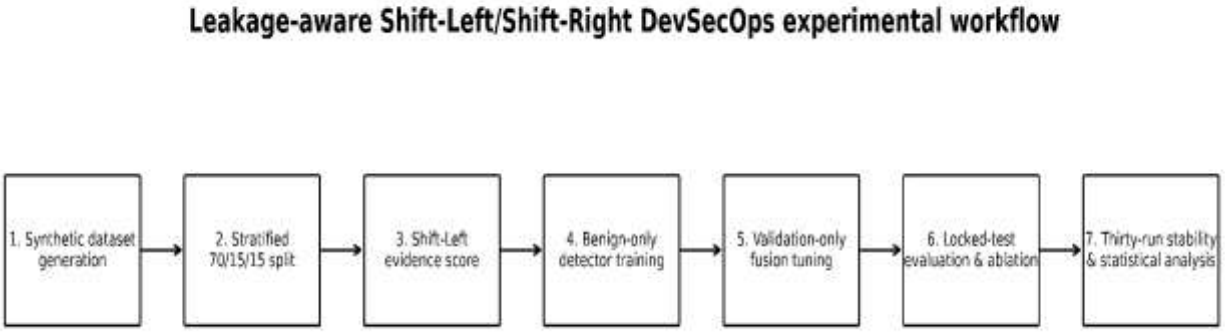


Figure 1: Leakage-aware Shift-Left/Shift-Right DevSecOps Experimental Workflow

A feature-oriented comparison with representative studies is presented in Table 1. Unlike previous work, the proposed framework simultaneously provides leakage-aware experimental design, validation-only tuning, repeated 30-run stability analysis, ablation studies, and statistical comparisons, thereby emphasizing methodological rigor instead of claiming universal superiority over existing approaches.

Table 1. Feature-Oriented Comparison of Representative DevSecOps Studies

Study	Primary Focus	Pre-deployment Evidence	Runtime ML	Evaluation Type
Putra and Kabetta (2022)	Static and dynamic testing in CI/CD	Yes	No	Pipeline implementation
Cankar et al. (2023)	Security tools with ML monitoring	Yes	Yes	Demonstration
Le-Thanh et al. (2023)	Runtime application self-protection	No	Limited	Runtime evaluation
Yasar et al. (2024)	Post-deployment container telemetry	No	Yes	Runtime metrics
Kwon et al. (2025)	Temporal graph-based anomaly detection	No	Yes	Containerized environment
Proposed study	Shift-Left/Shift-Right evidence fusion with statistical validation	Simulated	Yes	Controlled 30-run synthetic evaluation

3. Problem Formulation and proposed Framework

The proposed DevSecOps framework aims to integrate security evidence collected during software development (Shift-Left) with runtime behavioral evidence (Shift-Right) to produce a continuous security risk estimate for each software session. Unlike conventional DevSecOps studies that emphasize either pre-deployment security scanning or runtime anomaly detection independently, the proposed framework combines both evidence sources within a leakage-aware experimental protocol that preserves strict separation between training, validation, and testing stages. As illustrated in Figure 1, the framework consists of seven sequential phases: (1) synthetic dataset generation, (2) stratified dataset partitioning, (3) Shift-Left evidence extraction, (4) benign-only training of runtime anomaly detectors, (5) score normalization and validation-based evidence fusion, (6) locked test evaluation and ablation analysis, and (7) repeated validation with statistical significance testing. This workflow ensures that no information from the validation or test partitions is used during model training or preprocessing.

The complete experimental workflow implemented in this study follows the corrected leakage-aware protocol summarized in Figure 1, where feature scaling, anomaly detector training, fusion-weight optimization, and threshold selection are performed using separate data partitions.

3.1 Problem Formulation

Each software session is represented by two complementary feature vectors,

$$x = [x\ SL, x\ SR],\ x\ SL \in R\text{^}8,\ x\ SR \in R\text{^}10 \qquad (1)$$

where

- x_{SL} represents the Shift-Left security evidence collected before deployment,
- x_{SR} represents the Shift-Right runtime telemetry collected after deployment.

The objective is to estimate a continuous security risk score

$R(x) \in [0,1]$ (2)

where higher values indicate greater likelihood that the observed session is malicious.

Each session is assigned a binary class label

$$y \in \{0,1\}, \quad y = 0\ benign, \quad y = 1\ malicious \qquad (3)$$

Malicious sessions include both signature-based attacks and runtime-only synthetic anomalies generated during the experimental process.

The overall framework described above is summarized in Figure 1, while the mathematical components of the evidence-fusion model are described in the following subsections.

3.2 Shift-Left Security Evidence

The Shift-Left component simulates the outputs of common DevSecOps security scanners. Instead of relying on a single security tool, eight complementary security indicators are generated to emulate typical CI/CD security assessments:

- Static Application Security Testing (SAST)
- Software Composition Analysis (SCA)
- Infrastructure-as-Code (IaC) analysis
- Container image vulnerability assessment
- Secret detection
- Dynamic Application Security Testing (DAST)
- Software Bill of Materials (SBOM) risk assessment
- Policy compliance violations

These eight indicators constitute the pre-deployment feature vector shown in Figure 2.

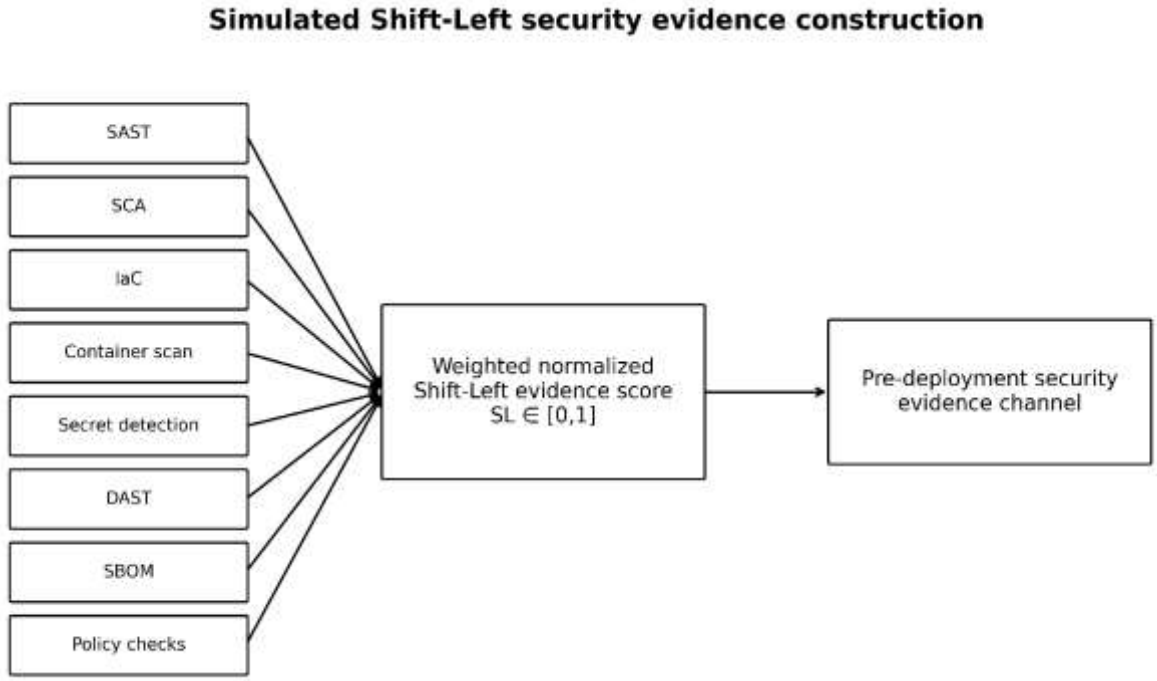


Figure 2: Simulated Shift-Left Security Evidence Construction

To obtain a continuous evidence score, the individual indicators are aggregated using weighted linear combination,

$$S_{SL} = 0.20 \cdot SAST\sim + 0.17 \cdot SCA\sim + 0.13 \cdot IaC\sim + 0.15 \cdot Container\sim + 0.10 \cdot Secrets\sim + 0.10 \cdot DAST\sim + 0.08 \cdot SBOM\sim + 0.07 \cdot Policy\sim \qquad (4)$$

The selected weights reflect the relative contribution assigned to each simulated security source. Since the underlying findings are synthetically generated rather than collected from operational security scanners, this score should be interpreted as synthetic security evidence rather than real scanner output.

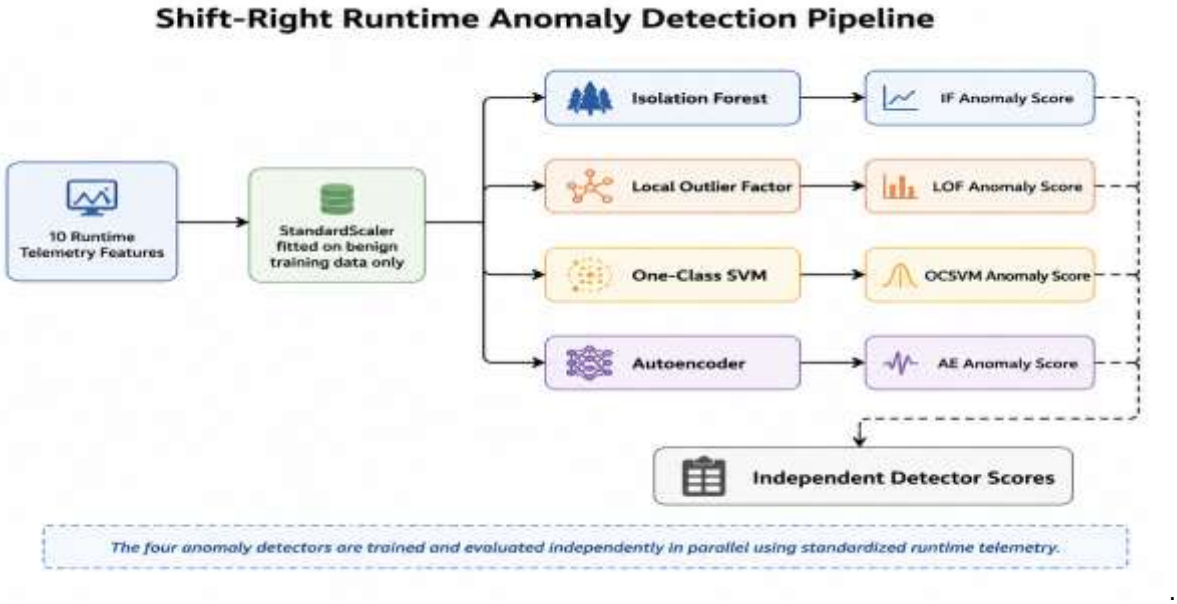
3.3 Runtime Anomaly Detection

Following deployment, software behavior is monitored through ten runtime telemetry features representing service execution characteristics, including request behavior, latency, CPU utilization, memory consumption, privileged system calls, outbound connections, payload entropy, and process anomalies.The runtime detection pipeline is illustrated in Figure 3.

Prior to detector training, only the ten Shift-Right runtime telemetry features were standardized. The StandardScaler was fitted exclusively on benign runtime samples from the training partition and was then reused unchanged for validation and test data. Isolation Forest, LOF, One-Class SVM, and the Autoencoder were trained only on these standardized benign runtime features, ensuring that the runtime anomaly detectors did not directly consume Shift-Left evidence.

Four unsupervised anomaly detection algorithms are evaluated:

- Isolation Forest (IF)
- Local Outlier Factor (LOF)
- One-Class Support Vector Machine (OCSVM)
- Autoencoder (AE)



- Figure 3. Shift-Right Runtime Anomaly Detection Pipeline

Each detector is trained only on benign training samples, thereby learning the normal operational profile of the monitored software system.

The Autoencoder employs a symmetric 10–8–4–3–4–8–10 encoder–decoder architecture, where the ten-dimensional input corresponds exclusively to the Shift-Right runtime telemetry features. The three-neuron bottleneck layer provides a compressed representation of benign runtime behavior

The anomaly score is computed using the mean squared reconstruction error,

$S\ AE(x\ SR) = (1/d\ SR)\ \Sigma\ \{j=1\}^{d\ SR}\ (x\ SR,j - \hat{x}\ SR,j)^2,\ \ d\ SR = 10$ (5)

where (d {SR}=10) denotes the number of Shift-Right runtime telemetry features.

Higher reconstruction errors indicate increasing deviation from normal behavior.

3.4 Score Normalization and Evidence Fusion

3.4 Score Normalization and Evidence Fusion

The raw outputs produced by the individual anomaly detectors are measured on different numerical scales. Therefore, each detector score is independently normalized using the minimum and maximum values estimated exclusively from the benign training samples. This prevents information from the validation or test partitions from influencing the normalization process.

For detector d, the normalized anomaly score for session i is defined as

$\tilde{s}_{i,}^{d} = (s_{i,}^{d} - s^{d},_{min}\text{^}(train)) / (s^{d},_{max}\text{^}(train) - s^{d},_{min}\text{^}(train) + \varepsilon)$ (6)

where $s_{i,}^{d}$ denotes the raw anomaly score produced by detector d for session i, $s^{d},_{min}\text{^}(train)$ and $s^{d},_{max}\text{^}(train)$ denote the minimum and maximum detector scores observed exclusively in the benign training partition, respectively, and ε is a small positive constant introduced to avoid division by zero.

To prevent values outside the training-derived range from producing unbounded normalized scores, the resulting values are clipped to the interval [0,1]:

$\hat{s}_i^{d} = \min(1, \max(0, \tilde{s}_i^{d}))$ (7)

The normalized runtime detector scores are then integrated with the normalized Shift-Left evidence score to obtain a single hybrid evidence score for each session:

$H_i = wSL\ s_i,SL + wIF\ \hat{s}_i,IF + wLOF\ \hat{s}_i,LOF + wAE\ \hat{s}_i,AE$ (8)

$wSL + wIF + wLOF + wAE = 1, \quad w_j \geq 0$

Here, s_i,SL denotes the normalized Shift-Left evidence score for session i, while $\hat{s}_i$,IF, $\hat{s}_i$,LOF, and $\hat{s}_i$,AE represent the normalized anomaly scores produced by Isolation Forest, Local Outlier Factor, and the Autoencoder, respectively. The corresponding weights wSL, wIF, wLOF, and wAE determine the contribution of each evidence source to the final hybrid score.

OCSVM is not included in the validation-tuned weighted fusion defined in Equation (8). Instead, it is retained as an independent anomaly-detection comparator and is also included as an input to the supervised Logistic Stacking baseline, as described in Section 3.6. This distinction ensures consistency between the mathematical formulation and the experimental configurations evaluated in this study.

The complete evidence-fusion process is illustrated in Figure 4. Unlike conventional score-fusion approaches, the proposed framework does not assume that combining multiple evidence sources necessarily improves predictive performance. Instead, the contribution of each evidence source is determined empirically through validation-only weight optimization, as described in Section 3.5. Consequently, individual components may receive zero weight when their inclusion does not improve the predefined validation criteria.

3.5 Validation-Only Weight Optimization

Candidate fusion weights are enumerated in increments of 0.1 while satisfying the constraint defined in Equation (8). For every candidate weight vector, decision thresholds corresponding to the 90th–99th percentiles of benign validation scores are evaluated.

The optimal configuration is selected using the following hierarchical criteria:

1. Maximum validation F1-score.
2. Highest Matthews Correlation Coefficient (MCC).
3. Lowest False Positive Rate (FPR).

The selected weights and threshold remain fixed throughout the final test evaluation.

The validation-based optimization procedure illustrated in Figure 4 guarantees that the test partition remains completely isolated until the final evaluation, thereby preventing optimistic performance estimation caused by data leakage.

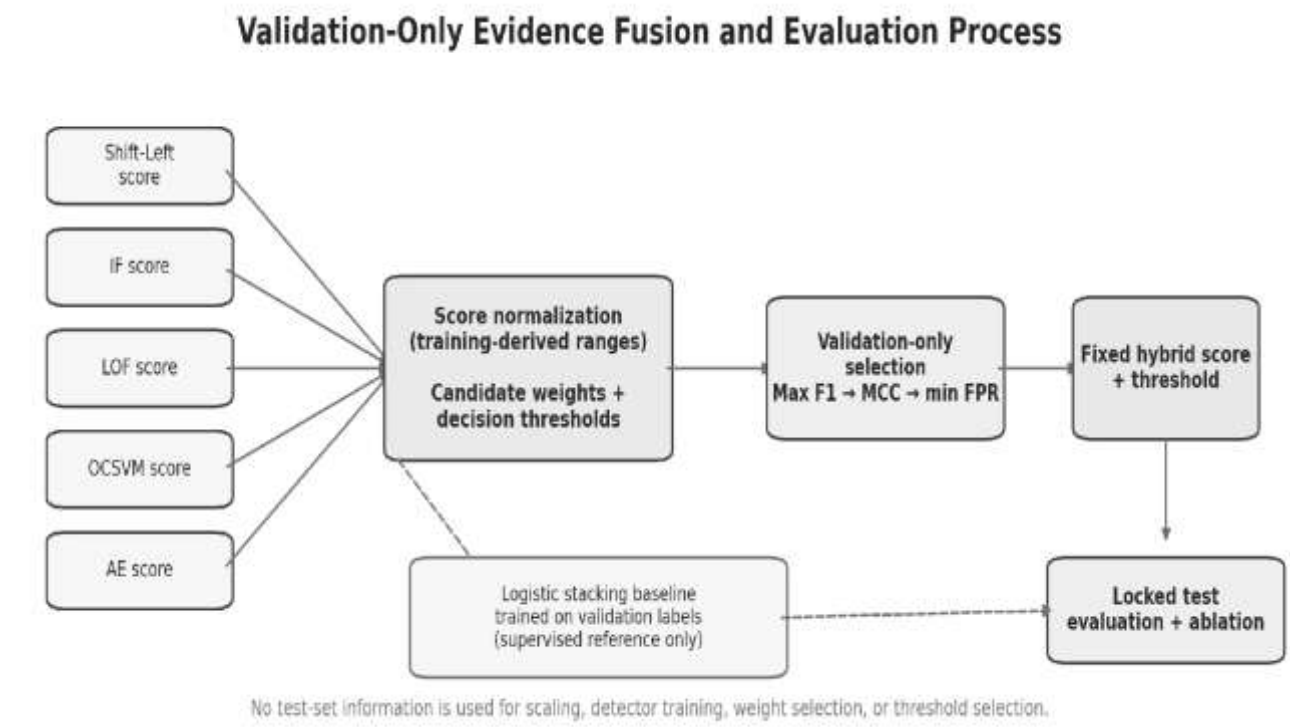


Figure 4: Evidence Fusion Process

3.6 Logistic Stacking Baseline

In order to have a supervised reference model, a Logistic Regression meta-classifier is assessed apart from the proposed fusion framework based on unsupervised base detectors and label-guided validation calibration.
The classifier accepts five normalized inputs (Shift-Left evidence score, Isolation Forest score, LOF score, OCSVM score, Autoencoder score).

Unlike the proposed hybrid framework, Logistic Stacking is trained directly on validation labels. Consequently, it is reported only as a supervised baseline and is not considered part of the proposed unsupervised evidence-fusion methodology.

3.7 Experimental Workflow

The complete experimental workflow is illustrated in Figure 4. After the model configuration and decision threshold are selected exclusively using the validation partition, the selected configuration is evaluated once on the locked test partition. To assess the contribution of individual evidence sources and detector combinations, 14 predefined ablation configurations are evaluated using the same validation-based threshold-selection and locked-test protocol.

To assess the stability of the results, the entire experimental pipeline is independently repeated across 30 runs using different random seeds. Each run includes synthetic data generation, stratified data partitioning, preprocessing based exclusively on the training data, benign-only training of the unsupervised anomaly detectors, validation-only calibration of fusion weights and decision thresholds, and final evaluation on the corresponding locked test partition. This repeated procedure captures variability arising from synthetic data generation, data partitioning, and model initialization.

Performance across the 30 independent runs is summarized using the mean, sample standard deviation, and 95% empirical percentile intervals. Statistical comparisons are conducted using paired Wilcoxon signed-rank tests on the matched F1-scores across runs, with Holm correction applied to control the family-wise error rate arising from multiple comparisons. Rank-biserial correlation is additionally reported as an effect-size measure. These analyses provide statistical evidence for assessing whether the observed performance differences are consistent across repeated experiments, rather than implying universal statistical validity beyond the controlled experimental setting.

4. Materials and Methods

This section describes the experimental methodology used for testing the suggested DevSecOps evidence-fusion framework. The comprehensive experimental workflow, as illustrated in Figure 1, incorporates the creation of synthetic datasets, feature engineering, leakage-aware dataset partitioning, training of benign-only models, validation-based score fusion, ablation analysis, and iterative statistical validation. The entire data preparation and evaluation framework adopted in this research work is presented in Figure 5.

4.1 Dataset Design

Dataset designing made it possible to test the framework in a controlled way. The dataset, shown in Figure 5, is made from 4,600 software sessions, with 4,000 benign, 300 sessions with signature-based attacks, and 300 sessions with runtime-only synthetic anomalies.

For the known attacks, Shift-Left findings were deliberately made to be more aggressive, with moderate runtime deviations. On the other hand, runtime-only synthetic anomalies were designed to contain benign, pre-deployment evidence with considerable deviations in runtime telemetry. Hence, the dataset makes it easy to differentiate attacks that are to be detected in pre-deployment from anomalies that will be detected in post-deployment. The dataset is shown in Table 2.

Table 2. Dataset Composition

Session Type	Number of Sessions	Binary Label	Runtime-only Indicator
Benign	4000	0	0
Known attack	300	1	0
Runtime-only synthetic anomaly	300	1	1
Total	4600		

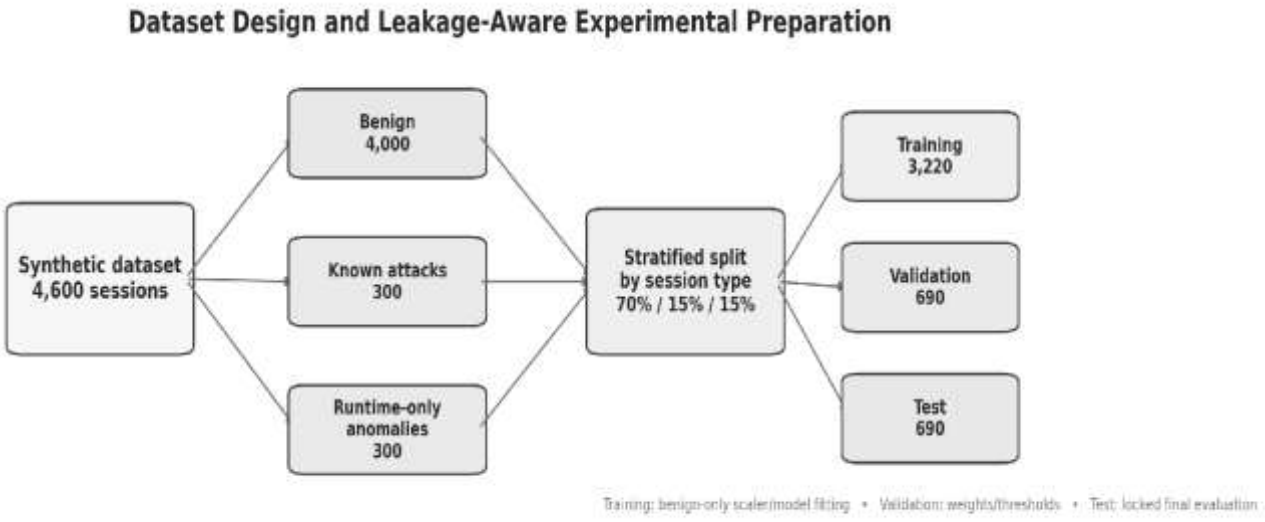


Figure 5: Dataset Design

4.2 Feature Engineering

Every software session follows the feature structure established in Equation (1) and comprises 18 developed features. Among these, eight features are identified as Shift-Left security indicators, while the remaining ten are recognized as Shift-Right runtime telemetry measurements.

According to this framework, Shift-Left features are represented by the evidence score in Equation (4). Runtime features are the inputs for the anomaly detection models, as described in Section 3. Table 3 provides a description of the developed features.

Table 3. Engineered Features Used in the Proposed Framework

Feature	Description	Evidence Channel
sast_findings	Simulated SAST findings	Shift-Left
sca_vulns	Vulnerable dependencies	Shift-Left
iac_misconfig	Infrastructure-as-Code misconfigurations	Shift-Left
container_vuln_score	Container vulnerability severity	Shift-Left
secrets_detected	Detected secrets	Shift-Left
dast_findings	Dynamic application testing findings	Shift-Left
sbom_risk_score	SBOM risk score	Shift-Left
policy_violations	Policy-as-Code violations	Shift-Left
request_rate	Request frequency	Shift-Right
avg_latency_ms	Average latency	Shift-Right
error_rate	Error rate	Shift-Right
cpu_usage	CPU utilization	Shift-Right
memory_usage	Memory utilization	Shift-Right
priv_syscalls	Privileged system calls	Shift-Right
unique_endpoints	Unique endpoints	Shift-Right
outbound_conns	Outbound connections	Shift-Right
payload_entropy	Payload entropy	Shift-Right
process_anomalies	Process anomaly count	Shift-Right

4.3 Dataset Partitioning and Leakage Control

To remain impartial, the dataset contained 3,220 training sessions, 690 validation sessions, and 690 testing sessions after the data was split into 70% training sessions, and 15% validation and testing sessions.

Stratified sampling was used according to the attack family in order to retain relative proportions in the training, validation, and test sessions for benign sessions, known attacks, and runtime-only synthetic anomalies. This is shown in Figure 5.

In this work, feature standardization was carried out by fitting a StandardScaler on benign training samples, and not refitted for the validation and test sets.
Also, all unsupervised anomaly detectors were trained on benign training samples only. For score normalization, only benign training samples were used, as shown in Equation (6). The validation set was used to optimize fusion weights and decision thresholds, as described by the evidence-fusion framework, Equations (7) and (8).
During the preprocessing phase, feature scaling, model training, score normalization, and the model selection process, the test partition was kept isolated to ensure no information was leaked for the experimental evaluation.
The framework proposed in this work was used to evaluate four unsupervised anomaly detection models, and two score-combination techniques.
The runtime models were Isolation Forest (IF), Local Outlier Factor (LOF), One-Class Support Vector Machine (OCSVM), and the Autoencoder from Equation (5). The outputs of these models were combined by the evidence-fusion model optimized on validation, described in Section 3.

A supervised Logistic Regression stacking classifier was also used as an independent baseline for comparison. The base anomaly detectors were trained without malicious labels. However, fusion weights and decision thresholds were calibrated on the labeled validation partition. Therefore, the framework uses benign-only unsupervised base detectors with label-guided validation calibration, while Logistic Stacking remains a supervised score-level baseline.

Table 4 summarizes the primary model configurations.

Model	Configuration
Isolation Forest	300 trees; contamination=auto; benign-only training
LOF	35 neighbors; novelty=True; contamination=auto
One-Class SVM	RBF kernel; ν=0.05; γ=scale
Autoencoder	Autoencoder 10–8–4–3–4–8–10; tanh; Adam optimizer; early stopping; trained only on benign Shift-Right runtime features.
Validation-tuned fusion	Shift-Left, IF, LOF, and AE; weights searched in increments of 0.1
Logistic Stacking	Balanced logistic regression using five normalized detector scores

4.5 Evaluation Metrics

For the evaluation of the performance, Accuracy, Precision, Recall, F1-score, Matthews Correlation Coefficient (MCC), Balanced Accuracy, ROC-AUC, PR-AUC, Specificity, False Positive Rate (FPR), Known-Attack Recall, and Runtime-Only Recall were calculated.
In this case, malicious sessions are the minority class, so MCC and PR-AUC were the main evaluation metrics. Known-Attack Recall and Runtime-Only Recall, on the other hand, measure the performance of detection of the two malicious subclasses.

4.6 Ablation Study and Repeated Validation

To measure how much each evidence channel contributes, we reviewed fourteen detector configurations: single detectors, pairs, several types of fusion models, and the validation-tuned hybrid framework.
For each configuration, only the validation partition was used to set decision thresholds before evaluation on the locked test set.
The pipeline used to measure the impact of each component on the overall model was also used to measure the robustness of the models. This was done by repeating the entire process, including generation of data, partitioning data (ensuring each subset has a representative sample), preprocessing data, training a benign-only model, weight optimization with validation, and the final evaluation on the test set, thirty times using the random seeds from 100 to 129, as shown in Figure 6.
Using the data from the thirty repetitions of each method, the average was calculated, as well as the sample standard deviation and the percentiles to create the 95% empirical percentile interval.

The validation-tuned hybrid framework was compared against the other methods using a paired Wilcoxon signed-rank test. The Holm correction was used to account for the multiple comparisons, and to measure the size of the effects, rank-biserial correlation coefficients were used. Figure 6 summarizes the repeated-validation protocol adopted in this study. For each of the 30 independent runs, a new synthetic dataset is generated, stratified into training, validation, and testing subsets, followed by benign-only preprocessing, validation-only tuning, locked-test evaluation, and aggregation of performance metrics.

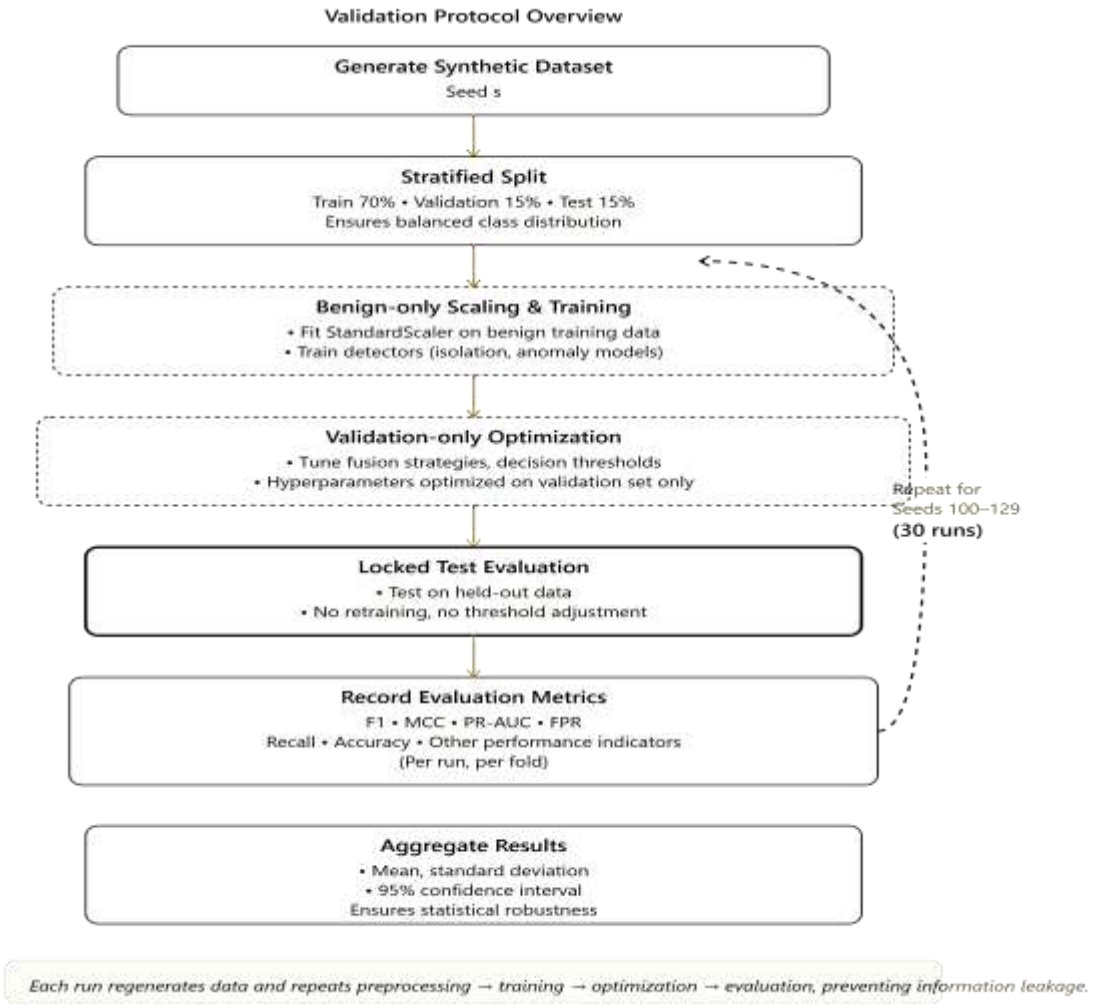


Figure 6. Stratified Partitioning and Thirty-Run Repeated Validation Protocol

5. Results

This section outlines the results of the primary experiment with validation selection, ablation study, 30-run stability analysis, and paired statistical comparisons. Results of the tests are from the locked test partition after model training, score normalization, and after calibration of fusion-weights and thresholds, with only the training and validation partitions utilized.

5.1 Primary-Run Validation-Selected Model

In the primary experimental run, the validation-only optimization procedure selected the following fusion weights:

$$w_{SL} = 0.00,\ w_{IF} = 0.00,\ w_{LOF} = 0.00,\ w_{AE} = 1.00,$$

with a validation-selected decision threshold of 0.559818. The optimized four-component formulation reduced to the Autoencoder as opposed to a multi-component weighted fusion.

The result is methodologically significant as it was permitted to assign zero-weight to components that did not improve the validation objective during the optimization procedure. Thus, the proposed method did not impose the inclusion of Shift-Left, Isolation Forest, or LOF evidence to maintain a somewhat hybrid structure.

According to Table 5, the validation-selected model showed an accuracy of 0.9884, an F1 score of 0.9570, an MCC of 0.9510, and a balanced accuracy of 0.9886. The model's ROC-AUC and PR-AUC values were 0.9997 and 0.9982, respectively. The model was also successful in the detection of the known attacks and of 97.78% of the synthetic anomalies that were generated during normal runtime. The resulting confusion matrix had 89 true positives, 593 true negatives, seven false positives, and one false negative.

Table 5. Performance of the Validation-Selected Model

Metric	Value
Accuracy	0.9884
Precision	0.9271
Recall	0.9889
F1-score	0.9570
MCC	0.9510
Balanced Accuracy	0.9886
ROC-AUC	0.9997
PR-AUC	0.9982
FPR	0.0117
Known Attack Recall	1.0000
Runtime-only Recall	0.9778
TN	593
FP	7
FN	1
TP	89

Although the primary validation process selected the Autoencoder alone, this outcome should not be interpreted as evidence that the other components are universally unnecessary. It indicates only that, for the primary validation partition and the predefined optimization hierarchy, adding the other weighted scores did not improve validation F1-score, MCC, or false-positive performance.

5.2 Primary-Run Ablation Study

The ablation study looked into individual detectors as well as selected multi-component combinations. For each combination considered, a distinct decision threshold was set using the validation set. The resulting test performance is detailed in Table 6.

The configuration of the equally weighted Isolation Forest–Autoencoder achieved the best primary run scores, with an F1-score of 0.9677 and an MCC of 0.9634. The configuration also achieved perfect recall for the known attacks and runtime-only synthetic anomalies and had a false-positive rate of 0.0100. F1-scores of 0.9626 were also achieved with the Isolation Forest alone and the equally weighted Isolation Forest–LOF configuration.

The validation-selected model, which collapsed to the Autoencoder, achieved an F1-score of 0.9570. Therefore, the configuration selected in the validation set was different from the configuration with the best performance in the locked test set. As expected, this difference is a result of finite-sample variation and does not warrant post hoc retuning on the test set. The final model would lose the test set's evaluation independence if the best test set's ablation result was considered.

The primary-run performance of the best-performing detector configurations is summarized in Table 6. The comparison includes the F1-score, Matthews Correlation Coefficient (MCC), and False Positive Rate (FPR), allowing direct evaluation of the most competitive models under the locked-test protocol.

Table 6. Primary-Run Performance Comparison of the Best Configurations

Method	F1	MCC	FPR
IF + AE	0.9677	0.9634	0.0100
IF only	0.9626	0.9576	0.0117
IF + LOF	0.9626	0.9576	0.0117
AE Only	0.9570	0.9510	0.0117
Tuned Fusion	0.9570	0.9510	0.0117
IF + LOF + AE	0.9524	0.9463	0.0150

Additional performance metrics for the same top-performing configurations are presented in Table 7. These results include ROC-AUC, PR-AUC, Known-Attack Recall, and Runtime-only Recall, providing a more comprehensive assessment of detection capability across different evaluation criteria.

Table 7. Additional Detection Performance of the Best Configurations

Method	ROC-AUC	PR-AUC	Known Attack Recall	Runtime-only Recall
IF + AE	0.9999	0.9995	1.000	1.000
IF Only	0.9998	0.9985	1.000	1.000
IF + LOF	1.0000	1.0000	1.000	1.000

AE Only	0.9997	0.9982	1.000	0.9778
Tuned Fusion	0.9997	0.9982	1.000	0.9778
IF + LOF + AE	0.99998	0.99988	1.000	1.000

The ten strongest primary-run configurations and their respective F1-scores are shown in Figure 7. The figure identifies IF + AE achieved the highest F1-score among the evaluated primary-run ablation configurations, followed by IF Only and IF + LOF. Performance differences for many of the top configurations were small.

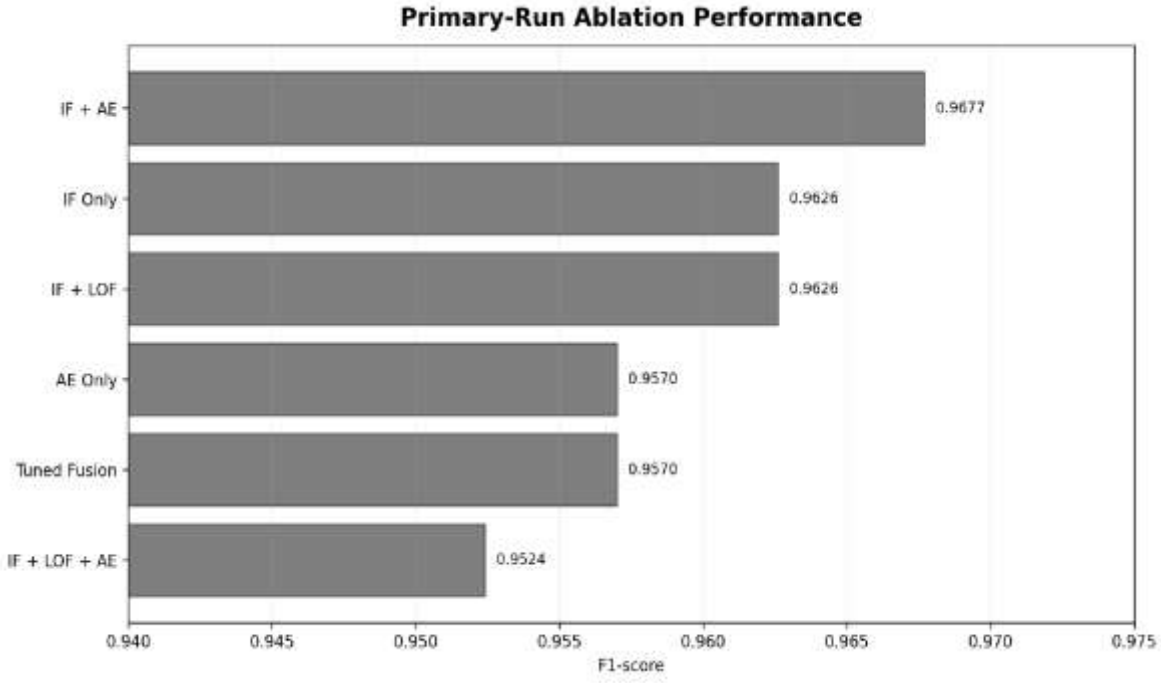


Figure 7: Ablation Performance

5.3 Thirty-Run Stability Analysis

A single experimental run may be influenced by stochastic factors, including synthetic data generation, data partitioning, model initialization, and validation-based model selection. To assess the stability of the reported results, the entire experimental process was independently repeated across 30 runs using random seeds 100–129. In each run, synthetic data generation, stratified data partitioning, training-based preprocessing, benign-only detector training, validation-only calibration, and locked-test evaluation were performed independently from scratch.

The repeated-run results presented in Table 8 show that Logistic Stacking achieved the highest mean F1-score of 0.9704, with a sample standard deviation of 0.0163 and a 95% empirical percentile interval of [0.9362, 0.9944]. It also achieved the highest mean MCC (0.9662) and the lowest mean false-positive rate (0.0079) among the evaluated methods.

Among the standalone unsupervised anomaly detectors, LOF achieved the highest mean F1-score of 0.9619, followed by OCSVM with 0.9584. The validation-tuned fusion achieved a mean F1-score of 0.9568, while the Autoencoder and Isolation Forest achieved mean F1-scores of 0.9521 and 0.9489, respectively. These numerical differences should be interpreted together with the statistical comparisons reported in Section 5.4 rather than as evidence of statistical superiority based solely on mean performance.

The Shift-Left-only method achieved a substantially lower mean F1-score of 0.6311 and a mean runtime-only recall of 0.0000. This result is consistent with the controlled data-generation design, in which runtime-only synthetic anomalies were generated with minimal pre-deployment evidence but substantial deviations in runtime behavior. Consequently, the Shift-Left evidence channel alone was not designed to detect this synthetic anomaly subgroup. Statistical significance between the Shift-Left-only method and the validation-tuned fusion is examined separately in Section 5.4.

Table 8. Repeated-Validation Summary over 30 Independent Experimental Runs

Method	Mean F1	SD	95% Empirical Percentile Interval	Mean MCC	Mean Runtime-Only Recall	Mean FPR
Logistic Stacking	0.9704	0.0163	[0.9362, 0.9944]	0.9662	0.9896	0.0079
LOF Only	0.9619	0.0169	[0.9306, 0.9866]	0.9567	0.9919	0.0108
OCSVM Only	0.9584	0.0154	[0.9260, 0.9851]	0.9529	0.9948	0.0121
Validation-Tuned Fusion	0.9568	0.0175	[0.9210, 0.9862]	0.9509	0.9881	0.0120
AE Only	0.9521	0.0173	[0.9143, 0.9759]	0.9454	0.9785	0.0125
IF Only	0.9489	0.0172	[0.9180, 0.9780]	0.9417	0.9541	0.0118
SL Only	0.6311	0.0187	[0.5919, 0.6583]	0.6204	0.0000	0.0123

The 95% empirical percentile intervals reported in Table 8 and illustrated in Figure 8 characterize stochastic variability across the 30 independent runs within the controlled synthetic experimental setting. They should not be interpreted as confidence intervals for expected performance in real-world DevSecOps environments.

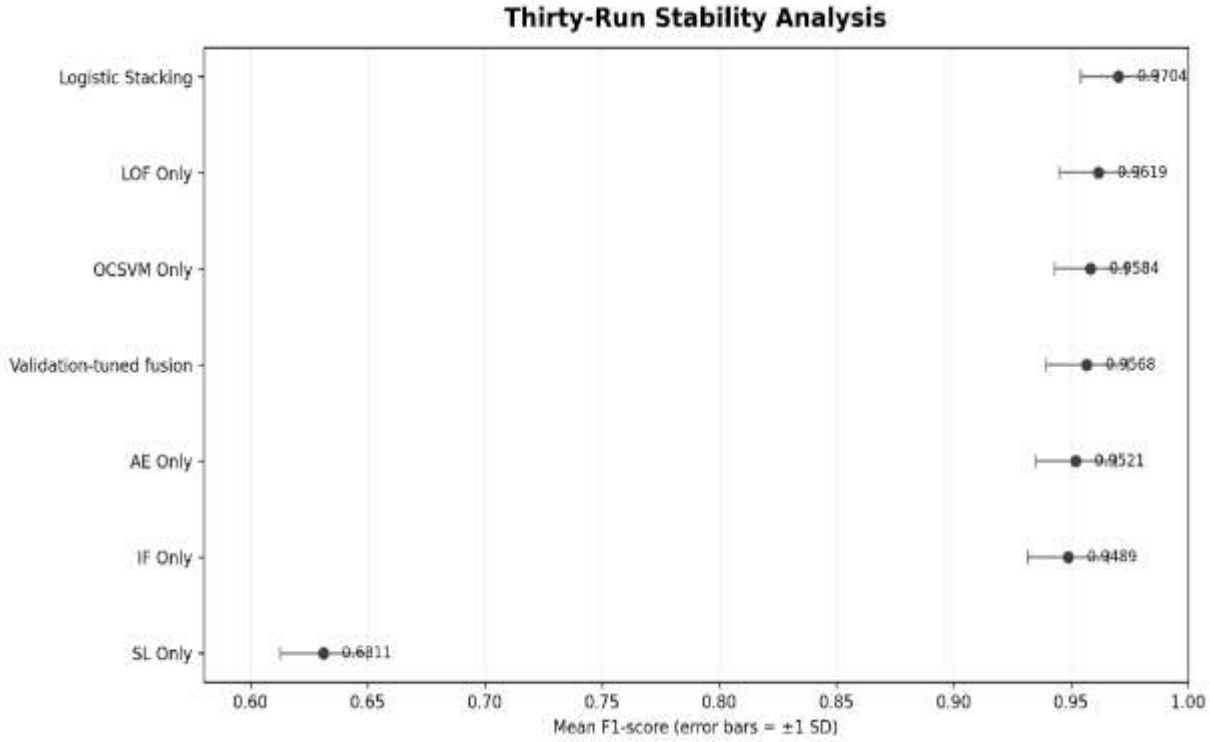


Figure 8: Stability Analysis

5.4 Statistical Comparison

Paired Wilcoxon signed-rank tests were carried out on all matched F1-scores from the 30 independent repetitions. Each comparator was measured against the validation-tuned fusion. To manage the family-wise error rate among the six comparisons, Holm correction was used. The statistical details can be seen in Table 8.

Post-holm adjustment, Logistic Stacking was superior to the validation-tuned fusion (Holm-adjusted $p = 0.0046$; rank-biserial effect = −0.7143). The rank-biserial effect being negative means the differences of the pairs (tuned-fusion F1 minus comparator F1) was, in general, in favor of Logistic Stacking.

The validation-tuned fusion outperformed the Shift-Left-only method (Holm-adjusted $p < 0.0001$; rank-biserial effect = 1.0000). The positive effect indicates that the validation-tuned fusion method of F1-Score fusion (tuned) was beneficial in all the observed pairs.

After Holm adjustment, no significant differences were observed for the validation-tuned fusion in comparison to AE, IF, LOF, or OCSVM. For the comparison with LOF, an unadjusted $p = 0.0143$ was observed, which, after adjustment, reflected an increase to 0.0574, thus, the result was nonsignificant ($\alpha = 0.05$). An unadjusted difference of the tuned fusion and AE was also observed but was lost after the multiple-comparison adjustment.

The repeated-run results indicate that several standalone runtime anomaly detectors achieved numerically higher mean F1-scores than the validation-tuned fusion; however, these differences were not statistically significant after Holm correction. The results say validation-tuned weighted fusion performs statistically similarly to the unsupervised detectors in this experiment. The validation-tuned fusion significantly outperformed the Shift-Left-only method but was significantly outperformed by the supervised Logistic Stacking baseline after Holm correction.
This segment looks at research questions and the validation-weighted zero-tuning solution alongside the practical importance of Shift-Left and Shift-Right evidence in a DevSecOps context. Findings of this investigation assume a controlled synthetic environment of the experiment and therefore should not be used to draw conclusions on production systems in the absence of external validation.

Table 9. Paired Wilcoxon signed-rank comparisons with Holm correction

Comparison	Wilcoxon Statistic	Raw PValue	Holm Adjusted PValue	Rank Biserial Effect	Significant At 0.05
SL + IF + LOF + AE (Tuned) vs AE Only	55	0.035479948	0.106439845	0.428571429	FALSE
SL + IF + LOF + AE (Tuned) vs IF Only	142.5	0.064140859	0.128281718	0.333333333	FALSE
SL + IF + LOF + AE (Tuned) vs LOF Only	57.5	0.014343439	0.057373756	-0.47826087	FALSE
SL + IF + LOF + AE (Tuned) vs Logistic Stacking	57.5	0.000921149	0.004605746	-0.714285714	TRUE
SL + IF + LOF + AE (Tuned) vs OCSVM Only	177	0.773101194	0.773101194	-0.111111111	FALSE
SL + IF + LOF + AE (Tuned) vs SL Only	0	1.86E-09	1.12E-08	1	TRUE

6.1 Answers to the Research Questions

RQ1: Are simulated Shift-Left evidence and runtime-only synthetic anomalies sufficiently different?
The results appear to show the required difference among the two malicious subgroups. The Shift-Left-only approach managed to spot the known attacks, while achieving a mean runtime-only recall score of 0.0000 after 30 attempts. This aligns with the design of the dataset, where known attacks were created with a significantly high number of pre-deployment findings, while the runtime-only synthetic anomalies were designed to contain minimal Shift-Left evidence with considerable runtime deviation.
The findings demonstrate complementary detection roles for the two evidence channels by construction: Shift-Left evidence primarily represents known pre-deployment findings, whereas runtime telemetry captures the simulated runtime-only anomalies. Nevertheless, since the result only captures the assumptions in the synthetic data creation, it is illogical to generalize to results produced from real SAST, SCA, IaC, DAST, SBOM, secret detection, or container scanning without an external assessment.
RQ2: What is the performance of anomaly detectors when trained with only benign examples?
Across the 30 independent runs, LOF and One-Class SVM achieved the highest mean F1-scores among the standalone unsupervised anomaly detectors, with values of 0.9619 and 0.9584, respectively. The Autoencoder achieved a mean F1-score of 0.9521 across the repeated runs. In the primary run, however, the validation-only optimization selected the Autoencoder, which achieved an F1-score of 0.9570 on the locked test set.
This result suggests that the Autoencoder reconstruction error provided useful discrimination within the primary validation split. Nevertheless, the repeated-run analysis did not establish a single universally superior unsupervised detector. LOF achieved the highest mean F1-score among the standalone unsupervised detectors, but the statistical comparisons did not establish a significant advantage of the leading standalone detectors over the validation-tuned fusion after correction for multiple comparisons. Logistic Stacking achieved the highest overall mean F1-score of 0.9704; however, it should be interpreted separately because it was trained using labeled validation data and therefore serves as a supervised score-level baseline rather than an unsupervised detector.
RQ3: Does weighted fusion outperform individual anomaly detectors?
The results do not demonstrate a consistent performance advantage for validation-tuned weighted fusion over the evaluated standalone runtime anomaly detectors. The tuned fusion achieved a mean F1-score of 0.9568, whereas LOF and One-Class SVM achieved mean F1-scores of 0.9619 and 0.9584, respectively. After Holm correction, no statistically significant differences were detected between the validation-tuned fusion and LOF, One-Class SVM, Isolation Forest, or the Autoencoder.

The principal value of weighted fusion in this study is therefore architectural rather than a demonstrated universal improvement in predictive performance. The proposed framework enables security evidence from different stages of the software development lifecycle to be integrated and evaluated within a unified DevSecOps pipeline. However, within the controlled synthetic setting evaluated here, integrating multiple evidence sources did not consistently improve predictive performance over the strongest standalone runtime detectors. Accordingly, the inclusion and weighting of evidence channels should be determined by validation evidence and operational objectives rather than by the assumption that adding more components will necessarily produce a better detector.

6.3 Operational Implications

Detection performance is only one reason to retain Shift-Left evidence in a DevSecOps pipeline. From an operational perspective, pre-deployment findings can remain valuable even when they do not maximize F1-score because they support vulnerability remediation, ownership assignment, policy compliance, auditing, and traceability between runtime incidents and software engineering artifacts.

Therefore, a production-oriented framework should not be evaluated solely on classification performance. Additional criteria may include alert interpretability, time to remediation, explanation quality, operational false-alert cost, compliance requirements, and the ability to trace runtime events back to development-stage evidence. Under these considerations, a Shift-Left component can remain justified within a production framework even when its direct contribution to runtime anomaly-detection performance is limited.

The present experiments also indicate that the appropriate evidence-combination strategy depends on the available validation data and deployment constraints. In this study, the supervised Logistic Stacking baseline achieved the highest mean performance when labeled validation data were available. When labeled validation data are available but insufficient for training a flexible meta-classifier, simpler validation-calibrated weighted fusion may provide a lower-complexity alternative. In either case, model selection should be guided by the available validation evidence together with deployment requirements, explainability needs, and the operational cost of false alerts.

Overall, the results support the controlled integration of lifecycle security evidence but do not support the assumption that weighted fusion is universally superior for detection. The main methodological contribution of this work is the leakage-aware experimental design, benign-only detector training, validation-only calibration, repeated stability analysis, ablation analysis, and statistically corrected comparison of alternative configurations.

7. Threats to Validity and Limitations

This study was intended to be a controlled proof-of-concept to analyze the validation-based security evidence fusion in DevSecOps. Even though the experimental protocol includes leakage-aware training, iterative validation, and validation through statistical significance, various dimensions may constrain the result interpretation and generalization.

7.1 Internal Validity

Generating data synthetically poses the greatest risk to the internal validity of the study. Though careful consideration of realistically possible DevSecOps scenarios was made in creating the synthetic data set, the benign sessions, known attacks, and runtime only anomalies differentiate themselves based on the simulation of the data generation. Therefore, some of the observed classification performance may be attributed to the synthetic distributions rather than the complexity of the real functional environments.

Additionally, the simulated environment excludes many characteristics seen in production, including, but not limited to, the presence of Kubernetes workloads, various service dependencies, burst traffic, concept drift, flexible antagonists, attack campaigns, and cross-service behavioral correlations. All of these may impact the performance of the detector in a real deployment scenario.

7.2 External Validity

The Shift-Left evidence channel proposal was created to be similar to results from some tools in DevSecOps automated security processes, as found in SAST, Software Composition Analysis, Infrastructure as Code Assessment, Dynamic Application Security Testing, Software Bill of Materials (SBOM) assessment, secret-detection tools, container image scanners, etc. Rather, these findings were fabricated rather than collected from any security operational platform.

As such, the proposed framework was to be relied upon as providing a particular data-generation methodology with a focus on evidence-fusion, not as providing direct examples of interoperability with the commercial/open-source DevSecOps security tools. Real outputs from scanners and real-time cyber operational security and cyber defense/attack platforms telemetry will be necessary to validate the proposed framework and make the framework usable for operations.

7.3 Methodological Limitations

The entire experimental workflow was performed independently 30 times, but all repetitions were generated from the same family of synthetic probability distributions. As a result, the repeated experiments reflect stochastic stability and, at best, indicate external validity for the simulation environment. They do not demonstrate external validity to other organizations, other applications, other infrastructures, other attack families, or other operational datasets.
The reported percentile-based 95% intervals account for the variability of the simulated experiments and should not be viewed as confidence intervals regarding the expected performance of real-world DevSecOps.
Another constraint relates to the Logistic Stacking baseline. Unlike the proposed evidence-fusion framework, Logistic Stacking uses validation labels and thus is trained with a form of supervision which is absent for the unsupervised fusion approaches. Thus, it is important to view Logistic Stacking's performance not as direct evidence of the inferior nature of score-level fusion, but rather as a supervised reference baseline.

8. Conclusion and Future Work

This study presented a reproducible proof-of-concept framework for integrating Shift-Left pre-deployment security evidence with Shift-Right runtime anomaly detection in DevSecOps CI/CD microservice pipelines. The primary contribution lies in the software security engineering methodology used to evaluate lifecycle evidence integration rather than in proposing a new anomaly detection algorithm. The experimental design incorporated leakage-aware data partitioning, benign-only training of unsupervised anomaly detectors, validation-only calibration of fusion weights and decision thresholds, locked-test evaluation, ablation analysis, and 30 repeated experiments with corrected statistical comparisons. Together, these elements provide a reproducible baseline for evaluating security evidence integration under controlled experimental conditions.
The results showed that the Shift-Left-only evidence channel was insufficient for detecting the simulated runtime-only anomaly subgroup, which was expected from the controlled data-generation design. The validation-tuned weighted fusion successfully provided a unified mechanism for combining security evidence across different stages of the software development lifecycle; however, it did not demonstrate a statistically significant performance advantage over the evaluated standalone runtime anomaly detectors after Holm correction. The supervised Logistic Stacking baseline achieved the highest mean F1-score and significantly outperformed the validation-tuned fusion, whereas no statistically significant differences were observed between the tuned fusion and AE, IF, LOF, or OCSVM. These findings indicate that combining additional evidence sources does not necessarily improve predictive performance and that evidence selection and weighting should instead be guided by validation results and operational requirements.
Accordingly, the principal contribution of this work is a leakage-aware and repeatable methodology for evaluating Shift-Left/Shift-Right security evidence integration within DevSecOps pipelines. The results support the architectural value of connecting pre-deployment security findings with runtime telemetry while also demonstrating the importance of distinguishing architectural integration benefits from improvements in predictive performance. Because the experiments were conducted using controlled synthetic data, the reported performance should be interpreted as evidence of methodological feasibility and internal experimental consistency rather than as an estimate of expected performance in production environments.
Future work should validate the proposed framework using real outputs from DevSecOps security tools, including SAST, SCA, IaC analysis, DAST, SBOM analysis, secret detection, and container scanning, together with runtime telemetry collected from containerized and Kubernetes-based environments. Evaluation on public cybersecurity benchmark datasets and long-term production workloads should also be conducted to assess external validity and performance under temporal distribution shifts.
Further studies should investigate leave-one-attack-family-out evaluation, temporal and rolling-window validation, concept-drift adaptation, cross-service generalization, score calibration, explainability, and robustness to adversarial conditions. Operational evaluation should additionally consider detection latency, false alarms per hour, system throughput, memory and computational overhead, CI/CD pipeline delay, model-update cost, and deployment overhead. Incorporating these criteria into a multi-objective evaluation framework would provide a more comprehensive assessment of the practical utility of lifecycle security evidence integration and support the transition of the proposed proof-of-concept toward evaluation in operational DevSecOps environments.

ACKNOWLEDGMENT The Authors would like to thank Mustansiriyah University(https://uomustansiriyah.edu.iq) in Baghdad, Iraq, for its support in the present work.

References

1. Akbar, M. A., Khan, A. A., Mahmood, S., & Hyrynsalmi, S. (2025). Management of DevSecOps process: An empirical investigation. Software: Practice and Experience, 55(7), 1234–1255. https://doi.org/10.1002/spe.3419

2. Akbar, M. A., Rafi, S., Hyrynsalmi, S., & Khan, A. A. (2024). Towards people maturity for secure development and operations: A vision. In Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (pp. 528–533). ACM. https://doi.org/10.1145/3661167.3661238
3. Alawneh, M., & Abbadi, I. M. (2022). Expanding DevSecOps practices and clarifying the concepts within Kubernetes ecosystem. In 2022 Ninth International Conference on Software Defined Systems (SDS) (pp. 1–7). IEEE. https://doi.org/10.1109/SDS57574.2022.10062874
4. Ashenden, D., & Ollis, G. (2020). Putting the Sec in DevSecOps: Using social practice theory to improve secure software development. In Proceedings of the New Security Paradigms Workshop 2020 (pp. 34–44). ACM. https://doi.org/10.1145/3442167.3442178
5. Azad, N., & Hyrynsalmi, S. (2024). Multivocal literature review on DevOps critical success factors. In Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (pp. 520–527). ACM. https://doi.org/10.1145/3661167.3661236
6. Billawa, P., Tukaram, A. B., Díaz Ferreyra, N. E., Steghöfer, J.-P., Scandariato, R., & Simhandl, G. (2022). SoK: Security of microservice applications: A practitioners' perspective on challenges and best practices. In Proceedings of the 17th International Conference on Availability, Reliability and Security (pp. 1–10). ACM. https://doi.org/10.1145/3538969.3538986
7. Cankar, M., Petrović, N., Pita Costa, J., Černivec, A., Antić, J., Martinčič, T., & Štepec, D. (2023). Security in DevSecOps: Applying tools and machine learning to verification and monitoring steps. In Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (pp. 201–205). ACM. https://doi.org/10.1145/3578245.3584943
8. Cheenepalli, J., Hastings, J. D., Ahmed, K. M., & Fenner, C. R. (2025). Advancing DevSecOps in SMEs: Challenges and best practices for secure CI/CD pipelines. In 2025 13th International Symposium on Digital Forensics and Security (ISDFS) (pp. 1–6). IEEE. https://doi.org/10.1109/ISDFS65363.2025.11011960
9. Feio, C., Santos, N., Escravana, N., & Pacheco, B. (2024). An empirical study of DevSecOps focused on continuous security testing. In 2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) (pp. 610–617). IEEE. https://doi.org/10.1109/EuroSPW61312.2024.00074
10. Haverinen, H., Janhunen, T., Päivärinta, T., Lempinen, S., Kaartinen, S., & Merilä, S. (2024). Automating cybersecurity compliance in DevSecOps with open information model for security as code. In Proceedings of the 4th Eclipse Security, AI, Architecture and Modelling Conference on Data Space (pp. 93–102). ACM. https://doi.org/10.1145/3685651.3686700
11. Hermann, K., Peldszus, S., Steghöfer, J.-P., & Berger, T. (2025). An exploratory study on the engineering of security features. In 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE) (pp. 2470–2482). IEEE. https://doi.org/10.1109/ICSE55347.2025.00184
12. Kwon, S., Son, W., & Lee, J.-H. (2025). Anomaly detection in containerized tactical systems using temporal graph neural networks. In MILCOM 2025—IEEE Military Communications Conference (pp. 1566–1571). IEEE. https://doi.org/10.1109/MILCOM64451.2025.11310523
13. Lazarus, J. I., Truett, L., Fischer, B., & Kershner, C. (2024). DevSecOps process assessment collaboration tool: A novel method to inject R&M into Agile development. In 2024 Annual Reliability and Maintainability Symposium (RAMS) (pp. 1–5). IEEE. https://doi.org/10.1109/RAMS51492.2024.10457824
14. Le-Thanh, P., Le-Anh, T., & Le-Trung, Q. (2023). Research and development of a smart solution for runtime web application self-protection. In Proceedings of the 12th International Symposium on Information and Communication Technology (pp. 304–311). ACM. https://doi.org/10.1145/3628797.3628901
15. Martínez-Moreno, P., Vergara-Camacho, J. A., Luévano-Mondragon, V. A., & Jiménez-Martínez, K. A. (2026). DevSecOps for secure software development: A systematic literature review of practices, benefits, and adoption barriers. International Journal of Combinatorial Optimization Problems and Informatics, 17(4), 25–40. https://doi.org/10.61467/2007.1558.2026.v17i4.1299
16. Morales, J. A., & Yasar, H. (2023). Experiences with secure pipelines in highly regulated environments. In Proceedings of the 18th International Conference on Availability, Reliability and Security (Article 57, pp. 1–9). ACM. https://doi.org/10.1145/3600160.3605466
17. Moyón, F., Angermeir, F., & Mendez, D. (2024). Industrial challenges in secure continuous development. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (pp. 309–311). ACM. https://doi.org/10.1145/3639477.3639736
18. Nagasundari, S., Manja, P., Mathur, P., & Honnavalli, P. B. (2025). Extensive review of threat models for DevSecOps. IEEE Access, 13, 45252–45271. https://doi.org/10.1109/ACCESS.2025.3547932
19. Nugraha, A. D. P., & Irsan, M. (2025). Integrating self-protection into DevSecOps framework for defense against threat. In 2025 International Conference on Software Engineering and Computer Systems (ICSECS) (pp. 287–291). IEEE. https://doi.org/10.1109/ICSECS65227.2025.11279258

20. Patel, A., Laudya, R., Ragothaman, H., Udayakumar, S. K., Pandey, P., & Sheth, A. (2025). Dynamic secret injection for microservices in the cloud. In 2025 8th International Conference on Information and Computer Technologies (ICICT) (pp. 72–79). IEEE. https://doi.org/10.1109/ICICT64582.2025.00018
21. Pecka, N., Ben Othmane, L., & Valani, A. (2022). Privilege escalation attack scenarios on the DevOps pipeline within a Kubernetes environment. In Proceedings of the International Conference on Software and System Processes and International Conference on Global Software Engineering (pp. 45–49). ACM. https://doi.org/10.1145/3529320.3529325
22. Putra, A. M., & Kabetta, H. (2022). Implementation of DevSecOps by integrating static and dynamic security testing in CI/CD pipelines. In 2022 IEEE International Conference of Computer Science and Information Technology (ICOSNIKOM) (pp. 1–6). IEEE. https://doi.org/10.1109/ICOSNIKOM56551.2022.10034883
23. Rajapakse, R. N., Zahedi, M., & Babar, M. A. (2021). An empirical analysis of practitioners' perspectives on security tool integration into DevOps. In Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 1–12). ACM. https://doi.org/10.1145/3475716.3475776
24. Scanlon, T., & Morales, J. (2022). Revelations from an Agile and DevSecOps transformation in a large organization: An experiential case study. In Proceedings of the International Conference on Software and System Processes and International Conference on Global Software Engineering (pp. 77–81). ACM. https://doi.org/10.1145/3529320.3529329
25. Sinan, M., Shahin, M., & Gondal, I. (2025). Integrating security controls in DevSecOps: Challenges, solutions, and future research directions. Journal of Software: Evolution and Process, 37(6), Article e70029. https://doi.org/10.1002/smr.70029
26. Solaimalai, G. (2025). AI-augmented DevSecOps for cloud-native security automation. In 2025 International Conference on Recent Innovation in Science Engineering and Technology (ICRISET) (pp. 1–8). IEEE. https://doi.org/10.1109/ICRISET64803.2025.11252281
27. Turner, R. (2021). Systems engineering and DevSecOps: Reviewing the principles. INSIGHT, 24(2), 38–43. https://doi.org/10.1002/inst.12339
28. Verderame, L., Caviglione, L., Carbone, R., & Merlo, A. (2023). SecCo: Automated services to secure containers in the DevOps paradigm. In Proceedings of the 2023 International Conference on Research in Adaptive and Convergent Systems (Article 10, pp. 1–6). ACM. https://doi.org/10.1145/3599957.3606222
29. Yasar, H., Morales, J., Antunes, L., Earl, P., Edman, R., Hamed, J., Reynolds, D., Maffey, K. R., & Yankel, J. (2024). Insights on implementing a metrics baseline for post-deployment AI container monitoring. In Proceedings of the 2024 International Conference on Software and Systems Processes (pp. 46–55). ACM. https://doi.org/10.1145/3666015.3666018
30. Yu, W., Qian, J., Xu, R., Jin, C., Fang, H., & Shi, X. (2024). Improving substation network security with DevSecOps and AIOps. In 2024 IEEE 10th Conference on Big Data Security on Cloud (BigDataSecurity) (pp. 113–118). IEEE. https://doi.org/10.1109/BigDataSecurity62737.2024.00027
31. Yulianto, S., & Ngo, G. N. C. (2024). Enhancing DevSecOps pipelines with AI-driven threat detection and response. In 2024 International Conference on ICT for Smart Society (ICISS) (pp. 1–8). IEEE. https://doi.org/10.1109/ICISS62896.2024.10751269
32. Zhou, X., Mao, R., Zhang, H., Dai, Q., Huang, H., Shen, H., Li, J., & Rong, G. (2023). Revisit security in the era of DevOps: An evidence-based inquiry into DevSecOps industry. IET Software, 17(4), 435–454. https://doi.org/10.1049/sfw2.12132