Journal of Informatics Unisla (JOUTICA)
P-ISSN: 2503-071X
E-ISSN: 2621-511X

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

CI/CD Pipeline Execution Time Optimization Using Code Churn-Based Selective-SAST on Jamstack Architecture

Received : Juni 07, 2026; Revised : July 20, 2026; Accepted : Okt 09, 2026; Published : Okt 12, 2026

Fadilah Tun Hazimah[1], Muhammad Faisal[2*], Emil Agusalim Habi Talib[3], Titik Khawa Abd Rahman[4]

[1] Informatika, Fakultas Teknik, Universitas Muhammdiyah Makassar,
Jl. Sultan Alauddin No. 259, Makassar, 90221, INDONESIA
105841118323@student.unismuh.ac.id

[2] Informatika, Fakultas Teknik, Universitas Muhammdiyah Makassar,
Jl. Sultan Alauddin No. 259, Makassar, 90221, INDONESIA
muhfaisal@unismuh.ac.id

[3] Informatika, Fakultas Teknik, Universitas Muhammdiyah Makassar,
Jl. Sultan Alauddin No. 259, Makassar, 90221, INDONESIA
emil@unismuh.ac.id

[4] School of Science and technology, Asia e University,
Wisma Subang Jaya, Jalan SS 15/4, Subang Jaya, Selangor, 47500, Malaysia
titik.khawa@aeu.edu.my

*Corresponding Author

Abstract:
Full execution of Static Application Security Testing (SAST) on every commit in a modern CI/CD pipeline creates significant bottlenecks, especially in Jamstack architecture characterized by high commit frequency and fragmented code changes. This study evaluates the effectiveness of the Selective-SAST approach based on code churn analysis as a strategy to optimize pipeline execution time. The proposed approach utilizes four indicators of code change risk, namely relative churn above the 75th percentile, developer entropy above 1.5, intersection with sensitive paths (authentication, API, configuration, CI/CD), and days since last modification above 14 days, to determine the scanning coverage. The experiment was conducted using a paired comparison design on 49 open-source Jamstack repositories, 2,450 unique commits, and 120,472 lines of data resulting from the combination of churn metrics and Semgrep findings. Empirical results show that Selective-SAST provides a median time saving of 15.21% (95% CI bootstrap 12.88%-17.28%; $Wilcoxon\ p < 0.001$; rank-biserial $r = 0.947$) while also demonstrating promising security retention potential (precision 1.00, recall 0.9412, and F1-score 0.9697). However, because the basis of SAST findings in the sub-sample is very sparse (only 17 baseline findings from 7 commits), these retention metrics should be interpreted as preliminary evidence. This study concludes that Selective-SAST shows promise in accelerating security feedback on the Jamstack CI/CD pipeline, but absolute validation requires testing on datasets with higher vulnerability density.

Keywords:
CI/CD pipeline, Selective-SAST, Code Churn, Jamstack, DevSecOps

This work is licensed under a *Creative Commons Attribution-ShareAlike 4.0 International License*. Allows readers to read, download, copy, distribute, print, search, or link to the full texts of its articles and allow readers to use them for any other lawful purpose.

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

1. Introduction

Continuous Integration and Continuous Deployment (CI/CD) are the basic pillars of modern software engineering for automating the development cycle [1], notably in enabling the current iterative development approaches that demand rapid and adaptive product releases [2]. The Jamstack architecture, which uses a decoupled architecture [3] and permits each component to be updated separately with a high commit frequency, makes the applicability of this technique more and more clear. In order to identify vulnerabilities as soon as feasible, the DevSecOps paradigm mainly depends on real-time monitoring [4] and Static Application Security Testing (SAST) [5], [6].

The meticulous execution of SAST on each commit really causes a bottleneck, despite its importance. Without contextual optimization, pipeline stages are susceptible to become delay sites, as demonstrated by empirical investigations [7], [8]. This thorough search also often produces false positives [9] amplified by architectural odors [10]. Small interface modifications shouldn't cause a full backend scan in Jamstack. The traditional continuous security strategy [11], [12] is still unable to correctly identify that structural context, in contrast to the traditional testing paradigm, which is hindered in automation by depending on manual validation at the conclusion of the development cycle [13].

Several earlier research have sought to lower that lag thru process efficiency. The efficacy of optimization through the selection of impacted code has been demonstrated by the HybridCISave technique [14] and the build skipping idea [15]. Historically, the use of code churn in Just-in-Time Software Defect Prediction (JIT-SDP) [16] and entropy-based weighting [17] has proven to be quite useful in forecasting abnormalities. This analytical method has considerable potential to be further adapted to increase the overall efficiency of the quality assurance process. Based on that assessment, four key holes underpinning this research are clear. First, pipeline optimization is still focussed on build and test without including the modification of SAST scanning. Second, the field of security optimization has not particularly benefited from code churn's success. Third, there hasn't been enough research done on integrating adaptive testing procedures [18] with security execution in CI/CD. Lastly, past research often overlooks the execution granularity requirement, which are an absolute property of the Jamstack architecture.

To tackle the highlighted problems, this research presents a Selective-SAST framework that use code churn analysis to contextually reduce the scanning scope. The evaluation of this framework is done through two basic problem formulations. RQ1 analyzes how much the application of this strategy can lower the pipeline execution time in the Jamstack ecosystem when compared to the Full-SAST baseline. Afterwards, RQ2 investigates the effect of this optimization on the quality of security finding retention in terms of precision, recall and F1-score.

The study has three main contributions: (1) the design of a decision model that incorporates relative churn, developer entropy, path sensitivity and staleness; (2) the measurement of the efficiency of scanning time, with clear documentation of the dataset limitations; and (3) the proposal of a scan prioritization mechanism, which is proposed as a complement, not a replacement, of comprehensive security scanning in the DevSecOps ecosystem.

2. Method

The study uses a quantitative experimental method, namely paired comparison, to assess the efficacy of the Selective-SAST strategy on the execution time and security retention in the CI/CD pipeline of Jamstack repositories via code churn analysis. The research stages are composed of six main steps, which are (1) formulating the research framework and scope, (2) selecting and acquiring open-source repository datasets, (3) mining commits and calculating code churn metrics, (4) designing the Full-SAST

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

mechanism as the ground truth and Selective-SAST as the proposed approach, (5) designing a paired comparison experiment with execution time and security retention evaluation metrics, and (6) measurement validation and statistical testing to confirm result reproducibility.

2.1. Research Framework

The research framework is to measure the effectiveness of the Selective-SAST approach based on the analysis of code churn to reduce the execution time of the CI/CD pipeline on the Jamstack architecture. This research considers the CI/CD pipeline as a system of interrelated structures, processes, tools, and stakeholders following the architectural framework of the pipeline formalized by Amou Najafabadi et al. [19]. Therefore, any optimization intervention has to be evaluated in the context of the whole system and not only a specific component. From this perspective, the research flow starts with defining methodological boundaries and scope, which means selecting open-source repositories based on Jamstack architecture and having static build artifacts, using static site generators (SSGs) or related frameworks, and meeting the selection criteria. Once the repository is selected, we clone it and snapshot it using a pinned SHA to ensure that the data is reproducible. Then, the commit history is mined with PyDriller to get commit metadata and file changes which are used as the basis to calculate the code churn statistic. The general flow of the research is illustrated in Figure 1.



Figure 1: Code Churn Analysis-based Selective-SAST research framework

After the code churn metrics are collected, the research conducts the two comparing scenarios: the ground truth (Full-SAST) and the optimization strategy (Selective-SAST). Full-SAST scans the complete code coverage on specified commits and Selective-SAST scans only the files that satisfy the selection predicates, such as high churn rate, developer entropy, modifications on sensitive pathways, and file

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

staleness level. We performed the evaluation on the same commits using a paired comparison approach to allow a better comparison of T_full and T_selective in terms of execution time and security retention. The assessment features are related to two major aims, i.e. to measure execution time efficiency and to test security retention through metrics of precision, recall and F1-score.

2.2. Datasets and Repository Selection Criteria

The dataset was assembled from open-source GitHub repositories representing Jamstack and static-site-generator (SSG) projects, drawing on the Next.js, Gatsby, Hugo, Astro, and Eleventy ecosystems. Framework usage alone was not treated as sufficient evidence of a Jamstack architecture; instead, a repository was classified as such only when it satisfied two binding operational indicators within the codebase: (1) the configuration files exhibited clear architectural decoupling between the front-end interface and the backend API, and (2) the target CI/CD pipeline contained an explicit static-site-generation instruction (for example, a next build command that outputs static HTML/CSS/JS assets). Enforcing both conditions was essential to exclude repositories that, despite relying on an SSG-capable framework, still functioned as monolithic applications built around conventional server-side rendering (SSR). Eligible candidates then moved through four sequential stages: discovery, filtering, cloning, and local validation. During discovery, repositories were required to be non-forked, publicly licensed, at least twelve months old, written in a relevant programming language, and above a minimum development-activity threshold. Local validation imposed stricter conditions, requiring a minimum of 200 commits, at least three distinct contributors, and the presence of CI/CD artifacts such as .github/workflows, .gitlab-ci.yml, netlify.toml, or .circleci/config.yml.

Sampling was done using stratified random sampling using the SSG ecosystem as the stratification variable to preserve the diversity of build characteristics and directory structure conventions among ecosystems. The final dataset consisted of 49 of the original 50 repositories, since one was eliminated owing to not matching the consistent processing criteria.

At the file level, the study ignores lockfiles, produced assets, vendor directories and other non-code artifacts. These exclusions are critical to guarantee that code churn measurements and SAST evaluation are not skewed by automated files or third-party dependencies. The data of the selection criteria are shown in Table 1.

Table 1: Dataset Selection Criteria

Components	Criteria
Data source	Open-source GitHub repository
Project domains	Jamstack projects or static site generators based on modern ecosystems such as Next.js, Gatsby, Hugo, Eleventy and Astro,
Inclusion criteria	The repository is active, non-forked, has a license, is at least 12 months old, has an adequate commit history, has a minimum of 200 commits, involves a minimum of three contributors, has CI/CD artifacts, and uses the relevant primary language
Exclusion criteria	Fork repositories, no licenses, not meeting development activity limits, no CI/CD artifacts, failed to process or incompatible at the local checkout/analysis stage, as well as vendor, generated, and lockfile files
Final Sum	49 repositories

2.3. Data Acquisition and Repository Snapshots

Data acquisition proceeded in two stages: repository discovery and local snapshot retrieval. During discovery, non-forked GitHub repositories within the Jamstack/SSG ecosystem were screened against two threshold-based indicators a minimum of 100 stars for popularity and at least 200 active commits

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

for development activity. Repository sampling was then conducted through stratified random sampling. The candidate population was partitioned into strata corresponding to the target framework ecosystems, namely Next.js, Gatsby, Hugo, Astro, and Eleventy, with the number of repositories drawn from each stratum allocated proportionally to that framework's actual prevalence on GitHub. Repositories were subsequently selected at random within each stratum, a procedure intended to yield an unbiased and representative sample of the overall population.

During the cloning phase the candidate is cloned with the --no-single-branch parameter to save the branch history for commit mining. Additional checks are performed to ensure the quantity of commits, contributors and CI/CD artifacts are sufficient. To make the tests reproducible, the repository version is locked by a pinned SHA from HEAD position and a SHA-256 hash. This acquisition process resulted in 49 valid repositories with 2450 unique commits and 120,472 lines of combined churn metrics and SAST data.

2.4. Mining Commit and Code Churn Calculation

The PyDriller tool is used for extracting the changes at the commit and file level in the specified repository. All the related attributes are extracted exhaustively using the framework of Nyamawe [20] for the usage of DevOps metadata for pipeline purposes. The dataset contains the hash identities (commit and parent), timestamps, encrypted (hashed) authors, merge/revert, file path, categories of modification, number of affected lines, and nloc_before.

In order to do the analysis, non-source code items like lock-files, vendor directories, created artifacts, and build results are excluded from the mining step. Then, each updated file is classified by specific annotations for Configuration, CI/CD, Authentication and API, which are the basic classification parameters for the Selective-SAST evaluation stage.

The final step of this process is applying the code churn measure to evaluate the intensity of the alterations. The absolute churn value is used as the key parameter in the mathematical calculation model (Equation 1) of code volatility to sum up the total lines added and removed in any single update.

$$Absolute\ Churn_i = Lines\ Added_i + Lines\ Deleted_i \quad (1)$$

Relative churn normalizes the magnitude of change by the size of the file before the change. The relative churn of the dataset is calculated as the sum of churn across a 30 days window divided by the value of nloc_before. Where nloc_before values of 0 are handled as 1 to avoid division by zero (Equation 2).

$$Relative\ Churn_{i,30d} = \frac{Churn_{i,30d}}{max(NLOC\ Before_i, 1} \quad (2)$$

Developer entropy is computed over a 90-day period to indicate how distributed is the contribution of modifications to a file among different developers, where pj is the proportion of contribution by developer j (Equation 3).

$$H = -\sum_{j=1}^{n} p_j \log_2(p_j) \quad (3)$$

The priorities of the Selective-SAST scan are based on parameters like absolute churn, relative churn, change frequency, developer entropy and file change patterns. This assures that the target selection is based on objectively measurable code properties only and not on arbitrary values.

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

2.5. Full-SAST and Selective-SAST Mechanisms

Experimental evaluation relied on two scenarios. The first scenario, Full-SAST powered by Semgrep, served as the reference baseline for measuring computational time overhead and mapping the pool of reference findings. Built on the SafeOps+ platform proposed by Achbarou et al. [21] and the multi-layer principles introduced by Wahid et al. [22], this mode represented the standard for comprehensive scanning. It should be emphasized that the evaluation metrics reported in this study, namely precision, recall, and F1-score, reflect the degree of retention relative to this Full-SAST baseline rather than an absolute measure of vulnerability-detection accuracy.

The second scenario, Selective-SAST, adopts the CAPRA method of Tang et al. [23] to limit scanning to dangerous files, but not as a replacement for Full-SAST. File priority is applied when relative churn is above the 75th percentile, developer entropy is above 1.5, it hits a critical path or the modification gap is above 14 days. All these choice rules including application of Windows and POSIX path normalization are summarized in Table 2.

Table 2: Selective-SAST Decision Rules

Conditions	Verdict
Relative churn > P75 on related repositories	Prioritized for scans
Developer entropy > 1.5	Prioritized for scans
Touch the authentication path, API, configuration, or CI/CD	Prioritized for scans
Days since last modification > 14 days	Prioritized for scans
Doesn't meet all trigger conditions	Not prioritized for scans

The four decision thresholds underlying Selective-SAST were formulated from a combination of empirical data distribution and prior collaboration-focused literature. The relative-churn parameter was fixed at the 75th percentile to capture the upper quartile of modification intensity across repositories. Consistent with evidence linking higher defect risk to multi-contributor code changes [16], [17], a developer-entropy threshold above 1.5 was adopted because, mathematically, this value corresponds to a balanced contribution from at least three distinct developers ($-\Sigma\ p_i\ log_2\ p_i\ \approx\ 1{,}58$), a condition prone to triggering structural miscommunication. The 14-day staleness threshold, by contrast, was not chosen from an arbitrary percentile; it was calibrated empirically to align with the standard two-week duration of a single Agile Sprint iteration. Within the context of modern continuous integration and delivery, a code file that remains unmodified across one full iteration cycle is classified as stale, since the likelihood of new vulnerabilities being introduced into such inactive areas approaches zero. This policy prevents the pipeline from expending computational resources on rescanning assets that were already validated during the preceding Sprint.

2.6. Experimental Design, Evaluation Metrics, and Measurement Validation

The experimental design follows a paired comparison strategy: the same commits are assessed using Full-SAST as the ground truth and Selective-SAST as the optimization approach. With this design, we are able to compare the time of executing Full-SAST (T_{full}) with Selective-SAST ($T_{selective}$) directly so as to remove the bias caused by the fluctuations of commits. The testing is targeting two primary metrics: (1) evaluation of the decrease of the execution time (time saving), and (2) measurement of the security discoveries retention by the precision, recall, and F1-score metrics against the performance of Full-SAST.

The Measurement validity was preserved through three replications performed on a 10% stratified random subsample, following the guidance of Hrusto et al. [24] on controlling systemic variance in DevOps settings. Of the 245 initial commits, 20 were excluded because the Selective-SAST algorithm

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

bypassed scanning entirely, flagging zero files for those commits. This exclusion was a deliberate, conservative step taken to avoid introducing positive skew into the evaluation metrics. Including commits with a near-100% time saving in the calculation would have artificially inflated the median efficiency figure. Consequently, the final sample of 225 paired commits reflects a worst-case scenario in which selective scanning still had to be executed, yielding a considerably more objective basis for the study's efficiency claims.

Time efficiency is evaluated using the time-saving metric, which is the percentage reduction in execution time from Full-SAST to Selective-SAST (Equation 4).

$$H = -\sum_{j=1}^{n} p_j \log_2(p_j) \tag{4}$$

Security retention is evaluated using precision, recall, and F1-score. In this study, true positive (TP) is the Selective-SAST results also found in Full-SAST, false positive (FP) is the additional discoveries not present in the reference, and false negative (FN) is the findings in Full-SAST missed by Selective-SAST (Equations 5–7).

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{7}$$

In the data analysis, descriptive statistics were used based on the median, the interquartile range (IQR), and the P5 and P95 percentiles. We used the Wilcoxon signed-rank test to examine the comparison between Full-SAST and Selective-SAST because the distribution of execution time is asymmetric. The uncertainty range in time saving is evaluated using a 95\% bootstrap confidence interval and the coefficient of variation (CV) is calculated to validate the stability of wall-clock time across replications.

3. Results and Discussion

3.1. Description of Codeset and Code Churn Distribution

The final dataset consists of 49 Jamstack/SSG repositories, which represent 2,450 unique commits in 120,472 lines of data at the file level modification. To boost the accuracy of regex identification, we performed path normalization (POSIX vs Windows) before analysis. The normalization worked well to re-enable the identification of relevant pattern flags: touches_api was found in 5579 lines, touches_ci in 1312 lines. However, the detection of touches_auth was reduced to 3,907 lines. The selection logic was executed with little danger of false positives. The dataset is summarized in Table 3.

Table 3: Final Dataset Summary

Components	Value	Remarks
Number of final repositories	49	Repositories that pass the local selection and validation criteria
Number of unique commits	2.450	Commit mining results using PyDriller

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

Number of rows of the final dataset	120.472	A combination of file changes, churn metrics, pattern flags, and SAST findings
Churn metrics	8 metrics	Includes absolute churn, relative churn, churn_30d, frequency_30d, entropy_90d, days_since_last_mod, total_devs, and primary_dev_ratio
Pattern flags	4 boolean	touches_config, touches_ci, touches_api, and touches_auth
SAST Findings	105.933	Semgrep scan results on experimental datasets
Unit of analysis	File-level change	Each row represents one file change in a single commit

Distribution of absolute and relative churn are calculated and have a very right-skewed distribution with median change of only 5 rows. This property is supported by the developer entropy metric over 90 days with a median of 0, suggesting the dominance of one developer. Genuine multi-developer collaboration, which would raise entropy, occurs only infrequently; this is precisely why the entropy threshold above 1.5 is used to flag these less common but higher-risk files. Operationally, these situations would necessitate the Selective-SAST selection process to rely more on a combination of relative churn, staleness and sensitive path identification factors.

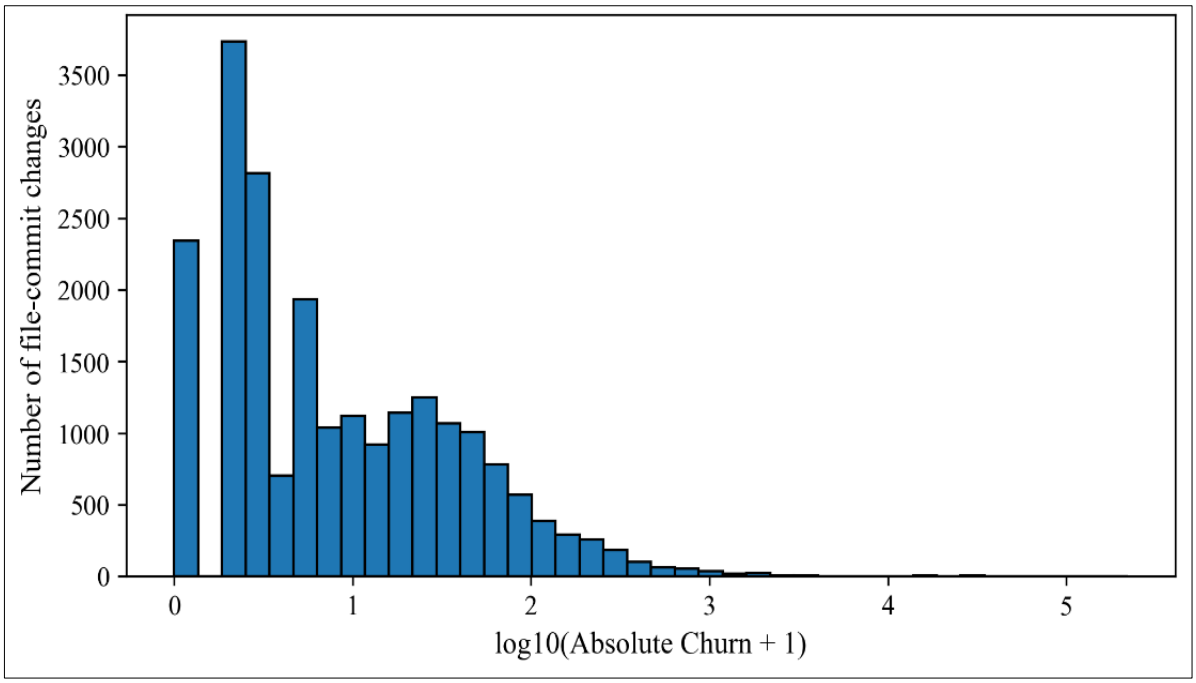


Figure 2: Absolute churn distribution (histogram, logarithmic scale)

The selection predicate was empirically validated through Spearman's Rank Correlation test applied to the full set of observations (N = 120,472). A linear analysis across the complete population revealed a small negative correlation for the two line-count-based volatility metrics, namely absolute churn ($\rho = -0.067$, $p < 0.001$) and relative churn ($\rho = -0.100$, $p < 0.001$). This mathematical anomaly in the churn metrics can be attributed to the large volume of static or boilerplate files within the Jamstack ecosystem, files that are modified frequently yet inherently carry little risk-bearing logic. Developer entropy, by contrast, showed a positive linear correlation ($\rho = 0.114$, $p < 0.001$), supporting the hypothesis that files edited concurrently by multiple developers carry a higher likelihood of harboring vulnerabilities.

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

These results indicate that the relationship between code-modification volatility and vulnerability risk within Jamstack architectures is non-linear and categorical rather than simply linear. The analytical approach was therefore shifted toward a conditional probability distribution instead. Operationally, the term "vulnerability probability" in this study is defined as the empirical conditional likelihood that a given file contains a SAST-detected vulnerability, given that the file falls within a particular volatility category.

Table 4: Probability of Vulnerability Findings by *Code Churn*

Volatility Category	Percentile range	Characteristics of Absolute Churn	Characteristics of Developer Entropy	Distribution of SAST Findings (%)	Vulnerability Probability
Static / Low	< P50	1 – 5 rows	0 (One developer)	3,1%	Very Low
Medium	< P50 – P75	6 – 23 lines	0 – 1	11,6%	Low
High (Selection Limit)	P75 – P95	24 – 130 lines	≥ 1	32,8%	Medium - High
Extreme	> P95	> 130 rows	≥ 2 (Multiple developers)	52,5%	Very High

As detailed in Table 4, an extreme clustering pattern of vulnerabilities becomes evident. The percentages reported under the "SAST Finding Distribution" column represent a conditional probability, computed as: (Number of SAST Findings within a Specific Volatility Category / Total SAST Findings across the Full Population) × 100. It should be made explicit that the denominator of 105,933 reference findings used in this population-level analysis represents the cumulative total of raw alerts generated across 2,450 repeated Full-SAST executions, a unit of analysis distinct from the file-commit rows used in the Spearman correlation test discussed earlier. This inflated aggregate arises naturally because Full-SAST indiscriminately rescans every file in the repository at each commit iteration, repeatedly regenerating alerts for the same underlying files. The figure therefore operates on a different scope than the evaluation reported in Table 7, where retention metrics were computed on a considerably smaller sample (17 validated findings within a 10% commit subset) following commit-level deduplication. The macro-level mapping in Table 4 shows that the cumulative risk proportion within the static range (< P50) amounts to only 3.1%, whereas the majority of findings accumulate within the High (32.8%) and Extreme (52.5%) volatility categories. This empirical evidence of non-linear clustering lends strong support to the effectiveness of setting the cutoff threshold at P75 to eliminate this redundant scanning of raw alerts.

3.2. Comparison of Full-SAST and Selective-SAST Execution Time

The execution-time comparison employed a paired-comparison design on identical commit units across the Full-SAST and Selective-SAST modes, ensuring a proportionally valid assessment of scanning efficiency. Evaluation of 225 commit pairs demonstrated that the Selective-SAST approach consistently reduced computational time. As detailed in Table 5, this efficiency gain is reflected in a median time saving of 15.21%, an interquartile range spanning 8.99% to 27.84%, and a 95% bootstrap confidence interval of 12.88% to 17.28% (Wilcoxon $p < 0.001$; rank-biserial $r = 0.947$). These statistical results indicate that the proposed method delivers a moderate efficiency improvement under standard execution conditions.

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

Table 5: Comparison of Full-SAST and Selective-SAST Runtimes

Metrics	Full-SAST	Selective-SAST	Remarks
Median execution time	7.52 seconds	6.30 seconds	Median is calculated from execution time per commit
IQR execution time	6.90–9.02 seconds	6.18–6.49 seconds	Main distribution of data without the influence of outliers
P95 Execution time	15.26 seconds	7.54 seconds	Describe the upper limit of the majority of the runtime
Median time saving	—	15,21%	Reduced time relative to Full-SAST
IQR time saving	—	8,99%–27,84%	Time-saving variations between commits
95% CI bootstrap	—	12,88%–17,28%	Median estimated time saving range
Commit with negative saving	—	8 commits (3.56%)	Cases when Selective-SAST is not faster than Full-SAST

This temporal difference is substantial (Wilcoxon signed-rank test, $p < 0.001$, rank-biserial effect size = 0.947). This computational performance validates a significant and measurable amount of time optimization, according to the runtime production assessment parameters of Viszkok and Hegedűs [25]. However, the discovery of negative savings in 8 commits (3.56%) should be underlined as proof that this strategy is not faster in all cases. These abnormalities are typically caused by the enormous volume of flagged files, the continual initialization load of the scan, or the large size of the target file. At the size of major operating pipelines, this ~3.6% proportion is critical to infer that Selective-SAST acts as a consistent optimization technique, not an absolute time-saver in every commit case.

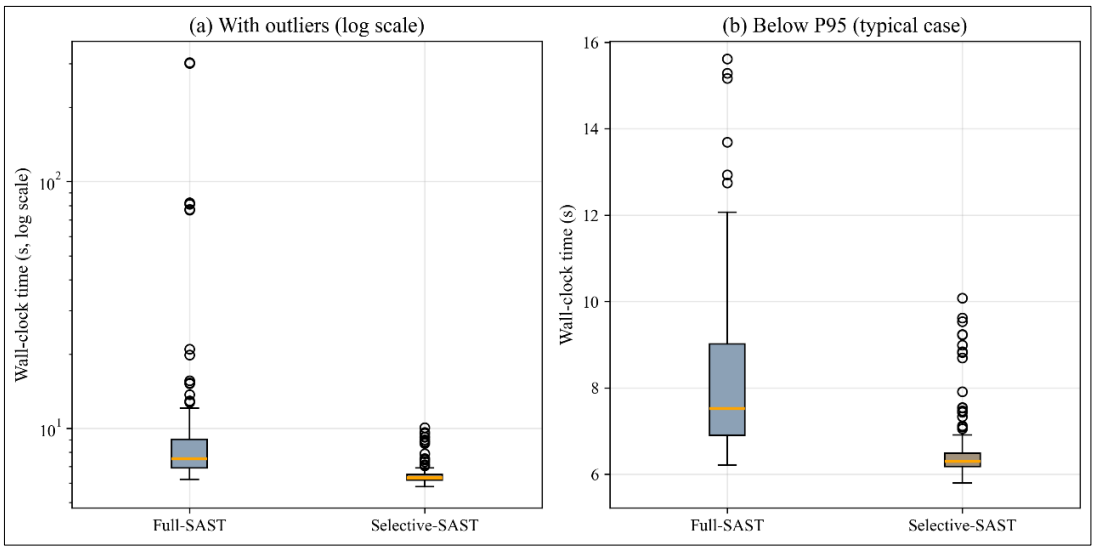


Figure 3: Comparison of Full-SAST and Selective-SAST execution times *(boxplot)*

3.3. Time Efficiency Analysis Based on Commit Characteristics

The effectiveness of Selective-SAST does not seem to be consistent for all commits, but depends on the properties of code changes, and the number of files selected for scanning. In general, Selective-SAST performs better when the number of files that match the selection predicate is smaller than the complete scan coverage, so Semgrep does not have to scan the entire repository. Table 6 displays the efficiency details by commit categories.

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

Table 6: Time Efficiency by Commit Category

Commit category	Median Full-SAST	Median Selective SAST	Time saving	Interpretation
1 Flagged file	7.38 seconds	6.28 seconds	12,75%	Efficiency emerged, but relatively moderate as the fixed cost of Semgrep initialization was still dominant
2–5 flagged files	7.73 seconds	6.28 seconds	18,72%	Efficiency is more pronounced because the scan coverage remains limited compared to a full scan
>5 Files Flagged	9.57 seconds	6.45 seconds	27,68%	Greater savings, especially when Full-SAST on related commit takes a higher time
Commit touches a critical path	8.13 seconds	6.35 seconds	19,67%	Stay efficient even if priority files cover risky areas such as APIs, authentication, configuration, or CI/CD
Commit without a critical path	7.46 seconds	6.29 seconds	13,26%	Efficiency remains, but lower as commit characteristics.

Based on Table 6, Selective-SAST works best when scanning is carefully limited to a selection of files. But a big number of flagged files does not automatically raise the causal efficiency. For the corresponding commit, the surge in the time difference is generally only caused by the high duration of the basic Full-SAST computation. The operational effectiveness of Selective-SAST needs to be examined due to the complicated relationship among scan coverage, commit characteristics, path sensitivity, and fixed load of the scanner.

The execution dynamics observed in the Mantinedev repository reveal a distinct computational anomaly. Although the repository itself is comparatively small in scale, totaling only 3.44 MB across 33 files, conventional Full-SAST execution became trapped in a pathological computation pattern that consumed as much as 1,380 seconds, a consequence of the scanning engine's inefficiency when analyzing complex static assets. The path-filtering mechanism within Selective-SAST, by contrast, was able to bypass these bottleneck files entirely, since they failed to meet the modification-volatility threshold, cutting the scanning time drastically down to just 11 seconds. This empirical finding confirms that the proposed algorithm does more than simply shorten queue duration; it also functions as an architectural fail-safe mechanism, guarding the CI/CD pipeline against extreme bottlenecks caused by anomalous behavior in the scanning tool itself.

3.4. The Impact of Selective-SAST on Security Retention

In addition to measuring time efficiency, evaluating security retention becomes crucial. In line with the concept of prioritizing high-risk commits by Kabadi et al. [26] the evaluation of 227 sub-sample commits resulted in 16 true positives, 1 false negative (FN), and 0 false positives. Referring to the FOBICS framework by Caniglia et al. [27], Selective-SAST maintains an excellent security retention rate (precision 1.00, recall 0.9412, F1-score 0.9697). An investigation into the 1 FN that was missed revealed a CWE-200 vulnerability (low severity), consisting of non-critical logging on stale files that naturally passed the filter threshold (< P75). This trimming is a very logical operational trade-off: the system sacrifices visibility on a minor vulnerability to massively reduce execution time, without sacrificing critical security integrity at all (Table 7).

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

Table 7: Impact of Selective-SAST on Security Retention

Metrics	Value	Interpretation
Total commit sub-sample	227	Commit for security retention evaluation
Commit with Full-SAST findings	7	Commit that has findings at full baseline
Commit with Selective-SAST findings	6	Commits that keep generating findings
Total baseline findings	17	Semgrep Findings in Full-SAST mode
True Positive (TP)	16	Full-SAST findings that remain detected by Selective-SAST
False Negative (FN)	1	Baseline findings not covered by Selective-SAST
False Positive (FP)	0	There were no additional findings beyond baseline
Precision	1.00	Each Selective-SAST finding is consistent with baseline
Recall	0.9412	16 of the 17 baseline findings were captured by Selective-SAST
F1 Score	0.9697	High safety retention with perfect precision

Table 7 presents the security evaluation metrics measured against the reference baseline mode. It should be emphasized that the findings within this experimental subset are extremely sparse, comprising only 17 reference findings drawn from 7 commits, or 3.08% of the total sub-sample. This scarcity of data introduces substantial sensitivity into the evaluation metrics; a single False Negative (FN) alone is enough to depress the recall score by approximately 5.88%. Accordingly, the achieved precision of 1.00, recall of 0.9412, and F1-score of 0.9697 should not be interpreted as absolute detection accuracy. Given that the number of findings remains relatively small and one vulnerability instance went undetected (FN = 1), these retention metrics must be strictly regarded as preliminary evidence. Further validation on repositories with a higher vulnerability density is essential to strengthen the generalizability of these findings.

3.5. Measurement Validation and Statistical Tests

Measurement validation was performed by three replications on 10% of the stratified random sub-sample with scan order randomization to suppress OS-level cache warming technical bias on runtime differences. The data exclusion stage sans flagged files resulted in 225 viable evaluation samples from the initial 245 commits. The wall-clock time stability across replications has been quite robust, as shown in Table 8, which has resulted in a low median coefficient of variation (CV) being acquired, i.e. 2.54% for the Full-SAST mode and 3.05% for the Selective-SAST technique.

Table 8: Validation of Runtime Measurements

Validation components	Value	Interpretation
Initial sub-samples	245 commits	10% stratified random sub-sample
Valid sub-samples	225 Commit	Commit valid after the exclusion of the commit without the flagged file
Replication	3 times per mode	Each commit is tested on both modes
Median CV Full-SAST	2.54%	Low measurement variation
Median CV Selective-SAST	3.05%	Low measurement variation
Median time saving	15.21%	Moderate and consistent savings
95% CI bootstrap	12.88%–17.28%	Estimated range of time savings
Wilcoxon Test	$p < 0.001$	Statistically significant runtime difference
Rank-biserial r	0.947	The effect of difference is very strong

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

3.6. Discussion, Implications, and Validity Threats

Experiments reveal that Selective-SAST can greatly reduce execution time with a median of 15.21% time saving at 95% CI of 12.88% to 17.28% based on Wilcoxon test $p < 0.001$ and rank-biserial $r = 0.947$. This approach, meantime, also effectively keeps security against Full-SAST with precision 1.00, recall 0.9412 and F1-score 0.9697. The frequent trade-off pattern in code-based scan pruning was effectively avoided, since the four selection predicates accurately captured most of the vulnerable files in the Jamstack dataset.

This optimization conceptually complements the full-stack security methodology proposed by Rey Rodriguez et al. [28] on frameworks such as Vue.js and Spring Boot. They focus on organizational secure-by-design, while Selective-SAST automates optimization at the CI/CD pipeline level. The computational save of 15.21% is moderate compared to the efficiency of the HybridCISave [14] approach for build/test selection. The disparity in these values is extremely sensible because the focus of the pruning is on the execution of SAST with an initial baseline duration that is already very short, with a median of Full-SAST at 7.52 seconds.

From a methodological point of view, the integration of SAST with Large Language Models (LLM) offers potential for further optimization. The CodeEnhancer tool from Lee et al. [29] also verifies the increase of code quality trough fine-tuning LLM, which is in line with the proof of enhancing vulnerability identification using static analysis by Ciavatta et al. [30]. This idea aligns with the implementation of semantic rule embedding in T4EDR by Nuzul et al. [31] to handle unstructured threats. These approaches are complementary to the Selective-SAST, where the churn-based selection reduces the area of computation, and the LLM integration enhances the scanning quality within the specific scopes.

There are also natural restrictions in the usage of code churn, Semgrep results, and wall-clock time regarding validity issues that do not always represent security problems in an absolute sense. To remove internal bias, we repeated the procedure three times on 10% of the stratified random sub-sample, randomizing the scan order, and obtained a median CV stability of 2.54% for Full-SAST and 3.05% for Selective-SAST. External restrictions are the population size of 49 Jamstack/SSG repositories. According to Steidl et al. [32], the population size does not allow an absolute generalization to alternative architectures due to pipeline variations. The sparse phenomena, with only 17 finds from 7 commits, placed the recall achievement at 0.9412 as just preliminary proof of investigation, however all reliability pathways were still rigorously checked through pinned SHA, SHA-256 hash, random seed, and dataset normalization.

An anomaly in the form of negative saving in 8 commits (3.56%), where the scanner startup overhead often exceeds the Full-SAST scanning time for very small code alterations. This shortcoming at the scale of industrial production requires the introduction of a fallback mechanism, for example a threshold on the absolute size of changes. If the number of lines of code altered is too small, the pipeline should run Full-SAST immediately instead of turning on the Selective-SAST evaluation filter, which slows down the first computation.

4. Conclusions and suggestions

This research demonstrates that Selective-SAST grounded in code churn analysis effectively optimizes Jamstack CI/CD pipelines by prioritizing high-risk targets. A comparative evaluation across 225 commit executions recorded a moderate efficiency gain, with a median time saving of 15.21%. Beyond this moderate baseline efficiency, the algorithm exhibited exceptional capability in an edge case involving a repository with a structurally complex static footprint, exemplified by the Mantinedev anomaly, where scanning time was reduced drastically from 1,380 seconds to just 11 seconds. This

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

finding indicates that the proposed method reaches its point of maximum effectiveness precisely when the pipeline is under its heaviest computational load, a consequence of pathological execution behavior.

From a quality standpoint, the method proved capable of preserving the majority of security findings relative to the baseline, as reflected in a precision of 1.00, a recall of 0.9412, and an F1-score of 0.9697. Because this conclusion rests on empirical evidence drawn from a relatively small number of findings (and includes one false negative), the reduction in execution time is more appropriately framed as a tactical, per-commit optimization mechanism intended to accelerate the developer feedback cycle, rather than as an absolute substitute for comprehensive scanning at the final merge stage.

However, given this sparse evaluation sub-sample with just 17 discoveries from 7 commits (out of a total of 227 commits), the generalization of this retention metric should be limited as a first indication that still needs validation on a denser dataset with a high vulnerability density. Thus, Selective-SAST is better characterized as an optimization and prioritizing solution rather than a full conversion of Full-SAST, which is still necessary at times and in high-risk scenarios.

Four directions for further research are recommended: (1) build an adaptive decision model that integrates code churn with historical vulnerability history, severity, and dependency risk; (2) experiment a hybrid scheduling strategy between Selective-SAST per commit and Full-SAST on merge to the main branch; (3) calibrate the selection thresholds (P75 churn, entropy 1.5, staleness 14 days) thru cross-validation on the hold-out set; and (4) extend the evaluation to non-Jamstack projects such as monolithic systems, microservices, and mobile to strengthen external validity.

Library

[1] G. Bermejo, Á. Macías, J. L. Herrera, S. Laso, and J. Berrocal, "Automating Docker image deployment across network-segmented environments," *SoftwareX*, vol. 32, Dec. 2025, doi: 10.1016/j.softx.2025.102414.

[2] I. D. S. Taufiq and Arda Gusema Susilowati, "Pengembangan Web Marketplace Catering Dengan Fitur Penjadwalan dan Menu menggunakan Metode Scrum," Joutica, vol. 11, no. 1, pp. 29–40, Apr. 2026, doi: 10.30736/jti.v11i1.1573

[3] P. Ataei and D. Staegemann, "Application of microservices patterns to big data systems," *J. Big Data*, vol. 10, no. 1, Dec. 2023, doi: 10.1186/s40537-023-00733-4.

[4] V. Casola, A. De Benedictis, C. Mazzocca, and V. Orbinato, "Secure software development and testing: A model-based methodology," *Comput. Secur.*, vol. 137, Feb. 2024, doi: 10.1016/j.cose.2023.103639.

[5] X. Zhao, T. Clear, and R. Lal, "Identifying the primary dimensions of DevSecOps: A multi-vocal literature review ☆," *J. Syst. Softw.*, vol. 214, p. 112063, 2024, doi: 10.5281/zenodo.7.

[6] L. Prates and R. Pereira, "DevSecOps practices and tools," *Int. J. Inf. Secur.*, vol. 24, no. 1, Feb. 2025, doi: 10.1007/s10207-024-00914-z.

[7] A. F. Nogueira and M. Zenha-Rela, "Process mining software engineering practices: A case study for deployment pipelines," *Inf. Softw. Technol.*, vol. 168, Apr. 2024, doi: 10.1016/j.infsof.2023.107392.

[8] S. J. Warnett, E. Ntentos, and U. Zdun, "MLOps pipeline generation for reinforcement learning: A low-code approach using large language models," *Journal of Systems and Software*, vol. 235, May 2026, doi: 10.1016/j.jss.2025.112760.

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

[9] F. Pagano, A. Romdhana, D. Caputo, L. Verderame, and A. Merlo, "SEBASTiAn: A static and extensible black-box application security testing tool for iOS and Android applications," *SoftwareX*, vol. 23, Jul. 2023, doi: 10.1016/j.softx.2023.101448.

[10] M. Bochicchio, D. Sas, A. Gilardi, and F. A. Fontana, "An empirical study on architectural smells through a pipeline for continuous technical debt assessment," *Inf. Softw. Technol.*, vol. 185, Sep. 2025, doi: 10.1016/j.infsof.2025.107783.

[11] Y. Mothanna, W. ElMedany, M. Hammad, R. Ksantini, and M. S. Sharif, "Adopting security practices in software development process: Security testing framework for sustainable smart cities," *Comput. Secur.*, vol. 144, Sep. 2024, doi: 10.1016/j.cose.2024.103985.

[12] P. Bellavista, N. Bicocchi, M. Fogli, C. Giannelli, M. Mamei, and M. Picone, "Exploiting microservices and serverless for Digital Twins in the cloud-to-edge continuum," *Future Generation Computer Systems*, vol. 157, pp. 275–287, Aug. 2024, doi: 10.1016/j.future.2024.03.052.

[13] Fitri Shafirawati, Rifqi Yovila Candra, Ikhwan Ningrattama, Nurul Aliyah, Tynton Prima Kusuma Wardhana, and Aditya Ramadhan, "Impelementasi Pengujian ALPHA dan BETA Testing pada Aplikasi Interpals," Joutica, vol. 9, no. 2, pp. 125–132, Sep. 2024, doi: 10.30736/informatika.v9i2.1279

[14] H. Xie, Z. Tan, and X. Lv, "HybridCISave: A Combined Build and Test Selection Approach in Continuous Integration," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Dec. 2024, pp. 349–354. doi: 10.1145/1122445.1122456.

[15] X. Jin and F. Servant, "Which builds are really safe to skip? Maximizing failure observation for build selection in continuous integration," *Journal of Systems and Software*, vol. 188, p. 111292, Jun. 2022, doi: 10.1016/J.JSS.2022.111292.

[16] Y. Zhao, K. Damevski, and H. Chen, "A Systematic Survey of Just-in-Time Software Defect Prediction," *ACM Comput. Surv.*, vol. 55, no. 10, Oct. 2023, doi: 10.1145/3567550.

[17] H. Dai, J. Xi, and H. L. Dai, "Improving effort-aware just-in-time defect prediction with weighted code churn and multi-objective slime mold algorithm," *Heliyon*, vol. 10, no. 18, Sep. 2024, doi: 10.1016/j.heliyon.2024.e37360.

[18] K. S. Rao, A. A. Rao, and P. R. Raju, "Design of an iterative adaptive method for volatility-aware test case prioritization in rapidly evolving software systems," *MethodsX*, vol. 15, Dec. 2025, doi: 10.1016/j.mex.2025.103582.

[19] F. Amou Najafabadi, J. Bogner, I. Gerostathopoulos, and P. Lago, "An architectural perspective on MLOps: Structures, processes, tools, and stakeholders," May 01, 2026, *Elsevier B.V.* doi: 10.1016/j.infsof.2026.108029.

[20] A. S. Nyamawe, "Mining commit messages to enhance software refactorings recommendation: A machine learning approach," *Machine Learning with Applications*, vol. 9, p. 100316, Sep. 2022, doi: 10.1016/j.mlwa.2022.100316.

[21] O. Achbarou, M. Lachgar, Y. El Badouri, M. Rhouddani, A. Tahalli, and R. El Dhimni, "SafeOps+: An integrated platform for automated security analysis and compliance in DevSecOps pipelines," *SoftwareX*, vol. 33, Feb. 2026, doi: 10.1016/j.softx.2025.102465.

Volume 11, No.2, Tahun 2026, Page. 241-256
DOI: 10.30736/jti.v11i2.1695
https://jurnalteknik.unisla.ac.id/index.php/informatika

[22] A. R. Wahid, D. Anggreani, M. A. M. Hayat, A. A. Rahman, and M. Faisal, "Stacking architecture-endpoint detection: a hybrid multi-layered architecture for endpoint threat detection," *International Journal of Advances in Applied Sciences*, vol. 14, no. 4, pp. 1263–1280, Dec. 2025, doi: 10.11591/ijaas.v14.i4.pp1263-1280.

[23] B. Tang, S. Zhang, F. Zhu, and A. Ye, "CAPRA: Context-Aware patch risk assessment for detecting immature vulnerability in open-source software," *Comput. Secur.*, vol. 157, Oct. 2025, doi: 10.1016/j.cose.2025.104540.

[24] A. Hrusto, E. Engström, and P. Runeson, "Towards optimization of anomaly detection in DevOps," *Inf. Softw. Technol.*, vol. 160, Aug. 2023, doi: 10.1016/j.infsof.2023.107241.

[25] T. Viszkok and P. Hegedűs, "Unified SAST benchmark: Compare AI-driven and traditional static analyzers," *SoftwareX*, vol. 31, Sep. 2025, doi: 10.1016/j.softx.2025.102336.

[26] V. Kabadi, X. B. D. Le, P. Thongtanunam, and C. Treude, "Just-in-time bug classifier: A step towards integrating Automated Program Repair in CI/CD pipelines," *Inf. Softw. Technol.*, vol. 195, Jul. 2026, doi: 10.1016/j.infsof.2026.108076.

[27] A. Caniglia, V. Dentamaro, S. Galantucci, and D. Impedovo, "FOBICS: Assessing project security level through a metrics framework that evaluates DevSecOps performance," *Inf. Softw. Technol.*, vol. 178, Feb. 2025, doi: 10.1016/j.infsof.2024.107605.

[28] K. S. Rey Rodriguez, J. D. Avellaneda Galindo, J. T. Juan, J. R. Bermejo Higuera, J. B. Higuera, and J. A. Sicilia Montalvo, "Secure Development Methodology for Full Stack Web Applications: Proof of the Methodology Applied to Vue.js, Spring Boot and MySQL," *Computers, Materials and Continua*, vol. 85, no. 1, pp. 1807–1858, 2025, doi: 10.32604/cmc.2025.067127.

[29] J. Lee, K. Mai, N. Ghate, T. Yagyu, R. Beuran, and Y. Tan, "CodeEnhancer: LLM-generated Python code enhancement through SAST integration and fine-tuning," *Knowl. Based. Syst.*, vol. 342, Jun. 2026, doi: 10.1016/j.knosys.2026.115925.

[30] J. A. S. Ciavatta, J. R. B. Higuera, J. B. Higuera, J. A. S. Montalvo, T. S. Riera, and J. P. Melero, "Integration of Large Language Models (LLMs) and Static Analysis for Improving the Efficacy of Security Vulnerability Detection in Source Code," *Computers, Materials and Continua*, vol. 86, no. 3, 2026, doi: 10.32604/cmc.2025.074566.

[31] M. D. A. Nuzul, D. Anggreani, Muhammad Faisal, Abd Rahman Wahid, and T. K. A. Rahman, "T4EDR: Hybrid Threat Detection Framework for EDR Based on Semantic Rule Embedding and Contextual Network Flow Analysis," *Jurnal Nasional Pendidikan Teknik Informatika (JANAPATI)*, vol. 15, no. 1, pp. 110–120, Mar. 2026, doi: 10.23887/janapati.v15i1.104059.

[32] M. Steidl, M. Felderer, and R. Ramler, "The pipeline for the continuous development of artificial intelligence models—Current state of research and practice," *Journal of Systems and Software*, vol. 199, May 2023, doi: 10.1016/j.jss.2023.111615.