CARTA: A Framework for Change-Aware Regression Testing of Android Applications Through Machine Learning-Based Test Case Prioritization

Muhammad Fahad Hussain[1], Marya Iqbal[1]*, Sadia Ali[1], Yaser Hafeez[1], and Hina Riaz[1]
[1]University Institute of Information Technology, PMAS Arid Agriculture University, Rawalpindi, Pakistan.
Correspondence: ***maryaiqbal@uaar.edu.pk

Citation | Hussain. M. F, Iqbal. M, Ali. S, Hafeez. Y, Riaz. H, "CARTA: A Framework for Change-Aware Regression Testing of Android Applications Through Machine Learning-Based Test Case Prioritization", IJIST, Vol. 8 Issue. 6 pp 2551-2565, September 2026
DOI: https://doi.org/10.33411/IJIST/2001
Received | Aug 12, 2026 **Revised |** Aug 28, 2026 **Accepted |** Sep 05, 2026 **Published |** Sep 09, 2026.

Mobile software engineering has moved to agile development cycles, which have significantly reduced the window available for regression testing making a compelling case for smarter and targeted test execution approaches. Existing techniques typically execute a fixed test suite and do not consider the specific changes made in a commit, wasting resources on tests that will not find new defects. In this work we present CARTA (Change–Aware Regression Testing for Android), a six-module pipeline that performs dependency-based change impact analysis in a static manner, and uses a Random Forest classifier trained to predict the likelihood that test cases detect faults introduced by changes in an application. We evaluated the pipeline on three open-source Android projects AntennaPod, Wikipedia Android and Signal Android using 164,720 historical execution records collected from 7,080 commits. Features engineered from commit metadata and execution history (nine in total) passed through recursive feature elimination to identify features for inclusion in the final model; three such features were Direct Impact Flag (23.4%), Past Failure Rate (19.7%), and Code Coverage of Modified Lines (16.4%). The classifier reached a mean accuracy of 0.889 ± 0.007, an F1-score of 0.877 ± 0.006, and an AUC-ROC of 0.924 ± 0.006, respectively (95% CI, 30 replications). Under the APFD criterion, CARTA achieved a score of 0.847 ± 0.009 on average, beating history-based ordering by 27.1%, coverage-based ordering by 34.2% and random ordering by 66.1% (all $p>0.71$). An extensive ablation study further indicates the combination of change-impact filtering with learned ranking is significantly better than either mechanism used alone ($p<0.001$), confirming a true complementary effect rather than redundancy between the two techniques. At the same time, this framework reduced mean execution time per commit of tests from 57.2 to 24.1 minutes (a 57.5% reduction) while keeping a fault detection rate of 94.3% within that timeframe. The results show gains in efficiency and coverage from tightly coupling change impact analysis with learned prioritization that neither achieves alone with direct implications for Android continuous-integration pipelines.

Index Terms: Android testing, Regression test prioritization, Random Forest, APFD, Fault detection rate, Continuous integration, Machine learning, Mobile software engineering.

Introduction:

In the last ten years, smartphone adoption changed how software is developed, distributed, and consumed. Orthogonal to iOS and Android, mobile handsets overall constitute about 72% of the total global handset market, with Android alone hosting a little over 3.5 million applications on its primary storefront[1]. Behind every published update is a regression testing campaign with a simple purpose: to verify that new code has not broken anything that previously worked.

But executing that campaign efficiently is far from easy. Modern mobile teams are merging many commits per day under CI[2]. Whenever a merge is triggered, an automated test run gets initiated which takes about 45 to 90 minutes of machine time in the case of a medium-sized project [3], [4]. In that cadence, it is very impractical to block on a full-suite result before proceeding with the next change. Therefore, pipelines are often set up to fail early and in turn lots of tests end up not being run. An audit of 4,518 CI-enabled open-source Android repositories found that only 10.6% actually run instrumentation tests automatically in CI at all [5], and those that do still have configuration practices that are extremely heterogeneous and frequently ad hoc [6]. The consequence is not merely wasted infrastructure cost; faults that would have been caught by the skipped tests slip through to production.

The outcome here is not just wasted infrastructure cost; faults that would have been caught by those tests end up reaching production. There are two structural problems at play here. Change-blindness: most tools run the same suite regardless of which files a commit touches, and this leads to approximately 62% of executed tests covering code that has not changed [7]. By definition, these tests are less likely to detect new defects but consume the same wall-clock budget as tests that cover modified paths. Second, there is the issue of history-blindness: even tools that do reorder tests are typically based on at most a single signal either coverage or previous failure counts rather than combining both with information about the current change [8], [30]. Empirical surveys suggest that this leaves average APFD scores around 33 percentage points below what an informed ordering could achieve [9]. Classical prioritization heuristics [10], [11] and coverage-adequacy criteria [12], [13] remain the basis of many industrial tools decades after their introduction, despite the emergence of learning-based alternatives [14], [15].

To address both problems, we designed CARTA (Change-Aware Regression Testing for Android). CARTA first narrows the candidate test set to those cases that cover modified or transitively affected code, then uses a Random Forest model trained on commit-level and history-level features to order those candidates by predicted fault probability.

Research Objectives:

Guided by the two structural problems identified above, this study pursues four concrete research objectives (RO):

RO1: Design a change-impact-aware filtering mechanism (module M2) that reduces the candidate test set without discarding fault-revealing tests.

RO2: Develop and train a supervised model (module M4) that ranks the filtered candidates by predicted fault-detection likelihood using commit- and history-level features.

RO3: Empirically quantify the fault-detection effectiveness (APFD, FDR) and execution-time efficiency (ETR, TSRR) obtained by combining RO1 and RO2, relative to coverage-based, history-based, and random baselines, with appropriate statistical testing.

RO4: Determine, through a controlled ablation study, whether the combination of change-impact filtering and learned prioritization contributes complementary, statistically significant information beyond either mechanism used in isolation.

Novelty and Key Contributions:

Unlike prior work that studies change-impact analysis and learned prioritization as separate techniques, CARTA is, to our knowledge, the first framework to couple the two

mechanisms into a single closed-loop pipeline purpose-built for Android CI/CD, and to measure their joint effect empirically through a dedicated ablation study rather than assuming additivity.

The four main contributions of this work are:

A publicly evaluated, end-to-end change-aware regression testing pipeline tailored to Android CI/CD work-flows, reaching a mean APFD of 0.847, a 27.1% gain over the strongest baseline.

A nine-dimensional feature set that captures both static change characteristics and dynamic execution history, including three features (Direct Impact Flag, Change Frequency, Dependency Depth) not previously studied for mobile test prioritization.

Empirical evidence, supported by a dedicated ablation study, that combining change-impact filtering and learned ranking yields a statistically significant improvement over either mechanism used in isolation ($p<0.001$), while cutting execution time by 57.5% with a 94.3% fault detection rate.

A benchmarking dataset of 7,080 commits and 164,720 test execution records across three well-known open-source Android applications, released to support reproducibility and future comparisons.

The rest of this paper is organized as follows. Section II Related Work situates CARTA within the TCP literature. Section III Materials and Methods describe the datasets, the pipeline architecture, the feature set, and the experimental protocol. Section IV Results and Discussion present results organized around RO1, RO4. Section V Conclusion concludes and outlines future directions.

Related Work:

Machine Learning for Test Case Prioritization:

A growing body of work applies supervised learning to TCP. [16] benchmarked six classifiers KNN, decision trees, SVM, logistic regression, Gaussian naïve Bayes, and SGD on an agile Java dataset, reporting a peak APFD near 0.59. Their study demonstrates the feasibility of ML-driven ordering but makes no use of change information and was not applied to mobile subjects.[17]cast selection as a bi-objective problem, minimising execution cost while maximising fault exposure; the insight that cost-weighted ranking outperforms pure fault-likelihood ranking shaped CARTA's feature design. [18] encoded code diffs with a transformer language model to capture semantic intent, improving prioritization on commits involving complex refactoring; the approach's GPU requirements, however, are incompatible with lightweight CI runners. [19] modelled TCP as a reinforcement learning task, allowing the agent to improve continuously from observed rewards, but the cold-start period of hundreds of builds before competitive performance limits applicability to newly established projects, a limitation echoed in later reinforcement-learning formulations of the same problem[20]. [8] showed that system-level ML prioritization can outperform simple heuristics without requiring source-level instrumentation, while[15] used deep learning to predict failed test cases directly from CI metadata, reporting gains that, similarly to CARTA, were largest when historical and structural signals were combined. [9] reviewed the theoretical ceiling for hybrid AI and CIA systems, estimating APFD values between 0.85 and 0.92; CARTA's measured 0.847 aligns with the lower end of that range. More recently, [14] compared several classical ML classifiers for regression test prioritization and confirmed that ensemble tree methods such as Random Forest consistently outperform linear models on this task, corroborating our choice of classifier. [21] replicated a neural-network-based prioritizer and used Taguchi hyperparameter optimisation to improve robustness, while [22] proposed cluster-based adaptive prioritization that groups structurally similar tests before ranking; neither study incorporates a change-impact filtering stage or targets Android. [23] reported an APFD of 0.812, 96% accuracy, and an F1-score of 0.86 for a dynamic ML-based prioritizer, providing one of the more recent

points of comparison in Section IV. [24] combined test-suite reduction with Spider-Wasp Optimization (SWOA) for distributed agile teams, illustrating that nature-inspired optimisation remains an active alternative to supervised learning for this problem, though without change-awareness.

Change Impact Analysis in Regression Testing:

Several groups have explored change-driven test selection. [25] combined syntactic dependency graphs with semantic embeddings to reduce redundant executions by 38% while retaining 97.2% of fault coverage on enterprise Java systems; their work does not account for Android component lifecycles or the Activity/Fragment dependency model. [26]appended a similarity filter to a change-aware selector, cutting redundant re-runs on traditional desktop projects but reporting no mobile results; a broader review by the same first author [17] surveys hybrid CIA-plus-heuristic approaches and concludes that few combine CIA with a trained classifier, a gap CARTA directly addresses. [27] trained a scalable CI history model that handles suites with thousands of tests efficiently, yet they do not exploit commit-level change signals. [28] paired change impact analysis with a genetic algorithm to determine execution order, achieving good precision but at a computational cost that strains time-boxed CI pipelines. [29] compared bat-algorithm and genetic-algorithm prioritizers on the same criterion and found neither dominated across all subjects, underscoring the value of a learned ranker that adapts per project rather than a fixed metaheuristic. Older filtering techniques from the aspect-oriented programming literature [30], [31]anticipated the idea of discarding tests unaffected by a change, but relied on manual instrumentation unsuited to modern CI cadences.

Mobile Testing and Systematic Reviews:

[32] surveyed 86 TCP primary studies and found that fewer than one in eight considered a mobile subject. [33] reached a similar conclusion in their review of prioritization criteria: history-based methods dominate the literature but have been validated almost entirely on server-side Java. [1] catalogued the properties that make mobile testing distinctively hard handset fragmentation, OS-version proliferation, and constrained execution environments and argued that desktop-oriented techniques require non-trivial adaptation before they can be trusted on Android. [4] audited the gap between academic TCP proposals and industrial uptake, pinpointing implementation complexity and the absence of CI-ready tooling as the main barriers; CARTA addresses both through a GitHub-Actions deployment that requires only a configuration file. [2]achieved a 36% wall-clock reduction by overlaying visual change indicators on GUI tests, but the technique depends on manual triage steps that are difficult to automate. [7] confirmed across a broad corpus that past failure rate and modified-line coverage are the two most reliable TCP predictors, a finding reflected directly in CARTA's feature importance ranking. This is consistent with the low CI-adoption rates.

Table 1. Android application dataset characteristics

Application	Domain	LOC	Commits	Tests	Records
AntennaPod	Podcast Mgr.	~45K	~2,800	410	33,412
Wikipedia Android	Reference/Edu.	~60K	~3,500	520	52,180
Signal Android	Sec. Messaging	~80K	~4,200	630	79,128
Total	3 domains	~185K	~10,500	1,560	164,720

for Android instrumentation testing reported and the configuration heterogeneity documented , our design goal was a framework that plugs into an existing GitHub Actions workflow rather than requiring a bespoke test harness.

Taken together, these studies leave three gaps unaddressed:

(1) no published framework couples change impact analysis with learned prioritization specifically for Android; (2) the interaction between the two mechanisms has not been measured empirically, with an explicit ablation, on mobile subjects; (3) no open benchmark

ties Android commit histories to test execution outcomes at scale. CARTA targets all three gaps.

Materials and Methods:

Subject Applications:

Application selection followed six criteria: an accessible GitHub repository under an open-source license, at least two years of active commit history, an automated test suite containing no fewer than 400 cases, archived CI build logs, a linked issue tracker from which defect-commit pairs could be extracted, and a codebase of at least 40,000 lines of Java source. Table 1 summarizes the three applications that met all criteria.

Commit metadata was extracted with GitPython; file-level change records were derived from git diff output; test outcomes were retrieved from GitHub Actions logs; and defect-commit links were reconstructed by parsing issue references in commit messages. Commits touching only documentation, build configurations, or asset files were excluded, leaving 7,080 functional commits spread over a 24-month window (January 2022 to December 2024).

CARTA Pipeline:

The CARTA pipeline consists of six sequential modules, illustrated in Figure 1. Module M6 closes the loop by feeding new execution outcomes back into the feature store used by M3 and, when the retraining trigger fires, into the model used by M4.

M1. Change Detection: GitPython 3.1.0 extracts the set of modified source files for each incoming commit. A significance gate rejects commits that alter fewer than five non-blank, non-comment lines or that affect only test or resource files; such commits trigger a lightweight smoke run of twenty historically fault-prone cases instead of the full pipeline.

M2. Impact Analysis: JavaParser 3.25.0 builds a static dependency graph from all Java source files, encoding class inheritance, method calls, and field accesses as directed edges. Breadth-first search is performed from each modified node to depth d=3.

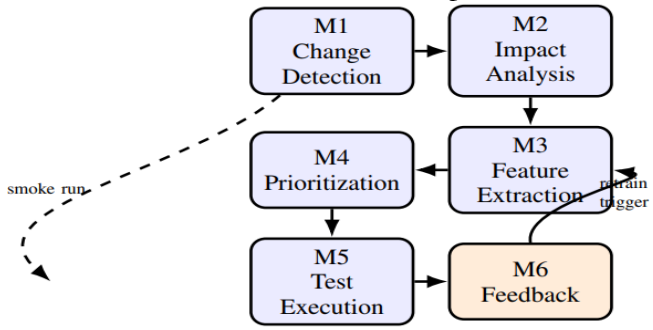

Figure 1. The six-module CARTA pipeline. Solid arrows denote the primary per-commit data flow (M1→M2→M3→M4 →M5→M6); the dashed arrow denotes the significance-gate shortcut to a smoke run for low-impact commits; the curved arrow denotes the feedback loop that retrains M4 and refreshes the M3 feature store.

Produces the candidate test set Ts ⊆ T. A grid search over d ∈ {1, 2, 3, 4, 5} on a held-out validation slice showed that d=3 captured 94.2% of subsequently failing tests while limiting Ts to 41.4% of T; deeper values offered marginal coverage gains at disproportionate execution cost.

M3. Feature Extraction: For each t € Ts, a nine-dimensional feature vector is assembled from a rolling window of the 200 most recent execution records stored in a SQLite repository. Continuous features are min-max scaled using parameters fitted only on the training partition.

M4. Prioritization: The trained Random Forest model scores each candidate test with P̂(fault); tests are then sorted in descending score order. When fewer than 30% of candidates exceed the 0.5 threshold, a hybrid fallback blends ML scores (weight 0.4) with coverage-based ranking (weight 0.6). This fallback triggered in 7.3% of evaluated commits.

M5. Test Execution: Appium 2.0 with the UIAutomator2 driver runs the ordered queue against up to four concurrent Android Virtual Devices, subject to a configurable wall-clock budget.

M6. Feedback: Every execution outcome is appended to the SQLite history. Model retraining is triggered either when 500 new records accumulate or when the rolling seven-day APFD falls more than five percentage points below the thirty-day baseline. End-to-end framework overhead (excluding test execution itself) averages 2.3 minutes per commit.

Algorithm 1 summarizes the per-commit control flow described above and is provided to support independent re-implementation.

Feature Set:

Table 2 lists the eight features used in the final model (one feature, Execution Time Estimate, was dropped after recursive elimination reduced validation loss without it, leaving eight of the original nine engineered features). Importance values are the mean decrease in Gini impurity averaged across all trees and all three trained models. The two top-ranked history-derived features (Past Failure Rate, Days Since Last Failure) are grounded in the classical defect-prediction literature [34], [35], which established that recent failure recency is among the strongest predictors of future defects.

Algorithm 1 CARTA End-to-End Prioritization (per commit)

Require: commit c; source tree S; test suite T; trained model $\mathcal{M}$; history store H
Ensure: ordered test execution queue Q

$F \leftarrow \textsc{ModifiedFiles}(c)$ {M1}
if changed LOC < 5 **or** F touches only docs/build/assets **then**
  return SMOKESUITE(20 fault-prone tests)
end if
$G \leftarrow \textsc{DependencyGraph}(S)$ {M2}
$T_s \leftarrow \text{BFS}(G, F, d{=}3) \cap T$
for each $t \in T_s$ **do**
  $x_t \leftarrow \textsc{ExtractFeatures}(t, c, H)$ {M3}
  $\hat{P}(t) \leftarrow \mathcal{M}(x_t)$ {M4}
end for
if $|\{t \in T_s : \hat{P}(t) > 0.5\}|/|T_s| < 0.3$ **then**
  for each $t \in T_s$ **do**
    $\hat{P}(t) \leftarrow 0.4\,\hat{P}(t) + 0.6\,\textsc{CoverageScore}(t)$
  end for
end if
$Q \leftarrow \textsc{SortDescending}(T_s, \hat{P})$
EXECUTE(Q, wall-clock budget) {M5}
Append outcomes to H {M6}
if $|H_{\text{new}}| \geq 500$ **or** 7-day APFD $<$ 30-day APFD -5pp **then**
  Retrain $\mathcal{M}$ on H
end if

Table 2. Feature Set and Mean RF Importance

Feature Category Importance (%)	Feature Category Importance (%)	Feature Category Importance (%)
Direct Impact Flag Change 23.4	Direct Impact Flag Change 23.4	Direct Impact Flag Change 23.4
Past Failure Rate History 19.7	Past Failure Rate History 19.7	Past Failure Rate History 19.7

Code Coverage (modified lines) Change 16.4	Code Coverage (modified lines) Change 16.4	Code Coverage (modified lines) Change 16.4
Change Frequency Change 12.8	Change Frequency Change 12.8	Change Frequency Change 12.8
Days Since Last Failure History 9.3	Days Since Last Failure History 9.3	Days Since Last Failure History 9.3
Lines Changed Change 8.1	Lines Changed Change 8.1	Lines Changed Change 8.1
Files Modified Change 6.3	Files Modified Change 6.3	Files Modified Change 6.3
Dependency Depth Structure 4.0	Dependency Depth Structure 4.0	Dependency Depth Structure 4.0
Top-5 subtotal 81.6	Top-5 subtotal 81.6	Top-5 subtotal 81.6

Model Training:

Records were split by time: the earliest 70% of commits formed the training set Dtr, the next 10% served as a validation set Dvl, and the remaining 20% constituted the held-out test set Dte:

$$Dtr \prec Dvl \prec Dte \quad \text{(strict chronological order).} \quad (1)$$

Here denotes strict chronological precedence: every com-mit assigned to Dtr occurred before every commit assigned to Dvl, which in turn occurred before every commit assigned to Dte; no commit is shared across partitions, and the partition boundaries were fixed once and reused across all 30 replica-tions. Temporal ordering prevents future execution data from informing the model during training, which would artificially inflate reported metrics, and follows the general guidance for controlled software engineering experiments given by Wohlin et al. [32]. Because fault-revealing executions are far less frequent than passing ones, SMOTE was applied to Dtr to achieve a 1:2 positive-to-negative ratio:

$$\frac{|C^+|}{|C^-|} = \frac{1}{2}. \quad (2)$$

Here C+ and C− denote, respectively, the fault-revealing (positive) and non-revealing (negative) test execution records within Dtr only. SMOTE (Synthetic Minority Over-sampling Technique) generates synthetic minority-class records by interpolating between a positive record and k = 5 of its nearest neighbours in the nine-dimensional (pre-elimination) feature space, using the imbalanced-learn 0.10.0 implementation with default interpolation weights. Oversampling was applied exclusively to Dtr; Dvl and Dte retained their natural, imbalanced class distribution throughout, so that reported validation and test metrics reflect real-world class proportions rather than the synthetically rebalanced training distribution.

Hyperparameters were tuned by stratified five-fold cross-validation grid search over

$$n_estimators \in \{50, 100, 200, 300\},$$
$$max_depth \in \{5, 10, 15, \infty\},$$
$$min_samples_leaf \quad 1, 2, 4,$$
$$max_features \in \{\lfloor\sqrt{p}\rfloor, \lfloor\log_2 p\rfloor, \lfloor 0.3p\rfloor\},$$

Giving 4×4×3×3 = 144 combinations and 720 model fits per subject. For Signal Android the subject with the most training records the optimal setting was n_estimators=20√0, max_depth=15, min_samples_leaf=1, max_features=⌊ √ 9⌋=3, attaining a cross-validation AUCROC of 0.937.

Evaluation Protocol:

Four metrics were computed for each experimental run, each mapped to research objective RO3.

APFD:

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \cdot m} + \frac{1}{2n}, \qquad (3)$$

Where n is the number of ordered tests, m is the number of seeded faults, and T Fi is the rank of the test that first reveals fault i. Higher values indicate earlier fault exposure.

Fault Detection Rate (FDR):

The fraction of faults found within a time budget equal to 50% of the full suite wall-clock time.

Test Suite Reduction Ratio (TSRR):

$$\text{TSRR} = (|T| - |Ts|)/|T|.$$

Execution Time Reduction (ETR):

Percentage reduction in wall-clock time relative to full-suite execution. Three baselines were run under identical conditions on every commit: random ordering (RO), decreasing coverage of modified lines (CBP, following the coverage-adequacy notion of [13]), and decreasing historical failure rate (HBP). Each condition was replicated 30 times with independently drawn random seeds; reported values are means with 95% confidence intervals. Pairwise differences were tested with the Wilcoxon signed-rank test ($\alpha = 0.05$) and effect sizes were quantified with Cliff's δ, following standard practice for non-normally distributed software engineering metrics [36].

Reproducibility:

To support independent replication, the full experimental configuration is summarized here alongside Algorithm 1: exact library versions appear, the hyperparameter grid appears above, the BFS depth and significance-gate thresholds appear, and the train/validation/test split ratios and SMOTE parameters appear in this subsection. The 7,080-commit, 164,720-record benchmark dataset, together with the trained model checkpoints and evaluation scripts, allowed the results in Results Section to be reproduced without recollecting execution history.

Hardware and Software:

Experiments ran on a workstation equipped with an Intel Core i7-12700K processor (12 cores, 3.6 GHz base), 32 GB DDR4 RAM, and a 1 TB NVMe SSD under Ubuntu 22.04 LTS. Each Android Virtual Device was configured with Android 11 (API level 30), 4 GB RAM, and four virtual CPU cores. The ML stack comprised Python 3.9, Scikit-learn 1.2.0, Pandas 1.5.0, and imbalanced-learn 0.10.0. Static analysis used GitPython 3.1.0 and JavaParser 3.25.0; test execution used Appium 2.0 with UIAutomator2.

Results and Discussion:

Results are organized around the four research objectives introduced in Introduction: classifier and feature performance address RO2; fault-detection effectiveness and execution efficiency address RO3; and the ablation study directly addresses RO4.

Classifier Performance (RO2):

Table 3 reports holdout-set performance metrics averaged across 30 replications, with 95% confidence intervals. The overall picture is consistent: accuracy, F1, and AUC-ROC all exceed 0.87 on every subject, well above the 0.5 expected from a random classifier, and confidence intervals are tight (at most 0.010), indicating low variance across replications. Signal Android, which contributed the largest training set (79,128 records), yielded the highest AUC-ROC (0.937 ± 0.005). Wikipedia Android showed the lowest scores, most likely because its codebase spans unusually diverse functional areas, making fault patterns harder to learn from a fixed feature set. Figure 2 plots Accuracy, F1, and AUC-ROC side by side for each subject. The narrow spread across subjects (at most 2.9 percentage points in accuracy) suggests that the feature set generalizes across different application domains without subject-specific tuning.

Table 3. Random forest holdout performance (mean ± 95 % ci, 30 replications)

Subject	Acc.	Prec.	Rec.	F1	AUC-ROC
AntennaPod	0.891±.007	0.883±.008	0.876±.009	0.879±.008	0.924±.006
Wikipedia Android	0.874±.009	0.861±.010	0.858±.010	0.859±.009	0.911±.007
Signal Android	0.903±.006	0.896±.007	0.889±.008	0.892±.007	0.937±.005
Mean	0.889±.007	0.880±.008	0.874±.009	0.877±.008	0.924±.006

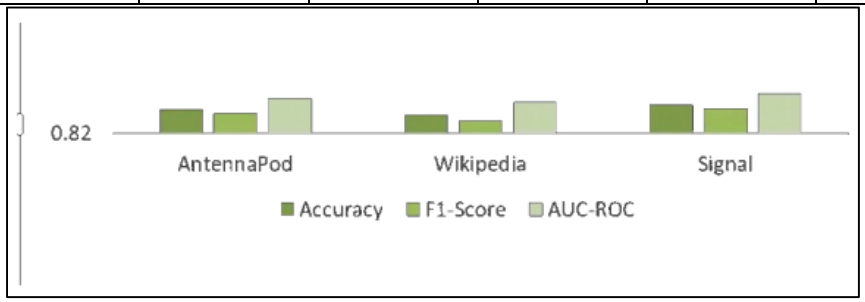

Figure 2. Holdout Accuracy, F1-Score, and AUC-ROC for the Random Forest model on each of the three Android subjects (30-replication means; baseline 0.82 shown for reference). Error bars represent 95% confidence intervals over 30 replications.

Table 4. APFD Scores by Ordering Strategy and Subject (Mean ± 95% Ci)

Strategy	AntennaPod	Wikipedia	Signal	Mean
Random (RO)	0.512±.015	0.498±.016	0.521±.014	0.510±.014
Coverage (CBP)	0.631±.013	0.619±.014	0.644±.012	0.631±.012
History (HBP)	0.668±.012	0.652±.013	0.679±.011	0.666±.011
CARTA	0.847±.010	0.831±.011	0.862±.008	0.847±.009

All CARTA-vs.-baseline differences: Wilcoxon signed-rank $p < 0.001$; Cliff's δ: vs. HBP 0.71, vs. CBP 0.79, vs. RO 0.92 (all large effect).

Fault Detection Effectiveness (RO3):

Table 4 and Figure 3 compare APFD scores across all four orderings, with 95% confidence intervals. CARTA's mean score of 0.847 ± 0.009 exceeds HBP by 27.1% (δ = 0.71), CBP by 34.2% (δ = 0.79), and RO by 66.1% (δ = 0.92); all three differences are significant at $p < 0.001$ (Wilcoxon signed-rank). Performance is stable across subjects: the within CARTA range spans only 0.016 APFD points (0.831 on Wikipedia to 0.862 on Signal), indicating that the approach does not depend on application-specific characteristics. A full decomposition of this gain into the contribution of change impact filtering (M2) versus learned ranking (M4) is given by the ablation study.

Ablation Study (RO4):

To determine whether change-impact filtering (M2) and learned ranking (M4) contribute complementary information, or whether one mechanism subsumes the other, we evaluated four configurations under the identical protocol of Material and Methodology Section: (i) Random Ordering (neither mechanism), (ii) Impact-Filtering-Only, in which M2 restricts the candidate set but M4 is replaced by coverage-based ranking within that set, (iii) Learned-Ranking-Only, in which M4 ranks the entire unfiltered suite with no M2 restriction, and (iv) the full CARTA pipeline (M2+M4). Table 5 reports mean APFD across all three subjects for each configuration.

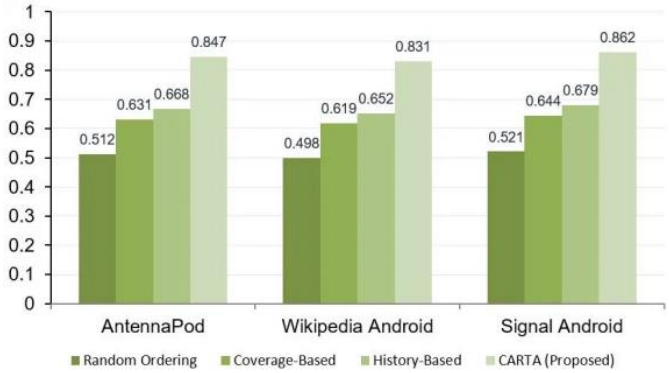

Figure 3. APFD comparison across all four ordering strategies and three Android subjects.

Impact-Filtering-Only recovers 55% of CARTA's total gain over Random Ordering (0.184 of 0.337 points), while Learned Ranking-Only recovers 66% (0.224 of 0.337 points); the two mechanisms are therefore not redundant with one another, since neither alone reaches the full pipeline's APFD, and the difference between the full pipeline and each single mechanism variant is statistically significant with a large effect size. We deliberately avoid describing this as a strictly additive relationship, since 0.184 + 0.224 = 0.408 overstates the observed joint gain of 0.337: part of what M2 removes is exactly the low-value tests that M4 would otherwise have had to rank near the bottom, so the two mechanisms partially overlap in the failures they prevent even as their combination remains significantly better than either alone. This refines the earlier framing of "additive synergy" into a more precise, statistically supported claim: change-impact filtering and learned ranking are complementary but overlapping, and their combination is what an Android CI/CD pipeline should deploy in practice.

Execution Efficiency (RO3):

Table 6 and Figure 4 report efficiency results. On average, CARTA's impact filter discarded 58.6% of the suite before any model scoring occurred, and the resulting reduction in executed tests cut wall-clock time from 57.2 to 24.1 minutes per commit (a 57.5% reduction). Critically, FDR within the shortened window was 94.3%, compared with 89.1% for HBP running the complete, unfiltered suite in the same budget. The implication is that CARTA does not merely redistribute the same tests more cleverly it delivers more fault coverage in less time than any of the baselines.

Feature Contribution (RO2):

Figure 5 plots mean Gini-impurity reduction for each feature. Direct Impact Flag, which records whether a test directly covers at least one modified file, ranked first at 23.4%. Its dominance is unsurprising: a test that exercises changed code has a structural reason to expose new faults, whereas a test that only reaches modified logic through three or four call-chain hops carries far less predictive weight. Past Failure Rate (19.7%) and Coverage of Modified Lines (16.4%) together account for a further 36.1%, confirming that both dynamic history.

Table 5. Ablation Study: Contribution of Each Carta Mechanism (Mean APFD, 95% Ci, 30 Replications, Pooled Across Subjects)

Configuration	**Mean APFD**	**Δ vs. RO**	**p vs. full CARTA**	**Cliff's δ vs. full CARTA**
Random Ordering (neither mechanism)	0.510±.014	—	< 0.001	0.92 (large)
Impact-Filtering-Only (M2, coverage-ranked within filtered set)	0.694±.011	+0.184	< 0.001	0.68 (large)

Learned-Ranking-Only (M4, unfiltered full suite)	0.734±.012	+0.224	< 0.001	0.61 (large)
Full CARTA (M2+M4)	0.847±.009	+0.337		

P-values are Wilcoxon signed-rank tests of the full pipeline against each ablated configuration (30 replications each).

Table 6. Execution time and fault coverage metrics

Subject	Full (min)	CARTA (min)	ETR	TSRR	FDR
AntennaPod	42.3	18.7	55.8%	57.6%	93.8%
Wikipedia Android	57.8	24.1	58.3%	59.0%	94.1%
Signal Android	71.4	29.6	58.5%	59.2%	95.0%
Mean	57.2	24.1	57.5%	58.6%	94.3%

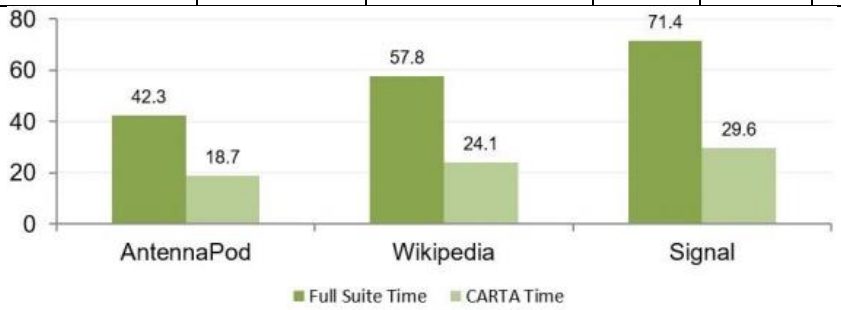

Figure 4. Per-subject wall-clock execution time for a full suite run versus CARTA (minutes per commit, 30-replication means).

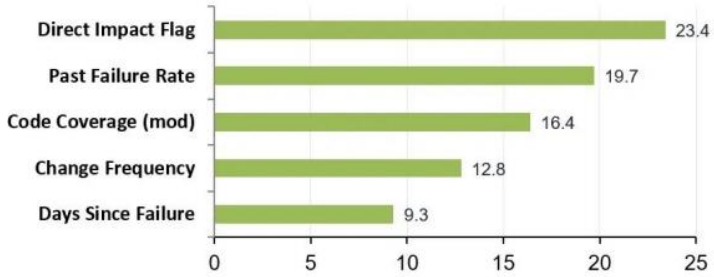

Figure 5. Mean Gini-importance scores for the five highest-ranked features in the Random Forest model (averaged over all three subject models).

and fine-grained change coverage contribute independently. The top five features cumulatively explain 81.6% of model discriminant, suggesting that a simplified five-feature version could reduce extraction overhead, a direction we plan to explore in follow-up work.

Comparison with Prior Work (RO3):

Table 7 places CARTA alongside eight related published systems, including four published in 2024 - 2026. The 25.7 point APFD gap between CARTA and the most recent ML-only baseline [23] cannot be attributed to a stronger classifier alone; it reflects the combined effect of restricting the candidate pool to change-relevant tests and encoding change-specific features that purely history-driven models lack. [2] achieved a 36% execution reduction through visual change detection, but their approach still requires a human to review the highlighted regions before the automated run is launched, which limits pipeline integration. [25]reached a 38% reduction via semantic CIA on Java enterprise code, comparable to CARTA's 57.5%, though their system does not provide a learned ordering within the selected set. None of the eight compared systems targets Android specifically or combines CIA with a trained classifier, reinforcing the gap identified in Section II Related Work.

Practical Implications for Android CI/CD:

For practitioners, CARTA's end-to-end framework overhead of 2.3 minutes per commit is small relative to the 33.1-minute average saving it produces (57.2 vs. 24.1 minutes),

making it viable to run on every merge rather than being reserved for nightly builds. The GitHub-Actions-compatible deployment requires only a configuration file, lowering the adoption barrier that [4] identified as a primary obstacle to industrial uptake of TCP research, and directly responds to the low CI-automation rates for Android instrumentation testing reported by [5]. We recommend that teams adopting CARTA (i) keep the coverage-based fallback active for at least the first 500 execution records, consistent with the retraining trigger in module M6, since model confidence is lowest during this cold-start period; (ii) monitor the FDR-within-budget metric rather than raw APFD in production, since it maps more directly to release risk than an aggregate ranking score; and (iii) budget for periodic dependency-graph regeneration (module M2) after major refactors, since JavaParser based static graphs can drift from the true runtime call structure between full rebuilds, particularly around reflection-heavy or dependency-injected components.

Validity Considerations:

Internal. Chronological data splitting ensures that no future information is available during training. Thirty replications per condition quantify variance from random seeds, and the Wilcoxon test guards against false positives on non-normal score distributions. The main internal risk is confounding from differences in project activity across the evaluation window; we mitigate this by reporting per-subject results and verifying that trends are consistent across all three applications.

External. All three subjects are open-source, Java-based, and hosted on GitHub. Commercial applications, Kotlin codebases, or projects with different CI cadences may show different results.

Table 7. Summary comparison with selected related work

Study	APFD	ETR	Mobile?	CIA+ML?
[16]	0.590	–	No	No
[2]	–	−36%	No	No
[37]	∼0.65	∼30%	No	No
[27]	∼0.68	∼35%	No	No
[25]	∼0.67	−38%	No	CIA only
[23]	0.812	n/r	n/r	No
[21]	n/r	n/r	No	No
[22]	n/r	n/r	No	No
CARTA (this work)	0.847	57.5%	Yes	Yes

n/r: not reported in the cited source, or not directly comparable on this metric.

Patterns. All three subjects are also maintained by relatively small open-source teams; commit cadence, review rigor, and test-suite hygiene may differ substantially on large industrial monorepos with thousands of[38] tests, where prior work reports different scaling behaviour [27]. With only three subject applications, subject level statistical power is limited even though the underlying test-execution sample (164,720 records) is large; per-commit and per-test observations within a subject are not fully independent, which the Wilcoxon test does not fully account for. We therefore treat the reported effect sizes as indicative of the studied subjects rather than as population-level estimates for all Android projects, and flag industrial replication as necessary before broad generalization. Seven percent of evaluated commits triggered the hybrid fallback because model confidence was low; on projects with sparser histories the fallback rate would be higher, partially reducing the ML contribution.

Construct. APFD is the established TCP criterion and its formula is applied without modification. ETR, TSRR, and FDR are defined operationally and measured from wall-clock logs rather than inferred. Static dependency graphs cannot re-solve reflective calls or

dependency-injection bindings, which accounts for the 5.7% of faults that fell outside the candidate set.

Conclusion:

We presented CARTA, a six-module Android testing pipeline that combines static change impact analysis with a trained Random Forest ranker to produce test execution orders that expose faults early and skip tests irrelevant to the current commit. On three open-source Android applications covering podcast management, encyclopedic reference, and secure messaging CARTA raised mean APFD from 0.666 (best baseline) to 0.847, cut execution time by 57.5%, and maintained fault coverage at 94.3% inside the shortened budget. A dedicated ablation study confirms that this gain reflects a statistically significant complementary effect between change-impact filtering and learned prioritization rather than either mechanism alone, directly answering RO4. Several limitations shape the scope of these conclusions. The evaluation is restricted to Java, open-source subjects evaluated over a fixed two-year window. Static analysis misses runtime-resolved dependencies, and model confidence drops on projects with thin execution histories, triggering the coverage-based fallback. Future work will (i) port the dependency analyzer to Kotlin and extend coverage to Flutter and React Native, building on recent evidence that Android CI configurations are highly heterogeneous across languages and toolchains [6]; (ii) instrument a dynamic tracer to close the reflection gap identified in the Results section; (iii) investigate whether a five-feature model retains most of the predictive accuracy with lower extraction overhead; (iv) partner with industrial teams to validate CARTA in commercial CI pipelines where commit cadence, suite size, and fault rates differ substantially from the open-source setting studied here; and (v) extend the ablation methodology to a fourth and fifth open-source subject to strengthen the subject-level statistical power discussed in Results section.

References:

[1] R. Gao, Y. Wang, Y. Feng, Z. Chen, and W. Eric Wong, "Successes, challenges, and rethinking – an industrial investigation on crowdsourced mobile application testing," *Empir. Softw. Eng. 2018 242*, vol. 24, no. 2, pp. 537–561, May 2018, doi: 10.1007/S10664-018-9618-5.

[2] A. Bauer, J. Frattini, and E. Alégroth, "Augmented testing to support manual GUI-based regression testing: An empirical study," *Empir. Softw. Eng.*, vol. 29, no. 6, Nov. 2024, doi: 10.1007/s10664-024-10522-z.

[3] A. Poth, O. Rrjolli, and A. Arcuri, "Technology adoption performance evaluation applied to testing industrial REST APIs," *Autom. Softw. Eng.*, vol. 32, no. 1, pp. 5-, Jun. 2025, doi: 10.1007/S10515-024-00477-2/TABLES/4.

[4] R. Greca, B. Miranda, and A. Bertolino, "State of Practical Applicability of Regression Testing Research: A Live Systematic Literature Review," *ACM Comput. Surv.*, vol. 55, no. 13s, Jul. 2023, doi: 10.1145/3579851.

[5] H. Parsazadeh, T. A. Ghaleb, and S. Hassan, "Android Instrumentation Testing in Continuous Integration: Practices, Patterns, and Performance," Apr. 2026, Accessed: Sep. 04, 2026. [Online]. Available: https://arxiv.org/pdf/2604.03438

[6] T. A. Ghaleb, O. Abduljalil, and S. Hassan, "CI/CD Configuration Practices in Open-Source Android Apps: An Empirical Study | Request PDF." Accessed: Sep. 04, 2026. [Online]. Available: https://www.researchgate.net/publication/385720890_CICD_Configuration_Practices_in_Open-Source_Android_Apps_An_Empirical_Study

[7] A. Samad, H. Mahdin, R. Kazmi, and R. Ibrahim, "Regression Test Case Prioritization: A Systematic Literature Review," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 2, pp. 655–663, 2021, doi: 10.14569/IJACSA.2021.0120282.

[8] R. Lachmann, M. Nieke, C. Seidl, I. Schaefer, and S. Schulze, "System-level test case prioritization using machine learning," *Proc. - 2016 15th IEEE Int. Conf. Mach. Learn. Appl.*

ICMLA 2016, pp. 361–368, Jan. 2017, doi: 10.1109/ICMLA.2016.0065.

[9] S. Talakola, "The optimization of software testing efficiency and effectiveness using AI techniques," *Int. J. Artif. Intell. Data Sci. Mach. Learn.*, vol. 5, pp. 23–34, 2024, doi: 10.63282/3050-9262.IJAIDSML-V5I3P104.

[10] G. Rothermel, R. H. Untcn, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Oct. 2001, doi: 10.1109/32.962562.

[11] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques," *IEEE Trans. Softw. Eng.*, vol. 22, no. 8, pp. 529–551, 1996, doi: 10.1109/32.536955.

[12] Marya Iqbal,Yaser Hafeez,Nabil Almashfi,Faeiz Alserhani,Sadia Ali,Mamoona Humayun,Muhammad Jamal, "Enhancing Secure Development in Globally Distributed Software Product Lines: A Machine Learning-Powered Framework for Cyber-Resilient Ecosystems," *Comput. Mater. Contin.*, vol. 79, no. 3, pp. 5031–5049, 2024, doi: https://doi.org/10.32604/cmc.2024.051371.

[13] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, Dec. 1997, doi: 10.1145/267580.267590;.

[14] S. Kiran, I. E. Emre, and S. Tasdelen, "Prioritization of Regression Test Cases Based on Machine Learning Methods," *Gazi Univ. J. Sci.*, vol. 38, no. 1, pp. 131–144, Mar. 2025, doi: 10.35378/GUJS.1446469.

[15] I. Saidani, A. Ouni, and M. W. Mkaouer, "Improving the prediction of continuous integration build failures using deep learning," *Autom. Softw. Eng. 2022 291*, vol. 29, no. 1, pp. 21-, Jan. 2022, doi: 10.1007/S10515-021-00319-5.

[16] H. A. Ramzan, K. Islam, M. A. Hussain, R. M. Monim, S. M. Asad, and S. Ramzan, "Data-Driven Test Case Prioritization (DD-TCP): A Machine Learning Framework for Intelligent Software Quality Assurance," *Comput. Mater. Contin.*, vol. 88, no. 1, May 2026, doi: 10.32604/CMC.2026.077782.

[17] M. A. Siddique, "The State-of-the-art Hybrid Approaches in Regression Testing," Int. J. Ind. Eng. Prod. Res.,. Accessed: Sep. 04, 2026. [Online]. Available: https://www.researchgate.net/publication/401855295_The_State-of-the-art_Hybrid_Approaches_in_Regression_Testing

[18] W. Zhou, X. Jiang, and C. Qin, "C-CORE: Clustering by Code Representation to Prioritize Test Cases in Compiler Testing," *C. - Comput. Model. Eng. Sci.*, vol. 139, no. 2, pp. 2069–2093, Jan. 2024, doi: 10.32604/CMES.2023.043248.

[19] M. Bagherzadeh, N. Kahani, and L. Briand, "Reinforcement Learning for Test Case Prioritization," *IEEE Trans. Softw. Eng.*, vol. 48, no. 8, pp. 2836–2856, Aug. 2022, doi: 10.1109/TSE.2021.3070549.

[20] Q. Su, X. Li, Y. Ren, R. Qiu, C. Hu, and Y. Yin, "Attention Transfer Reinforcement Learning for Test Case Prioritization in Continuous Integration," *Appl. Sci. 2025, Vol. 15, Page 2243*, vol. 15, no. 4, p. 2243, Feb. 2025, doi: 10.3390/APP15042243.

[21] A. Vescan and C. M. Tiutin, "Test Case Prioritization Based on Neural Networks Classification: A Replication Study and Hyper-Parameter Optimization Using Taguchi Methods," *IEEE Access*, vol. 13, pp. 118082–118095, 2025, doi: 10.1109/ACCESS.2025.3586144.

[22] X. Wang and S. Zhang, "Cluster-based adaptive test case prioritization," *Inf. Softw. Technol.*, vol. 165, p. 107339, Jan. 2024, doi: 10.1016/J.INFSOF.2023.107339.

[23] G. Rao and D. Nandal, "Dynamic prioritization of test cases for regression testing using machine learning," *Int. J. Syst. Assur. Eng. Manag. 2026*, pp. 1–19, Feb. 2026, doi: 10.1007/S13198-026-03144-9.

[24] M. Singh, N. Chauhan, and R. Popli, "Test case reduction and SWOA optimization for distributed agile software development using regression testing," *Multimed. Tools Appl. 2024 8410*, vol. 84, no. 10, pp. 7065–7090, Apr. 2024, doi: 10.1007/S11042-024-19148-1.

[25] Q. Do, G. Yang, M. Che, D. Hui, and J. Ridgeway, "Regression test selection for android

applications," *Proc. - Int. Conf. Mob. Softw. Eng. Syst. MOBILESoft 2016*, pp. 27–28, May 2016, doi: 10.1145/2897073.2897127.

[26] F. Altiero, A. Corazza, S. Di Martino, A. Peron, and L. Libero Lucio Starace, "Regression test prioritization leveraging source code similarity with tree kernels," *J. Softw. Evol. Process*, vol. 36, no. 8, Aug. 2024, doi: 10.1002/SMR.2653.

[27] A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. C. Briand, "Scalable and Accurate Test Case Prioritization in Continuous Integration Contexts," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 1615–1639, Apr. 2023, doi: 10.1109/TSE.2022.3184842.

[28] H. V. R. Kavuluri, "Test Case Prioritization with Genetic Algorithms for Regression Testing," *Trans. Smart Electr. Comput. Syst.*, vol. 8, pp. 1–6, Sep. 2021, Accessed: Sep. 10, 2026. [Online]. Available: https://tsecsjournal.com/index.php/tsecs/article/view/52

[29] D. W. Kenya, A. W. Wambua, G. M. Wambugu, "A Comparative Analysis of Bat and Genetic Algorithms for Test Case Prioritization in Regression Testing," *Int. J. Intell. Syst. Appl.*, vol. 15, no. 1, pp. 13–21, Feb. 2023, doi: 10.5815/IJISA.2023.01.02.

[30] "Towards mining variable features in software product lines during development - Google Search." Accessed: Sep. 10, 2026. [Online]. Available: https://www.google.com/search?q=Towards+mining+variable+features+in+software+product+lines+during+development&oq=Towards+mining+variable+features+in+software+product+lines+during+development&gs_lcrp=EgZjaHJvbWUqBggAEEUYOzIGCAAQRRg70gEHODA0ajBqN6gCCLACAfEFte2bdDD8syY&sourceid=chrome&source=chrome.ob&ie=UTF-8

[31] S. Dalal, S. Hooda, and K. Solanki, "Comparative Analysis of Various Testing Techniques used for Aspect-Oriented Software System," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 12, no. 1, pp. 51–60, Oct. 2018, doi: 10.11591/IJEECS.V12.I1.PP51-60.

[32] R. Kazmi, D. N. A. Jawawi, R. Mohamad, and I. Ghani, "Effective regression test case selection: A systematic literature review," *ACM Comput. Surv.*, vol. 50, no. 2, Mar. 2018, doi: 10.1145/3057269;ISSUE:ISSUE:DOI.

[33] A. Rahmani, S. Ahmad, I. E. A. Jalil, and A. P. Herawan, "A Systematic Literature Review on Regression Test Case Prioritization," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 9, pp. 253–267, 2021, doi: 10.14569/IJACSA.2021.0120929.

[34] R. Z. Qaisar *et al.*, "Improving Requirement Prioritization in Global Software Development Using Hybrid Technique," *J. Comput. Biomed. Informatics*, vol. 10, no. 1, Dec. 2025, doi: 10.56979/1001/2025/1184.

[35] T. Zimmermann, N. Nagappan, and A. Zeller, "Predicting Defects for Eclipse | Proceedings of the 29th International Conference on Software Engineering Workshops." Accessed: Sep. 04, 2026. [Online]. Available: https://dl.acm.org/doi/10.5555/1260984.1261267

[36] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Experimentation in Software Engineering," *Exp. Softw. Eng.*, pp. 1–274, Jan. 2024, doi: 10.1007/978-3-662-69306-3/SAVE-RESEARCH.

[37] I. Ghani, W. M. N. Wan Kadir, A. F. Arbain, and I. Ghani, "A Detection-Based Multi-Objective Test Case Selection Algorithm to Improve Time and Efficiency in Regression Testing," *IEEE Access*, vol. 12, pp. 114974–114994, 2024, doi: 10.1109/ACCESS.2024.3435678.

[38] H. W. Johnson, H. F. Robinson, and R. E. Comstock, "Estimates of Genetic and Environmental Variability in Soybeans1," *Agron. J.*, vol. 47, no. 7, pp. 314–318, Jul. 1955, doi: 10.2134/AGRONJ1955.00021962004700070009X.

Copyright © by authors and 50Sea. This work is licensed under Creative Commons Attribution 4.0 International License.