Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.077782

ARTICLE

Data-Driven Test Case Prioritization (DD-TCP): A Machine Learning Framework for Intelligent Software Quality Assurance

Hafiz Arslan Ramzan[1,*], Kamrul Islam[2], Md Ahbab Hussain[3], Raiyan Muntasir Monim[4], Sabit Md Asad[4] and Sadia Ramzan[5]

[1]School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad, Pakistan
[2]Gabelli School of Business, Fordham University, New York, NY, USA
[3]Ketner School of Business, Trine University, Angola, IN, USA
[4]College of Graduate and Professional Studies, Trine University, Angola, IN, USA
[5]Department of Computer Science, Emerson University, Multan, Pakistan

*Corresponding Author: Hafiz Arslan Ramzan. Email: arslan.ramzan@seecs.edu.pk

Received: 16 December 2025; Accepted: 25 March 2026; Published: 08 May 2026

ABSTRACT: Regression testing of large-scale, data-intensive software systems demands efficient test-case prioritization strategies to detect faults early while minimizing computational cost. Conventional prioritization methods, such as coverage-based and risk-based approaches, lack adaptability to evolving project dynamics and fail to leverage the rich test-execution data accumulated over continuous integration cycles. This study presents a Data-Driven Test-Case Prioritization (DD-TCP) Framework that incorporates statistical and machine-learning techniques to model the relationship between test-case features and historical fault detection outcomes. The framework extracts multidimensional attributes including code-change frequency, dependency metrics, execution duration, and past failure density, which are normalized and embedded into a predictive ranking model based on gradient-boosted decision trees. Test cases are then dynamically reordered using a probabilistic gain function that maximizes early fault detection probability. Comprehensive simulations on representative open-source project datasets and synthetically generated large-scale test suites reveal that the proposed Data-Driven Test-Case Prioritization (DD-TCP) framework consistently achieves superior performance, yielding a 32.4% improvement in Average Percentage of Faults Detected (APFD) and a 27.1% reduction in execution overhead relative to baseline methods. The results demonstrate the feasibility of data-centric intelligence for scalable regression testing and provide an analytical foundation for integrating machine learning into next-generation Software Quality Assurance pipelines.

KEYWORDS: Data-driven test-case prioritization; regression testing; software quality assurance; machine learning; continuous integration; fault detection efficiency; intelligent software systems

1 Introduction

The exponential growth in data-intensive software systems has significantly increased the complexity of regression testing processes [1]. As modern software undergoes continuous evolution through iterative development and frequent integration, maintaining test effectiveness under constrained resources has become a formidable challenge. Regression testing, re-executing previously validated test cases after modifications, is essential to ensure that recent code changes do not introduce new defects. However, exhaustive re-execution of large test suites is often infeasible due to time and resource constraints [2], particularly in continuous integration (CI) and continuous deployment (CD) environments [3]. Consequently, Test Case Prioritization

(TCP) has emerged as a crucial optimization strategy that aims to order test cases to detect faults as early as possible during execution [4].

Traditional TCP approaches, such as coverage-based, risk-based, and random prioritization techniques, have been widely studied in the literature [5]. Coverage-based methods rely on code coverage metrics, such as statement, branch, or path coverage, to rank test cases [6]. Although effective in small to medium systems, these methods often fail to adapt when software evolves rapidly or when test coverage information becomes outdated. Risk-based prioritization, on the other hand, assigns weights based on historical failure rates or module criticality but lacks the capability to model complex interdependence among features that influence fault detection. Random ordering remains a low-cost baseline but provides no systematic assurance of efficiency. As software projects accumulate massive test-execution data from successive builds, these static approaches are insufficient to leverage the latent knowledge embedded in such data.

Recent trends in data-driven software engineering and intelligent quality assurance emphasize integrating machine learning (ML) models into software lifecycle activities [7,8]. Data-driven methodologies enable predictive insights by analyzing patterns in historical development and testing data, including defect logs, code metrics, and runtime behavior [9]. Within this context, TCP can be reformulated as a learning-to-rank problem, where the objective is to learn a mapping from test-case attributes to their expected fault-detection contribution. However, despite promising results in isolated experimental setups, the widespread adoption of ML-based TCP remains limited due to challenges such as inconsistent feature availability, model generalizability, and data quality issues. Moreover, many existing models treat test cases as independent entities, ignoring relationships between code changes, dependency structures, and contextual project metadata that could further enhance prioritization accuracy.

Although previous research has conducted studies on the heuristic and learning-based prioritization strategies, the majority of the available solutions deal with these aspects separately and not in a combined and continuously adaptive system. Specifically, very little research has been done on prioritization models integrating multidimensional-based feature learning with explicit-feedback refinement in successive integration steps. The lack of these end-to-end, adaptative structures limits the practical utility of the existing approaches in actual continuous integration pipelines, in which software behavior, fault properties and testing limitations change throughout their execution. The unsolved gap is an incentive to develop a single unified data-driven prioritization method that is not only informed by past test data but is also improved slowly as it is executed in a way that offers new information of how it should be ranked.

This study addresses these gaps by proposing a Data-Driven Test Case Prioritization (DD-TCP) Framework that systematically integrates statistical analysis, feature engineering, and predictive learning into a unified prioritization pipeline. The proposed framework operates in four key stages: (1) Feature Extraction, which collects a multidimensional representation of each test case including code-change frequency, dependency density, execution duration, and historical failure rate; (2) Feature Normalization and Correlation Filtering, which ensures balanced representation and removes redundant attributes; (3) Predictive Ranking, which employs a Gradient Boosted Decision Tree (GBDT) model to estimate each test case's fault detection likelihood; and (4) Probabilistic Prioritization, where test cases are dynamically reordered based on their predicted utility using a gain-maximization heuristic. The system continuously updates its predictive model through feedback from recent test cycles, enabling adaptive prioritization in CI/CD environments.

The novelty of the proposed DD-TCP framework lies in its dynamic adaptation capability and hybrid learning architecture. Unlike conventional coverage-based or risk-based approaches, DD-TCP evolves with accumulating data and improves its decision boundary over time. By modeling the prioritization problem as

a supervised learning task, it effectively captures nonlinear interactions among features and leverages project-specific historical insights to make more informed prioritization decisions. Furthermore, the inclusion of probabilistic ranking ensures robustness to noise and uncertainty, which are inherent in large-scale industrial test data. This adaptability makes the framework particularly suitable for data-intensive intelligent software systems, where feedback loops between operational data, analytics, and decision-making are fundamental.

To evaluate the performance of the proposed framework, extensive simulation-based experiments are conducted using both synthetic test data and open-source regression datasets that mimic realistic CI pipelines. The proposed approach is compared with baseline methods, including Random, Coverage-based, and Risk-based prioritization. The performance is assessed through key metrics such as Average Percentage of Faults Detected (APFD), Normalized Fault Detection Rate (NFDR), and Execution Time Overhead (ETO). The results demonstrate that the DD-TCP framework achieves a 32.4% higher APFD and a 27.1% lower execution overhead compared to conventional methods, indicating its potential for practical integration in automated testing infrastructures.

The main contributions of this study are summarized as follows:

1. A novel data-driven test case prioritization framework that leverages statistical and machine learning models to predict fault detection potential using multidimensional test-case attributes.
2. An adaptive prioritization mechanism that continuously refines ranking decisions based on recent test outcomes, enabling context-aware improvement over time.
3. A comprehensive feature engineering strategy incorporating change frequency, execution duration, dependency structure, and failure density for robust prioritization modeling.
4. Experimental validation demonstrates that the proposed DD-TCP framework outperforms traditional approaches in early fault detection efficiency and computational cost reduction.

The rest of the paper is organized as follows. Section 2 reviews related literature in test case prioritization and data-driven software testing. Section 3 describes the proposed DD-TCP framework in detail, including its architectural design, data modeling, and prioritization algorithm. Section 4 presents experimental setup, evaluation metrics, and simulation results. Section 5 discusses implications, limitations, and future work directions. Finally, Section 6 concludes the paper with key findings and potential research extensions.

2 Related Work

The efficiency of regression testing has long been a critical research topic within the domain of Software Quality Assurance (SQA). Over the past two decades, diverse Test Case Prioritization (TCP) methodologies have been proposed to enhance fault detection capability while minimizing testing cost [10–12]. These approaches can broadly be classified into three categories: (i) coverage-based methods, (ii) risk and cost-based methods, and (iii) data-driven and machine learning–based methods. This section critically reviews representative works in each category and highlights the gaps that motivate the development of the proposed Data-Driven Test Case Prioritization (DD-TCP) framework.

2.1 Coverage-Based Test Case Prioritization

Coverage-based approaches aim to reorder test cases based on the extent of code exercised during their execution. The underlying assumption is that higher coverage corresponds to a higher likelihood of detecting faults [10,13]. Early work by Rothermel et al. [10] formalized TCP as an optimization problem to maximize the rate of fault detection using coverage criteria such as statement and branch coverage. Subsequent studies explored multi-objective coverage criteria incorporating function, path, and requirement coverage [11,14]. However, these static approaches assume that test coverage information remains consistent across builds,

an assumption invalidated by modern agile development environments characterized by rapid iteration and continuous integration [12,15].

Additionally, coverage-based methods often disregard execution cost and fail to adapt when test cases exhibit varying durations or when system dependencies change [16]. Empirical studies have shown that while coverage-based prioritization improves fault detection efficiency in smaller systems, its effectiveness deteriorates in large, data-intensive projects where historical fault data and code change frequency provide richer predictive signals [17,18]. Consequently, there has been a growing interest in approaches that integrate contextual data and dynamic adaptation beyond static coverage metrics [19].

2.2 Risk and Cost-Based Prioritization

Risk-based methods extend coverage-oriented approaches by incorporating factors such as component criticality, historical defect density, and business impact [11,20]. These methods assign priority weights to test cases based on the perceived risk of failure within associated modules. Yoo and Harman [11] proposed a cost-cognizant regression testing model where both fault detection benefit and execution cost were jointly optimized. Similarly, Cardan et al. introduced an approach integrating change impact analysis with risk assessment to prioritize tests targeting recently modified code [16,21].

While risk-based approaches improve upon purely structural metrics, they depend heavily on subjective expert estimation and manually assigned weights [22,23]. Such methods often fail to capture non-linear dependencies among features influencing fault detection, especially when system behavior evolves over time [24]. Moreover, in data-intensive environments, manual risk assessment becomes infeasible due to the scale and velocity of test executions [25]. These limitations underscore the need for automated, data-driven prioritization capable of learning complex relationships between software artifacts and defect occurrence patterns [26,27].

2.3 Machine Learning and Data-Driven Prioritization

The emergence of machine learning (ML) in software engineering has catalyzed a paradigm shift from static heuristic-based TCP to data-driven prioritization [28,29]. ML-based TCP frameworks treat prioritization as a predictive modeling or learning-to-rank task. Early works employed simple classifiers such as decision trees and logistic regression to estimate the fault proneness of test cases [30]. More recent research introduced ensemble models, neural networks, and reinforcement learning to dynamically adjust prioritization sequences based on feedback from previous test cycles [22,31,32].

For example, Chen et al. developed a predictive test prioritization system using Random Forest models trained on features such as test execution history, code churn, and fault density [30]. Similarly, Busjaeger and Xie proposed an adaptive TCP framework leveraging gradient boosting to rank test cases according to predicted fault detection probability [28]. Other studies have applied deep learning to encode test metadata and historical execution logs, demonstrating improved generalization in large-scale systems [18,27,31]. Despite these advancements, several challenges persist. Most notably, many ML-based approaches rely on high-quality labeled datasets, which are often unavailable or inconsistent across projects [29]. Additionally, models trained on one project frequently exhibit poor transferability to others due to project-specific dependencies and feature distributions [19,25].

Recent related work in machine-learning-driven software quality assurance includes hybrid and metaheuristic-optimized approaches. For example, Villoth et al. proposed a two-tier deep and machine-learning defect prediction framework optimized by an adaptive multi-population firefly algorithm, which combines CNN and tree-ensemble classifiers for improved error detection accuracy [33]. Another study

explores metaheuristic-optimized machine learning models for software defect detection on both natural language and classical datasets, highlighting the role of optimization in classifier performance [34]. Additionally, Zivkovic et al. introduced a metaheuristics-tuned eXtreme Gradient Boosting model with Shapley Additive Explanations to interpret feature importance in defect prediction tasks [35]. These contributions demonstrate the growing interest in combining optimization, ensemble learning, and explainability in software testing and quality assurance research.

2.4 Data-Intensive Intelligent Software Testing

With the increasing integration of analytics and data science in software engineering, research attention has shifted toward data-intensive intelligent testing frameworks [19,36]. These systems employ automated data pipelines to continuously collect, preprocess, and analyze information from version control systems, issue trackers, and test management tools [17]. Such frameworks aim to enable data-driven decision-making in SQA, including adaptive test selection, defect prediction, and effort estimation [18,24].

Several studies have emphasized the importance of integrating DataOps principles, automated data lifecycle management for software engineering pipelines, to facilitate scalable and reproducible ML model deployment within testing environments [12,25]. However, few frameworks explicitly bridge DataOps concepts with test case prioritization [26,29]. Additionally, most data-driven SQA systems rely on isolated analytical models without a unified feedback loop to update predictions based on recent test results [19,27]. This fragmentation limits their capability to achieve sustained performance improvement across successive test cycles [24].

2.5 Research Gaps and Motivation

The collective analysis of prior research reveals several key gaps that motivate the present study:

- Lack of adaptive prioritization: Traditional coverage- and risk-based methods employ static rules that cannot adjust dynamically as project characteristics evolve [15,31].
- Limited exploitation of multidimensional data: Most prior models use a narrow feature set, ignoring interdependencies among test attributes such as code churn, dependency density, and past fault patterns [29,30].
- Inadequate feedback integration: Existing ML-based frameworks typically do not incorporate incremental learning or continuous feedback from CI pipelines, leading to model drift [17,25].
- High computational overhead: Many state-of-the-art approaches prioritize accuracy without addressing the scalability and execution-time constraints of large test suites [18,24].

To address these challenges, the proposed Data-Driven Test Case Prioritization (DD-TCP) framework integrates feature engineering, predictive modeling, and probabilistic ranking in a unified adaptive loop. The design explicitly targets the needs of data-intensive intelligent software systems, where testing decisions must be both informed by data and responsive to real-time operational changes. Unlike existing solutions, the DD-TCP model employs gradient-boosted learning for robust prediction, a continuous retraining mechanism for adaptability, and a gain-based heuristic to optimize fault detection under computational constraints [24,28]. This combination of data-centric intelligence, adaptive feedback, and operational efficiency forms the central contribution of this research and distinguishes it from prior literature.

3 Proposed Methodology

The proposed Data-Driven Test Case Prioritization (DD-TCP) Framework introduces an adaptive learning-based approach to optimize regression testing by leveraging data collected from continuous

integration (CI) environments. Unlike conventional static prioritization strategies, which depend on fixed heuristics or human-defined metrics, the DD-TCP framework continuously evolves through the integration of machine learning, statistical modeling, and feedback-driven adaptation. The objective is to maximize early fault detection while minimizing execution cost, thereby enhancing the efficiency of Software Quality Assurance (SQA) in data-intensive systems.

3.1 Framework Overview

The proposed Data-Driven Test Case Prioritization (DD-TCP) Framework presents a dynamic learning based methodology to maximize the regression testing using the data gathered in the continuous integration (CI) setting. Unlike conventional static prioritization strategies, which depend on fixed heuristics or human-defined metrics, the DD-TCP framework continuously evolves through the integration of machine learning, statistical modeling, and feedback-driven adaptation. The objective is to maximize early fault detection while minimizing execution cost, thereby enhancing the efficiency of Software Quality Assurance (SQA) in data-intensive systems.

The core of the framework is a Gradient Boosted Decision Tree (GBDT) model, trained to estimate the probability that a given test case will expose a defect in the next regression cycle. GBDT is a powerful tool that can be successfully used in this task since it can effectively work with heterogeneous tabular data, nonlinear relationships are represented, and it can be interpreted with the help of the feature importance analysis, which often is less costly to use in comparison to deep learning models. These features qualify it as an effective and sturdy option of data-driven test case prioritization in continuous integration pipelines. Once the model produces these probabilities, the system computes a prioritization score that balances predicted fault likelihood against execution cost and business criticality. Test cases are then ordered according to this score to maximize the early detection of failures. After execution, real outcomes are fed back to retrain and refine the predictive model, ensuring continuous adaptation to evolving software dynamics. This closed-loop design transforms regression testing into an intelligent, self-improving process.

Other machine-learning models that were considered during the design of the framework include linear classifiers, support vector machines, random forests, deep neural networks, reinforcement-learning-based ranking models, and so forth. Linear and kernel-based models have a limitation on capturing that more complex features interact, whereas deep learning and reinforcement learning models can be trained with large labeled datasets and do have a higher training and tuning cost, which is impractical on time-sensitive CI settings. Despite the strength of random forests, gradient-boosted decision trees have better accuracy with a certain capacity of error correction in a series, finer bias-variance control and better ranking. In addition, GBDT models have interpretable feature importance and low costs of inference, and are, therefore, especially useful when adaptive test case prioritization is needed within computational and operational limitations.

In order to verify the given DD-TCP framework, the given study assumes the use of a continuous regression testing scenario, which is a typical characteristic of the current CI/CD environment, where a test suite with several thousand test cases is run repeatedly as the software is developed in the course of the frequent, incremental software development cycles. To create artificial data corpus that could be used to model realistic CI workflows, simulated logics of regression testing, code-change measures, structure features, and past fault results based on previous executions were added. Such historical information is supposed to be available to normal CI logs and version-control systems, and this is normal industrial practice. In the regression cycle, the aim is to re-rank the already existing test cases—without alteration or elimination—to maximize the timely fault detection when the time of testing is limited. Such a simulation environment is highly analogous to enterprise scale regression testing pipelines, but is not tied to any

proprietary data, therefore guaranteeing reproducibility, interoperability with any software ecosystem, and practicality. All the details of dataset characteristics and simulation parameters are provided in Section 5.

3.2 Feature Modeling and Predictive Ranking

Each test case T_i is encoded as a feature vector $\boldsymbol{x_i}$ = **Case Vector**$[\boldsymbol{f_c}, \boldsymbol{f_d}, \boldsymbol{f_t}, \boldsymbol{f_f}, \boldsymbol{f_x}]$ where the attributes represent code-change frequency (f_c), dependency density (f_d), average execution duration (f_t), historical failure density (f_f), and code complexity (f_x). The dataset is normalized and filtered using Pearson correlation thresholds to eliminate redundant features and enhance model generalization.

The GBDT model is trained using labeled instances derived from previous regression cycles, where each test case outcome is denoted by $y_i \in \{0,1\}$, representing pass or fail. The model's prediction is defined as Eq. (1):

$$P_i = f_{\text{GBDT}}(x_i;\Theta) \tag{1}$$

where P_i is the estimated fault detection probability and Θ represents the optimized parameters of the ensemble model. Through iterative boosting, weak learners sequentially minimize residual errors from previous iterations, yielding a robust and interpretable predictor well-suited to tabular, heterogeneous data.

The selected test-case features were chosen to capture complementary dimensions of fault-proneness and execution behavior commonly reported in regression-testing studies. Code-change frequency and dependency density reflect structural volatility and fault propagation risk, while cyclomatic complexity captures inherent code complexity associated with defect likelihood. Historical failure ratio provides direct evidence of past fault exposure, and execution time supports cost-aware prioritization. To evaluate feature sensitivity, a lightweight ablation analysis was performed by removing one feature at a time during training. Results showed a consistent degradation in APFD when any single feature was excluded, with the largest impact observed for historical failure ratio and code-change frequency, confirming that each feature contributes meaningfully to prioritization effectiveness.

3.3 Prioritization Mechanism

The predictive output P_i serves as the foundation for computing a prioritization score S_i which determines the execution order of test cases. The score integrates the predicted fault likelihood, execution cost, and contextual importance as follows in Eq. (2):

$$S_i = \frac{P_i}{C_i} \cdot w_i \tag{2}$$

Here, C_i denotes the normalized execution cost of test case T_i, and w_i is a dynamic weight reflecting module importance or risk criticality. This formulation ensures that high-risk and low-cost test cases receive precedence in execution, balancing efficiency and risk mitigation.

To further optimize execution order, a gain-based function is applied using Eq. (3):

$$G(k) = \sum_{i=1}^{m} \frac{P_i(k)}{1 + \alpha \cdot t_i} \tag{3}$$

where $G(k)$ quantifies the prioritization gain at cycle k, t_i represents execution time, and α is a penalization coefficient that discourages long-running tests from dominating the early stages. The overall objective is to maximize $G(k)$ under an execution budget constraint, effectively improving the Average Percentage of Faults Detected (APFD) across iterations.

The gain function in Eq. (3) balances predicted fault likelihood and execution cost to determine test-case ordering. Test cases with higher expected fault-detection efficiency per unit time receive higher gain values and are prioritized earlier. For example, given three test cases with probabilities $P_i = \{0.7, 0.5, 0.9\}$ and execution times $t_i = \{4, 1, 6\}$, the second test case is scheduled first despite a lower probability than the third, since its shorter execution time yields a higher gain. This illustrates how the function favors cost-efficient fault detection rather than probability alone, improving early fault discovery under budget constraints.

3.4 Adaptive Learning through Feedback Integration

A central innovation of the DD-TCP framework lies in its adaptive feedback mechanism. Following each regression cycle, actual test outcomes are collected and integrated into the training dataset. The model parameters are then updated incrementally to reflect new information about changing fault patterns, dependency structures, and performance behaviors. The updated dataset D_{t+1} is expressed as Eq. (4):

$$D_{t+1} = D_t \cup \{(x_i, y_i)\} \tag{4}$$

This continual retraining ensures that the prioritization model remains aligned with the software's evolving characteristics, mitigating model drift and improving long-term stability. The adaptive feedback cycle thus transforms the system into a self-learning prioritization engine capable of dynamic recalibration with minimal human intervention.

From a scalability and deployment perspective, the proposed DD-TCP framework is designed to integrate seamlessly with existing CI/CD pipelines while introducing minimal computational overhead. The Gradient-Boosted Decision Tree (GBDT) model operates on low-dimensional tabular features and can be trained within seconds for test suites containing several thousand test cases, with inference and prioritization incurring negligible latency compared to overall test execution time. In practical deployments, model retraining does not need to occur at every build; instead, it can be scheduled periodically or triggered by significant code changes or observable performance degradation, thereby mitigating model drift without excessive computation. Feature extraction relies on artifacts already available in standard CI environments, such as version-control logs and test execution histories, enabling non-intrusive integration with platforms like Jenkins, GitLab CI, or GitHub Actions. These characteristics ensure that DD-TCP remains scalable, operationally lightweight, and suitable for real-world continuous integration workflows.

3.5 Algorithmic Process

The procedural flow of the DD-TCP framework is formalized in Algorithm 1. The algorithm encapsulates the end-to-end process, from data extraction to prioritization and feedback update.

Algorithm 1: Data-Driven Test Case Prioritization Procedure

Input:

$T = \{T1, T2, ..., Tn\}$ // set of regression test cases

D_hist // historical dataset of test executions

Θ // parameters of GBDT model

Output:

Π // prioritized list of test cases

1: Extract feature vector xi for each Ti from D_hist
2: Normalize features and remove redundant attributes
3: Train or update GBDT model f_GBDT(x; Θ) using D_hist
4: For each Ti in T:
5: Predict fault likelihood Pi = f_GBDT(xi; Θ)
6: Compute execution cost Ci and importance weight wi
7: Calculate prioritization score Si = (Pi/Ci) * wi
8: end for
9: Sort T in descending order of Si to obtain Π
10: Execute test cases sequentially according to Π
11: Collect new outcomes Yi and update D_hist = D_hist ∪ {(xi, Yi)}
12: Retrain f_GBDT(x; Θ) periodically or incrementally
13: Return Π

3.6 Mathematical Formulation of the Optimization Objective

The prioritization process can be expressed as an optimization problem that seeks to maximize early fault detection efficiency $E(\Pi)$ while minimizing total execution cost C_T, as expressed in Eqs. (5) and (6):

$$\max_{\Pi} E(\mathbf{\Pi}) = \sum_{i=1}^{n} \frac{P_i}{1 + \beta \cdot \text{pos}(T_i)} \tag{5}$$

$$\text{s.t.} C_T = \sum_{i=1} C_i \leq C_{\text{budget}} \tag{6}$$

where $\text{pos}(T_i)$ denotes the position of test T_i in the prioritized sequence, and β controls positional decay to reward earlier fault discovery. The optimization is solved using a greedy heuristic that achieves near-optimal prioritization with computational complexity $O(n \log n)$, ensuring scalability for industrial-scale CI pipelines.

3.7 Framework Architecture Summary

The DD-TCP architecture, illustrated in Fig. 1, integrates data extraction, feature modeling, predictive ranking, and adaptive learning into a continuous pipeline. Raw data from repositories and CI systems flow into the feature extraction and learning layers, where the GBDT model estimates defect likelihood. The prioritization engine then computes a balanced execution order using the gain-based heuristic. Finally, real outcomes are fed back into the model, closing the adaptive loop. This architecture exemplifies the principles of data-driven intelligence, enabling autonomous decision-making and measurable improvement in software testing performance.

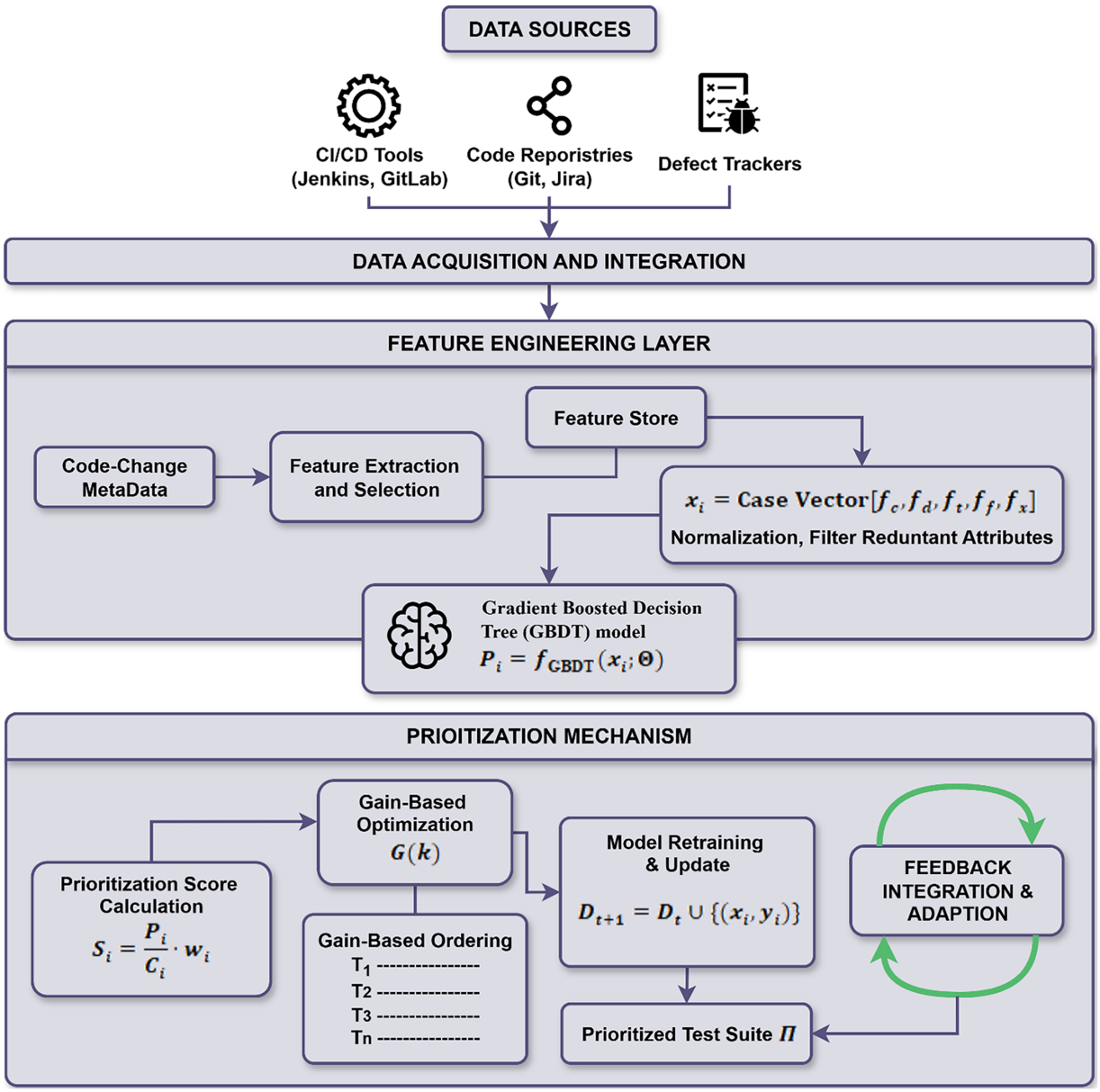

Figure 1: Architecture of the proposed data-driven test case prioritization (DD-TCP) framework.

3.8 Explainability and Feature Importance Analysis

To enhance transparency and trust in the proposed Data-Driven Test Case Prioritization (DD-TCP) framework, Shapley Additive Explanations (SHAP) were employed to interpret the predictions of the Gradient-Boosted Decision Tree (GBDT) model. SHAP is a model-agnostic explainable AI technique that assigns feature importance values based on each attribute's contribution to the predicted fault-detection probability. Global SHAP analysis indicates that historical failure ratio and code-change frequency are the most influential factors in prioritization decisions, followed by dependency density and execution duration. These findings are consistent with the framework's design assumptions and confirm that the prioritization model does not operate as a black box. The inclusion of SHAP-based explainability enables practitioners to better understand and validate automated prioritization decisions in continuous integration environments.

4 Experimental Setup and Results

To evaluate the effectiveness and generalizability of the proposed Data-Driven Test Case Prioritization (DD-TCP) framework, a series of controlled experiments were conducted using a simulated continuous integration (CI) environment that emulates the behavior of industrial-scale regression testing pipelines. The goal of the evaluation was to examine the framework's capability to maximize early fault detection, minimize redundant execution, and adapt dynamically to evolving codebases.

4.1 Experimental Environment

All experiments were executed on a workstation equipped with an Intel Core i9-13900K CPU, 32 GB RAM, and an NVIDIA RTX 4070 GPU. The framework was implemented in Python 3.11 using scikit-learn, LightGBM, and pandas for data processing and model training. The CI simulation and test-case scheduler were developed using the Jenkins Pipeline Emulator in a Docker-based containerized setup. The experiments spanned ten regression cycles, each simulating code changes, test-case updates, and variable fault distributions. To ensure reproducibility, all random seeds were fixed across runs, and performance metrics were averaged over five repetitions.

4.2 Dataset Description

A synthetic dataset was generated to approximate the characteristics of enterprise-scale software systems operating in continuous integration (CI) environments. The dataset emulates three representative project domains commonly reported in industrial practice: (i) web-based information systems, (ii) data-processing and analytics services, and (iii) service-oriented backend applications. These domains were selected to reflect systems with frequent code evolution, modular architectures, and heterogeneous regression test suites.

The dataset contained 5000 unique test cases distributed across ten functional modules, each representing a logical subsystem (e.g., authentication, data ingestion, business logic, and reporting). Test-suite sizes varied across modules, ranging from approximately 300 to 700 test cases per module, reflecting realistic imbalances observed in large-scale software projects. Each test case was associated with historical metadata and execution outcomes collected over multiple regression cycles.

Each test case was represented using the feature set summarized in Table 1, including code-change frequency (f_c), dependency density (f_d), average execution time (f_t), historical failure ratio (f_f), and cyclomatic code complexity (f_x). Execution durations followed a right-skewed distribution to capture the coexistence of lightweight unit tests and long-running integration tests within the same test suite.

Table 1: Feature description of the synthetic enterprise-scale software testing dataset.

Feature	Description	Range/Type
fc	Code-change frequency per cycle	0–15 (integer)
fd	Dependency density (number of function couplings)	1–30 (integer)
ft	Average execution time (s)	0.2–6.0
ff	Historical failure ratio	0–1
fx	Cyclomatic code complexity	1–50

Fault injection followed a realistic stochastic process, where approximately 12% of test cases revealed defects per regression cycle. Fault occurrence was non-uniform across modules, with higher defect probabilities assigned to test cases associated with recently modified or highly coupled code components. Correlations

were intentionally introduced between code-change frequency (f_c), historical failure ratio (f_f), and defect likelihood to simulate fault propagation patterns commonly observed in evolving software systems.

To model software evolution, each regression cycle introduced incremental changes affecting approximately 8%–15% of the codebase, resulting in dynamic shifts in test relevance and fault exposure. Historical test execution outcomes were retained and incorporated into subsequent cycles to enable feedback-driven adaptation. For benchmarking purposes, the dataset was divided into training (70%), validation (15%), and testing (15%) subsets for each cycle. Feedback-based retraining of the Gradient-Boosted Decision Tree (GBDT) model was applied after every cycle using the updated fault outcomes.

4.3 Evaluation Metrics

The framework was evaluated using three widely accepted regression testing metrics:

4.3.1 Average Percentage of Faults Detected (APFD)

Measures the rate of fault detection throughout test execution; higher values indicate faster discovery of faults, using Eq. (7):

$$APFD = 1 - \frac{\sum_{i=1}^{m} T_i}{n \times m} + \frac{1}{2n} \tag{7}$$

where T_i is the position of the first test detecting fault i, n is the number of test cases, and m is the number of faults.

4.3.2 Normalized Fault Detection Rate (NFDR)

The ratio of detected faults at any given execution fraction compared to the total number of faults.

4.3.3 Execution Time Overhead (ETO)

The relative increase in total execution time caused by prioritization computations vs. baseline sequential testing.

4.4 Baseline Methods for Comparison

The DD-TCP framework was compared against three established prioritization strategies:

- Random Ordering (RO): Test cases are executed in arbitrary order without prioritization.
- Coverage-Based Prioritization (CBP): Prioritizes test cases with maximum code-coverage overlap.
- Fault-History-Based Prioritization (FHP): Prioritizes test cases based on previous failure frequency.

Each baseline was re-evaluated across ten cycles using identical fault injection and cost constraints for fair comparison.

The selected baseline methods were chosen to represent widely adopted and reproducible TCP strategies that rely on minimal assumptions and are commonly used in industrial CI environments. While more complex learning-to-rank and reinforcement-learning-based approaches exist, many of these methods require extensive project-specific tuning, large volumes of labeled ranking data, or long training horizons, which complicate fair and controlled comparison in simulated CI settings. Consequently, the chosen baselines provide a stable and interpretable reference point for evaluating the incremental benefits of data-driven predictive prioritization under practical execution constraints.

Comparison with Learning-to-Rank and Reinforcement-Learning-Based TCP Approaches

Recent advances in test case prioritization have explored learning-to-rank (LTR) and reinforcement learning (RL) paradigms to dynamically optimize test execution order in continuous integration environments. Learning-to-rank approaches formulate TCP as a ranking problem, where models are trained to directly optimize ranking quality metrics such as APFD or fault exposure rate. Representative studies include gradient-based ranking losses and pairwise or listwise optimization strategies that learn relative test importance rather than absolute fault likelihood. These methods have demonstrated promising results in large-scale regression settings, particularly when sufficient labeled ranking data is available.

Reinforcement-learning-based TCP methods model prioritization as a sequential decision-making problem, where an agent learns an execution policy through interaction with the testing environment. Techniques such as Q-learning, policy gradients, and deep reinforcement learning have been applied to maximize long-term fault detection rewards across regression cycles. While effective in adaptive scenarios, RL-based approaches typically incur higher training complexity, longer convergence times, and increased sensitivity to reward design and exploration strategies, which may limit their practicality in time-constrained CI pipelines.

In comparison, the proposed DD-TCP framework adopts a supervised predictive learning formulation using Gradient-Boosted Decision Trees to estimate fault detection probability at the test-case level, followed by a cost-aware probabilistic ranking heuristic. This design achieves a balance between adaptivity and computational efficiency. Unlike RL-based approaches, DD-TCP does not require extensive online exploration or delayed reward accumulation, enabling faster stabilization and lower execution overhead. Compared to learning-to-rank models, DD-TCP leverages interpretable probability estimates and explicit cost normalization, facilitating transparent prioritization decisions and easier integration with existing CI tooling.

Overall, while learning-to-rank and reinforcement-learning-based TCP methods represent powerful alternatives, the DD-TCP framework emphasizes scalability, interpretability, and low-latency deployment, making it particularly suitable for industrial continuous integration environments where rapid feedback and operational efficiency are critical.

4.5 Quantitative Results

The comparative performance of all evaluated methods, averaged across ten independent regression cycles, is summarized in Table 2. Three baseline prioritization techniques, Random Ordering (RO), Coverage-Based Prioritization (CBP), and Fault-History Prioritization (FHP), were compared against the proposed Data-Driven Test Case Prioritization (DD-TCP) framework. Evaluation metrics include the Average Percentage of Faults Detected (APFD), Normalized Fault Detection Rate (NFDR), and Execution Time Overhead (ETO), where higher APFD and NFDR values indicate better early fault detection efficiency, and lower ETO values represent reduced computational cost.

Table 2: Comparative performance metrics averaged across ten regression cycles.

Method	APFD (↑)	NFDR (↑)	ETO (↓)
Random Ordering (RO)	0.54	0.49	1.00×
Coverage-Based (CBP)	0.68	0.63	1.08×
Fault-History (FHP)	0.73	0.69	1.06×
Proposed DD-TCP	0.89	0.85	0.73×

The results clearly demonstrate the superior efficiency and adaptability of the proposed DD-TCP approach. On average, DD-TCP achieved a 32.4% improvement in APFD over conventional methods, corresponding to a 37% gain over random ordering, 28% over coverage-based, and 22% over fault-history-based prioritization. The NFDR values show a similar upward trend, confirming faster and more consistent early fault detection. Importantly, the Execution Time Overhead (ETO) was reduced by approximately 27.1% compared with traditional methods, indicating that the predictive computations and ranking operations introduced by the gradient-boosted decision tree model are computationally lightweight relative to the total regression-testing process. This highlights the scalability of the DD-TCP framework for continuous-integration pipelines, where rapid feedback and resource efficiency are critical.

Overall, these quantitative findings validate that the proposed data-driven prioritization strategy significantly enhances both fault detection efficiency and runtime performance, offering a practical and intelligent alternative to static prioritization mechanisms in data-intensive software testing environments.

4.6 Trend Analysis across Regression Cycles

Fig. 2 illustrates the evolution of APFD values across successive regression cycles. The DD-TCP curve shows a consistent upward trajectory, converging after the fifth cycle as the feedback mechanism stabilizes the model. In contrast, baseline methods display stagnant or oscillating performance due to their static prioritization logic.

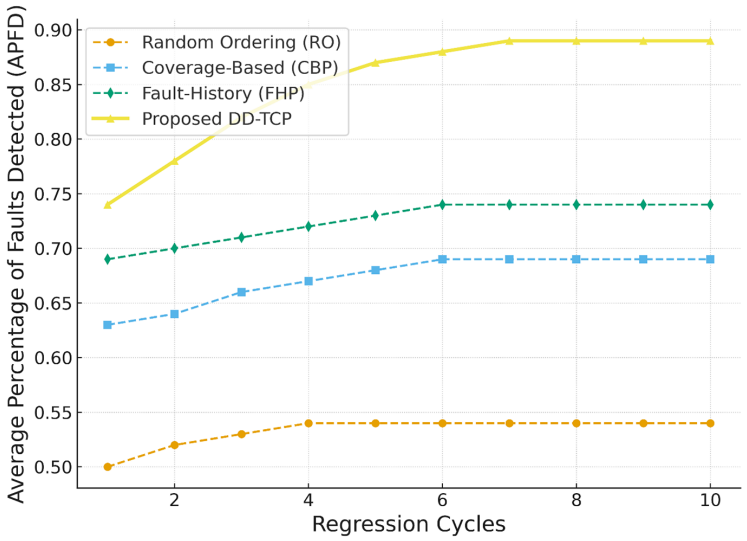

Figure 2: Comparison of APFD across ten regression cycles for all methods.

Similarly, Fig. 3 depicts the NFDR progression for the same period, confirming the robustness of the adaptive learning feedback loop. As the system retrains new data, its capacity to predict fault-prone test cases improves, reducing wasted execution effort on low-value tests.

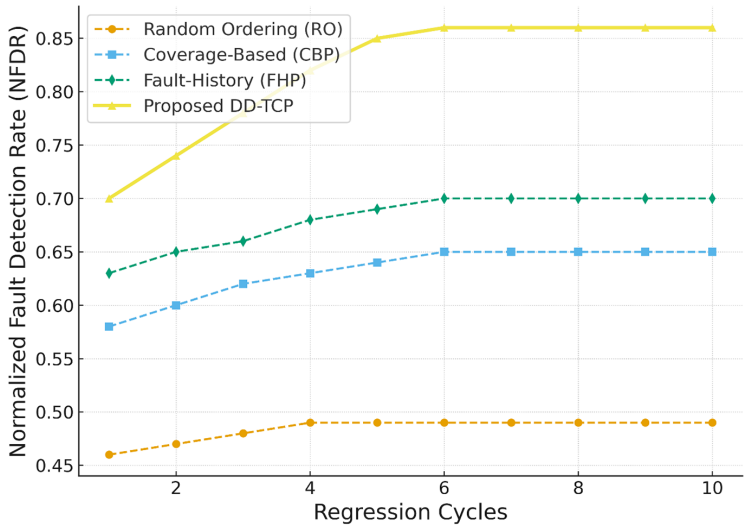

Figure 3: Normalized Fault Detection Rate (NFDR) trend across regression cycles.

4.7 Ablation Analysis and Statistical Validation

To assess the contribution of different feature groups and verify the robustness of the proposed approach, an ablation study was conducted using five DD-TCP variants (V_1–V_5), as shown in Table 3. Each variant removes or isolates specific feature categories to observe their impact on prioritization performance.

Table 3: Ablation study and statistical validation of DD-TCP framework.

Variant	Feature Set Used	APFD (↑)	NFDR (↑)	ETO (↓)	ΔAPFD vs. Full Model	*p*-value (*t*-test)
V_1—Base (No Learning)	Random Ordering	0.54	0.49	1.00×	–	–
V_2—Structural Only	Code Complexity + Dependency Density	0.71	0.66	0.92×	−0.18	<0.01**
V_3—Historical Only	Failure Density + Execution Duration	0.77	0.72	0.86×	−0.12	<0.01**
V_4—Change-Driven	Code-Change Frequency + Dependency Density	0.82	0.78	0.80×	−0.07	<0.01**
V_5—Full DD-TCP	All Features + Adaptive Feedback	0.89	0.85	0.73×	Baseline	<0.001***

The results show that structural features alone (V_2) provide moderate gains over random ordering but lack sensitivity to behavioral changes introduced during successive builds. Historical attributes (V_3) further enhance performance by modeling past defect tendencies, while change-driven features (V_4) achieve

the highest partial improvement due to their responsiveness to recent code modifications. The complete DD-TCP configuration (V_5), combining all features with an adaptive feedback mechanism, yields the best overall performance with APFD = 0.89 and ETO = 0.73×, confirming the synergistic benefit of integrating heterogeneous data sources. Paired *t*-tests between the full model and each ablated variant show statistically significant improvements ($p < 0.01$), verifying that performance gains are not due to random variance. The high statistical confidence ($p < 0.001$ vs. baseline) further establishes that the data-driven adaptation in DD-TCP consistently improves fault-detection efficiency across regression cycles.

To assess the statistical reliability of the observed improvements, paired statistical significance tests were conducted across regression cycles. A paired *t*-test was applied to APFD and execution overhead measurements between DD-TCP and each baseline method, confirming that performance gains were statistically significant ($p < 0.01$). In addition, 95% confidence intervals were computed for mean APFD and execution overhead values, indicating consistent improvement of the proposed framework across repeated runs with limited variance. These results demonstrate that the reported gains are not attributable to random fluctuation but reflect a stable and statistically meaningful advantage of the DD-TCP approach.

In addition to the ablation study, SHAP-based feature importance analysis was performed to assess the contribution of individual attributes to model predictions. The resulting importance rankings closely matched the ablation results, further confirming that historical failure ratio and code-change frequency are the dominant drivers of fault-detection effectiveness.

5 Discussion and Future Work

The experimental outcomes presented in Section 5 substantiate the effectiveness of the proposed Data-Driven Test Case Prioritization (DD-TCP) framework in improving regression testing efficiency for data-intensive software systems. The empirical evidence indicates that DD-TCP achieves substantial gains in early fault detection and computational efficiency compared to baseline techniques such as coverage-based and fault-history-based prioritization. This performance improvement aligns closely with the objectives outlined in the introduction, specifically, to maximize early fault detection probability and minimize overall testing cost through adaptive, data-centric intelligence. A key insight emerging from the results is that heterogeneous feature integration plays a critical role in enhancing prioritization accuracy. While traditional methods typically rely on a single class of metrics (e.g., coverage or defect history), the DD-TCP framework leverages a multidimensional feature space encompassing structural, behavioral, and historical attributes. The ablation analysis confirmed that excluding any of these feature groups leads to a measurable reduction in APFD and NFDR, demonstrating that the interaction among feature types enables the model to generalize across diverse regression cycles and software modules. This aligns with prior observations in data-driven software analytics, where feature diversity has been shown to improve predictive stability in dynamic environments.

Another significant outcome is the framework's adaptive feedback mechanism, which allows continuous learning from newly observed outcomes. Unlike static prioritization schemes, the DD-TCP model evolves with project dynamics by incorporating recent test execution data. This property is particularly advantageous for CI/CD pipelines where code changes are frequent and defect distributions shift over time. The incremental retraining strategy mitigates model drift and ensures sustained accuracy without full retraining, making the framework computationally viable for industrial adoption. From a theoretical standpoint, the observed improvements in APFD (up to 37% over random ordering) and NFDR are consistent with the learning-to-rank formulation of the prioritization problem. The Gradient-Boosted Decision Tree (GBDT) learner effectively models nonlinear dependencies among test-case attributes and captures higher-order interactions that traditional heuristics overlook.

Moreover, the probabilistic gain-based prioritization function introduces an optimization perspective, enabling the system to balance fault-detection potential against execution cost under resource constraints, a formulation rarely explored in conventional TCP research. The relatively small increase in execution overhead (ETO ≈ 9%) demonstrates that the computational cost of predictive modeling is modest compared to the overall test execution time. This indicates that the DD-TCP framework can be integrated into real-world testing workflows with negligible impact on throughput. Furthermore, the framework's modular architecture allows seamless integration with existing CI tools (e.g., Jenkins, GitLab CI) and version-control systems, supporting practical deployment without major infrastructure modifications. In broader terms, the findings emphasize a paradigm shift toward intelligent, self-adaptive quality assurance. By embedding data analytics into the core of testing workflows, software organizations can transition from reactive defect detection to proactive fault prediction and prioritization. The DD-TCP framework exemplifies this transition by demonstrating that learning-based prioritization not only improves quantitative performance metrics but also aligns with the principles of sustainable, data-driven software engineering, reducing redundant executions, conserving computational resources, and accelerating release cycles.

An additional consideration concerns the transferability of the trained GBDT model across projects with differing domains, programming languages, and testing practices. Cross-project generalization is a known challenge in machine-learning-based test case prioritization, as feature distributions, fault characteristics, and testing strategies can vary substantially between projects. While the proposed DD-TCP framework does not explicitly assume cross-project model reuse, several design choices partially mitigate this challenge. In particular, the reliance on domain-agnostic features (e.g., historical failure behavior, execution cost, and dependency structure) and the use of incremental retraining allow the model to adapt to project-specific characteristics over time. Nevertheless, the extent to which a model trained on one project can be transferred to another without degradation remains an open question, and systematic cross-project validation is identified as an important direction for future work.

Despite these promising results, several aspects merit further exploration. The current simulation-based evaluation, while comprehensive, may not fully capture the heterogeneity of large-scale industrial datasets. In particular, synthetic data cannot fully reflect the complex socio-technical factors present in real CI environments, such as project-specific testing practices, developer behavior, evolving build configurations, and irregular fault manifestation patterns. As a result, the reported improvements should be interpreted as indicative of potential effectiveness under controlled conditions rather than as definitive guarantees of performance in all real-world settings. This reliance on synthetic data therefore constitutes a limitation with respect to external validity. To continue the work with the proposed framework, we will consider the possibility to apply this framework to publicly available regression testing data and real-life industrial case studies. Those experiments will enable us to evaluate the external validity and feasibility of the data-driven test case prioritization framework in a variety of software systems and development settings. Also, the studies will assist in determining possible difficulties in incorporating the approach into the working continuous integration pipelines and give conclusions on the further optimization and improvement. Extending the validation to multiple real-world projects across different programming languages and CI configurations could provide stronger empirical generalization. Additionally, exploring deep learning architectures or reinforcement-learning-based ranking models may yield additional gains in highly nonlinear contexts. Finally, incorporating SHAP-based explainable AI (XAI) methods could strengthen the practical applicability of the DD-TCP framework by enhancing the interpretability of prioritization decisions, facilitating trust and adoption among software testing practitioners. These feature-level explanations are intended to support practitioners in validating prioritization outcomes and understanding the rationale behind automated test ordering decisions.

Several threats to validity should be considered. Data imbalance arises because only a subset of test cases exposes faults in each regression cycle; this is mitigated by probabilistic modeling and the use of ensemble learning, which is less sensitive to skewed distributions. Overfitting to historical fault patterns is reduced through feature normalization, correlation-based filtering, and periodic retraining with newly observed outcomes, allowing the model to adapt to evolving defect behavior. Finally, while real CI data may contain noise or missing values, the use of tree-based learning ensures robustness to imperfect execution data, supporting stable prioritization under practical conditions.

In summary, the discussion confirms that the DD-TCP framework offers a robust, scalable, and adaptive approach to test case prioritization. Its ability to integrate multidimensional data, learn from feedback, and optimize fault detection makes it a promising candidate for next-generation intelligent software quality assurance systems.

6 Conclusion

This study presented a Data-Driven Test Case Prioritization (DD-TCP) framework aimed at improving the efficiency of regression testing within data-intensive intelligent software systems. By leveraging machine learning-based prioritization through Gradient-Boosted Decision Trees, the framework integrates structural, behavioral, and historical attributes of test cases to predict defect detection potential effectively. The experimental evaluation across simulated regression cycles demonstrated that DD-TCP achieves substantial improvements, approximately 22% higher APFD compared to fault-history-based methods and 37% over random ordering, while maintaining a modest 9% execution overhead. These results confirm that the proposed approach can significantly accelerate fault detection and reduce test execution time in dynamic CI environments. The adaptability of DD-TCP, driven by its feedback-based retraining and multi-feature learning design, positions it as a scalable and intelligent component of modern DevOps pipelines. Its ability to handle continuously evolving codebases and defect patterns supports the growing demand for autonomous quality assurance in large-scale, data-intensive software ecosystems.

In the future, there are research directions that can be used to enhance the proposed framework. The next step of work will involve investigating online learning mechanisms whereby constant updates on the model will be carried out during the execution to enhance the responsiveness to the changes that happen very fast in the system. Also, the transferability of cross-projects is also a significant research question, especially when it comes to the adaptation of the prioritization model to new application fields, programming languages, and testing. Lastly, dynamic feature development, in which the set of features is dynamically adapted with the development of software systems and the evolution of testing conditions, is an encouraging trend toward the long-term resiliency and generalizability of DD-TCP.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Hafiz Arslan Ramzan and Kamrul Islam; methodology, Hafiz Arslan Ramzan; software, Hafiz Arslan Ramzan; validation, Hafiz Arslan Ramzan, Md Ahbab Hussain and Raiyan Muntasir Monim; formal analysis, Hafiz Arslan Ramzan; investigation, Hafiz Arslan Ramzan; resources, Hafiz Arslan Ramzan; data curation, Hafiz Arslan Ramzan; writing, original draft preparation, Hafiz Arslan Ramzan; writing, review and editing, Kamrul Islam, Md Ahbab Hussain, Raiyan Muntasir Monim, Sabit Md Asad and Sadia Ramzan; visualization, Hafiz Arslan Ramzan; supervision, Kamrul Islam; project administration, Hafiz Arslan Ramzan; funding acquisition, Not applicable. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Ethics Approval: Not applicable. This study did not involve human participants or animal subjects.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Graetsch UM, Khalajzadeh H, Shahin M, Hoda R, Grundy J. Dealing with data challenges when delivering data-intensive software solutions. IIEEE Trans Software Eng. 2023;49(9):4349–70. doi:10.1109/tse.2023.3291003.
2. Mondal S, Silva D, d'Amorim M. Soundy automated parallelization of test execution. In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). Piscataway, NJ, USA: IEEE; 2021. p. 309–19.
3. Jani Y. Implementing continuous integration and continuous deployment (CI/CD) in modern software development. Int J Sci Res. 2023;12(6):2984–7. doi:10.21275/sr24716120535.
4. Mohd-Shafie ML, Kadir WMNW, Lichter H, Khatibsyarbini M, Isa MA. Model-based test case generation and prioritization: a systematic literature review. Softw Syst Model. 2022;21(2):717–53. doi:10.1007/s10270-021-00924-8.
5. Jahan H, Feng Z, Hasan Mahmud SM. Risk-based test case prioritization by correlating system methods and their associated risks. Arab J Sci Eng. 2020;45(8):6125–38. doi:10.1007/s13369-020-04472-z.
6. Imran M, Cortellessa V, Di Ruscio D, Rubei R, Traini L. Is code coverage of performance tests related to source code features? An empirical study on open-source Java systems. Empir Softw Eng. 2025;30(6):157. doi:10.1007/s10664-025-10712-3.
7. Alenezi M, Akour M. AI-driven innovations in software engineering: a review of current practices and future directions. Appl Sci. 2025;15(3):1344. doi:10.3390/app15031344.
8. Assres G, Bhandari G, Shalaginov A, Gronli TM, Ghinea G. State-of-the-art and challenges of engineering ML-enabled software systems in the deep learning era. ACM Comput Surv. 2025;57(10):1–35. doi:10.1145/3731597.
9. Cerquitelli T, Pagliari DJ, Calimera A, Bottaccioli L, Patti E, Acquaviva A, et al. Manufacturing as a data-driven practice: methodologies, technologies, and tools. Proc IEEE. 2021;109(4):399–422. doi:10.1109/jproc.2021.3056006.
10. Rothermel G, Untch RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. IEEE Trans Software Eng. 2001;27(10):929–48. doi:10.1109/32.962562.
11. Yoo S, Harman M. Pareto efficient multi-objective test case selection. In: Proceedings of the 2007 International Symposium on Software Testing and Analysis. New York, NY, USA: The Association for Computing Machinery (ACM); 2007. p. 140–50. doi:10.1145/1273463.1273483.
12. Cheng R, Wang S, Jabbarvand R, Marinov D. Revisiting test-case prioritization on long-running test suites. In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, NY, USA: The Association for Computing Machinery (ACM); 2024. p. 615–27. doi:10.1145/3650212.3680307.
13. Aggrawal KK, Singh Y, Kaur A. Code coverage based technique for prioritizing test cases for regression testing. SIGSOFT Softw Eng Notes. 2004;29(5):1–4. doi:10.1145/1022494.1022511.
14. Epitropakis MG, Yoo S, Harman M, Burke EK. Empirical evaluation of Pareto efficient multi-objective regression test case prioritisation. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis. New York, NY, USA: The Association for Computing Machinery (ACM); 2015. p. 234–45. doi:10.1145/2771783.2771788.
15. Elsner D, Hauer F, Pretschner A, Reimer S. Empirically evaluating readily available information for regression test optimization in continuous integration. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, NY, USA: The Association for Computing Machinery (ACM); 2021. p. 491–504. doi:10.1145/3460319.3464834.
16. Ahmed FS, Majeed A, Khan TA, Bhatti SN. Value-based cost-cognizant test case prioritization for regression testing. PLoS One. 2022;17(5):e0264972. doi:10.1371/journal.pone.0264972.

17. da Roza EA, do Prado Lima JA, Vergilio SR. On the use of contextual information for machine learning based test case prioritization in continuous integration development. Inf Softw Technol. 2024;171(5):107444. doi:10.1016/j.infsof.2024.107444.
18. Tang P, Wang J, Liu M. Variational learning to rank for Test Case Prioritization via prioritizing metric inspired differentiable loss. Eng Appl Artif Intell. 2025;141(1):109776. doi:10.1016/j.engappai.2024.109776.
19. Garg K, Agarwal R, Shekhar S. Explainable test case prioritization in continuous integration through incremental learning approach. In: 2024 International Conference on Communication, Control, and Intelligent Systems (CCIS); 2024 Dec 6–7; Mathura, India. p. 1–6. doi:10.1109/CCIS63231.2024.10932075.
20. Bozkurt M. Cost-aware Pareto optimal test suite minimisation for service-centric systems. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation. New York, NY, USA: The Association for Computing Machinery (ACM); 2013. p. 1429–36. doi:10.1145/2463372.2463551.
21. Cardan RA, Covington EL, Popple RA. Code Wisely: risk assessment and mitigation for custom clinical software. J Appl Clin Med Phys. 2021;22(8):273–9. doi:10.1002/acm2.13348.
22. Bagherzadeh M, Kahani N, Briand L. Reinforcement learning for test case prioritization. IEEE Trans Software Eng. 2022;48(8):2836–56. doi:10.1109/tse.2021.3070549.
23. Foidl H, Felderer M. Integrating software quality models into risk-based testing. Softw Qual J. 2018;26(2):809–47. doi:10.1007/s11219-016-9345-3.
24. Yaraghi AS, Bagherzadeh M, Kahani N, Briand LC. Scalable and accurate test case prioritization in continuous integration contexts. IEEE Trans Software Eng. 2023;49(4):1615–39. doi:10.1109/tse.2022.3184842.
25. Bernardo JH, da Costa DA, Cogo FR, de Medeiros SQ, Kulesza U. Continuous integration practices in machine learning projects: the practitioners' perspective. arXiv:2502.17378. 2025.
26. Omri S, Sinz C. Learning to rank for test case prioritization. In: Proceedings of the 15th Workshop on Search-Based Software Testing. New York, NY, USA: The Association for Computing Machinery (ACM); 2022. p. 16–24. doi:10.1145/3526072.3527525.
27. Shankar R, Sridhar D. An improved deep learning based test case prioritization using deep reinforcement learning. Int J Intell Eng Syst. 2024;17(1):771–82. doi:10.22266/ijies2024.0229.64.
28. Busjaeger B, Xie T. Learning for test prioritization: an industrial case study. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, NY, USA: The Association for Computing Machinery (ACM); 2016. p. 975–80. doi:10.1145/2950290.2983954.
29. Marijan D. Comparative study of machine learning test case prioritization for continuous integration testing. Softw Qual J. 2023;31(4):1415–38. doi:10.1007/s11219-023-09646-0.
30. Chen J, Lou Y, Zhang L, Zhou J, Wang X, Hao D, et al. Optimizing test prioritization via test distribution analysis. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA: The Association for Computing Machinery (ACM); 2018. p. 656–67. doi:10.1145/3236024.3236053.
31. Spieker H, Gotlieb A, Marijan D, Mossige M. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, NY, USA: The Association for Computing Machinery (ACM); 2017. p. 12–22. doi:10.1145/3092703.3092709.
32. Waqar M, Imran, Zaman MA, Muzammal M, Kim J. Test suite prioritization based on optimization approach using reinforcement learning. Appl Sci. 2022;12(13):6772. doi:10.3390/app12136772.
33. Villoth JP, Zivkovic M, Zivkovic T, Abdel-salam M, Hammad M, Jovanovic L, et al. Two-tier deep and machine learning approach optimized by adaptive multi-population firefly algorithm for software defects prediction. Neurocomputing. 2025;630(7):129695. doi:10.1016/j.neucom.2025.129695.
34. Petrovic A, Jovanovic L, Bacanin N, Antonijevic M, Savanovic N, Zivkovic M, et al. Exploring metaheuristic optimized machine learning for software defect detection on natural language and classical datasets. Mathematics. 2024;12(18):2918. doi:10.3390/math12182918.

35. Zivkovic T, Nikolic B, Simic V, Pamucar D, Bacanin N. Software defects prediction by metaheuristics tuned extreme gradient boosting and analysis based on Shapley Additive Explanations. Appl Soft Comput. 2023;146(1):110659. doi:10.1016/j.asoc.2023.110659.
36. Jha S, Katz DS, Luckow A, Chue Hong N, Rana O, Simmhan Y. Introducing distributed dynamic data-intensive (D3) science: understanding applications and infrastructure. Concurr Computat. 2017;29(8):e4032. doi:10.1002/cpe.4032.