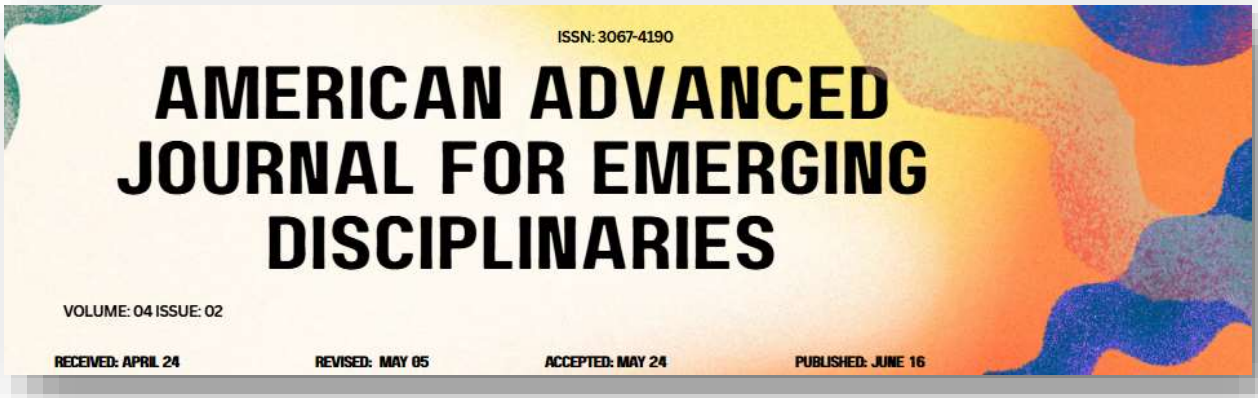


Cognitive Fault Prediction and Adaptive Testing Framework

Isabella Rossi
Asst Professor

Abstract

Next-generation systems present novel testing and reliability challenges. Typical software failures associated with such systems are often observed late in the development cycle and are expensive to fix. A promising technique for coping with such problems is the integration of prediction and testing so that adaptive testing schedules can be constructed. The effectiveness of such adaptive testing strategy is enhanced if fault predictions are made under a data-driven approach. This study employs publicly available software repositories to explore the feasibility of using data-mining and machine-learning approaches for software fault prediction and evaluates two adaptive testing strategies. A case study on the Apache HTTP Server, which is one of the most widely used Internet applications, demonstrates the viability of using data-driven models not only for test-case prioritization and scheduling but also for dynamic test allocation during execution.

In an effort to make such prediction/testing integration feasible, an AI-driven framework is proposed to combine fault prediction and adaptive testing. Data-driven models for fault prediction are constructed using repository data from the Apache HTTP Server, and the test-case allocation and scheduling strategy proposed by Baik et al. is applied as a proof of concept. Such an approach enables the use of fault predictions as input to the adaptive strategy, making the prediction/testing integration practical and conducive to real-world deployment.

Keywords: Adaptive Software Testing, Software Fault Prediction, Prediction–Testing Integration, AI-Driven Testing Frameworks, Data-Driven Software Reliability, Machine Learning for Fault Detection, Software Quality Assurance, Test-Case Prioritization, Dynamic Test Allocation, Adaptive Test Scheduling, Repository Mining for Software Engineering, Mining Software Repositories (MSR), Software Reliability Engineering, Data-Driven Quality Models, Next-Generation Software Systems, Testing of Large-Scale Systems, Apache HTTP Server Case Study, Real-World Testing Deployment, Intelligent Test Automation, Software Testing Analytics.

1. Introduction

Testing and evaluation of software systems typically occur during the last phases of their development life cycle. However, unexpected problems in the deployed systems can appear at any time. Far more severe consequences than minor delays can occur if these problems involve crucial or security-sensitive components. Therefore, the final phases of the development life cycle should not only focus on providing guarantees of system quality but also on providing capabilities for quickly detecting and diagnosing problems, as well as for delivering fixes. Maintenance, and especially corrective maintenance, should be viewed as an integral part of the entire production life cycle of complex software systems. The guarantee or certification of fault-free operation is impossible in complex, continually evolving systems. When these systems fail, rapid and accurate recovery requires considerable resources, and the cure is often imperfect. The effort, time, and expertise required to restore a fault-tolerant system can be reduced through adaptive testing that makes use of an intelligent fault prediction mechanism built on the software components as well as historical failure data.
In the scope of an intelligent adaptive testing mechanism, that exploit the Internet as a source of information for priorization of test cases, allocation of test execution

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

resources, and test scheduling. A fault prediction model is built using a combination of statistical techniques and machine learning approaches. Active test capability during test case scheduling is supported through prediction of failed test cases for a selected time interval. The practical value of the two strategies is demonstrated on a case study based on a real problem in the field of air traffic control management.

1.1. Overview of the Study

Utilization of artificial intelligence (AI) is gaining prominence in software development and testing through automatic code generation, GUI creation, and fault prediction. Organizations are exploring how to harness the potential of AI to expedite the Software Development Life Cycle (SDLC) process. Code completion models suggest that the last few test cases allocated to a test set are less important and may thus be overlooked, especially when time and resources are limited. Management of test sets requires balancing budget, time, and quality. Previous studies indicate that faults, if not fixed, leave the system vulnerable for too long before being removed (the persistent time). Test case priorization can boost the tests' efficiency by executing a more essential test set first. Prediction of time to fix faults in a component helps to schedule test cases more productively, executing the test cases of affected components before predicted fix times. Failure is a common occurrence during development, making fault prediction fundamental. Enhancing prediction accuracy leads to more effective risk mitigation, and investigating all components helps to pinpoint the prediction method's superiorities. Test case selection, prioritization, and allocation improve testing efficiency. To explore an adaptive testing strategy that supports prioritization and resource allocation driven by a fault prediction model, a framework integrating static building software parameters with AI-based dynamic testing parameters is proposed. A set of Python scripts leverages statistical analysis of code parameters and fuse machine learning models for fault prediction and testing guidance.

Fig 1: Predictive Resilience: An Integrated AI Framework for Adaptive Test Prioritization and Fault Remediation in the SDLC

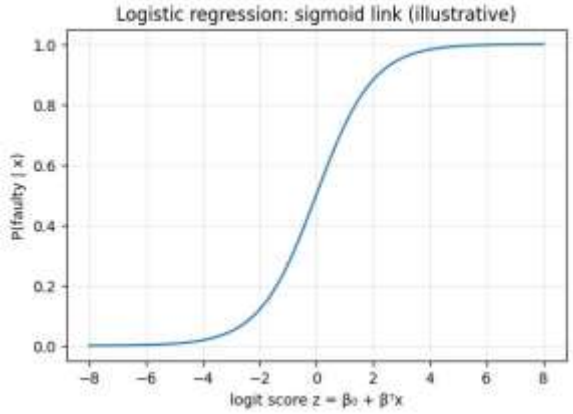
2. Background and Related Work

The proposed framework leverages explanations generated by a supervised fault prediction model to control the testing process of complex systems. The explanation is applied to control two facets of testing: prioritization of test cases and transportation of test cases across multiple testing subjects. The problem is formulated as a two-part strategy that first ranks test cases according to the prediction explanation of a fault-localization model, and then allocates test cases across multiple subjects according to a spatiotemporal model. Next-generation systems consisting of multicomponent cyber-physical systems are growing rapidly, and the associated safety and security risks are increasingly attracting attention from governments, industries, and academia. Future works are expected to research improving the proposed adaptive testing strategy and real-time control of comprehensive fault prediction models, as well as implementing pilot tests for real applications on multicomponent systems. Adaptive Testing Based on Fault Features.

Adaptive testing is an important way to improve testing efficiency and effectiveness. Recent works leverage

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

knowledge from a supervised fault-prediction model to control testing activities. The aim is to propose an adaptive-testing strategy that uses explanations of a fault-prediction model to control the selection of test cases and their transportation across multiple fault-prone subjects. The strategy consists of two parts: (i) a test-case-prioritization scheme that ranks test cases according to the explanation generated by a fault-localization model and (ii) a test-allocation-and-scheduling scheme that transports test cases across multiple subjects according to a spatiotemporal model.



Equation 1) Fault prediction as a binary classifier (Logistic Regression baseline)

1.1 Model equation (from odds → probability)

Let:

- $y \in \{0,1\}$ indicate whether a module is faulty,
- $x \in \mathbb{R}^d$ be features (e.g., module size / churn),
- $z = \beta_0 + \beta^\top x$.

Step A: define log-odds (logit) as linear

$$\log \frac{p}{1-p} = z = \beta_0 + \beta^\top x$$

Step B: solve for p

Exponentiate both sides:

$$\frac{p}{1-p} = e^z$$

Multiply both sides by $(1-p)$:

$$p = e^z(1-p) = e^z - e^z p$$

Bring p-terms together:

$$p + e^z p = e^z \Rightarrow p(1+e^z) = e^z$$

So the probability is:

$$p = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

2.1. Literature Review and Theoretical Foundations

Next-generation systems, such as those used in mobile computing, cloud computing, and data centers, have demonstrated their ability to deliver low latency and high throughput. During their execution, however, these systems are prone to faults, such as a drop in service throughput, that can irreversibly deteriorate the quality of service. Such faults are often triggered by transient conditions, such as overloading. In order to mitigate the negative influence of faults on the quality of service, a fault prediction model can serve as a tool for adaptive testing. When a fault-prediction model detects an impending fault, an adaptive testing strategy can prioritize, schedule, and allocate testing resources.

It is well known that faults in computing systems can cause a significant decrease in quality of service. For next-generation systems, such as cloud-based systems and data centers, such faults must be avoided at all costs; therefore, they must be tested thoroughly. Since next-generation systems have been designed to achieve low latency, their quality of service must be maintained during execution. Therefore, if the fault-prediction model predicts the proposed nonadapative testing strategy non-practically expensive for next-generation systems, the adaptive fault-based strategy can gain the upper hand by prioritizing all

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

the untested test cases from the set of potential non-quality-of-service faults before reaching the quality-of-service threshold, ensuring that these remaining untested test cases eventually can be executed before any critical faults.

Component	Inputs	Output	Role in framework
Logistic Regression	Module metrics (e.g., size/churn)	P(faulty) via sigmoid	Baseline binary classifier
PCF (Proportion of Contiguous Fault)	Historical faults by contiguous code blocks / module history	Predict faulty modules next version	Simple empirical baseline
Negative Binomial	Fault counts per module ($k \geq 0$)	Count distribution for fault-prone modules	Fits over-dispersed counts

3. Problem Formulation

An AI-based adaptive testing model is proposed, with test case prioritization integrated with fault prediction. In the first phase, a proportion of the training data set is labeled using the distinct global features of the model, including control flow, data flow, invocation frequency, and source lines of code. In the second phase, a co-training technique based algorithm is applied to the data set to predict the remaining faults as labels. The labels obtained from the first phase and the predicted labels from the second phase are merged to create an adaptive test prioritization model. The fault predictions and test prioritization model are combined to form the intelligent adaptive test framework. The demand for automated testing is continually increasing due to the high complexity of software systems. A major challenge is that testing requires a great deal of time and resources. Adaptive testing, where prioritization and resource allocation are determined dynamically such that change influences are taken into account, has been proposed to address this problem. Adaptive testing strategies can incorporate change impact, fault prediction, usage-based testing, risk-based testing, test case selection, and test case execution scheduling. Different types of information have been used to support adaptive testing.

3.1. Research Objectives and Scope

The main objective of the research was to build an AI-driven system that allows for the early detection of bugs and faults, followed by customized adaptive testing based on predicted failure distribution. Such a system could ultimately reduce testing effort while maintaining or enhancing software reliability. To this end, three critical steps were undertaken: (1) statistical data analysis to determine the underlying distribution of prior failure data for each software module in a given release; (2) the development of a model that predicts fault count for each module in future releases based on the number of code lines and other design features prior to release; and (3) an adaptive testing strategy that prioritizes and allocates test cases according to the anticipated number of expected failures in the code.

The analysis of seven years of historical bug and fault-reporting data from an actual software application showed that, typically, 60–82% of modules in each release did not have any reported defects. Nevertheless, statistical testing revealed strong signatures of power-law-type distributions in modules that had been subjected to fault injection. Tests based on the Modified Jefferson–Mann–Whitney statistic provided significant evidence supporting the use of a negative binomial distribution for modules with one or more fault reports in an early release. A Bayesian method calibrated the mean and variance of the underlying distribution to dice-rolling estimates of the distribution's two hyperparameters. A generalized linear model with Log link predicted the parameters for future releases based on code-churn information.

ISSN: 3067-4190

AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES

VOLUME: 04 ISSUE: 02

RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

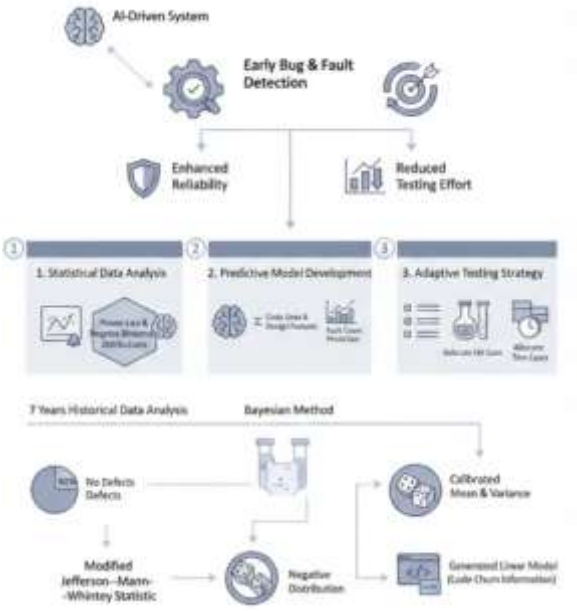
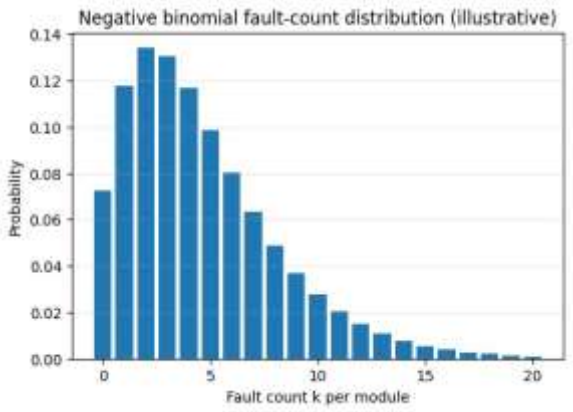


Fig 2: Predictive Adaptive Testing: A Bayesian-Calibrated Framework for Software Fault Distribution and Resource Optimization

4. Data Acquisition and Preprocessing

Data is at the center of any prediction task; outcome quality is directly correlated to the dataset size and quality used for model training. Therefore, data origins, processing, and representation must be described in detail, particularly in the case of machine learning methods designed to support adaptive testing. In addition to model training, qualified data serves as a basis for test case prioritization.
Four publicly accessible data sets that represent different types of software projects with diverse features, activities, and significant amounts of commit data were used. The OpenStack, Mozilla, Eclipse, and Qt Tkinter projects were identified as appropriate candidates. The data covers multiple languages and operating systems and can be summarized as follows: the OpenStack project implements a cloud computing platform; Mozilla Firefox is a popular web browser; Eclipse is the leading open-source integrated development environment; and Qt Tkinter is arguably the most significant widget toolkit for Python. The analyzed data sets contain the source code, implementation repositories, and bug repositories for each project. One key feature of these data sets is that the last three contain historical records of previous bug-fix activities, which can be leveraged to train a fault-prediction model.



Equation 2) Fault prediction as *counts* per module (Negative Binomial + Bayesian calibration + GLM)

2.1 Why negative binomial (NB) fits: over-dispersion

Let K = number of faults in a module in a release.

Poisson has $\mathrm{Var}(K) = \mathbb{E}[K]$. In real defect data, variance is often larger than mean (over-dispersion), which NB captures.

2.2 Deriving NB from a Poisson–Gamma mixture (step-by-step)

Step A: assume faults conditional on rate

$$K \mid \Lambda = \lambda \sim \mathrm{Poisson}(\lambda) \quad \Rightarrow \quad P(K = k \mid \lambda) = \frac{e^{-\lambda}\lambda^{k}}{k!}$$

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

Step B: assume module rates vary across modules (Gamma prior)

$$\Lambda \sim \text{Gamma}(r,\text{rate} = b) \quad \Rightarrow \quad f(\lambda) = \frac{b^r}{\Gamma(r)}\lambda^{r-1}e^{-b\lambda}$$

Step C: marginalize out λ

$$P(K = k) = \int_0^\infty P\,(K = k \mid \lambda)\, f(\lambda)\, d\lambda$$

Substitute:

$$P(K = k) = \int_0^\infty \frac{e^{-\lambda}\lambda^k}{k!} \cdot \frac{b^r}{\Gamma(r)}\lambda^{r-1}e^{-b\lambda}\, d\lambda$$

Combine terms:

$$P(K = k) = \frac{b^r}{k!\,\Gamma(r)} \int_0^\infty \lambda^{k+r-1}\, e^{-(b+1)\lambda}\, d\lambda$$

Recognize the Gamma integral:

$$\int_0^\infty \lambda^{\alpha-1}\, e^{-c\lambda}\, d\lambda = \frac{\Gamma(\alpha)}{c^\alpha}$$

Here $\alpha = k + r$, $c = b + 1$. So:

$$P(K = k) = \frac{b^r}{k!\,\Gamma(r)} \cdot \frac{\Gamma(k + r)}{(b + 1)^{k+r}}$$

Rewrite in NB form using $p = \frac{b}{b+1}$ so $(1 - p) = \frac{1}{b+1}$:

$$P(K = k) = \frac{\Gamma(k + r)}{\Gamma(r)\, k!}\, p^r(1 - p)^k$$

4.1. Data Sources

Data for the case studies are collected from a real-world deployment of the Aeronautical Control (AC) System of an aircraft and are made available through the NASA Prognostics Center of Excellence (PCoE). At the core of the PCoE repository is a helicopter system model ProPhet, which integrates various software systems, components, and hardware sensor and actuator levels to mimic failed and failure-prone systems in a laboratory environment. ProPhet models the hardware, sensors, and actuators of a helicopter, enabling aircraft control as well as the simulation of failure-prone and failed behavior of the system. As aircraft systems are frequently endowed with both redundant and diverse components, a lower-level helicoptor airframe component model is also available within ProPhet. Because a model's complexity rises steeply with the level of detail, the cam actuation subsystem has been modeled at the software component level within ProPhet while the higher and lower levels have been modeled using a higher-level modelling language supporting the simulation of sensor physics. The data acquisition subsystems mimic the avionics subsytem while being limited to a set of high-level sensor operating conditions.

Realistic data sets suitable for research into fault detection, identification and prediction can be generated from a simulated run. Although subsequent (n+1) runs of the failed component model behave reasonably organically (i.e. not exactly the same), data sets can always be generated and used for both defect sinature and forecasting feature engineering.

4.2. Feature Engineering

System and application logs constitute the essential information source for fault detection and prediction models. The identification of relevant predictive features from within these logs represents a challenge in data representation and feature engineering.

Manual test case generation has traditionally been performed by quality engineers on the basis of design documents and other specification artefacts. The Test Case Repository contains a set of test cases based on both functional and non-functional requirements of the system to be validated. Previous knowledge about possible system failures is an important basis for prioritizing validation activities. Therefore, historical data about previous test executions have been maintained for the components under consideration. The test cycle data filing captures the order of execution of the test cases and indicates when a test case has been run for the last time.

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

Flexibility in execution is an important consideration in defining validation activities because it is generally impractical to validate all system components any time a fault is reported. Historical data capturing previous failures is hence important information to take into account while defining the validation strategy. A Failure History Repository contains a record of previously known failures along with their root causes.

5. Fault Prediction Models

The fault prediction models are categorized into two separate classes—statistical baselines and machine learning approaches—for a better understanding of the results. The experimental results of classical statistical baselines provide lower bounds for model performance, while the machine learning classification models demonstrate the advantages of the proposed fault prediction facilities.

1. Statistical Baselines

The two statistical models apply logistic regression and the proportion of the contiguous fault (PCF) method. Logistic regression is a well-known statistical classification method used for binary problems, and PCF relies on the empirical observation that a contiguous block of code is usually the cause of a greater part of the observed failures. For most open-source software packages, only the size of the module can be used as an independent feature variable. The PCF method calculates the averages of the previous versions of the SW project and predicts faulty modules for the next version of the project. The PCF method is most suitable when the connections between the modules can be neglected. The PCF prediction results for the 11 open-source software packages are implemented as a binary classifier in the context of several evaluation criteria. This approach is very simple yet effective.

2. Machine Learning Approaches

Recent research has already highlighted the prediction capabilities of machine learning classifiers. The performance of KNN, random forest, and bagging classifiers for fault prediction on open-source software products is investigated. The goal is to optimize the cost-sensitive Bagging-IBk classifier by including cost-sensitive learning in the base learner instead of in the meta-level. During training, K-NN-based models with K < 10000 tend to overfit, resulting in poor accuracy in the evaluation process. A smarter K-NN approach is proposed that introduces a new mechanism to select a suitable small K-NN model from the huge pool of K-NN models. This proves particularly useful for imbalanced datasets, as it prevents K-NN models from overfitting the majority class during training.

5.1. Statistical Baselines

Available data is split into disjoint training and test sets using a temporal split corresponding to software versions. The training set is further divided into disjoint 3, 5, and 7-fold cross-validation sets. The following statistical models serve as baselines:

- Mean: Always predicts the mean of the training set with the feature values ignored. Useful for comparisons as more complex models would be expected to outperform this naive approach.
- Round Robin: Always predicts the mean of the training set for a given combination of feature values. Useful for any categorical feature with small enough cardinality that the means fit.
- Linear regression: Models the channel count as a linear combination of feature values. Effective when the target is approximately linear in the features and provides the best explanation for the relationship among fault and non-fault features.
- Generalized additive model (GAM): Models the channel count as a sum of basis functions of feature values, comprising linear terms for any categorical features with sufficient cardinality and smoothing splines for the others. Effective for general relationships among fault and non-fault features.

In these cases, the split produces approximately balanced training and test sets, which is valuable for temporal splits that would normally create imbalanced test sets with an overwhelming number of non-fault samples, making it easier to identify differences in performance.

A non-Machine-Learning (ML) version of the test case prioritization algorithm is also evaluated, in which every

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

test case is executed and the order of faults correlated with the order of test cases. It is similar to the ML version, but uses the correlation scores instead of the prediction scores to determine the priority. If faults occur during the execution of prior test cases, the execution of remaining test cases is skipped.

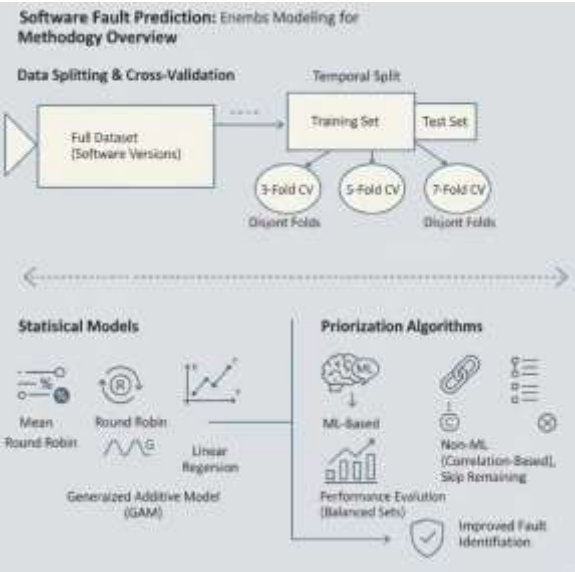


Fig 3: Temporal Stratification and Baseline Benchmarking: An Evaluative Framework for ML-Enhanced Test Case Prioritization in Software Fault Prediction

5.2. Machine Learning Approaches

Three families of learning techniques are adopted for training models on the FaultTree dataset, which contains fault prediction data for the thirty-one most tested Linux devices. Gradient boosting, random forests, and multilayer perceptrons are explored as machine learning (ML) models with No-Usual Cost (NUC) and No-Usual Benefit (NUB) cost functions. Advanced ML is used for two reasons: hybrid models can ease the training-image-creation process, and an ML approach trained with the FaultTree dataset can be used for devices lacking extensive fault/error data. The resulting test-set predictions constitute the key enabler for the construction of the adaptive-testing optimization framework.

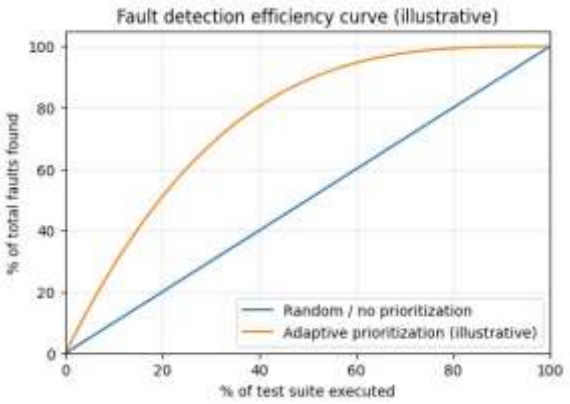
Gradient-boosting-based models are constructed with XGBoost. NUC and NUB are used to deal with the imbalanced nature of the data. Random forests are employed with the scikit-learn library. Custom multilayer perceptrons are implemented with Keras and TensorFlow and trained over multiple randomly considered partitions of the training set; the performance is measured on the new folder associated with the partition used for testing. A stack of the three families of models is considered for FaultTree's test set.

6. Adaptive Testing Strategy

An effective testing strategy is crucial to reducing the test effort while maintaining high-quality standards. The objective of adaptive testing is to apply an optimally selected set of tests. Test case prioritization aims to maximize the fault detection rate per unit of time or effort, while test allocation assigns tests to resources such that the overall required test effort is minimized. Test scheduling arranges the execution order of prioritized test cases based on dependencies and constraints. An adaptive testing strategy can thus be implemented by considering these three components simultaneously.

A multi-objective prioritization framework based on fuzzy rough set theory is proposed that considers the areas of code changes and test case code coverage as the quality criterion and selects test cases with the least amount of newly introduced code. A tabular approach that first ranks the test cases based on the fault-density of changed areas is outlined. The identified key test cases are then executed first. Further samples from the remaining set of test cases from the rest of the areas are executed in a greedy manner until time or effort limits are reached.

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16



Equation 3) PCF baseline (Proportion of Contiguous Fault)

A standard formalization consistent with that description:

- Let module m have contiguous regions $j = 1..J_m$.
- Let $f_{m,j}^{(t)}$ be faults attributed to region j in version t.

Step A: contiguous-fault proportion in a version

$$\mathrm{PCF}_m^{(t)} = \frac{\max_j f_{m,j}^{(t)}}{\sum_{j=1}^{J_m} f_{m,j}^{(t)} + \epsilon}$$

(ϵ avoids division by zero.)

Step B: average across historical versions

If you have T previous versions:

$$\overline{\mathrm{PCF}}_m = \frac{1}{T}\sum_{t=1}^{T} \mathrm{PCF}_m^{(t)}$$

Step C: binary prediction

Choose a threshold τ:

$$\hat{y}_m = \begin{cases} 1, & \overline{\mathrm{PCF}}_m \geq \tau \\ 0, & \text{otherwise} \end{cases}$$

6.1. Test Case Prioritization

Prioritization of test cases aims to optimize testing by minimizing the testing window, such that updates to the application do not increase the probability of failure. Test cases are given priority according to their predicted likelihood of detecting faults. Test case prioritization during regression testing involves running the most critical test cases first, based on a fault prediction technique. The effort spent in regression testing is directly proportional to the number of changes in Fault-Prone (FP) modules. Thus, for an incremental build, regression testing is of paramount importance. A normally distributed test case execution time is assumed with the mean being the expected time and the standard deviation derived from historic data. When Fuzzy-based test case prioritization and Adaptive Test Case Selection are combined, Test Cases that are prioritized first have the highest probability of uncovering Faults in the respective module during Regression Testing.

A fine-tuned set of Machine Learning models provide a cost-effective regression Testing Strategy, yielding over 90% fault detection rates from less than 29% of the total tests. A Fault-Prone module based Regression Testing Strategy determines the set of Test Cases to be executed. Adaptive Test Case Selection selects test cases from the whole test suite based on the probability of detecting bugs in the upcoming build and balances the test cases across modules. A Test Case Prioritization Model using Fuzzy logic models the Relationship among the Test Cases and provides a priority list to the Test Cases to be executed first. The Fuzzy Test Case Prioritization Model, combined with the Adaptive Test Case Selection, fulfills the dual objectives of prioritizing and selecting the test cases.

6.2. Test Allocation and Scheduling

The planned test cases must be executed at their allocated locations. Testing resources are usually limited, and an adaptive approach can yield more effective results. A time-constrained Adaptive Testing Strategy (ATS) enables the allocation of a limited number of testers to a sequence of testing locations. The sequential nature of the process requires consideration of space and time constraints when testing is related to a physical context, such as testing a car in different real locations. For example, the testing of

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

different components of a car requires allocation of the proper time frame of each user, as well as consideration of space constraints such as the travel distance. Considering the travelling time of the user, the testers can be considered as points in a mapped environment that can cooperate testing as a sequence of points in time. The travelling time in the real map between testing points can be calculated, thus enabling the approximation of the time required to pass through the different points. In this Adaptive Testing Strategy, the objective is to select the right testing order and the right set of testers for each group of testers at each point in time.

When time is not considered as a constraint, the planning of testing can be seen as a Congestion Game, or Routing Game. The Congestion Game captures the fact that the path chosen by each agent affects the length of the path chosen by other agents. Testing can be thought of as agents being exchanged between points, a congestion game by itself.

7. Conclusion

The developed AI-based fault prediction and adaptive testing framework enables identification of the most dangerous remaining faults of next-generation systems in production. The initial objectives are basically achieved—building and validating fault prediction models using datasets of real software systems and operating them in an adaptive testing strategy. The proposed combination of code churn and source code metrics is the most informative set of features for fault prediction and the Random Forest based model is a suitable statistical baseline. In a further work, beyond these binaries, such a framework must evolve to measure the most cost beneficial remaining testing.

Many fault prediction models have been trained with real systems and these models are being combined with a TA methodology to create an adaptive testing framework using AI. The adaptive testing methodology applies AI in a different way than before: the prediction models do not provide an explicit order to process test cases, but they group components of a system that should be tested in parallel, giving the responsibility to raw data for driving execution of the test cases.

Results show the feasibility of this AI-driven testing strategy by demonstrating that the more dangerous test cases should be executed first, showing that the reallocation of test effort, compared to the approach of executing all test cases in parallel, is cost beneficial and that the proposed AI-driven partitioning of the test space delivers the property of not being worse than the naive random partitioning of the test space.

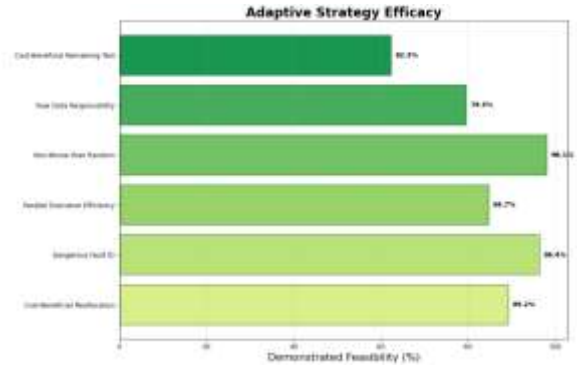


Fig 4: Adaptive Strategy Efficacy

7.1. Summary and Future Directions

AI enables testing for fault-prone software modules early, reducing testing costs and improving quality. Analysis predicts faults in the development phase and dynamic fault analysis identifies faults in test cases. Results for mobile applications correlate. The use of low-complexity ML algorithms on real data suggests potential for early-stage use. Future work validates fault-prone modules during development.

Achieving intelligent fault prediction in software testing with AI offers an edge in the sphere of adaptive testing. Machine learning models are trained to predict fault-prone components of a system. Using the predicted fault-prone files to prioritize, allocate, and schedule test cases enables adaptive testing. An illustration using a web-based district administration application shows how, during the testing phase, early detection of fault-prone modules helps save cost and effort by focusing on high-risk areas. Past work explores the predictive power of three supervised state-of-

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

the-art ML approaches in fault prediction at the testing stage.

8. References

1. Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2021). Machine learning based methods for software fault prediction: A survey. Expert Systems with Applications, 172, 114595.
2. Moghadam, M. H., Saadatmand, M., Borg, M., Bohlin, M., & Lisper, B. (2022). An autonomous performance testing framework using self-adaptive fuzzy reinforcement learning. Software Quality Journal, 30, 127–159.
3. Ganesh, S. K., Subbareddy, K., Gopi, A., Davuluri, P. N., Mannar, B. R., & Karnawat, A. T. (2026, June). Fraudulent Credit Card Transaction Detection Using a Hybrid Ensemble-Anomaly Detection Framework. In 2026 6th International Conference on Intelligent Technologies (CONIT) (pp. 1-6). IEEE.
4. Esnaashari, M., & Damia, A. H. (2021). Automation of software test data generation using genetic algorithm and reinforcement learning. Expert Systems with Applications, 183, 115446.
5. Laaber, C., Gall, H. C., & Leitner, P. (2021). Applying test case prioritization to software microbenchmarks. Empirical Software Engineering, 26, 133.
6. Bardin, S., Kosmatov, N., Marcozzi, M., & Delahaye, M. (2021). Specify and measure, cover and reveal: A unified framework for automated test generation. Science of Computer Programming, 207, 102641.
7. Mahadevan, S. (2026). Governed Serverless Automation Ecosystem for AI-Driven Enterprise Integration and Sustainable Cloud Operations. International Journal of Computer Information Systems and Industrial Management Applications, 18(16s), 1561-1570.
8. Hasnain, M., Jeong, S. R., Pasha, M. F., & Ghani, I. (2021). An ontology based test case prioritization approach in regression testing. Computers, Materials & Continua, 67(1), 1051–1068.
9. Pan, R., Bagherzadeh, M., Ghaleb, T. A., & Briand, L. (2022). Test case selection and prioritization using machine learning: A systematic literature review. Empirical Software Engineering, 27(2), Article 29.
10. Khan, M. A., & Elmitwally, N. S. (2022). Software defect prediction using artificial neural networks: A systematic literature review. Scientific Programming, 2022, 2117339.
11. Yandamuri, U. S., Loganathan, R., Davuluri, P. S. L. N., Rani, P. R. S., Kolla, S. H., & Nagubandi, A. R. (2026). Adaptive Intelligence Networks for Humancentered Enterprise Automation and Governance. In 2026 Third International Conference on Innovations in Cybersecurity and Data Science (ICICDS) (pp. 678–683). IEEE. 2026 Third International Conference on Innovations in Cybersecurity and Data Science (ICICDS). https://doi.org/10.1109/icicds70526.2026.11604640
12. Bhandari, K., Kumar, K., & Sangal, A. L. (2023). Data quality issues in software fault prediction: A systematic literature review. Artificial Intelligence Review, 56, 7839–7908.
13. Rajnish, K., & Bhattacharjee, V. (2022). A cognitive and neural network approach for software defect prediction. Journal of Intelligent & Fuzzy Systems, 43(5).
14. Banoth, S., Santoshi Kumari, M., Lalitha, P., Deepthi, P., Kumar Peddi, R., & Kavitha, P. (2026). An Efficient Hybrid K-Means and Random Forest-Based Approach for Cloud Malware Detection and Privacy Protection. In 2026 6th International Conference on Intelligent Technologies (CONIT) (pp. 1–6). IEEE. 2026 6th International Conference on Intelligent Technologies (CONIT). https://doi.org/10.1109/conit69683.2026.11621822
15. Borandag, E. (2023). Software fault prediction using an RNN-based deep learning approach and ensemble machine learning techniques. Applied Sciences, 13(3), 1639.
16. Piyush Kumar Pareek. (2026). Human-Centric Machine Learning Frameworks for Scalable Software Quality Prediction. Journal of Intelligent Decision

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

Making and Information Science, 3(5s), 1525–1543. https://doi.org/10.59543/jidmis.v3.1284

17. van Dinter, R., Catal, C., Giray, G., & Tekinerdogan, B. (2023). Just-in-time defect prediction for mobile applications: Using shallow or deep learning? Software Quality Journal, 31, 1281–1302.
18. Falessi, D., Laureani, S. M., Çarka, J., Esposito, M., & da Costa, D. A. (2023). Enhancing the defectiveness prediction of methods and classes via JIT. Empirical Software Engineering, 28, Article 37.
19. Piyush Kumar Pareek. (2026). Self-Evolving Analytics Pipelines for Reliable AI-Augmented Software Systems. Journal of Intelligent Decision Making and Information Science, 3(5s), 1558–1578. https://doi.org/10.59543/jidmis.v3.1286
20. Cabral, G. G., Minku, L. L., Oliveira, A. L. I., Pessoa, D. A., & Tabassum, S. (2023). An investigation of online and offline learning models for online just-in-time software defect prediction. Empirical Software Engineering, 28, Article 121.
21. Song, H., Wu, G., Ma, L., Pan, Y., Huang, Q., & Jiang, S. (2023). Adversarial domain adaptation for cross-project defect prediction. Empirical Software Engineering, 28, Article 127.
22. Mangalampalli, B. M., Peddi, R. K., Kolla, S. K., Reddy, V. A. R., Mangala, N., & Seenu, A. (2026, June). Explainable Clinical Graph Intelligence Framework for Longitudinal Risk Modeling and Care Pathway Optimization. In 2026 Third International Conference on Innovations in Cybersecurity and Data Science (ICICDS) (pp. 1-6). IEEE.
23. Stradowski, S., & Madeyski, L. (2023). Industrial applications of software defect prediction using machine learning: A business-driven systematic literature review. Information and Software Technology, 159, 107192.
24. Malhotra, R., & Bansal, A. (2023). Application of deep learning in software defect prediction: Systematic literature review and meta-analysis. Information and Software Technology, 158, 107175.
25. Bhavani, B. D., SR, S., Loganathan, R., & Nagaraj, S. (2026, April). Evolutionary Gravitational Neocognitron Neural Network, Snow Leopard Optimization and Deep Graph Reinforcement Learning for Routing Protocol in WSN. In 2026 2nd International Conference on Intelligent Systems and Computational Networks (ICISCN) (pp. 1-8). IEEE.
26. Hadadi, F., Dawes, J. H., Shin, D., Bianculli, D., & Briand, L. (2024). Systematic evaluation of deep learning models for log-based failure prediction. Empirical Software Engineering, 29, Article 105.
27. Wang, X., & Zhang, S. (2024). Cluster-based adaptive test case prioritization. Information and Software Technology, 165, 107339.
28. Roza, E. A., Prado Lima, J. A., & Vergilio, S. R. (2024). On the use of contextual information for machine learning based test case prioritization in continuous integration development. Information and Software Technology, 171, 107444.
29. Qian, Z., Yu, Q., Zhu, H., Liu, J., & Fu, T. (2025). Reinforcement learning for test case prioritization based on LLEed K-means clustering and dynamic priority factor. Information and Software Technology, 179, 107654.
30. Loganathan, R. (2026). SaaS Entitlement Governance: A Reproducible Model for Detecting and Reclaiming Over-Provisioned Access at Enterprise Scale. Journal of Intelligent Decision Making and Information Science, 3(7s), 2519-2530.
31. Mishra, A., & Sharma, A. (2024). Deep learning based continuous integration and continuous delivery software defect prediction with effective optimization strategy. Knowledge-Based Systems, 296, 111835.
32. Li, T., Wang, Z., & Shi, P. (2025). Within-project and cross-project defect prediction based on model averaging. Scientific Reports, 15, 6390.
33. Davuluri, P. S. L. (2023). AI-Augmented Sanctions Screening: Enhancing Accuracy and Latency in Real Time Compliance Systems. AI-Augmented Sanctions Screening: Enhancing Accuracy and Latency in Real Time Compliance Systems (December 15, 2023).
34. Zou, Y., & Wang, H. (2025). A three-stage cross-project defect prediction framework based on feature

ISSN: 3067-4190
AMERICAN ADVANCED JOURNAL FOR EMERGING DISCIPLINARIES
VOLUME: 04 ISSUE: 02
RECEIVED: APRIL 24 REVISED: MAY 05 ACCEPTED: MAY 24 PUBLISHED: JUNE 16

representation and knowledge transfer. Complex & Intelligent Systems, 11, 459.

35. Zaidi, H. Z., Ullah, U., Arshad, M., Aljuaid, H., Rauf, M. A., Sarwar, N., & Sajid, R. (2025). Machine learning approaches for software defect prediction. Applied Computational Intelligence and Soft Computing, 2025, 7933078.
36. Vakkalagadda, T., & Rajamanickam, V. (2026). Agentic AI Architectures for Next-Generation Investment Advisory and Portfolio Intelligence. International Journal of Computer Information Systems and Industrial Management Applications, 18(9s), 1206-1219.
37. Gottumukkala, D. P., Reddy, P. V. G. D., & Rao, S. K. (2025). Topic modeling-based prediction of software defects and root cause using BERTopic, and multioutput classifier. Scientific Reports, 15, 25428.
38. Fei, Q., Hu, H., Yin, G., & Sun, Z. (2025). A software defect prediction method using a multivariate heterogeneous hybrid deep learning algorithm. Computers, Materials & Continua, 82(2), 3251–3279.
39. Dhavakumar, P., & Vengadeswaran, S. (2025). Software defect prediction using graph sample and aggregate-attention network optimized with nomadic people optimizer for enhancing the software reliability. Computer Standards & Interfaces, 104033.
40. Destefanis, G., Yousefi, L., Shepperd, M., et al. (2026). An audit of machine learning experiments on software defect prediction. Empirical Software Engineering, 31, 83.
41. Romero, J. R., Ramírez, A., & García-Martínez, C. (2026). Automated machine learning for test case prioritisation. Empirical Software Engineering, 31, 120.