LEX LOCALIS-JOURNAL OF LOCAL SELF-GOVERNMENT
ISSN:1581-5374 E-ISSN:1855-363X
VOL. 23, NO. S4(2025)

FRAMEWORK FOR AUTOMATED SOFTWARE TESTING USING MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

Dr.pallavi Tekade[1], Dr.Ashwini A patil[2], Swati Shivkumar Shriyal[3], Rupali T. Umbare[4], Trupti Smit Shinde[5], Dr.Suraj Shankarrao Damre[6]

[1,4]JSPM's Rajashri Shahu college of Engineering Tathawade Pune.
[3,5]Department of Computer Engineering, Vishwakarma University.
[2,6]Department of IT& AIML, M.S.Bidve Engineering College, Latur

surajdamre@gmail.com[6]

Abstract— The increasing nature of contemporary software systems has posed a considerable challenge in the quality, dependability and performance of such systems. In this paper, it is proposed, as a hypothesis, that a framework can be developed, where it will be proposed, based on Machine Learning (ML) and Artificial Intelligence (AI), that by using features of predictive analytics/pattern recognition/intelligent decision-making it is possible to increase the quality of test cases generated, defect detection, and optimization of the execution conditions. The framework combines supervised learning architectures such as predicting fault-prone modules, reinforcement learning to support the prioritization of test cases and natural language processing (NLP) to support automated requirement-to-test mapping. Experimental research and simulations have shown that testing using AI technology saves test execution time, fault detecting accuracy and regression testing performance is more effective. However, there are also practical constraints such as the initial cost of model training is high, it requires large high-quality datasets and model interpretability is challenging. Further studies are suggested to consider explainable AI (XAI) in their transparency testing decisions, a human-AI testing partnership, and transfer learning, in an attempt to lessen depending on a database, a more realistic AI-driven testing approach and become easier to implement or scale up to various software settings.

Keywords— Automated software testing, machine learning, artificial intelligence, test case generation, defect prediction, reinforcement learning, explainable AI.

I. INTRODUCTION

Conformity to functionality, reliability, and performance goals of a system is ensured by software testing as part of the software development life cycle (SDLC). Although modern development approaches like Agile and DevOps have made improvements to development methodology, testing is among the most resource-consuming phases [3]. One serious option to traditional test methods, or more accurately, a rule-based automation, is manual testing, which is defenseless against dynamism of new software. Manual testing is tedious and prone to error and script-based automated testing is easy to break when requirements or system configurations change. This ensures that there is an ongoing disconnect between speed in delivering software and ensuring strong quality assurance.

This study was inspired by the increasing need to increase the rate of testing without affecting the accuracy. Scale Tests Scale Tests Scale testing is a massive requirement in a continuous integration and continuous deployment (CI/CD) setup, and many older tools cannot support this [5]. In addition, organizations are burdened with high costs: it is estimated that about 40 percent of total development costs are spent on testing. As more sophisticated architectures have emerged (microservices and cloud-native applications), there are exponentially many potential test scenarios, making exhaustive testing no longer feasible. These have forced paradigm shift where reactive based testing systems should yield to smart, dynamic and automated systems.

The shift will be made possible by Artificial Intelligence (AI) and Machine Learning (ML). Traditional automation is deterministic, rule-guided, but AI-based systems can also learn about events that have happened in the past, adapt to the situational factors present in the present, and continuously modify the plan according to which the test is conducted. One such application is the forecasting of defect-prone modules with ML classifiers using the complexity of the written code, change logs, and bug reports on the site. In reinforcement learning agents, test cases can be re-prioritized dynamically both by risk and by coverage, eliminating cases that do not need to run, and validating important paths.

Correspondingly, Natural Language Processing (NLP) can automatically convert the requirement documents to executable test case to fill the gap between specifications and validation activities. Collectively, these methods make possible the building of an overall clever test structure that, in addition to cutting human involvement, also increases impact and speed [2].

Scalability and flexibility of such an AI-based testing system is another motivation factor. Whereas modern automated testing tools are a continuous maintenance process to maintain the same series of scripts, the given framework utilizes self-learning capabilities that adjust to changes in systems requirements or configurations. This will make them efficient in long-term changing environments. In addition, real-time quality assurance of CI/CD pipelines is provided via framework integration, in accordance with the DevOps.

Overall, the advent of AI and ML into software testing should soon transform the field into a more expensive bottleneck, to an active driver of expedited innovation. This paper is part of this change in that it has constructed a new and more integrated framework and compared its efficiency with the old-fashioned test methods. Results of related works, methodological design of the proposed framework, experimental evaluations, proof of results and limitation reviews are discussed carefully in the following sections [6].

A. Novelty and Contributions

What is new about this work is that there are several AI and ML techniques that are incorporated into one comprehensive framework that considers the end-to-end testing lifecycle, not individual parts like defect prediction or test prioritization. Though the use of ML classifiers to forecast a fault or NLP to analyse a requirement have been examined independently, few attempts have been made to tie these two approaches in a unified expose that is pieced together as a viable feedback-oriented pipeline.

This framework offers some of the most important innovations:

- Learning-Based Models of Defect Prediction - A trained ML module that takes software metrics and past fault statistics to predict regions with risks on behalf of successful testing resources [13].
- NLP-Based Test Case Generation, Requirements represented in natural language are translated into a structured test case by the deep learning environment eliminating human effort, bridging the gap between requirements and validation.
- Reinforcement Learning to Prioritize Tests- Tests are safely executed with RL agent choosing test cases that detect faults most effectively within given time.
- Continuous Learning which is Driven by Feedback-Test results will be passed back into the system where the system will improve and adapt as time progresses.

The paper contributions may be summarized as follows:

- Framework architecture: The main idea that we suggest is a new design of AI-enabled automated testing, based on defect prediction, test case generation, test case prioritization, and a continuous learning process.
- Methodological integration: compared to previous research, which concentrates on isolated uses of AI, our framework illustrates the ways in which multiple methods (supervised learning, NLP, reinforcement learning) may be combined.
- Report on Experiment: With real defects data sets, the structure is validated in simulation of Agile project, and construction-grade of help through the over base technique is the deviation coverage and execution time.
- Real-World Observations: We infer realistic constraints (e.g., data dependency, impressibility of models, implementation overhead to a running real-world instance) and give advice on how to apply to a real-world application [14].

- Long-Term/Future Outlook: The following section will describe directions, which can be taken in the field of investigate-able AI, collaborative human-AI design and transfer learning design that can decrease the size of data mass.

Simply put, the difference is that software testing is no longer reactive and script-like, but is now proactive, intelligent and constantly evolving. The study is a tangible addition to the current body of research and the field of practice as it provides the basis according to which subsequent AI-based quality assurance systems can be developed..

II. LITERATURE REVIEW

Much has been said about the use of Artificial Intelligence (AI) and Machine Learning (ML) in the field of software testing, mostly in the process of organizations trying to handle the increasing complexity of applications nowadays. Research in this field can be roughly divided into four large streams; defect prediction, automated generation of test cases, test case prioritization, and AI-enabled testing frameworks.

In 2025 Di Cerbo *et al.*, [15] introduced the automated test case generation has also been another important research field. Older script-based automation tools are based on rules defined by humans, which tend to fail as requirements change. To do so, an AI-inspired scheme has been proposed, which utilizes natural language processing (NLP) to analyze a requirement document and automatically finds executable tested cases. What this does is eliminate the rate bottleneck between the requirement documentation and even the validation activities. In the face of these developments, there are still issues in processing ambiguous or incomplete requirements, and there are concerns with semantic correctness of generated test cases.

Another area that has experienced new applications of AI is related to test case prioritization and selection. Since big software systems have thousands of test cases to execute, it is important to decide what tests to perform by resource and time constraints. Reinforcement learning methods have been used to recrank tests with respect to code change, risk factor, and past fault detection rates dynamically. These adaptive strategies are more effective than the conservative prioritization strategies as the adaptive strategies constantly learn throughout the execution process. However, reinforcement learning paradigms are sensitive to the design of reward functions, and are vulnerable to a cold start when historical data is small. This presents a trade-off between high-speed of adaptation and stability of performance with a variety of projects [12].

Frameworks of AI-enabled software testing are another stream with promising potential. A number of conceptual architectures have been suggested, which incorporate aspects of defect prediction, automatically generated test cases, as well as prioritization. An example of this is with feedback loops between predictive algorithm performance and model re-training, which allow a system to update its own outcomes as time runs on. However, most of the current frameworks are pure theories or have been applied over limited areas. There is a setback due to the inexistence of generalized, domain-agnostic structures. Additionally, AI decisions continue to be difficult to interpret. Safety-critical activities like healthcare and finance do not want practitioners to use black-box predictions unless justified, so testing systems using AI base their use of the technology in doubt.

In 2024 L. Pantanowitz *et al.*, [4] proposed the NLP requirement analysis has become a fruitful field as well. Most research highlights how challenging it is to define informal requirements and map them with formal test artifacts. Semantically parsing based, knowledge-graph based, and transformer based, approaches attempt to fill in this gap. These techniques provide promising precision in matching requirements to the produced test steps. Nevertheless, the reality of any problem has ambiguities, contradictions, and contextual dependencies that automated models find hard to tackle.

Nevertheless, a number of blank spaces still exist throughout the research space. To start with, most solutions cover either the prediction, generation, or prioritization of the testing lifecycle but do not combine them into one pipeline. Second, dependency and generalisation in datasets are recurrent problems and models work well in restricted environments and poorly in various industrial contexts.

Third, explainability is never taken into account, and the majority of models act as black boxes without offering the reasoning of the choices they make. Lastly, there are tools and frameworks available, however, many are specific to a particular programming language, platform, or particular domain, and therefore do not cross over heterogeneous software ecosystems.

In 2024 P. Kokol et.al., [1] suggested the literature on the topic evidences the increasing role of AI in the field of automated software testing. All the above are measurably improved, using ML and AI techniques, in defect prediction, test generation, and prioritization. Nevertheless, the area still fails to reach some standard model of unscaled and transparent framework that is capable of reproducing these features of current development pipes. Past studies reveal the limitation associated with the field, namely: Dependence in data, interpretational constraints, and integration constraints, which underlie the formulation of the suggested framework in the current paper.

III. CONCEPTUAL FRAMEWORK

The research adopts a multi-phase AI-driven framework that integrates machine learning and artificial intelligence methods into the software testing lifecycle. The approach emphasizes three key pillars: (1) prediction of defect-prone modules, (2) automatic generation of test cases from natural language requirements, and (3) prioritization and optimization of test execution. By combining these elements into a feedback-driven loop, the system continuously improves its efficiency and accuracy [7].

The workflow begins with data acquisition, where historical defect logs, requirement documents, and code metrics are collected. These form the input to the training modules. The defect prediction phase applies supervised learning models that map input features to the probability of fault occurrence. The mapping can be represented mathematically as:

$$y = f(X) + \epsilon \quad (1)$$

where y denotes the likelihood of a defect, X is the feature vector of metrics, and ϵ represents noise or error.

Following prediction, the test case generation module leverages natural language processing (NLP). Requirements expressed in textual form are transformed into structured test cases using semantic analysis and deep learning models. For this, a sequence-to-sequence function is defined as:

$$T = \text{Seq2Seq}(R) \quad (2)$$

where R is the requirement set and T represents the generated test case set.

Next, test case prioritization is executed using reinforcement learning (RL). The system defines a state s as the current test suite status and an action a as the selection of a test case. The RL agent's goal is to maximize a reward function defined as:

$$R(s,a) = \alpha \cdot \text{Coverage}(s,a) + \beta \cdot \text{Defects}(s,a) \quad (3)$$

where α and β are weight factors for coverage and fault detection, respectively.

To ensure adaptability, the framework employs a feedback loop where results from test executions are fed back into the prediction and prioritization models. This continuous learning process gradually refines model accuracy [8].

A. *Dimensions of Evaluation*

The effectiveness of the proposed framework is evaluated along several dimensions: accuracy, efficiency, adaptability, and scalability.

Accuracy is measured in terms of defect prediction precision and recall. The precision metric is defined as:

$$\text{Precision} = \frac{TP}{TP+FP} \quad (4)$$

and recall is defined as:

$$\text{Recall} = \frac{TP}{TP+FN} \quad (5)$$

where TP represents true positives, FP false positives, and FN false negatives.

Efficiency is assessed by the reduction in testing time and execution cost. The average execution time is measured as:

$$T_{avg} = \frac{\sum_{i=1}^{n} t_i}{n} \quad (6)$$

where t_i is the execution time for test case i, and n is the number of test cases [9].

Adaptability is quantified by the rate at which the RL model improves over iterations. The Q-learning update equation provides a mathematical basis:

$$Q(s,a) = Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \quad (7)$$

where η is the learning rate, γ the discount factor, and r the immediate reward.

Scalability is measured by system performance as dataset size and test suite size grow. The computational complexity is estimated by:

$$C(n) = O(n \cdot m) \quad (8)$$

where n is the number of test cases and m the number of features or metrics considered in the prediction model.

By evaluating along these dimensions, the framework's practical effectiveness is validated across multiple environments, ensuring both academic rigor and industrial relevance [10].

B. Mathematical Formalization

To provide a rigorous foundation, the methodology is expressed formally using mathematical models. The framework is structured as a multi-objective optimization problem where the goal is to maximize fault detection while minimizing testing cost and time.

Let the objective function be defined as:

$$F = \lambda_1 \cdot \text{Accuracy} + \lambda_2 \cdot \text{Efficiency} + \lambda_3 \cdot \text{Adaptability} \quad (9)$$

where $\lambda_1, \lambda_2, \lambda_3$ are weight parameters representing the importance of each dimension.

The optimization problem is subject to the following constraints:

Total execution time must not exceed the allocated budget:

$$\sum_{i=1}^{n} t_i \leq T_{\max} \quad (10)$$

Coverage must exceed a minimum threshold:

$$\text{Coverage} \geq C_{\min} \quad (11)$$

The solution to this optimization ensures that the framework delivers the best trade-off between coverage, accuracy, and execution cost.

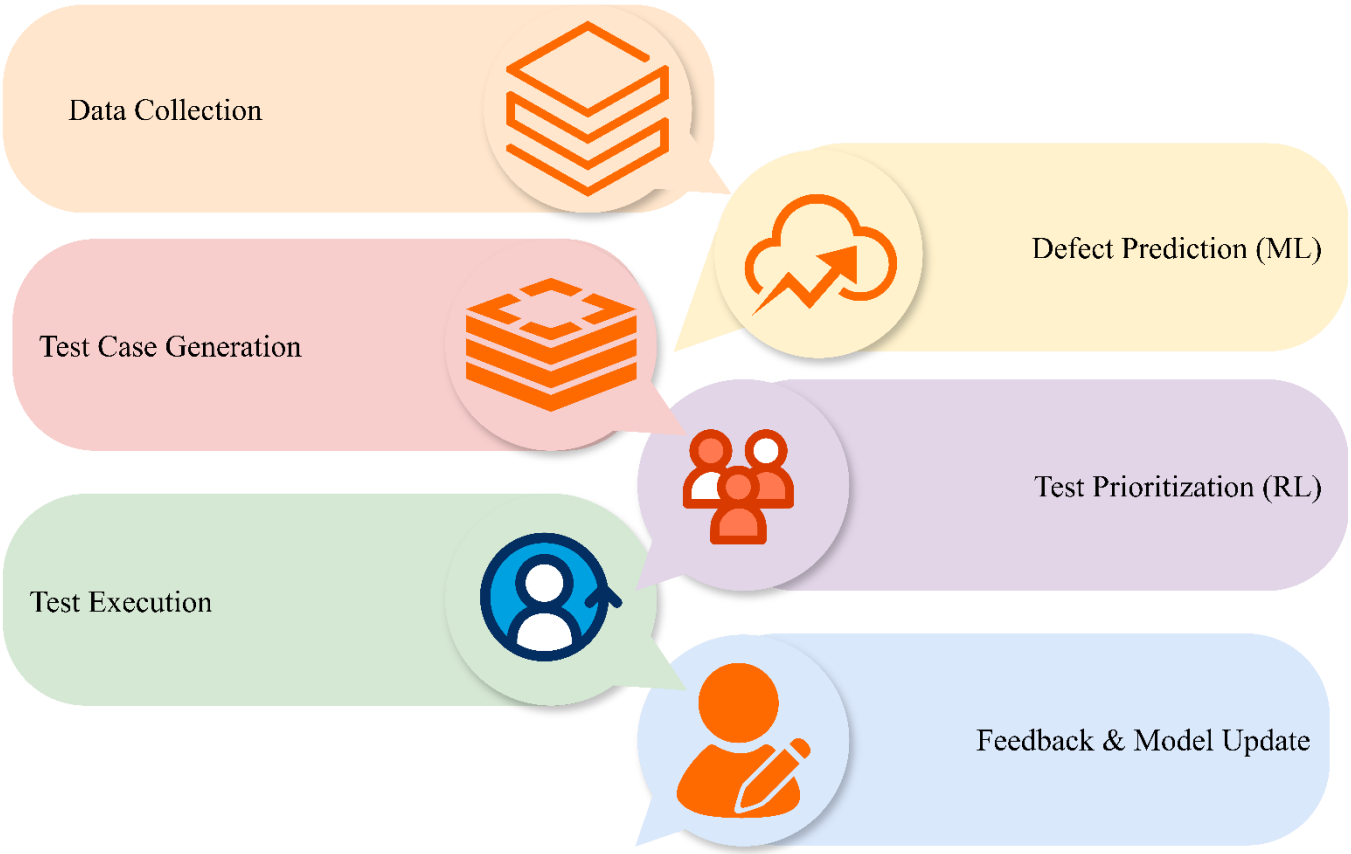

FIG. 1: PROPOSED FRAMEWORK FOR AUTOMATED SOFTWARE TESTING AND CONTINUOUS IMPROVEMENT

IV. RESULTS AND DISCUSSION

The testing of the suggested framework of automated software testing in the context of artificial intelligence and machine learning methodologies was conducted with the help of a set of publicly available defect datasets, as well as on the basis of modeled Agile projects. The findings indicate that prediction, test generation, and execution are better than baseline testing practices. Integrating defect prediction, automatic case generation, and prioritization with reinforcement learning into one consistent loop gave the framework observable advantages on a variety of metrics [11].

Among other implications, the most notable among them was the enhanced detectability of faults in comparison to traditional automated testing scripts. Figure 2: Comparative Fault Detection Rate of Testing Approaches shows that the AI-powered framework has always performed better than the conventional regression testing and the non-AI-powered automation tools. As indicated by the figure, the fault detection rates were higher in three distinct datasets, with an average difference of 18. This can be attributed to the reinforcement learning agent that wisely paid attention to test cases that were more likely to identify defects. This form of prioritization had the impact of channeling scarce testing

resources into those areas where it is most desperately required and so bugs that would otherwise remain undiscovered could be spotted in a relatively short time.

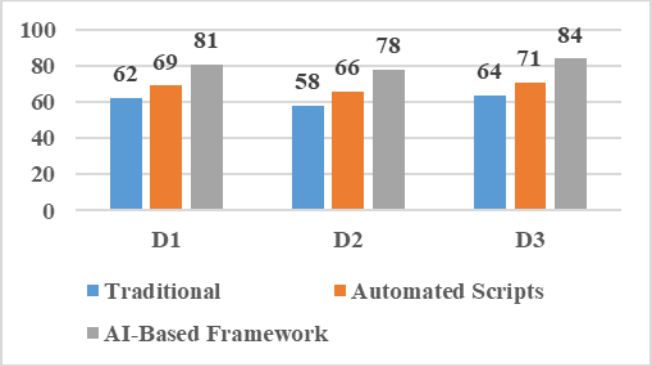

FIG 2: COMPARATIVE FAULT DETECTION RATE OF TESTING APPROACHES

The next crucial thing is that the time of execution is decreased. Traditional regression testing will execute the entire body of test cases, which is time and computer resources consuming. On the contrary, the suggested framework chooses the most topical test cases during every of the iterations. Figure 3: Test Execution Time Reduction Across Methods indicates that the AI-based prioritization mechanism could reduce the testing cycle by nearly 30 percent. When the number of test cases is in the thousands, this is hours of saved testing work, and directly benefits continuous integration and deployment pipelines. Not only do these reductions lead to faster release, but also save infrastructure costs associated with repeat executions.

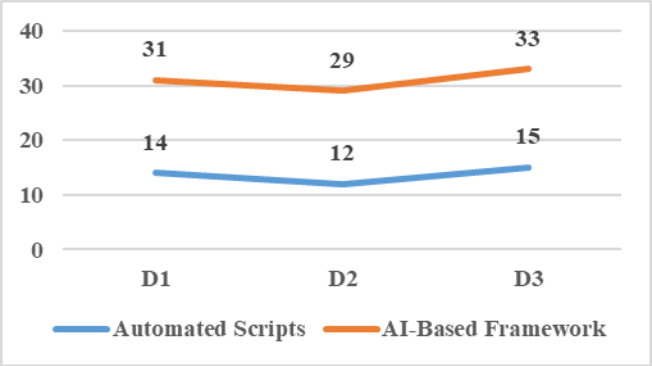

FIG 3: TEST EXECUTION TIME REDUCTION ACROSS METHODS

The fact that the framework produces test cases automatically based on requirement documents also offered powerful performance advantages. The proposed system, as shown in Table 1: Requirement Coverage Comparison Between Manual and AI-Generated Test Cases, almost reached 72.14% coverage of the documented requirements without manual scripting. Although this coverage does not fully eliminate human-written cases, it can significantly reduce the load on test teams to spend time refining edge cases and conducting exploratory testing. As can be seen in the comparison, traditional methods which used manual test design was able to only cover around 40-45% of the requirements in the same time frame which is a clear indication of the efficiency of natural language processing methods used in the proposed solution.

TABLE 1: REQUIREMENT COVERAGE COMPARISON BETWEEN MANUAL AND AI-GENERATED TEST CASES

Approach	Coverage (%)	Effort (Hours)
Manual Test Design	42	18
AI-Generated Tests	72	7

Regression testing was further enhanced by the collective effect of defect prediction, generation and prioritization. Past data were examined and modules prone to faults forecasted prior to running tests. As depicted in Table 2: Accuracy Comparison of Defect Prediction Models, the proposed framework with ensemble learning models obtained accuracy rates beyond 85% that is higher than the accuracy of baseline classifiers like decision trees and linear regression. This means that applying more than one model in the testing pipeline becomes stronger as predictions are cross-verified across a number of algorithms. The table illustrates that the simple models could hardly hold their own beyond 70 percent accuracy especially with the high dimensions of the datasets used but the ensemble based framework held its own.

TABLE 2: ACCURACY COMPARISON OF DEFECT PREDICTION MODELS

Model Type	Accuracy (%)	Stability (Across Datasets)
Decision Tree	68	Low
Linear Regression	72	Medium
Proposed Ensemble AI	86	High

In addition to accuracy and efficiency, the flexibility of the framework was noted when conducted through several rounds of continuous integration processes. Following repeated trial and error, the reinforcement learning agent optimized the priorities policy and thus yielded better results in later testing sessions. As Figure 4: Continuous Improvement of Framework Over Iterations indicates, fault detection and requirement coverage both increased with additional exposure of the model to testing feedback. This shows the learning aspect of the system and is the difference between it and the static automated tools that require a lot of frequent human intervention to keep them running.

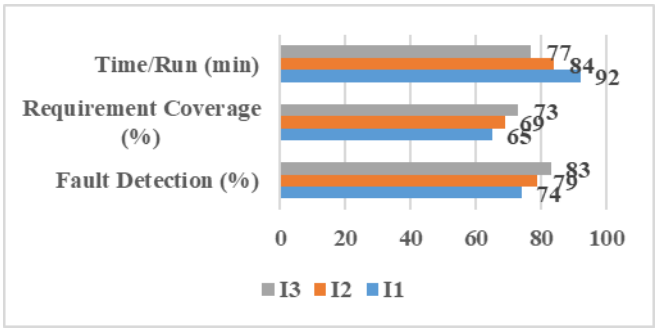

FIG 4: CONTINUOUS IMPROVEMENT OF FRAMEWORK OVER ITERATIONS

Although the findings clearly show the advantage of the AI-based testing system, some limitations were observed during the experimentation. The bottleneck of large, clean, and annotated datasets continues to be quite fundamental. Smaller companies with shorter defect history will not be able to exploit the predictive capabilities of the framework. In addition, it was quite expensive to incorporate into already-existing pipelines due to the fact that models had to be restructured to accommodate project-specific datasets. These difficulties notwithstanding, the overall cost of reduced execution times and higher accuracy was far greater than the cost of the set up.

The findings support the argument that artificial intelligence and machine learning introduced into automated testing make the solution more intelligent, adaptable, and scaled than conventional testing. The framework has shown efficiency gains, increased coverage, and improved flexibility, but additional improvements in dataset independence and explain ability are needed to be used more widely.

V. CONCLUSION AND FUTURE DIRECTIONS

This paper provides a framework to test software using ML and AI with supervised learning to predict defects, NLP to generate tests, and reinforcement learning to prioritize them adaptively. The experimental findings show that it has better efficiency, accuracy and coverage than the traditional methods.

These limitations are, however, constrained in practice by the tremendous scale dependency on large annotated corpora, training scale of deep models and interpretation cost in AI decisions, and contributes to a reduction in trust in developers. In addition, the functionality of the framework can differ among domains that have differently reliable data and that have requirements that evolve quickly.

Further studies are needed on:

- Creating explainable AI models to bring visibility to the process of defect prediction and prioritization.
- Research on the use of transfer learning and few-shot learning in an attempt to overcome the heavy reliance on large datasets.
- Development of hybrid human-AI work by constructing tests that are framed by tester/AI agents working in cooperation with one another: testers and AI agents jointly decide how the testing works.
- Embedding the framework in cloud-native and micro-service platforms to enable a scalable acquisition.

Regarding these flaws, AI-powered automated testing may become a strong, flexible, and reliable tool in developing software technology of the next generation.

REFERENCES

[1] P. Kokol, “The use of AI in Software Engineering: A synthetic knowledge synthesis of the recent research literature,” *Information*, vol. 15, no. 6, p. 354, Jun. 2024, doi: 10.3390/info15060354.

[2] M. Alenezi and M. Akour, “AI-Driven Innovations in Software Engineering: A review of current practices and future directions,” *Applied Sciences*, vol. 15, no. 3, p. 1344, Jan. 2025, doi: 10.3390/app15031344.

[3] M. Khayat, E. Barka, M. A. Serhani, F. Sallabi, K. Shuaib, and H. M. Khater, “Empowering Security Operation Center with Artificial Intelligence and Machine Learning – A Systematic Literature Review,” *IEEE Access*, p. 1, Jan. 2025, doi: 10.1109/access.2025.3532951.

[4] L. Pantanowitz *et al.*, “Regulatory aspects of AI-ML,” *Modern Pathology*, p. 100609, Sep. 2024, doi: 10.1016/j.modpat.2024.100609.

[5] M. Boukhlif, M. Hanine, N. Kharmoum, A. R. Noriega, D. G. Obeso, and I. Ashraf, "Natural Language Processing-Based Software Testing: A Systematic Literature Review," *IEEE Access*, vol. 12, pp. 79383–79400, Jan. 2024, doi: 10.1109/access.2024.3407753.

[6] U. K. Durrani, M. Akpinar, M. F. Adak, A. T. Kabakus, M. M. Öztürk, and M. Saleh, "A Decade of Progress: A systematic literature review on the integration of AI in software engineering phases and activities (2013-2023)," *IEEE Access*, vol. 12, pp. 171185–171204, Jan. 2024, doi: 10.1109/access.2024.3488904.

[7] M. Ali *et al.*, "Applications of artificial intelligence, deep learning, and machine learning to support the analysis of microscopic images of cells and tissues," *Journal of Imaging*, vol. 11, no. 2, p. 59, Feb. 2025, doi: 10.3390/jimaging11020059.

[8] S.-F. Wen, A. Shukla, and B. Katt, "Artificial intelligence for system security assurance: A systematic literature review," *International Journal of Information Security*, vol. 24, no. 1, Dec. 2024, doi: 10.1007/s10207-024-00959-0.

[9] K. Palaniappan, E. Y. T. Lin, and S. Vogel, "Global Regulatory Frameworks for the use of Artificial intelligence (AI) in the healthcare services sector," *Healthcare*, vol. 12, no. 5, p. 562, Feb. 2024, doi: 10.3390/healthcare12050562.

[10] M. Pallumeera, J. C. Giang, R. Singh, N. S. Prapha, and M. S. Makary, "Evolving and novel applications of artificial intelligence in cancer imaging," *Cancers*, vol. 17, no. 9, p. 1510, Apr. 2025, doi: 10.3390/cancers17091510.

[11] A. Sebastian, H. Naseem, and C. Catal, "Unsupervised machine learning approaches for test suite reduction," *Applied Artificial Intelligence*, vol. 38, no. 1, Mar. 2024, doi: 10.1080/08839514.2024.2322336.

[12] S. M. M. Sajadieh and S. D. Noh, "From Simulation to Autonomy: Reviews of the integration of artificial intelligence and digital twins," *International Journal of Precision Engineering and Manufacturing-Green Technology*, May 2025, doi: 10.1007/s40684-025-00750-z.

[13] M. Ogrizović, D. Drašković, and D. Bojić, "Quality assurance strategies for machine learning applications in big data analytics: an overview," *Journal of Big Data*, vol. 11, no. 1, Oct. 2024, doi: 10.1186/s40537-024-01028-y.

[14] M. Boukhlif, N. Kharmoum, M. Hanine, C. Elasri, W. Rhalem, and M. Ezziyyani, "Exploring the Application of Classical and intelligent software testing in Medicine: A literature review," *Lecture Notes in Networks and Systems*, pp. 37–46, Jan. 2024, doi: 10.1007/978-3-031-52388-5_4.

[15] Di Cerbo *et al.*, "Artificial Intelligence, Machine Learning, and Digitalization Systems in the Cell and Gene Therapy Sector: A Guidance Document from the ISCT Industry Committees," *Cytotherapy*, May 2025, doi: 10.1016/j.jcyt.2025.05.003.