UDC 004.89:004.42 DOI https://doi.org/10.35546/kntu2078-4481.2025.2.2.37

M. O. REIDA
Postgraduate Student at the Department of Computer Science
Zaporizhzhia National University
ORCID: 0009-0002-3098-113X

G. A. DOBROVOLSKY
Candidate of Technical Sciences,
Professor at the Department of Computer Science
Zaporizhzhia National University
ORCID: 0000-0001-5742-104X

SURVEY ON AI IN THE FUNCTIONAL SOFTWARE TESTING

The survey conducts a systematic review of the application of artificial intelligence (AI) in software quality control and testing, covering both functional and non-functional quality aspects across all testing stages. The study analyzes various AI techniques, including machine learning, deep learning, evolutionary algorithms, and natural language processing, while also examining AI-based testing tools. It highlights advancements, challenges, and research gaps, drawing on scientific publications from the last decade, supplemented by key foundational works.

Functional software quality is defined by its compliance with requirements and correct execution of functions, verified through unit, integration, system, and acceptance testing. AI facilitates automation of test case generation, defect prediction, and oracle design. Non-functional quality encompasses attributes such as performance, reliability, security, and usability, which are more challenging to automate. The review emphasizes AI's role in automating testing, particularly regression testing, to ensure product stability after changes.

Key AI applications include test case generation and test automation using search-based algorithms (e.g., EvoSuite, Sapienz), deep learning, and large language models (LLMs) like Codex. AI is also utilized for defect prediction, code quality assessment, test selection, prioritization, and optimization of test execution in continuous integration environments. Tools like Diffblue Cover demonstrate industrial adoption of AI for generating compact test suites.

Major challenges include the oracle problem (determining correct test outcomes), data dependency, the need for model transparency, and limited applicability in safety-critical domains. Future research directions include automating oracles, integrating AI with DevOps, developing human-in-the-loop hybrid systems, and establishing standards for AI in testing. The review calls for unified datasets and benchmarks to evaluate AI testing techniques, particularly in safety-critical sectors like healthcare and automotive.

Key words: *Survey, Artificial intelligence, Machine learning, Test Case Generation, Test Automation, Defect Prediction, Test Selection, Test Prioritization, Test Execution Optimization.*

М. О. РЕЙДА
аспірант кафедри комп'ютерних наук
Запорізький національний університет
ORCID: 0009-0002-3098-113X

Г. А. ДОБРОВОЛЬСЬКИЙ
кандидат технічних наук,
доцент кафедри комп'ютерних наук
Запорізький національний університет
ORCID: 0000-0001-5742-104X

ОГЛЯД ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ У ФУНКЦІОНАЛЬНОМУ ТЕСТУВАННІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Ця стаття представляє систематичний огляд застосування штучного інтелекту (ШІ) у контролі якості та тестуванні програмного забезпечення, охоплюючи як функціональні, так і нефункціональні аспекти якості на всіх етапах тестування. Дослідження аналізує різноманітні техніки ШІ, включаючи машинне навчання, глибоке навчання, еволюційні алгоритми та обробку природної мови, а також розглядає інструменти тестування на основі ШІ. У роботі висвітлено досягнення, виклики та прогалини в дослідженнях, використовуючи наукові публікації за останні 10 років, доповнені ключовими фундаментальними роботами.

Функціональна якість програмного забезпечення визначається його відповідністю вимогам і правильним виконанням функцій, що перевіряється через модульне, інтеграційне, системне та приймальне тестування. ШІ сприяє автоматизації створення тестових випадків, прогнозуванню потенційних помилок і розробці оракулів.

Нефункціональна якість охоплює такі аспекти, як продуктивність, надійність, безпека та зручність використання, які складніше автоматизувати. Огляд підкреслює роль ШІ в автоматизації тестування, зокрема регресійного, для забезпечення стабільності продукту після змін.

Серед ключових застосувань ШІ виділено генерацію тестових випадків і автоматизацію тестування за допомогою пошукових алгоритмів (наприклад, EvoSuite, Sapienz), глибокого навчання та великих мовних моделей (LLM), таких як Codex. ШІ також використовується для передбачення дефектів, оцінки якості коду, вибору та пріоритизації тестів, а також оптимізації виконання тестів у середовищах безперервної інтеграції. Інструменти, такі як Diffblue Cover, демонструють промислове впровадження ШІ для створення компактних тестових наборів.

Основні виклики включають проблему оракула (визначення правильних результатів тестів), залежність від даних, потребу в прозорості моделей ШІ та обмежену застосовність у критичних галузях. Перспективні напрями досліджень охоплюють автоматизацію оракулів, інтеграцію ШІ з DevOps, створення гібридних систем із залученням людини та розробку стандартів для використання ШІ у тестуванні. Огляд закликає до створення уніфікованих наборів даних і стандартів для оцінки ШІ-технік у тестуванні, особливо в безпекокритичних галузях, таких як охорона здоров'я та автомобілебудування.

Ключові слова: *Огляд, Штучний інтелект, Машинне навчання, Генерація тестових випадків, Автоматизація тестування, Прогнозування дефектів, Вибір тестів, Пріоритизація тестів, Оптимізація виконання тестів.*

Introduction

Software quality is a central concern in contemporary software engineering, defined by a system's compliance with requirements, user expectations, and standards. Quality is typically categorized into functional and non-functional aspects. Functional quality refers to the software's ability to perform its intended functions correctly. It is ensured by verifying each feature against requirements through various testing stages: unit, integration, system, and acceptance tests [1]. AI assists functional testing by generating effective test cases, predicting likely failures, and automating oracle design. Non-functional quality encompasses attributes beyond specific functions, including performance, reliability, security, usability, maintainability, portability, and compatibility [2]. For example, a web service must return correct results (functional) and meet performance benchmarks under load (non-functional). Non-functional tests often require specialized tools and environments and are more challenging to automate [3, 4]. Quality control must address both aspects. Automated testing is now standard practice, reducing risks and ensuring product stability [5]. Regression testing covers functional and non-functional aspects, verifying no regressions after changes [6].

Analysis of recent publications

Kolmogorov-Arnold Networks (KAN) is a novel class of neural networks grounded in a 1957 theorem by A. N. Kolmogorov (extended by V. Arnold) providing both flexibility and interpretability [51]. KANs have gained attention mainly in the past few years, so their applications to different tasks are being actively explored. Several surveys have been published on AI application to software testing covering mutation testing [5], artificial intelligence [7, 10] and machine learning [11, 33, 35] in software test automation. But none of them mention Kolmogorov–Arnold Networks (KANs) as a potential modeling paradigm.

The objective of this review is to re-examine how AI advances software quality control, surveys state-of-the-art AI techniques across testing stages, and identifies their impact, best practices, and challenges, providing a foundation for future research in AI-driven quality assurance. and paying the special attention to an application of KANs.

Publication selection

The publication selection combines automated snowball sampling search with manual filtering and citation network exploration. The method described below yielded 54 primary references representing the research on AI-driven software testing and quality assurance over the past decade [7].

We defined explicit inclusion and exclusion criteria for selecting publications to ensure a high-quality and relevant literature base.

Inclusion Criteria: We focused on peer-reviewed research articles (including conference papers and journal articles) and credible systematic literature reviews published roughly in the last 10 years (2015–2025). We included studies explicitly addressing the application of AI (e.g., machine learning, deep learning, evolutionary algorithms, knowledge-based systems) to software testing or quality control. Also, we included older seminal works (pre-2015) if they were highly cited and foundational to the field (to provide historical context or definitions). We limited sources to English-language publications from reputable venues in software engineering and AI. For tools or industry solutions, official documentation or whitepapers (software manuals) were considered if they provided technical details [8].

Exclusion Criteria: We excluded papers that did not specifically connect AI with software testing/quality (for example, general AI papers with no software engineering context or testing papers with no AI aspect). Non-peer-reviewed sources (blogs, opinion articles) were generally excluded unless they described essential tools (official product documentation was preferred). We also excluded publications focusing on testing AI systems rather than using AI for testing software to keep the scope on AI as a means for software quality improvement. Studies before 2015 were excluded unless identified

as high-impact classics (Artificial Intelligence in Software Testing: A Systematic Review). Non-English papers and purely theoretical works with no evaluation were also left out.

While selecting papers, we evaluated each publication against these criteria to ensure relevance and credibility. This filtering ensures the included literature comes from respected journals, conferences, or academic repositories and emphasizes contemporary developments while acknowledging key prior work [9].

At the initial stage of the publication selection process, a search using the query "AI in software testing review" in Scopus, Openalex, and Google Scholar returned a pool of the existing review studies and all the related scientific papers. Then, titles and abstracts were screened to berrypick the 20 most relevant publications. Next, we followed the controlled snowball sampling method [10] to collect the minimal complete list of publications, download full texts if freely available, and extract the basic keywords. In the last step, we removed the irrelevant references by screening the titles and abstracts.

AI Applications in Functional Software Quality Control and Testing

This section overviews how different AI methods contribute to software testing and quality control, highlights recent advances and challenges, and discusses AI-based testing tools. The discussion is organized by major testing and quality areas, such as test generation and defect prediction, encompassing functional and non-functional aspects [11].

AI for Test Case Generation and Test Automation

A leading application of AI in software testing is automatically generating test cases, scripts, or data – reducing manual effort [12, 13]. Search-based testing leverages evolutionary algorithms or meta-heuristic search to create test inputs optimizing objectives like code coverage or fault detection. Notable tools include EvoSuite, which applies genetic algorithms to evolve unit test suites for Java classes, using coverage as a fitness function over multiple generations [14]. Sapienz, another prominent tool, employs a multi-objective genetic algorithm (NSGA-II) to generate GUI action sequences for Android apps, optimizing for both crash discovery and coverage [15]. Sapienz outperformed earlier random testing tools, demonstrating that evolutionary computation is highly effective where coverage or fault detection guides the search process [16]. Beyond search-based approaches, machine learning methods have gained traction. For instance, reinforcement learning (RL) techniques treat the software under test as an environment, learning input sequences to achieve goals like reaching rare states or maximizing code coverage. Recent studies have combined deep learning with RL to generate tests from requirements [17]. In 2024, Alagarsamy et al. introduced PyTester, which employs deep RL to generate executable test cases from natural language requirements, demonstrating promise for early-stage test automation [18]. RL has also been used for test case prioritization based on feedback [19]. Large language models (LLMs) have opened new avenues for neural network-based test code generation. Modern GPT-based coding assistants can create test cases from natural language or code contexts, but these techniques are still emerging in the literature. The key challenge remains the "oracle problem": ensuring that generated tests accurately check expected outcomes. Some research has tackled this by generating assertions or leveraging specifications/contracts when available [20]. Natural Language Processing (NLP) facilitates the linkage between requirements and testing by analyzing requirement documents to generate test scenarios. Recent advances include improved constraint-solving algorithms, AI-augmented model-based testing (where state models are learned via exploratory interaction), and graph-based AI for path coverage. These techniques significantly reduce manual test writing effort and often enhance coverage and bug detection over manual methods [21]. However, several challenges persist. Generated tests must be valid and maintainable, and redundancy or brittleness can be an issue. The oracle problem remains significant: while AI readily generates inputs, determining correct outputs requires human insight or detailed specifications. Scalability to complex systems and developer trust in AI-generated tests are ongoing concerns. Research focuses on human-AI collaboration, such as AI-suggested tests reviewed by humans or AI-formulated assertions based on human-labeled outcomes. AI-based testing tools have gained industrial adoption. Diffblue Cover, for example, uses AI to autonomously generate Java unit tests by analyzing bytecode and applying AI planning for meaningful coverage [22]. Compared to EvoSuite and Randoop, Diffblue produces more concise test suites with slightly lower coverage AI has advanced test generation and automation, reducing human effort and improving maintenance. The remaining gaps include addressing the oracle problem and enhancing the semantic relevance of generated tests [23, 24, 25]. Ongoing work aims to combine large code models with domain knowledge and integrate AI-based tools into continuous integration pipelines [26, 27].

AI for Defect Prediction and Software Quality Assessment

AI plays a significant role in software quality analytics by leveraging machine learning on software data – such as source code metrics, defect histories, and test coverage – to predict where bugs are likely to occur and assess other quality outcomes. These predictions act as upstream quality controls, focusing review and testing where risk is highest [28].

Supervised machine learning is widely used to predict defect-prone code modules. Models are trained on historical data (code metrics and known defects) to classify new or modified code as likely defective. Malhotra's systematic review surveys various machine learning techniques for fault prediction, including decision trees, random forests, support vector machines, and neural networks [29]. These models often achieve good recall for defectprone files, enabling more targeted testing and refactoring [30]. Recent work applies deep and graph neural networks to learn rich code representations. For example, Wang et al. (2020) used machine learning to guide fuzz testing by predicting which inputs would yield deeper code coverage, thereby improving vulnerability discovery [31]. While fuzzing is dynamic, ML's role is analogous to

defect prediction. Despite these advances, model accuracy varies, and retraining is necessary as software or processes evolve [32].

Beyond defect prediction, AI is applied to **internal code quality evaluation**.

Machine learning classifiers can detect code smells and poorly designed code sections. AI-augmented static analysis tools learn error-prone patterns from large codebases. Unsupervised anomaly detection identifies unusual code changes, and clustering methods group code modules by quality, flagging outliers as potentially problematic [33]. AI can also estimate ”technical debt” by learning historical links between code metrics and subsequent refactoring. Bayesian networks and statistical machine learning models can predict the number of bugs remaining or expected. These models take into account factors like test execution trends, code churn, and complexity. While traditional reliability models exist, ML can exploit project-specific patterns for better forecasting.

AI also enhances test result analysis and debugging. Clustering failure reports aids bug triage, while ML classifiers can route bug reports to the right component or team. NLP on bug reports and logs identifies duplicates or suggests likely fault locations (bug localization). Some methods treat debugging as a search problem, using AI planning to propose likely fault sources or fixes – bridging toward automated program repair (e.g., GenProg, which uses evolutionary algorithms). AI-driven predictive models thus function as decision support.

A notable advancement is the direct use of deep learning on code (e.g., code tokens, ASTs, as in code2vec or codeBERT) for defect prediction, which captures richer semantic features than handcrafted metrics. The challenge, however, is the need for large labeled datasets and overcoming the cross-project prediction problem. Transfer learning and domain knowledge integration are common remedies. Additionally, while models may flag a module as risky, actionable explanations (”Why is this file risky?”) are essential for developer trust and adoption.

Ensemble methods in machine learning often surpass traditional statistical models in defect prediction, but success depends heavily on data quality and feature selection. Wider adoption in the industry is still limited by trust and integration challenges. However, as data collection in DevOps environments becomes ubiquitous, the opportunities for AI-powered predictive analytics in software quality continue to expand.

AI in Test Selection, Prioritization, and Execution Optimization

AI techniques are increasingly applied to optimize software testing after test cases are created, with a particular focus on test case selection (choosing a subset of tests to run, e.g., during regression testing) and test case prioritization (ordering tests so those most likely to detect faults run earlier). The aim of **Test Case Prioritization (TCP)** is to schedule tests to maximize objectives such as early fault detection or coverage. Machine learning and meta-heuristic search methods are widely used for this. For instance, classifiers can predict the likelihood of a test revealing a fault based on historical test failures and code changes, enabling ranking by expected value. Reinforcement learning (RL) is also prominent: RL agents learn to select and order tests dynamically by receiving feedback on test outcomes, adapting strategies over time [34]. Su et al. introduced an attention-transfer RL method for TCP in Continuous Integration (CI) environments, allowing prioritization models to adapt between individual test features and broader suite patterns, thereby improving adaptability and fault detection ordering across CI scenarios [35, 36]. Such RL-based approaches are promising for CI, where time constraints require constant test prioritization, and learning-based models can refine their rankings as they observe test failures after code changes [37].

Regression test selection focuses on running only those tests relevant to recent software changes. While traditional methods rely on dependency analysis, AI augments this by learning from historical data, which tests are most likely to catch faults after specific code changes [38]. This has been approached as an information retrieval or recommender system problem, matching tests to code commits [39].

AI is also employed to create a **self-healing testing systems** making test execution more robust. Anomaly detection on test logs can identify flaky tests (which pass and fail inconsistently due to environmental or non-deterministic issues) [40].

In large-scale environments, AI can **optimize how tests are allocated and parallelized**. By forecasting runtimes and failure probabilities, AI-driven schedulers can allocate resources to maximize feedback speed and minimize waste, for example, by scheduling likely-to-fail short tests first or evolving test orders through genetic algorithms to optimize multiple objectives (e.g., fault detection, coverage, execution time) [41].

Despite the benefits, several challenges exist. Models may become outdated if development practices change, requiring continuous learning and adaptation – something RL can help address. The benefits of AI-driven selection are most significant in large, time-constrained suites (e.g., CI for large projects); its value is reduced in small or cheap-to-run test suites. Safety-critical domains may be slower in adopting AI selection due to risk concerns. Nonetheless, studies show that AI-based prioritization improves early fault detection and generalizes across projects [42]. Future research may investigate federated learning for cross-project adaptation and deep learning approaches that predict affected tests directly from code changes.

Conclusion and future research directions

The review demonstrates that up to review date the Kolmogorov-Arnold Networks were not applied to software testing tasks but other AI methods became increasingly vital to software quality control and testing, delivering notable

advancements across test automation, defect prediction, and non- functional testing such as performance and security analysis [43]. AI techniques – including machine learning, deep learning, evolutionary algorithms, and NLP – have enabled greater test coverage, reduced manual effort, faster fault detection, and more intelligent allocation of testing resources [44]. AI-driven tools streamline test generation and maintenance, often catching bugs overlooked by manual approaches, while ML-guided fuzzing has advanced vulnerability discovery. Early identification of risky components through ML enables proactive quality actions [45, 46].

However, several challenges persist. The “oracle problem” remains unresolved; fully automated testing often cannot determine correct outputs without human or formal input. AI-generated tests may only indicate that an event occurred without verifying its correctness [45, 51].

Data dependency is another barrier: supervised models require representative historical data, which may be unavailable, and even when present, model results must be interpreted by engineers. Transparency and trust are ongoing issues – testers and developers may resist ”black-box” AI quality judgments [48]. Certain testing stages, such as usability or requirements validation, also rely on human expertise, where AI offers little assistance. These gaps suggest several future research directions [49].

Progress is needed in automating oracles. Promising directions include leveraging AI to learn expected behaviors or generating metamorphic relations, enabling the creation of assertions for AI-generated tests to check correctness automatically [47]. Integrating specification mining with AI could further support this goal.

Advances in NLP and large language models (LLMs) offer opportunities for AI assistants that interpret requirements and collaborate with testers to generate or refine test scenarios, improving both requirements coverage and clarity.

To keep pace with rapidly evolving software, research should pursue AI that continuously adapts in CI/CD environments, learning from new test outcomes and code changes without manual retraining. Online learning and feedback from production (e.g., DevOps telemetry) could yield more context-aware, up-to-date testing models [50].

While most current research applies individual AI techniques, future systems should combine multiple approaches for holistic QA pipelines. For example, a predictive model could flag risky components, an evolutionary algorithm generate targeted tests, deep learning models cluster failures, and a recommender system suggest fixes – yielding automated, end-to-end QA processes [50].

Rather than aiming for fully autonomous testing, an effective path forward is interactive tools that let testers guide and refine AI-generated tests. Human-in-the-loop approaches leverage human domain knowledge alongside AI’s automation speed, and transparent AI explanations will foster trust and adoption.

Standard benchmarks and datasets are needed for rigorous, comparable evaluation of AI testing techniques. Establishing guidelines or standards for AI use in safety-critical contexts (e.g., healthcare, automotive) will encourage adoption in high- assurance industries.

While AI is making strides in security, further research should extend to attributes such as privacy, ethical compliance, and accessibility.

References

1. BrowserStack: Functional testing: Definition, types & examples (January 2025). URL: https://www.browserstack.com/guide/functional-testing

2. BrowserStack: What is non-functional testing? Definition, types, and tools (January 2025). URL: https://www.browserstack.com/guide/what-is-non-functional-testing

3. The Apache Software Foundation: Apache jmeter (2025). URL: https://jmeter.apache.org/

4. OWASP: Free for open source application security tools (2025). URL: https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools

5. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering 37(5), 649–678 (2010). doi: 10.1109/TSE.2010.62

6. Alshayeb, M.: Empirical investigation of refactoring effect on software quality. International Journal of Electrical and Computer Engineering (IJECE) 13(2), 1823–1832 (2023). doi: 10.11591/ijece.v13i2.pp1823-1832

7. Harman, M., McMinn, P.: A theoretical and empirical study of search- based testing: Local, global, and hybrid search. IEEE Transactions on Software Engineering 36(2), 226–247 (2009). doi: 10.1109/TSE.2009.71

8. Ricca, F., Marchetto, A., Stocco, A.: Ai-based test automation: A grey literature analysis. 2021 IEEE International Conference on Software Testing, Verification and Validation Work- shops pp. 263–270 (2021). doi: 10.1109/ICSTW52544.2021.00051

9. Politowski, C., Guéhéneuc, Y., Petrillo, F.: Towards automated video game testing. Proceedings of the 1st International Workshop on Search-Based Software Engineering pp. 57–64 (2022). doi: 10.1145/3524494.3527627

10. Dobrovolskyi, H., Keberle, N.: Obtaining the minimal terminologically saturated document set with controlled snowball sampling. In: CEUR Workshop Proceedings. vol. 2740, pp. 87–101 (2020).

11. Ali, S., Briand, L., Hemmati, H., Panesar-Walawege, R. K.: A systematic review of the application and empirical investigation of search-based test case generation. IEEE Transactions on Software Engineering 36(6), 742–762 (2009). doi: 10.1109/TSE.2009.52

12. Su, T., Meng, G., Chen, Y., Wu, K., Yang, W., Yao, Y., Pu, G., Liu, Y., Su, Z.: Guided, stochastic model-based gui testing of android apps. In: Proceedings of the 2017 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE). pp. 245–256 (2017). doi: 10.1145/3106237.3106298

13. Liu, Z., Chen, C., Wang, J., Chen, M., Wu, B., Che, X., Wang, D., Wang, Q.: Make llm a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions. Proceedings of the 46th International Conference on Software Engineering pp. 1–13 (2024). doi: 10.1145/3597503.3639180

14. Fraser, G., Wyrich, M.: State of the art in search based software testing: A report from the sbst'18 tool competition (2018).

15. Arcuschin, I. G., Rojas, J. M., Fraser, G., Campos, J.: Evosuite at the sbst 2020 tool competition. In: 2020 IEEE/ACM 13th International Workshop on Search-Based Software Testing (SBST). pp. 33–36. IEEE (2020). doi: 10.1145/3387940.3392186

16. Fraser, G., Zeller, A.: Mutation-driven generation of unit tests and oracles. Proceedings of the 19th International Symposium on Software Testing and Analysis pp. 147–158 (2010). doi: 10.1145/1831708.1831728

17. Dong, Z., Böhme, M., Cojocaru, L., Roychoudhury, A.: Time-travel testing of android apps. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering pp. 687–698 (2020). doi: 10.1145/3377811.3380402

18. Alagarsamy, S., Tantithamthavorn, C., Takerngsaksiri, W., Arora, C., Aleti, A.: Enhancing large language models for text-to-testcase generation. arXiv preprint arXiv:2402.11910 (2024).

19. Li, Z., Harman, M., Hierons, R.M.: Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering 33(4), 225–237 (2007). doi: 10.1109/TSE.2007.38

20. Ellis, K., Wong, L., Nye, M., Sablé-Meyer, M., Morales, L., Hewitt, L., Solar-Lezama, A., Tenenbaum, J. B.: Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning (2023). doi: 10.48550/arXiv.2307.00404

21. Spichkova, M., Muxiddinova, N.: Automated analysis of the scrum agile process using process mining. In: Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). pp. 266–274. SCITEPRESS – Science and Technology Publications (2020). doi: 10.5220/0009417802660274

22. Diffblue: What is diffblue cover? (March2025). URL: https://www.diffblue.com/diffblue-cover/

23. Liu, P., Zhang, X., Pistoia, M., Zheng, Y., Marques, M., Zeng, L.: Automatic text input generation for mobile testing. 2017 IEEE/ACM 39th International Conference on Software Engineering pp. 643–653 (2017). doi: 10.1109/ICSE.2017.65

24. Gu, T., Cao, C., Liu, T., Sun, C.P., Deng, J., Ma, X., Lü, J.: Aimdroid: Activity-insulated multi-level automated testing for android applications. 2017 IEEE International Conference on Software Maintenance and Evolution pp. 103–114 (2017). doi: 10.1109/ICSME.2017.72

25. Liu, Z., Chen, C., Wang, J., Che, X., Huang, Y.K., Hu, J., Wang, Q.: Fill in the blank: Context-aware automated text input generation for mobile gui testing. 2023 IEEE/ACM 45th International Conference on Software Engineering pp. 1355–1367 (2023). doi: 10.1109/ICSE48619.2023.00119

26. Mao, K., Harman, M., Jia, Y.: Sapienz: Multi-objective automated testing for android applications. In: Proceedings of the 38th International Conference on Software Engineering (ICSE). pp. 352–362 (2016). doi: 10.1145/2931037.2931054

27. Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., Yoo, S.: The oracle problem in software testing: A survey. IEEE Transactions on Software Engineering 41(5), 507–525 (2014). doi: 10.1109/TSE.2014.2372785

28. Fraser, G., Arcuri, A.: Evosuite. Proceedings of the 19th ACM SIG-SOFT Symposium and the 13th European Conference on Foundations of Software Engineering pp. 416–419 (2011). doi: 10.1145/2025113.2025179

29. Malhotra, R.: A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing 27, 504–518 (2015). doi: 10.1016/j.asoc.2014.11.023

30. Alshammari, A., Morris, C., Hilton, M., Bell, J.: Flake- flagger: Predicting flakiness without rerunning tests. 2021 IEEE/ACM 43rd International Conference on Software Engineering pp. 1572–1584 (2021). doi: 10.1109/ICSE43902.2021.00140

31. Wang, Y., Jia, P., Liu, L., Huang, C., Liu, Z.: A systematic review of fuzzing based on machine learning techniques. PLOS ONE 15(8), e0237749 (2020). doi: 10.1371/journal.pone.0237749

32. Xie, Q., Memon, A.M.: Designing and comparing automated test oracles for guibased software applications. ACM Transactions on Software Engineering and Methodology 16(1), 4 (2007). doi: 10.1145/1189748.1189752

33. Basavegowda Ramu, V.: Performance testing using machine learning. SSRG International Journal of Computer Science and Engineering 10(6), 36–42 (2023). doi: 10.14445/23488387/IJCSE-V10I6P105

34. Li, Z., Harman, M., Hierons, R.M.: Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering 33(4), 225–237 (2007). doi: 10.1109/TSE.2007.38

35. Su, Q., Li, X., Ren, Y., Qiu, R., Hu, C., Yin, Y.: Attention transfer reinforcement learning for test case prioritization in continuous integration. Applied Sciences 15(4), 2243 (2025). doi: 10.3390/app15042243

36. Pinto, G., Miranda, B., Dissanayake, S., d'Amorim, M., Treude, C., Bertolino, A.: What is the vocabulary of flaky tests? Proceedings of the 17th International Conference on Mining Software Repositories pp. 492–502 (2020). doi: 10.1145/3379597.3387482

37. Chekam, T. T., Papadakis, M., Traon, Y. L., Harman, M.: An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. 2017 IEEE/ACM 39th International Conference on Software Engineering pp. 597–608 (2017). doi: 10.1109/ICSE.2017.61

38. Abdessalem, R. B., Nejati, S., Briand, L., Stifter, T.: Testing vision-based control systems using learnable evolutionary algorithms. 2018 IEEE/ACM 40th International Conference on Software Engineering pp. 1016–1026 (2018). doi: 10.1145/3180155.3180160

39. Zhang, X., Li, Y., Wang, Z.: Rtl regression test selection using machine learning. In: Proceedings of the 27th Asia and South Pacific Design Automation Conference (ASP-DAC). pp. 1–6 (2022). doi: 10.1109/ASP-DAC52403.2022.9712550

40. Mohammed, A. S., Boddapati, N., Mallikarjunaradhya, V., Jiwani, N., Sreeramulu, M. D., Natarajan, Y.: Optimizing real-time task scheduling in cloud-based ai systems using genetic algorithms. In: Proceedings of the 7th International Conference on Contemporary Computing and Informatics (IC3I). pp. 1649–1654 (2025). doi: 10.1109/IC3I61595.2024.10829055, doi: 10.5220/0009417802660274

41. A multi-objective optimization design to generate surrogate machine learning models for predictive maintenance. Computers in Industry (2023). doi: https://www.sciencedirect.com/science/article/pii/S2193943823000134

42. Su, T., Meng, G., Chen, Y., Wu, K., Yang, W., Yao, Y., Pu, G., Liu, Y., Su, Z.: Guided, stochastic model-based gui testing of android apps. In: Proceedings of the 2017 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE). pp. 245–256 (2017). doi: 10.1145/3106237.3106298

43. Leotta, M., Ricca, F., Marchetto, A., Olianas, D.: An empirical study to compare three web test automation approaches: Nlp and capturereplay. Journal of Software: Evolution and Process 35(9), e2606 (2023). doi: 10.1002/smr.2606

44. Harman, M., Hassoun, Y., Lakhotia, K., McMinn, P., Wegener, J.: The impact of input domain reduction on search-based test data generation. Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering pp. 155–164 (2007). doi: 10.1145/1287624.1287647

45. Su, T., Wang, J., Su, Z.: Benchmarking automated gui testing for android against real-world bugs. Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering pp. 119–130 (2021). doi: 10.1145/3468264.3468620

46. Chen, X., Hu, X., Yuan, H., Jiang, H., Ji, W., Jiang, Y., Jiang, Y., Liu, B., Liu, H., Li, X., Lian, X., Meng, G., Xin, P., Sun, H., Shi, L., Wang, B., Wang, C., Wang, J., Wang, T., Xuan, J., Xia, X., Yang, Y., Yang, Y., Li, Z., Zhou, Y., Zhang, L.: Deep learning-based software engineering: Progress, challenges, and opportunities. Science China Information Sciences 67(5), 150101 (2024). doi: 10.1007/s11432-023-4127-5

47. Choudhary, S.R., Gorla, A., Orso, A.: Automated test input generation for android: Are we there yet? arXiv preprint (2015). doi: 10.48550/arXiv.1503.07217

48. Harman, M., Hassoun, Y., Lakhotia, K., McMinn, P., Wegener, J.: The impact of input domain reduction on search-based test data generation. Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering pp. 155–164 (2007). doi: 10.1145/1287624.1287647

49. Adamsen, C.Q., Mezzetti, G., Møller, A.: Systematic execution of android test suites in adverse conditions. Proceedings of the 2015 International Symposium on Software Testing and Analysis pp. 83–93 (2015) doi: 10.1145/2771783.2771786

50. Harman, M., McMinn, P.: A theoretical empirical analysis of evolutionary testing and hill climbing for structural test data generation. Proceedings of the 2007 International Symposium on Software Testing and Analysis pp. 73–83 (2007). doi: 10.1145/1273463.1273475

51. Ji T., Hou Y., Zhang D. A comprehensive survey on Kolmogorov-Arnold Networks (KAN). arXiv preprint arXiv:2407.11075. 2024. DOI 10.48550/arXiv.2407.11075