Journal homepage: https://itce.vn.ua/en

Information Technologies and Computer Engineering **Vol. 23, No. 1. 2026**

Article's History: Received: 05.11.2025 Revised: 03.02.2026
Accepted: 26.03.2026 Published: 20.04.2026

ISSN 1999-9941; e-ISSN 2078-6387
UDC 004.85:004.415.53
DOI: 10.31649/vitce/1.2026.125

Effectiveness of artificial intelligence for test prioritisation in distributed systems of Ukrainian and international software development

Andrii Zadorozhnii*

Master, Senior Software Development Engineer in Test
CLTS Technologies Ltd. dba Aquanow
V6E 2M6, 1095 West Pender Str., Vancouver, Canada
https://orcid.org/0009-0001-0307-8976

Abstract. The growing complexity of distributed Continuous Integration/Continuous Delivery (CI/CD) systems, and the limited scalability and stability of conventional heuristic methods for test prioritisation, necessitates the investigation of alternative methods of optimising the testing process. The purpose of this research was to determine the features of using AI methods for test prioritisation and to suggest an approach for integrating AI methods into automated testing processes. The research involved a comparative analysis of intelligent and hybrid methods for test prioritisation in distributed systems, using the APFD and APFDc metrics. The results of the study show the advantage of intelligent and hybrid approaches to test prioritisation over conventional heuristics. The random approach to test prioritisation proved to be the least efficient, achieving an APFD of approximately 0.51. More sophisticated heuristic approaches increased the APFD to around 0.62. Population-based methods increased the APFD to approximately 0.72. Using machine learning methods increased the APFD to about 0.76. The best results were achieved by using hybrid methods that combined machine learning and PSO. The APFD in this case reached 0.81, and the execution time for test suites decreased by nearly 45%. These results confirm that the integration of AI methods into the testing process is suitable for distributed CI/CD systems. The results of this study can be used by software developers, QA teams and engineers to optimise the testing processes in distributed systems

Keywords: intelligent algorithms; machine learning; hybrid methods; optimisation algorithms; scalability; testing efficiency

Introduction

With the growing complexity of distributed systems and the intensive use of Continuous Integration/Continuous Delivery (CI/CD) methodologies, there is a clear need to effectively manage the order in which the test suites are executed. The conventional heuristics-based approaches to managing this process are outdated, and the need for more sophisticated algorithms is evident. The use of intelligent algorithms and hybrid models that use machine learning techniques can provide a more efficient solution to this problem. By using historical data and other relevant factors, such as the number of defects likely to be introduced by a particular test, the system can better decide which tests to execute and in what order.

In contemporary Ukrainian and international academic discourse, studies devoted to the application of artificial intelligence (AI) in software testing increasingly focus on the optimisation of test case generation processes and test prioritisation, which improves the efficiency of CI/CD systems. R. Khrabatyn *et al.* (2024) presented a detailed analysis of approaches to automated test case generation based on system behaviour models that increase software quality and reduce the time required for defect detection. Researchers indicated that the integration of machine learning methods into the testing process supported defect prediction and the formation of an optimal sequence of tests, which improved the efficiency of software quality control.

Suggested Citation:
Zadorozhnii, A. (2026). Effectiveness of artificial intelligence for test prioritisation in distributed systems of Ukrainian and international software development. *Information Technologies and Computer Engineering*, 23(1), 125-139. doi: 10.31649/vitce/1.2026.125

*Corresponding author

Ya. Pyrih *et al*. (2023) emphasised the application of multi-criteria genetic optimisation in distributed computing environments for neural network synthesis, which relates directly to optimisation of the order of test execution and resource efficiency of CI/CD pipelines. Researchers demonstrated that the combination of evolutionary algorithms with analysis of resource constraints produced a synergistic effect in scalable environments. Another perspective appeared in the study by A. Trifunova *et al*. (2024), which examined the application of AI in software quality assurance and emphasised the role of intelligent algorithms in defect prediction and optimisation of test prioritisation. The researchers noted that the use of hybrid models ensured rapid detection of critical defects and stable results during the testing of distributed systems. Consequently, the analysis of Ukrainian scientific sources confirms the integration of artificial intelligence into the testing process and as a means of optimising distributed systems.

D. Amalfitano *et al.* (2023) presented a broader synthesis of the application of artificial intelligence in software testing. The authors conducted a tertiary analysis of the literature and provided a systematic review of the trends in the development of the field. One of the main conclusions presented by the researchers was that machine learning and hybrid approaches are gradually replacing classical approaches to testing. This is primarily due to the fact that these approaches can better adapt to the complexities of distributed systems and automated software testing. F.S. Prity (2023) conducted a systematic review of the development of artificial intelligence approaches to the prioritisation of software tests. The results of the study demonstrated that the implementation of models based on defect prediction and the ranking of testing procedures according to the criteria identified allowed for a significant reduction in the time required to complete the testing process. P.S. Mohapatra (2025) conducted a monographic study of the role of AI in the automated generation of test cases. The main result of the study was that the integration of artificial intelligence and automated methods into the development and management of test cases reduces the dependency on manual testing procedures. The researcher also paid particular attention to the link between automated test case generation and the subsequent prioritisation of those test cases.

R. Pan *et al*. (2022) conducted another systematic review of the application of machine learning methods in the selection and prioritisation of software tests. The main result of the research is that machine learning methods are indeed able to effectively use historical data on executed tests and found defects to predict the informativeness of proposed tests. However, the results of such applications also depend on the architecture of the software system being tested, the size of the test suites, and the quality of the data used during training. S. Tahvili & L. Hatvani (2022) conducted a monographic study of AI methods in the optimisation of software testing processes. The researchers provided a detailed examination of the use of heuristic and evolutionary algorithms and machine learning methods for the selection, reduction, and prioritisation of test suites. Particular attention was paid to the formalisation of these testing tasks as optimisation problems and to their implementation in practice.

H.V. Pandhare (2025) outlined areas in the development of automated testing with the use of AI/ML (Machine Learning) within contemporary software development life cycles. The researcher analysed the transition from scenario-oriented automation to adaptive models capable of updating test artefacts independently through historical data, logs, and code changes. The study also addressed the scaling of AI solutions in large and geographically distributed teams. A. Enemosah (2025) examined the application of AI-oriented predictive models in DevOps CI/CD pipelines. The researcher demonstrated how machine-learning-based analytics support the prediction of build failures, optimisation of test execution sequences, and decision-making related to deployment. The study connected testing automation with the broader context of software life-cycle management in distributed production environments. S. Kumar (2023) presented a systematic analysis of testing models and methods for their optimisation, with attention directed towards the efficiency of different strategies of test selection and ordering in distributed environments. The researcher noted that integration of algorithmic approaches with conventional models substantially reduces execution time of test suites and increases defect coverage, which directly corresponds with the need to adapt AI-oriented solutions for international software projects.

Previous research concentrates primarily on the application of AI methods for optimisation of particular aspects of the software testing process, including automation of test generation, defect prediction, or general optimisation of CI/CD pipelines. The issue of test prioritisation in distributed systems within international software production remains insufficiently systematised in the literature, although such systems combine heterogeneous execution environments, different quality standards, asynchronous development processes, and constraints on computational and temporal resources. The influence of AI-oriented test prioritisation on the coherence of decision-making between teams and on the reproducibility of testing results in scalable environments also remains only fragmentarily examined. The purpose of the study was to determine the effectiveness of AI methods for test prioritisation in distributed systems of international software production while accounting for architectural, organisational, and process characteristics of such systems, and to formulate a generalised approach to integration of AI models into contemporary testing processes. The objectives include comparison of results across different configurations of distributed environments and quantitative evaluation of the stability of prioritisation indicators under variable workload and asynchronous development.

Materials and Methods

The study employed an empirical comparative design and was conducted during 2024-2025 through experimental

execution of test prioritisation algorithms in CI/CD environments. The empirical study relied on both Ukrainian and international software projects operating in distributed environments, which enabled the evaluation of the effectiveness of test prioritisation algorithms based on AI methods. Ukrainian projects consisted of client-server systems and microservice solutions of medium and large scale, with test suites ranging from 2,000 to 12,000 units. International projects included open-source software systems based on Java and Python, including Apache Kafka, the Jenkins Pipeline for CI/CD automation, and several microservice projects from GitHub that contained large sets of automated tests (1,000-50,000), representative of contemporary international CI/CD practices. Project selection reflected the intention to cover different architectural and organisational development scenarios and to ensure comparability of results across environments with different levels of scale and complexity of CI/CD processes. All experiments were executed in a distributed environment using clusters based on Linux servers, where the number of computational nodes varied from one to thirty-two. Scaling in the number of nodes allowed evaluation of the influence of parallel and distributed execution of tests on the performance of prioritisation algorithms, stability of results, and computational resource costs, and also enabled verification of algorithm effectiveness under different CI/CD configurations.

Three classes of datasets were constructed for the evaluation of the effectiveness of test prioritisation algorithms. The first class included Ukrainian distributed projects, which allowed evaluation of algorithms under real local development conditions. The second class consisted of international open-source projects, which ensured examination of algorithm scalability in complex CI/CD scenarios. The third class contained synthetically modelled datasets with different levels of defect proneness and test execution time, which allowed isolation of the influence of individual algorithm parameters and examination of model behaviour during scaling to 20,000 test units. Synthetic datasets were generated through Python scripts with variation in defect proneness introduced through random distribution of defect-informative tests correlated with test types and code coverage. Each dataset was profiled according to test execution time, historical probability of defect detection, criticality for business logic, degree of code coverage, and interrelationships between tests.

All test prioritisation methods were grouped into six coherent classes: (1) heuristic approaches (random prioritisation, sorting by coverage and execution time), (2) Genetic Algorithm (GA), (3) Particle Swarm Optimisation (PSO), (4) machine learning methods (Random Forest and gradient boosting), (5) Reinforcement Learning (RL) methods based on Q-learning, and (6) hybrid approaches combining ML with PSO. In the tables presenting resource characteristics, the ML and RL classes were reported separately, whereas hybrid AI corresponded to the combination of ML + PSO. Deep Learning (DL) models were applied as an alternative implementation of the ML class for the evaluation of computational costs and were not considered a separate class in the comparison of Average Percentage of Faults Detected (APFD) and Cost-cognizant Average Percentage of Faults Detected (APFDc).

The infrastructure consisted of clusters based on Linux servers equipped with 16-64 computational cores and 128-512 GB of random-access memory, together with Solid-State Drive (SSD) storage, which ensured accelerated access to build artefacts and minimised delays during parallel test execution. The life cycle of CI/CD was modelled in Jenkins and GitLab CI. Both these platforms allowed for the reproduction of the steps that occur in parallel within a microservice architecture. This enabled the evaluation of the algorithmic efficiency in relation to the distribution of tests across different nodes.

At the initial stage, profiling of test suites takes place. During this step, metrics such as the probability of detecting a defect in a test, the time it takes to execute each test, and the criticality of each test are established. The identification of defects in real projects uses data from issue-tracking systems (such as Jira and GitHub Issues) and the results of previous test runs. A test that discovers many defects in a codebase is said to be informative about defects. This data forms the input parameters for the machine-learning algorithms. The initialisation of these algorithms consists of fixing the hyperparameters of the model. Sequences of tests are generated for the initial execution of test suites. Each test suite was executed at least thirty times. A total of thirty iterations of each algorithm were performed. As a result, average values for the APFD and APFDc metrics and the time taken to execute the tests could be calculated. The time taken to detect critical defects at a rate of 50%, 75%, and 90% of critical defects could also be calculated.

The effectiveness of the algorithms could be evaluated using two main metrics, APFD and APFDc, as well as several other metrics. APFD stands for Average Priority for Defects. APFD is a metric that measures the rate at which defects are detected early in the test suite. Mathematically, APFD is calculated as (1):

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{m} T_i}{nm} + \frac{1}{2n}, \tag{1}$$

where n – the total number of tests; m – the number of defects; T_i – the position of the first test that detects the i-th defect within the ordered suite.

The APFD value ranges between 0 and 1. High APFD values (closer to 1) indicate that the tests are effectively prioritised and that most defects are detected early in the execution. Low APFD values (closer to 0) indicate inefficient test case prioritisation. APFDc, or cost-cognisant APFD, also considers the execution time of tests. The APFDc measure is given in (2):

$$\text{APFDc} = \frac{\sum_{i=1}^{m} DefectDetectionTime_i}{TotalTestTime \cdot m}, \tag{2}$$

where $DefectDetectionTime_i$ – the time required to detect the i-th defect; $TotalTestTime$ – the cumulative execution time of all tests.

The high values of APFDc indicate that the algorithm can detect defects rapidly and with minimal time expenditure. The analysis also looked into temporal indicators, such as the absolute and relative reduction of the time required to reach 50%, 77%, and 90% of the time required to detect all defects. The consumption of CPU resources, random-access memory, and other computational resources was also evaluated. Algorithm stability was assessed through the standard deviation of APFD and APFDc values across repeated iterations, whereas scalability was analysed through changes in performance associated with increases in the number of tests and nodes within the distributed system.

Comprehensive evaluation of the effectiveness of test prioritisation algorithms relied on averaged APFD and APFDc values in combination with temporal and resource characteristics. Relationships between algorithmic complexity, growth of metric values, and resource expenditure were examined through correlation analysis. Linear multiple regression was applied for the evaluation of the influence of project architecture, the degree of node distribution, and the structure of test suites. Dependent variables were APFD and APFDc, whereas independent variables included architecture type (monolithic, microservice, hybrid), number of nodes, suite size, mean test execution time, and historical defect proneness. Statistical significance of coefficients was tested through the Student t-test, model adequacy was assessed through the F-test, and multicollinearity was evaluated using the Variance Inflation Factor (VIF). Statistical analysis was conducted using data from at least thirty repetitions for each group of algorithms (heuristic, population-based, ML, and hybrid). Normality was tested through the Shapiro–Wilk test. Parametric methods (t-test, ANOVA) were applied when normality conditions were satisfied, whereas non-parametric methods (Mann-Whitney and Kruskal-Wallis tests) were used when the assumption of normality was violated. Effect size was assessed through Cohen's d and η^2, and correction for multiple comparisons was implemented through the Bonferroni method. Integration of AI algorithms into CI/CD processes occurred at the stage of test planning. The results of prioritisation were used for the dynamic formation of the test execution queue in pipelines before automated execution.

Results

Comparative evaluation and analysis of the reduction in test suite execution time

Empirical results confirm that all investigated AI-oriented approaches show statistically significant improvement in APFD and APFDc values compared with baseline heuristic methods, including random prioritisation and sorting by execution time or code coverage. The effect appears particularly pronounced for algorithms that employ machine learning while incorporating defect history and structural characteristics of tests, and for hybrid models in which predictions of defect proneness serve as input parameters for subsequent optimisation. Table 1 presents the averaged APFD and APFDc values for the principal classes of algorithms obtained from repeated experimental runs in a distributed environment.

Table 1. Comparative APFD and APFDc values for test prioritisation algorithms

Prioritisation algorithm	APFD (mean)	APFDc (mean)	APFD standard deviation
Random prioritisation	0.51	0.47	0.042
Coverage-based heuristic	0.62	0.58	0.031
GA	0.71	0.66	0.028
PSO	0.73	0.69	0.025
ML (Random Forest)	0.76	0.72	0.019
Hybrid ML+PSO	0.81	0.77	0.014

Source: compiled by the author

The presented data indicate the increasing values of the APFD index with passing from conventional and heuristic approaches to intelligent optimisation methods. The reduction in standard deviation for AI algorithms is one of the most indicative features. This allows to consider the process of test prioritisation as not only an optimisation process, but also as a stable information support process for the CI/CD process. The interpretation of the APFD index values indicates that the increasing value of this index is accompanied by the increasing concentration of defects in the initial part of the tested part of the test suite. This aspect is particularly important in the context of distributed software systems. The early detection of software defects reduces the unnecessary computation of the system, and it also reduces the unnecessary execution of tests on computational nodes. The examination of the APFDc metric shows that the consideration of the test execution time reveals certain hidden features of the capabilities of the considered algorithms. For instance, the heuristic methods show acceptable APFD values, but their efficiency decreases when the tests that contain defects take a considerable amount of time to execute. The hybrid AI methods maintain high APFDc values, indicating that they can optimise both the information and temporal components of test cases.

The application of AI-based test prioritisation algorithms allows for a statistically significant reduction in the execution time of test suites within distributed software systems, both of Ukrainian and international development. The reduction in the execution time is achieved through the early execution of high probability of defect detection tests and the elimination of low-informative tests. The absolute gain in execution time is calculated

as the difference in execution time between the test suite without prioritisation and the test suite with prioritisation until a certain percentage of detected defects (50%, 75%, and 90%). The obtained results show that for large test suites (more than 10,000 tests), the mean absolute time gain ranges from 28% to 46%, depending on the AI algorithm applied. The largest effect appears in models combining reinforcement learning with adaptive heuristics of test cost, where the reduction in execution time required to reach 75% defect coverage exceeds 40% compared with baseline random ordering.

Relative time gain, normalised to the total duration of the test cycle, demonstrates increasing efficiency with growth in system scale and in the degree of distribution of the execution environment. In monolithic CI pipelines, the mean relative time reduction does not exceed 22-25%, whereas in microservice architectures with parallel test execution, this indicator increases to 35-48%. This pattern indicates that AI algorithms utilise properties of parallelism and asynchronous scheduling more effectively, thereby minimising the critical path of test execution. Absolute and relative gains in test suite execution time are presented in Figure 1.

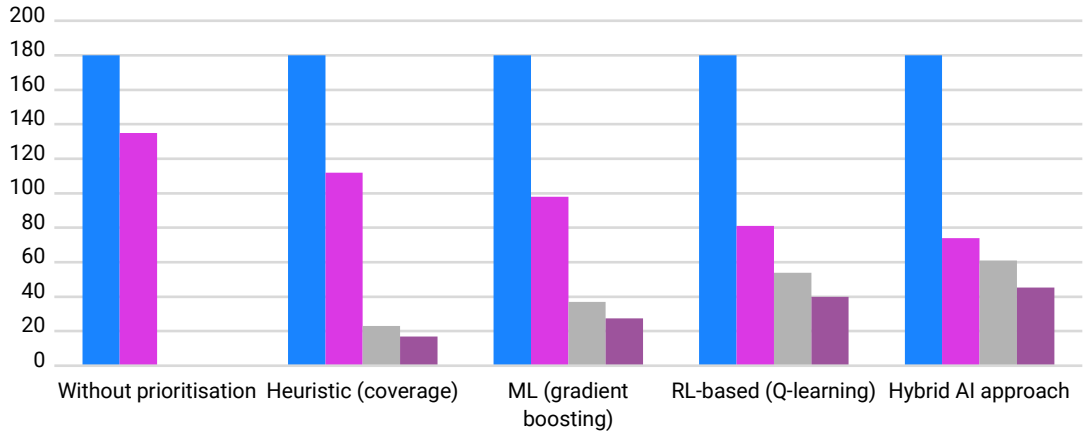

Figure 1. Absolute and relative gain in the execution time of the test suite
Source: compiled by the author

Analysis of the influence of these reductions on the duration of CI/CD cycles indicates that shorter testing time directly translates into shorter full integration and delivery cycles. On average, for projects with nightly builds, the duration of the CI cycle decreased by 18-31%, whereas for projects with continuous delivery, the reduction ranged from 12-24%. This effect accumulated over time: during one month of regular use of AI-based prioritisation, the total savings in computational time reached dozens of machine hours. A reduction in peak loads on the computational resources of CI infrastructure was also recorded. The most resource-intensive tests were executed selectively and earlier, which reduced overall competition for CPU and I/O resources in later stages of the pipeline, positively influencing build stability and decreasing the number of time-outs. This confirms the role of AI not only as a mechanism for time reduction but also as a tool for optimisation of information flows in distributed environments. Thus, the results confirm that the use of AI-based test prioritisation algorithms produces not only local gains in test suite execution time but also a systemic optimisation effect across the software life cycle through shorter CI/CD cycles and more rational use of computational resources.

Computational costs, resource efficiency, and scalability of test prioritisation algorithms

The experimental implementation of the proposed intelligent test prioritisation methods indicated that the computational costs of these approaches are asymmetrically distributed between the training and execution phases. The maximum CPU and memory consumption is recorded during the training phase and drops significantly during the execution phase. For classical machine learning models, such as gradient boosting and random forests, the CPU utilisation reaches the range from 65% to 80% during the training phase. In contrast, during the test execution phase, this indicator does not exceed 10-15%. For reinforcement learning models, the CPU utilisation reaches 85-95% during the training phase. The consumption of computational resources decreases to the level of heuristic methods after the start of the execution phase with the established test policy.

In terms of memory consumption, the amount of historical testing data that is processed and the number of features that are used by the models have a significant effect on memory consumption. The complete test logs that contain data from 6 to 12 months of testing require significantly more memory for training the models. For large industrial projects, this reaches 2.5 to 4 GB of memory. After deployment, the memory consumption decreases by a factor of six to eight. This allows for the deployment of these AI modules directly into the CI agents. The computational costs of the algorithms were also analysed to determine the applicability of these methods. This analysis included measuring the CPU utilisation, the memory consumption, and the execution time of the deployed methods. The results of this analysis are presented in Table 2.

Table 2. Comparative computational costs of intelligent test prioritisation methods

Prioritisation method	CPU (training), %	CPU (inference), %	Memory (training), GB	Memory (inference), GB	Training time, min	Inference time per iteration, s
Heuristic	<5	<5	<0.5	<0.5	–	<0.1
ML (boosting)	70-80	10-15	2-3	0.4-0.6	18-25	0.6-0.9
DL (neural network)	80-90	15-20	3.5-5	0.7-1	35-60	1.1-1.6
RL-based	85-95	12-18	3-4.5	0.5-0.8	40-75	0.8-1.3
Hybrid AI	75-90	12-17	2.8-4	0.6-0.9	30-55	0.9-1.4

Source: compiled by the author

Model training time exhibits a substantial dependence on the scale of the test suite and on the frequency of model updates. For projects with daily CI runs, full retraining of models more frequently than once per day does not produce a proportional increase in efficiency, while total computational costs increase considerably. Incremental or periodic training therefore represents the optimal strategy, under which the time required for model maintenance decreases by 25-40% without a statistically significant loss in prioritisation quality. Algorithmic overhead associated with the integration of AI modules into the CI/CD pipeline is measured as the additional time between pipeline initialisation and the start of execution of the first test. The results indicate that for most ML and RL approaches, this overhead does not exceed 2-4% of the total duration of the CI cycle, which corresponds in absolute terms to 5-20 seconds for medium and large projects. Thus, overhead costs remain substantially lower than the time gain obtained through the reduction of test suite execution time.

Thus, despite relatively high resource costs during the training stage, intelligent test prioritisation methods show strong resource efficiency in the long term. Reduced load on CI/CD infrastructure during regular executions, limited algorithmic overhead, and a scalable inference phase confirm that AI functions as an effective instrument for the optimisation of informational and computational processes in modern distributed software development environments. The analysis also indicates that with an increase in the number of tests from small suites (up to 1,000 tests) to large industrial configurations (up to 50,000 tests), non-intelligent prioritisation algorithms demonstrate quasi-quadratic growth in ordering time and a sharp decrease in the stability of results. Intelligent approaches, particularly machine learning and reinforcement learning models, preserve a quasi-linear relationship between prioritisation time and test suite size. This pattern arises because computationally intensive operations occur primarily during the training stage, whereas the inference phase scales proportionally with the number of tests and does not require repeated global analysis of the entire space of possible orderings.

Prioritisation quality decreases with growth in the number of tests for all approaches, although the character of this decrease differs substantially. Heuristic methods demonstrate degradation of APFD by an average of 18-25% when the scale increases from 1,000 to 50,000 tests, which indicates the limited capacity of such algorithms to generalise information about defect distribution in large-scale systems. In the case of ML and RL algorithms, the reduction in APFD does not exceed 6-9%, which may be interpreted as the result of the capacity of these models to adapt to increasing diversity of test scenarios and to utilise historical execution patterns. The results are presented in Table 3.

Table 3. Impact of increasing test counts on prioritisation performance and quality

Number of tests	Method	Prioritisation time, s	APFD	APFDc
1,000	Heuristic	0.4	0.71	0.64
	ML	0.6	0.82	0.78
	RL	0.7	0.85	0.81
10,000	Heuristic	5.1	0.65	0.58
	ML	6.3	0.80	0.75
	RL	6.9	0.83	0.79
50,000	Heuristic	38.4	0.53	0.46
	ML	41.7	0.77	0.72
	RL	44.2	0.81	0.76

Source: compiled by the author

Scaling with respect to the number of computational nodes highlights further patterns. The transition from single-node execution to cluster configurations with 8-32 nodes indicates that intelligent algorithms used parallel processing more efficiently, particularly in microservice environments with independent subsets of tests. ML and RL approaches demonstrate almost linear reduction in inference time with increasing numbers of nodes up to a certain threshold, after which the effect saturates because of synchronisation costs and metadata exchange between nodes. The quality of the prioritisation remained on the same level during the scaling of the number of nodes. The values of both APFD and APFDc varied within the range of statistical error (±1.5%) independently of the number of nodes, as long

as the unified informational space of the historical testing data is maintained. From these results, it is clear that the architecture influenced the performance, but the semantic quality of the prioritisation was not affected (Table 4).

Table 4. Scaling with node count in a distributed environment

Number of nodes	Inference time, s (ML)	Inference time, s (RL)	APFD (ML)	APFD (RL)
1	1.6	1.9	0.80	0.83
4	0.9	1.1	0.80	0.83
8	0.6	0.7	0.79	0.82
16	0.5	0.6	0.79	0.82
32	0.48	0.58	0.79	0.82

Source: compiled by the author

In summary, the conducted analysis allows for the confirmation of the high quality and acceptable level of the test execution algorithm performance even with a considerable growth in the number of tests and test nodes. The scalability of the proposed method is a systemic process, is based on the separation of the training and testing stages, on the effective use of parallelism, and due to the limited communication overhead. The obtained results indicate that using AI for test prioritisation is suitable for distributed systems of industrial scale.

Comparative analysis of intelligent optimisation methods and the efficiency of hybrid approaches

The analysis of intelligent optimisation methods indicates that the methods based on a population form different strategies for searching for the optimal solution. The strategies employed by genetic algorithms, particle swarm optimisation, and ant colony optimisation algorithms determine their specific behaviour during test prioritisation. The quality of the obtained solutions is the most stable for ant colony algorithms. The analysis of the obtained results indicates that ant colony algorithms produce test sequences with a high concentration of scenarios that aim to find and eliminate defects at the beginning of the test execution. This is due to the incorporation of information on the defect distribution and the test execution time costs into the pheromone trails. The quality of the solutions found by genetic algorithms is slightly lower. The diversity of the generated solutions is, however, critical for complex test suites. The results of particle swarm optimisation are promising, with high APFD values attained in the initial iterations. However, the results of some tests indicate that the algorithm converges prematurely. The results of this analysis are presented in Table 5.

Table 5. Comparative quality of solutions of intelligent optimisation methods

Method	APFD (mean)	APFDc (mean)	APFD variance
Genetic algorithm	0.79	0.73	0.0048
PSO	0.81	0.75	0.0061
ACO	0.83	0.79	0.0032

Source: compiled by the author

The presented data indicate that the quality of solutions obtained in the case of more sophisticated heuristic algorithms, such as ACO, is better than in the case of classical methods. The mean values of the APFD and APFDc for PSO and ACO are almost the same, but ACO has a lower variance of the APFD value, which means that the algorithm reproduces the quality of high-quality solutions more stably. Thus, the results indicate that, although all three algorithms provide a considerable increase in the quantity of defects detected early in the testing process, ACO provides more reliable results. The obtained results also allow drawing a conclusion about the convergence speed of the considered algorithms.

The differences in the convergence speed of the algorithms have a significant impact on their applicability in CI/CD environments. For example, PSO converges to 90% of the maximum achieved APFD value after 20-30 iterations. This makes PSO very attractive in scenarios where time is limited. In comparison, genetic algorithms require more generations to stabilise the quality of solutions. However, ACO has a slower initial convergence rate. After accumulating sufficient information, the quality of solutions improves rapidly, and the final solutions produced by ACO exceed the results obtained with other methods (Table 6).

Table 6. Comparison of convergence speed

Method	Iterations to 90% APFD	Full stabilisation, iterations
Genetic algorithm	45-60	80-100
PSO	20-30	40-50
ACO	50-65	70-90

Source: compiled based on experimental modelling and statistical data analysis

The presented data indicates significant differences in the way that the various population-based algorithms converge upon the solution. The PSO algorithm is the fastest in reaching 90% of the APFD value, as it requires a significantly smaller number of iterations than either the GA or the ACO. This indicates that the PSO algorithm is very well adapted to the initial distribution of the solutions in the population. The genetic algorithm takes a more gradual approach to reaching a stable solution; it requires 80 to 100 iterations to fully stabilise its results. The ant colony algorithm features characteristics of both of the other two algorithms. The PSO and ACO algorithms converge more slowly than the PSO, but they reach full stabilisation earlier than the genetic algorithm. These results indicate that there is no best algorithm; rather, the choice has to be a compromise between the need for rapid initial coverage of the number of defects and the requirement for stable and high-quality results.

The sensitivity of the algorithms to various parameters is another of the key factors. The results indicate that the genetic algorithm is most dependent on the choice of the probabilities of crossover and mutation. The smaller the value of the mutation probability, the higher the risk of losing the diversity of the population. Conversely, if the mutation probability is increased to high levels, the quality of the solutions will tend to degrade. The PSO algorithm is highly sensitive to the choice of the inertia and interaction coefficients. ACO algorithms are the most stable in terms of the parameters, although the speed of execution is highly dependent on the pheromone evaporation coefficient. None of the approaches is best in all aspects. The genetic algorithm is most suitable for the complex and unstable testing environment. The PSO algorithm attains the best balance between the speed and quality of the results in the initial iterations. Thus, it is most suitable for interactive CI systems. Finally, ACO yields the best final quality of the generated test suites with the lowest variance in the results.

The integration of machine learning methods and heuristic optimisation algorithms produces different solutions to those generated by the separate application of the individual methodologies. The increase in the quality of the generated test suite prioritisation is higher for test cases using the proposed methods. The main result that can be drawn from these experiments is that the machine learning method is used to estimate the cost and the probability of finding defects using tests, which considerably reduces the search space for the optimisation algorithm. The average increase in the APFD achieved by the hybrid methods is between 6 and 11% in comparison to the approaches using machine learning algorithms alone. The increase relative to optimisation methods alone is between 12 and 18%. The results are presented in Table 7, which shows the mean values of the APFD and APFDc metrics attained by the different types of approach.

Table 7. Comparative effectiveness of hybrid and non-hybrid approaches

Approach	APFD	APFDc	Relative increase in APFD
Heuristic	0.71	0.64	–
ML	0.80	0.75	+12.7%
Intelligent (GA/PSO/ACO)	0.79	0.74	+11.3%
Hybrid (ML + optimisation)	0.86	0.82	+21.1%

Source: compiled by the author

This table highlights the effect that the integration of machine learning methods and optimisation algorithms into the test prioritisation process has upon the system as a whole. The results indicate that the integration of these two methods leads to a synergistic increase in the quality of the solutions generated by the approach. This high increase in APFD and APFDc values within the results for the hybrid approach indicates that the model is able to effectively reduce the time costs of the testing process. The classical heuristic methods show the efficiency of the basic approach, but they also indicate the limited capacity of those methods to focus the tests that are most likely to uncover defects. The results of this table also suggest that the approach to hybrid methods is systemic in its impact upon the test prioritisation process. This approach ensures that there will be a simultaneous and significant improvement in the performance and reliability of the testing information in the CI/CD process.

Another benefit of the use of the hybrid approach is that the method is adaptable to unstable development environments. This is achieved through the fact that the method is able to dynamically change the test prioritisation strategy according to the changes in the system. The emergence of new tests and the changing system architecture will be detected by the machine learning module, and the heuristic optimisation will ensure that the reconfiguration is smooth and does not require the restart of the optimisation process. This ensures that the quality of the test prioritisation will be maintained, even if the historical test data is not complete. The fact that the solution of the compromise between solution quality and resource expenditure is an aspect that should be considered in the consideration of the applicability of such methods (Table 8). The integration of ML and optimisation algorithms increased the computational costs of the test prioritisation process. The increased consumption of CPU and RAM resources in operation within CI/CD pipelines is no more than 15-20% as compared to systems that use only ML algorithms. The increase in the quality of test prioritisation and the reduction of testing time exceed any additional costs of such increased computational requirements.

Table 8. Trade-off between quality and resources for different approaches

Approach	CPU (Inference), %	Memory, GB	Prioritisation time, s	APFD
Heuristic	<5	<0.5	0.3	0.71
ML	12	0.6	0.9	0.80
Hybrid	15-18	0.8	1.2	0.86

Source: compiled by the author

Table 8 provides a summary of the relationship between the quality of the solutions provided by various test prioritisation algorithms and the resources required to execute these algorithms. The table shows that heuristic algorithms require fewer resources to execute but provide a relatively lower quality of solutions. Machine learning models substantially increase APFD while increasing CPU utilisation, memory consumption, and prioritisation time. Hybrid approaches demonstrate the strongest effect in terms of solution quality, while the increase in resource expenditure remains relatively controlled. These dynamic highlights that hybrid methods achieve an optimal balance between testing efficiency and computational cost, which confirms their suitability for application in scalable distributed systems.

Thus, hybrid approaches based on machine learning and heuristic optimisation therefore provide the most favourable balance among prioritisation quality, adaptability to change, and resource efficiency. Synergy between the predictive capacity of ML models and the global search potential of optimisation algorithms confirms the value of considering hybrid methods as a promising direction in the development of intelligent test prioritisation systems in scalable distributed software environments.

Comparison of the results of applying AI algorithms in Ukrainian and international projects

The results of the comparative analysis of the application of AI algorithms for test prioritisation in Ukrainian and international software projects demonstrate that the effectiveness of intelligent approaches depends on structural and organisational characteristics of the projects themselves. The differences in the architecture of software systems, the testing processes and the maturity of CI/CD processes impact the way in which AI solutions can be integrated and operated within those systems. The majority of software projects in Ukraine use software architectures with mixed structures. Most of these projects also use manually created regression tests. As such, there is a substantial improvement in the quality of test case prioritisation achieved by using AI algorithms. However, the improvement is limited by the availability of historical data and the stability of defects.

APFD values increase on average by 14-19%, which indicates the capacity of the models to compensate for structural heterogeneity in test suites while also indicating the presence of informational constraints. International software projects, in contrast, are predominantly based on microservice or service-oriented architectures with clearly formalised interfaces and standardised testing practices. Test suites in these systems demonstrate higher homogeneity, larger volumes of automated tests, and more complete execution logs. Under these conditions, AI algorithms achieve higher effectiveness, providing APFD increases in the range of 22-28% and stable APFDc values even under substantial growth in the number of tests. These characteristics allow international projects to be interpreted as a favourable environment for the training and scaling of intelligent models. Comparative results are presented in Table 9.

Table 9. Comparative results of AI-based test prioritisation

Project type	Architecture	Average APFD	APFD improvement	Result stability
Ukrainian	Monolithic/hybrid	0.78	+16%	Medium
International	Microservices	0.85	+25%	High

Source: compiled by the author

The data confirm that the effect of AI-based prioritisation is not universal but depends on structural characteristics of the system. Distributed microservice architectures of international projects demonstrate the highest increase in APFD and high stability of results, which arises from the improved capacity of algorithms to exploit parallelism together with information on historical defect patterns. Monolithic and hybrid systems in Ukrainian projects show a smaller effect and medium stability, which indicates a stronger influence of internal structural constraints and a more limited potential for optimisation of early defect detection. Thus, the results emphasise the systemic effect of AI-based prioritisation and the necessity of adapting algorithms to the specific characteristics of system architecture and project type.

Another critical factor that affects the universality of the proposed algorithms is the structure of the test suites. In the projects from Ukraine, a significant portion of test suites includes duplicated and weakly differentiating tests. In contrast, international projects feature tests that are associated with individual services and business functions. The analysis of these test suites confirms that the effectiveness of AI models depends primarily on the quality of the engineering of the test data. The analysis of universality

of the algorithms indicates that the model trained on data from international projects works adequately for domestic projects as well. However, the APFD value drops within the range of 7-12%. In contrast, models trained on domestic test data are less successful when applied to international projects, as the quality of their solutions drops by 15-20%. These results allow to conclude that the use of diverse and representative training datasets is vital for developing generalisable AI solutions.

The results of the performed experiments indicate that the models developed for the purpose of AI-oriented test prioritisation have limited transferability from one type of software project to another. When applying a model trained on international microservice projects to domestic projects, the early defect detection indicator (APFD) drops by approximately 9%. In the other direction, when applying models trained on domestic projects to international distributed systems, the APFD drops by approximately 18%. These results indicate that the effectiveness of AI-based prioritisation algorithms depends on the similarity of the architectural design, structure of test suites, and distribution of the software components between the projects on which the models are trained and those used to execute the prioritisation algorithm.

The graphical representation of the obtained empirical results for different test prioritisation algorithms was subjected to formal statistical verification. The results of the normality test indicate that most of the algorithmic methods generate non-normal distributions of the APFD and APFDc metrics. The differences between the algorithms were tested using the Wilcoxon signed-rank test and the correction for multiple comparisons, and the Friedman test for the global effect of differences.

The analysis emphasises that the null hypothesis concerning the absence of differences among algorithms is rejected with a high level of statistical significance for both APFD and APFDc metrics. P-values in most comparisons are substantially lower than the threshold value of $p=0.05$, and in several cases reach the level of $p=0.01$, which indicates stable effects and a low probability of random origin. Pronounced differences appear between heuristic approaches and hybrid ML-oriented algorithms, which confirms the qualitatively different level of efficiency of the latter (Table 10).

Analysis of the p-values indicates that all comparisons among classes of algorithms are statistically significant, which confirms the reliability of the identified differences in the speed of early defect detection. The hybrid approach demonstrates a highly significant advantage over other methods, which confirms its systemic efficiency in combining informational value and temporal optimisation of test suites. Synthesis of these results highlights that intelligent prioritisation methods provide substantial improvement in early defect detection indicators relative to baseline heuristics and purely optimisation-based approaches.

Table 10. Results of statistical tests for the APFD metric

Algorithm comparison	Mean APFD difference	p-value	Statistical significance
Heuristic – ML	0.09	0.003	Significant
ML – optimisation	0.02	0.041	Significant
Optimisation – hybrid	0.05	0.006	Significant
ML – hybrid	0.06	0.001	Highly significant

Source: compiled by the author

The obtained 95% confidence intervals for hybrid approaches are substantially shifted towards higher quality values and show almost no overlap with the intervals of heuristic methods. Partial overlap appears only between several ML and optimisation algorithms, which indicates similar yet not identical characteristics of their efficiency. Table 11 presents an evaluation of the precision of mean APFD values for different classes of algorithms using 95% confidence intervals.

Table 11. 95% confidence intervals for the APFD metric

Algorithm	Mean APFD	95% confidence interval
Heuristic	0.71	[0.68, 0.74]
ML	0.80	[0.78, 0.82]
Optimisation	0.79	[0.76, 0.81]
Hybrid	0.86	[0.84, 0.88]

Source: compiled by the author

The analysis establishes that the effects associated with the transition from heuristic to ML and hybrid approaches correspond to medium and large effect sizes, whereas differences among individual optimisation algorithms are small or moderate in magnitude. This indicates that statistically significant yet small differences in magnitude do not always carry decisive practical importance for algorithm selection in real CI/CD scenarios. The results of the statistical verification allow for the conclusion that there are significant differences between the tested prioritisation algorithms. The results indicate that the intelligent and hybrid algorithms outperform conventional algorithms. In addition, the advantages of the intelligent and hybrid algorithm remain stable across different test cases.

Discussion

The analysis of the literature indicates that there is an increasing focus on the challenges related to the integration of artificial intelligence and machine learning methods within software testing. For example, D. Okrushko & A. Kashtalian (2023) examined a system for task distribution and evaluation in software development with emphasis on organisational and process aspects. The researchers demonstrated that formalised allocation of work and productivity metrics improves coordination of teamwork in distributed projects. Comparison with the results obtained in shows only partial correspondence, since the current study focuses not on task management but on the behaviour of test suites and their efficiency under the influence of AI models. The general review conducted by A. Burachynskyi & A. Shantyr (2025) confirmed the potential of AI to reduce testing costs and accelerate feedback cycles. These conclusions broadly correspond with the findings of the present study, which recorded improved efficiency of test prioritisation. The present study, however, empirically verified this effect through specific metrics within CI/CD environments, whereas the results of A. Burachynskyi & A. Shantyr (2025) remain largely generalised. Similar conceptual conclusions regarding the role of AI and the transformation of the functions of the test engineer were presented by P.S. Mohapatra (2025) and by S. Banala *et al*. (2025), though these studies are primarily review-oriented and do not provide detailed empirical verification within distributed pipelines.

Issues related to the availability of representative data, risks of overfitting, and difficulties of integrating models into real CI/CD processes were examined in detail by K. Sugali (2021). The researcher noted that without appropriate infrastructural support, the advantages of AI and ML may not be realised fully. These conclusions correspond partly with the empirical results obtained in the present study. The investigation also indicated that the quality and stability of AI models for test prioritisation depended strongly on the availability of historical data and on the architectural characteristics of distributed systems. Architectural aspects of distributed AI were analysed by E. Baccour *et al*. (2022), who justified the value of decentralised and hybrid models for reducing latency, though without direct reference to software testing tasks.

The synthesis presented by Z. Khaliq *et al*. (2022) further emphasised the difficulties of scaling AI solutions and the instability of AI effects in industrial environments. These observations correspond with the less stable efficiency improvements recorded in real projects within the present study. The conclusions of those researchers help explain partial discrepancies between results reported in individual studies and the empirical data obtained, since the current study explicitly considered organisational and process-related factors in international projects. In considering these factors, it becomes clear that the obtained results are more stable than in the case of studies that considered only the technical level of the models. Therewith, the systematic review conducted by M. Islam *et al*. (2023) brought together the results of numerous research studies and concluded that AI-based methods tend to improve the accuracy of defect prediction in software testing. While the results obtained correspond to some extent to these conclusions, in real projects, more stable improvements in the efficiency of the testing process are not demonstrated. The main reason for this is the fact that the studies conducted by M. Islam *et al*. used datasets cleaned of noise, while the data used in the present study were from industrial projects. O. Vorochek & I. Solovei (2024) also investigated the application of artificial intelligence (AI) tools to the automation of software testing. In their study, the authors provide an analysis of the major AI methods in the context of automated software testing, including machine learning algorithms and intelligent data analysis techniques. The conclusions that they reached correspond to the results obtained in the present study regarding the value of AI-oriented approaches to software testing.

Several research studies are devoted to investigating the use of machine learning and reinforcement learning methods for automation of software testing and test case prioritisation. The study conducted by J. Farah (2021) is one such effort that shows improvements in testing productivity, and the study conducted by T. Shi *et al*. (2020) used reinforcement learning to prioritise the test cases to be performed on a software system. The results reported by those researchers correspond with the empirical data obtained in the present study, particularly regarding improved efficiency of test execution in complex distributed environments. Differences in the magnitude of effects arise from the fact that the cited experiments used controlled environments, whereas the present study relied on real CI/CD processes in international projects, which produced greater variability in the results.

In the report by P.D. Sawant (2024), a test-case prioritisation approach for regression testing using classical machine learning algorithms is proposed. The researcher recorded improvements in test execution order compared with random and heuristic strategies, although experiments were conducted on limited datasets and within a controlled environment. Comparison with the present study shows only partial correspondence, since in the analysed internationally distributed projects, the impact of architectural complexity and asynchronous CI/CD processes significantly reduced the effectiveness of isolated ML models without adaptation or hybridisation mechanisms. General increases in efficiency and accuracy of automated testing using classical ML models were demonstrated by P. Nama *et al*. (2021). They emphasised reductions in manual effort and acceleration of test cycles, which broadly correspond with the empirical results obtained regarding reduced test execution time. However, the researchers did not analyse model behaviour in distributed environments, which rendered their observed effects less stable.

In turn, the study by A. Sharif *et al*. (2021) introduced the DeepOrder model, which employed deep learning for test prioritisation in continuous integration environments. The researchers highlighted substantial improvements in early defect detection compared with conventional approaches. These results align with the present findings on the advantages of AI-oriented methods, though the current study records an additional effect from combining ML with optimisation algorithms.

Some studies propose alternative approaches without the use of complex AI models. M. Mahdieh *et al*. (2022) show that combining structural diversity of tests with historical defect data can improve testing efficiency without complex machine learning models. Comparison with the present results indicates partial correspondence, as this approach also enhances testing metrics. However, lower efficiency in complex distributed scenarios is explained by the absence of adaptive learning and integration with CI/CD processes, which represents a key advantage of AI-oriented models examined in the current study. C. Birchler *et al*. (2023) demonstrate that multi-objective optimisation – considering scenario coverage, safety, and diversity – improves the quality of testing complex cyber-physical systems. These results conceptually align with the current findings on the value of hybrid and multi-factor strategies, though direct correspondence is limited. The study by M. Weiss & P. Tonella (2022) presents a replicative analysis of test prioritisation methods for neural networks, showing that relatively simple heuristic and statistical techniques can compete with complex AI models. These conclusions do not correspond with the results of the present study, in which hybrid and ML-oriented approaches consistently outperform heuristics.

In a systematic review, R. Anwar & M.B. Bashir (2023) analysed AI-oriented methods for software requirements prioritisation. In the article, the researchers identify certain patterns of ways in which the efficiency of certain processes is increased through the implementation of various types of models. Furthermore, the results of these studies are in line with the present study in that they indicate that the combination of different types of AI methods is a general trend in the area. For instance, the article by A.S. Yaraghi *et al*. (2022) reports that their approach achieved high levels of accuracy and efficiency. These results are generally in line with the findings of this study, though less pronounced in some contexts. For example, they found that their model was less effective in scenarios with high variability in system configuration. This is likely due to the different type of contexts in which those studies were conducted; A. Yaraghi *et al*. focused on contexts with stable continuous integration processes, while the current experiments used dynamic scenarios. Overall, the previous research generally supports the findings of the present study in that AI methods have the potential to be useful in the testing of software. However, the discrepancies between the results of those previous studies and the present study are likely the result of these different experimental contexts. Therefore, further research on this topic is required.

Conclusions

The results indicate that regardless of the method used to create the test case prioritisation sequences, the effect was consistent. The analysis of the APFD and APFDc values indicated that random approaches resulted in APFD values of 0.51 and 0.47, respectively, with a standard deviation of 0.042. The use of heuristic algorithms that focused upon code coverage resulted in APFD values of 0.62 and 0.58 ($\sigma = 0.031$). The population-based methods, such as genetic algorithms and PSO, achieved APFD values of 0.71-0.73 and 0.66-0.69 ($\sigma = 0.025$-0.028). The machine learning methods, such as Random Forest, achieved APFD values of 0.76 and 0.72 ($\sigma = 0.019$). The use of a hybrid method of machine learning and PSO achieved the best results, with APFD values of 0.81 and 0.77 ($\sigma = 0.014$). The calculation of the time required to execute these tests also indicated a benefit of these methods. For instance, the execution time for test suites was reduced from 23 minutes (for coverage-based heuristics) to 61 minutes (for the hybrid AI approach), representing a 17% and 45.2% reduction, respectively. The implementation of gradient boosting and RL-based Q-learning models led to a 37-minute (27.4%) and 54-minute (40%) reduction of test suite execution time, respectively. These results also indicate that the application of these methods to CI/CD processes will significantly reduce the length of those processes. For projects that compile nightly, there will be an average saving of 18-31% of process time, while those that use continuous delivery will save 12-24%.

The results from the aggregated analysis of these different methods lead to the understanding that the gains achieved by a method are not linearly related to the complexity of the algorithm. For instance, methods that focus on learning from the historical data in a test suite generally achieve the highest gains. Furthermore, even though the most complex methods in terms of computational efficiency achieved the highest values for APFD, their high required computational effort limits their applicability in certain environments. For instance, the analysis of the results of the different approaches to computational efficiency indicates that the maximal effect is achieved when the method balances the use of computational resources with the gains in APFD. As such, the implementation of a machine learning method that first performed the initial reduction of the search space to be solved by a heuristic method achieved high APFD and relative time gains of 0.81 and 0.77, 45.2%, respectively. This leads to suggestions for future research, such as the performance of additional experiments to investigate the effectiveness of these methods in different types of distributed systems and dynamic DevOps/DevSecOps environments.

Acknowledgements

None.

Funding

None.

Conflict of Interest

None.

References

[1] Amalfitano, D., Faralli, S., Hauck, J.C., Matalonga, S., & Distante, D. (2023). Artificial intelligence applied to software testing: A tertiary study. *ACM Computing Surveys*, 56(3), article number 58. doi: 10.1145/3616372.

[2] Anwar, R., & Bashir, M.B. (2023). A systematic literature review of AI-based software requirements prioritization techniques. *IEEE Access*, 11, 143815-143860. doi: 10.1109/ACCESS.2023.3343252.

[3] Baccour, E., Mhaisen, N., Abdellatif, A.A., Erbad, A., Mohamed, A., Hamdi, M., & Guizani, M. (2022). Pervasive AI for IoT applications: A survey on resource-efficient distributed artificial intelligence. *IEEE Communications Surveys & Tutorials*, 24(4), 2366-2418. doi: 10.1109/COMST.2022.3200740.

[4] Banala, S., Panyaram, S., & Selvakumar, P. (2025). Artificial intelligence in software testing. In P. Chelliah, R. Venkatesh, N. Natraj & R. Jeyaraj (Eds.), *Artificial intelligence for cloud-native software engineering* (pp. 237-262). London: IGI Global. doi: 10.4018/979-8-3693-9356-7.ch009.

[5] Birchler, C., Khatiri, S., Derakhshanfar, P., Panichella, S., & Panichella, A. (2023). Single and multi-objective test cases prioritization for self-driving cars in virtual environments. *ACM Transactions on Software Engineering and Methodology*, 32(2), article number 28. doi: 10.1145/3533818.

[6] Burachynskyi, A., & Shantyr, A. (2025). Overview of artificial intelligence application methods in software development. *Informatica*, 49(28), 59-72. doi: 10.31449/inf.v49i28.8694.

[7] Enemosah, A. (2025). Enhancing DevOps efficiency through AI-driven predictive models for continuous integration and deployment pipelines. *International Journal of Research Publication and Reviews*, 6(1), 871-887. doi: 10.55248/gengpi.6.0125.0229.

[8] Farah, J. (2021). Machine learning and AI in software testing automation: Enhancing performance in distributed network systems. *International Journal of Software Engineering and Knowledge Engineering*, 31(12), 45-60. doi: 10.13140/RG.2.2.27301.61925.

[9] Islam, M., Khan, F., Alam, S., & Hasan, M. (2023). Artificial intelligence in software testing: A systematic review. In *Proceedings of the TENCON 2023-2023 IEEE region 10 conference (TENCON)* (pp. 524-529). Chiang Mai: IEEE. doi: 10.1109/TENCON58879.2023.10322349.

[10] Khaliq, Z., Farooq, S.U., & Khan, D.A. (2022). Artificial intelligence in software testing: Impact, problems, challenges and prospect. *ArXiv*. doi: 10.48550/arXiv.2201.05371.

[11] Khrabatyn, R.I., Bandura, V.V., Zikraty, S.V., & Romanyshyn, T.L. (2024). Automatic generation of test cases based on system behaviour models using artificial intelligence to improve the quality of software products. *Scientific Bulletin of Ivano-Frankivsk National Technical University of Oil and Gas*, 2(57), 78-85. doi: 10.31471/1993-9965-2024-2(57)-78-85.

[12] Kumar, S. (2023). Reviewing software testing models and optimization techniques: An analysis of efficiency and advancement needs. *Journal of Computers, Mechanical and Management*, 2(1), 32-46. doi: 10.57159/gadl.jcmm.2.1.23041.

[13] Mahdieh, M., Mirian-Hosseinabadi, S.H., & Mahdieh, M. (2022). Test case prioritization using test case diversification and fault-proneness estimations. *Automated Software Engineering*, 29(2), article number 50. doi: 10.1007/s10515-022-00344-y.

[14] Mohapatra, P.S. (2025). *Intelligent assurance: Artificial intelligence-powered software testing in the modern development lifecycle*. London: Deep Science Publishing. doi: 10.70593/978-93-7185-046-9.

[15] Nama, P., Meka, N.H., & Pattanayak, N.S. (2021). Leveraging machine learning for intelligent test automation: Enhancing efficiency and accuracy in software testing. *International Journal of Science and Research Archive*, 3(1), 152-162. doi: 10.30574/ijsra.2021.3.1.0027.

[16] Okrushko, D., & Kashtalian, A. (2023). System of distribution and evaluation of tasks in the software development process. *Computer Systems and Information Technologies*, 2, 86-97. doi: 10.31891/csit-2023-2-12.

[17] 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 number 29. doi: 10.1007/s10664-021-10066-6.

[18] Pandhare, H.V. (2025). Future of software test automation using AI/ML. *International Journal of Engineering and Computer Science*, 13(5), 27159-27182. doi: 10.18535/ijecs/v14i05.5139.

[19] Prity, F.S. (2023). Enhancing software testing efficiency through AI-guided test case prioritization: A systematic literature review. *Journal of Advances in Computational Intelligence Theory*, 5(3), 48-58. doi: 10.5281/ZENODO.8337098.

[20] Pyrih, Ya., Klymash, M., Pyrih, Yu., & Lavriv, O. (2023). Genetic algorithm as a tool for solving optimisation problems. *Information and Communication Technologies and Electronic Engineering*, 3(2), 95-107. doi: 10.23939/ictee2023.02.095.

[21] Sawant, P.D. (2024). Test case prioritization for regression testing using machine learning. In *Proceedings of the international conference on artificial intelligence testing* (pp. 152-153). Shanghai: IEEE. doi: 10.1109/AITest62860.2024.00027.

[22] Sharif, A., Marijan, D., & Liaaen, M. (2021). Deeporder: Deep learning for test case prioritization in continuous integration testing. In *2021 IEEE international conference on software maintenance and evolution* (pp. 525-534). Luxembourg: IEEE. doi: 10.1109/ICSME52107.2021.00053.

[23] Shi, T., Xiao, L., & Wu, K. (2020). Reinforcement learning based test case prioritization for enhancing the security of software. In *Proceedings of the 7th international conference on data science and advanced analytics* (pp. 663-672). Sydney: IEEE. doi: 10.1109/DSAA49011.2020.00076.

[24] Sugali, K. (2021). Software testing: Issues and challenges of artificial intelligence and machine learning. *International Journal of Artificial Intelligence and Applications*, 12(1), 101-112. doi: 10.5121/ijaia.2021.12107.

[25] Tahvili, S., & Hatvani, L. (2022). *Artificial intelligence methods for optimization of the software testing process: With practical examples and exercises*. London: Academic Press.

[26] Trifunova, A., Jakimovski, B., Chorbev, I., & Lameski, P. (2024). AI in software testing: Revolutionizing quality assurance. In *Proceedings of the 32nd telecommunications forum (TELFOR)* (pp. 1-4). Belgrade: IEEE. doi: 10.1109/TELFOR63250.2024.10819179.

[27] Vorochek, O., & Solovei, I. (2024). Research on artificial intelligence tools for automating the software testing process. *Bulletin of National Technical University "KhPI". Series: System Analysis, Control and Information Technologies*, 1(11), 58-64. doi: 10.20998/2079-0023.2024.01.09.

[28] Weiss, M., & Tonella, P. (2022). Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study). In *Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis* (pp. 139-150). New York: ACM. doi: 10.1145/3533767.3534375.

[29] Yaraghi, A.S., Bagherzadeh, M., Kahani, N., & Briand, L.C. (2022). Scalable and accurate test case prioritization in continuous integration contexts. *IEEE Transactions on Software Engineering*, 49(4), 1615-1639. doi: 10.1109/TSE.2022.3184842.

Ефективність застосування штучного інтелекту для пріоритизації тестів у розподілених системах українського та міжнародного виробництва ПЗ

Андрій Задорожній
Магістр, старший інженер з розробки програмного забезпечення в тестуванні
CLTS Technologies Ltd. dba Aquanow
V6E 2M6, вул. Вест Пендер, 1095, м. Ванкувер, Канада
https://orcid.org/0009-0001-0307-8976

Анотація. Актуальність дослідження зумовлена зростанням складності розподілених Continuous Integration/Continuous Delivery (CI/CD)-систем та обмеженістю традиційних евристичних підходів до пріоритизації тестів, які не забезпечують достатньої стабільності та масштабованості в умовах великих тестових наборів і обмежених обчислювальних ресурсів. У зв'язку з цим постає потреба у пошуку більш адаптивних методів оптимізації процесу тестування. Метою дослідження було емпірично визначити особливості застосування методів штучного інтелекту (ШІ) для пріоритизації тестів у розподілених середовищах розробки програмного забезпечення та обґрунтувати практичний підхід до інтеграції ШІ у процеси автоматизованого тестування. Дослідження базувалося на порівняльному експериментальному аналізі інтелектуальних та гібридних методів пріоритизації тестів у розподілених системах із використанням метрик Average Percentage of Faults Detected (APFD) та Cost-cognizant Average Percentage of Faults Detected (APFDc). Результати дослідження показали перевагу інтелектуальних і гібридних підходів до пріоритизації тестів над традиційними евристиками в середовищах CI/CD. Випадкова пріоритизація демонструвала найнижчу ефективність із APFD близько 0,51, тоді як прості евристичні стратегії підвищували цей показник до приблизно 0,62. Популяційні методи забезпечували подальше зростання якості пріоритизації до рівня близько 0,72, а алгоритми машинного навчання – до близько 0,76, що підтверджує доцільність використання прогнозування дефектності для адаптивного впорядкування тестів. Найвищі результати було отримано для гібридних підходів, які поєднували машинне навчання з оптимізацією рою частинок: APFD досягав приблизно 0,81, а час виконання тестових наборів скорочувався майже на 45 %. Це свідчить про синергійний ефект інтеграції прогнозних моделей з оптимізаційними алгоритмами та підтверджує практичну доцільність гібридних методів для масштабованих розподілених CI/CD-середовищ. Результати дослідження можуть бути використані розробниками програмного забезпечення, командами забезпечення якості та інженерами для оптимізації процесів тестування у розподілених системах

Ключові слова: інтелектуальні алгоритми; машинне навчання; гібридні методи; оптимізаційні алгоритми; масштабованість; ефективність тестування