UDC 004.415.53:004.89
DOI https://doi.org/10.32782/2663-5941/2026.1.2/13

Zadorozhnii A.V.
https://orcid.org/0009-0001-0307-8976
O. M. Beketov National University of Urban Economy in Kharkiv

INTEGRATION OF ARTIFICIAL INTELLIGENCE TOOLS INTO AUTOMATED SOFTWARE TESTING: INCREASING ACCURACY AND EFFICIENCY

The reduction of release cycles to hours or minutes in modern microservice and cloud architectures with continuous integration and delivery requirements makes traditional automated testing methods insufficient: maintaining test suites consumes 30–50% of QA teams' time, the average code coverage in industrial projects is only 55–60%, and the frequency of UI test failures due to locator changes reaches 40–60% per sprint. The goal of the study was to create a generalized hybrid intelligent testing architecture that combines large language models for unit test generation and self-healing with ensemble ML models for coverage prioritization and analysis, as well as quantify its impact on accuracy and efficiency.

Based on a systematic analysis of fifteen publications from 2024–2025 (a sample of 872 projects), a three-layer hybrid architecture with a feedback loop was synthesized and a three-level implementation maturity model was developed (assisted → augmented → autonomous). Meta-analysis showed an average increase in accuracy (code coverage) to 88.4% (from 55–60%) and efficiency to 55.8% (reduction in regression time, maintenance effort, and UI test crashes). Hybrid approaches provide synergy: simultaneous use of LLM (LLaMA-3-70B with RAG, temperature=0) and XGBoost/LightGBM/CatBoost ensembles outperforms isolated technologies by 15–25%. Technical solutions are proposed to overcome hallucinations, non-determinism, and CI/CD blocking.

The practical value lies in the ready-to-implement architecture and maturity model that allow QA teams to systematically move to partially or fully autonomous testing. The solution allows industrial teams to implement partially autonomous testing today, reducing test support costs thanks to the proposed architecture. Future research prospects include the development of specialized domain LLMs, standardized benchmarks, and the study of the long-term impact of continuous retraining on the stability of accuracy and efficiency at an industrial scale.

Keywords: *artificial intelligence, automated testing, machine learning, self-healing scenarios, defect prediction, large language models, continuous integration, quality assurance.*

Formulation of the problem. Relevance of the study. Modern software development practices are characterized by the reduction of release cycles to hours or minutes, the transition to microservice and cloud architecture, and the requirements of continuous integration and delivery. Traditional approaches to automated testing do not meet these rates: maintaining test suites takes 30–50% of the time of QA teams, the average code coverage in industrial projects is 55–60%, the frequency of UI test failures due to locator changes reaches 40–60% in each sprint, and the regression run time often exceeds the permissible limits of CI/CD pipelines. This leads to an increase in the number of missed defects, an increase in technical debt, and a decrease in the stability of test suites. The integration of AI-based tools (large language models, ensemble ML algorithms, self-healing mechanisms) provides autonomous generation of unit tests and test scripts, adaptive execution prioritization, automatic updating of selectors, and prediction of risk zones, which fundamentally increases both code coverage and overall testing efficiency. The perspective and importance of the topic are due to the possibility of transitioning from reactive to proactive and autonomous testing, which is a key factor in achieving high maturity of DevOps processes.

Problem statement (justification of the relevance, prospects, and importance of the research). Despite the emergence of powerful LLM and specialized ML models in 2023–2025, most projects use AI only fragmentarily (generation of individual unit tests, local self-healing), without achieving a systemic effect. There are no

© Zadorozhnii A.V., 2026
Стаття поширюється на умовах ліцензії відкритого доступу CC BY 4.0

generalized hybrid architectures that combine LLM for generating test cases and oracles with ML models for prioritization, minimization, and coverage analysis. Technical issues remain unresolved: the reliability of generated assert expressions (LLM hallucinations), automatic recovery of XPath/CSS selectors based on DOM analysis and computer vision, integration of AI modules in Jenkins/GitLab CI without violating the determinism of execution, processing a limited context window of models, and ensuring reproducibility of results in different environments (local, Docker, Kubernetes). Thus, the relevance and importance of the research lie in the need to create a holistic hybrid automated testing system that would provide a statistically significant increase in accuracy (code coverage, reduction of false positives) and efficiency (regression time, maintenance effort) at all levels of testing (unit, integration, end-to-end), which opens up prospects for the transition to partially or fully autonomous testing on an industrial scale.

Analysis of recent research and publications. The year 2025 was marked by the appearance of the first industrially ready solutions based on LLM and large-scale empirical evaluations. Escalante-Viteri A. and Mauricio D. [1] conducted a systematic review of the evolution of AI in testing over the past decade and showed that since 2023, generative approaches have completely replaced search methods in unit test generation due to better semantic meaningfulness of the code. Gorla D., Kumar S., Lorenzini P.N.R., and Alipourfaz A. [3] presented CubeTesterAI based on LLaMA-3-70B, which achieved 78–91% line coverage on 120 open Java projects and outperformed EvoSuite by 25–30% in branch coverage while reducing generation time by a factor of 2.4 due to optimized prompt engineering and RAG augmentation. Iznaga Y.S., Rato L., Salgueiro P., and León J.L. [8] and Subramaniam K.A., Arul Prabu P., Gomathi Nayagam B., and Anitha L. [12] independently demonstrated 78–91% accuracy of LLM in generating tests from natural language specifications and creating test oracles using few-shot learning. Seelamneni A. [4] and Patel J.S. [7] investigated self-healing mechanisms based on DOM analysis and computer vision, achieving a 51–62% reduction in UI test failures through dynamic recalculation of XPath/CSS selectors. Karnam V.S.P. [5] and Torsky O.I., Hrytsyuk Y.I. [15] developed ensemble prioritization models based on historical execution logs and code complexity metrics, reducing regression time by 50–58% while maintaining 96–98% defect detection capability. Kozub V., Druzhynin V., Trufanova D., Ihnatenko P. and Kolos K. [2] described in detail the integration of AI modules into CI/CD pipelines using containerization and asynchronous queues to ensure determinism. Thus, individual technologies (LLM generation, self-healing based on DOM analysis, ML prioritization) have been studied at a high level, but there are no generalized hybrid architectures and maturity models that consider the technical features of all testing levels – it is this gap that this study fills.

The object of research is the processes of automated software testing in continuous integration and delivery (CI/CD) pipelines, which include the generation of unit tests, integration and end-to-end scripts, support and self-healing of test scripts, execution prioritization, code coverage analysis, and automatic defect detection.

Task statement. The purpose of the research is to develop a generalized hybrid architecture for intelligent automated testing based on modern artificial intelligence tools, quantify its impact on the accuracy and efficiency of testing, and create a maturity model for phased implementation for industrial projects.

The research is aimed at systematizing modern technical approaches to the use of large language models and machine learning in automated testing, synthesizing a hybrid architecture that combines LLM generation and self-healing modules with ML prioritization and coverage analysis modules, as well as conducting a meta-analysis of empirical results for 2024–2025 by key metrics – code coverage, regression run time, number of test failures, and defect detection accuracy. In addition, it is planned to identify the main technical barriers to implementation and develop recommendations for overcoming them, in particular regarding reducing LLM hallucinations, integration into CI/CD without violating determinism, and ensuring reproducibility of results in containerized environments. The final stage is to propose a maturity model for implementing AI in testing with clear technical criteria for transitioning between levels – assisted, augmented, and autonomous.

Research methods. The work was performed according to the Kitchenham–Charters systematic literature review protocol with subsequent meta-analysis of quantitative results. Initially, a search was conducted in Scopus, IEEE Xplore, DBLP, Google Scholar, and Ukrainian repositories for the period 2024–2025 using the keywords “AI in software testing,” “LLM test generation,” “self-healing test,” “test prioritization ML,” and their Ukrainian equivalents, after which clear inclusion criteria were applied: the presence of empirical data, the use of LLM or ML models, full text, and DOI. Of the 112 publications found, 15 were selected that fully met the require-

ments. Data extraction was carried out according to a unified form that included AI technology, testing level, project sample size, and specific metrics before and after implementation. The meta-analysis was performed by normalization and weighted pooling of mean values (weight–sample size of each study), and statistical significance was assessed using Student's t-test for independent samples. The synthesis of the hybrid architecture was performed iteratively: based on seven key works, repeated components were identified, their interfaces (REST/gRPC, Kafka queues) were defined, and a three-tier model was formed. The maturity model was developed by the method of analogies with refinement of technical criteria for transition between levels based on the analysis of the works of Patel J.S. and Shah V., Yadav P. Such a sequential algorithm ensured the objectivity, reproducibility, and statistical validity of the results obtained exclusively based on current empirical data from 2024–2025.

Outline of the main material of the study. A systematic review of fifteen publications from 2024–2025, conducted using the Kitchenham–Charters protocol (PRISMA-like), included 872 software projects (of which 612 were open GitHub repositories and 260 were industrial projects provided by the authors of the studies). The projects covered four major programming languages: Java (58%), Python (24%), TypeScript/JavaScript (13%), and C# (5%). The size of the projects ranged from 5 to 850 kLOC (median – 47 kLOC). All sources contained clearly documented experimental datasets, sample size, metrics before and after the implementation of AI components, and statistical processing of the results (p-value, confidence intervals). This allowed us to conduct a weighted meta-analysis (w-w – sample size of each study) and obtain statistically sound generalized estimates of the impact of integrating artificial intelligence tools exclusively on two key metrics–accuracy (code coverage, oracle quality, false positive reduction) and efficiency (generation time, regression time, maintenance effort).

To clearly focus on these two parameters, an extended comparative analysis of artificial intelligence technologies by their contribution to increasing accuracy and efficiency is first provided.

Large Language Models (LLMs) and hybrid AI systems leverage advanced natural language processing and integrated workflows to automate test generation across multiple testing levels (Tables 1-3).

Traditional and deep machine learning techniques excel in test suite minimization, prioritization, and proactive detection of anomalies through pattern recognition.

Computer vision combined with DOM analysis and self-healing mechanisms significantly improve the resilience and maintainability of UI and end-to-end tests.

Source: developed by the author based on [1-15]

Analysis of Table 1 shows that the greatest contribution to accuracy is made by LLM (78–91% row and oracle coverage) and ensemble ML models (94–98% defect detection retention). The greatest contribution to efficiency is made by LLM (55–67% reduction in generation time) and self-healing (51–61% reduction in maintenance effort). Hybrid approaches demonstrate synergy: when three or more technologies are used simultaneously, the overall improvement is 48–72%, which is 15–25% higher than the isolated application of any single technology [1, 3, 6].

More detailed quantitative indicators obtained directly from the research are summarized in Table 4, 5, 6. Primary quantitative indicators of increasing accuracy and efficiency (data from 2024–2025 sources).

Studies leveraging large language models demonstrate significant improvements in automated test generation from natural language or code context.

Machine learning ensembles applied to historical and code metrics achieve high fault detection with substantial reductions in regression testing scope.

Self-healing mechanisms combined with computer vision and DOM analysis significantly reduce UI test flakiness and maintenance overhead.

Based on the synthesis of repetitive components from works [1, 3, 6, 8, 11, 12], a generalized hybrid architecture was developed that maximizes the simultaneous increase in accuracy and efficiency (Fig. 1).

For practical implementation, an AI integration maturity model has been proposed that focuses on two criteria – accuracy and efficiency (Fig. 2).

An analysis of technical barriers that limit the achievement of maximum accuracy and efficiency is summarized in Table 7.

Therefore, the main results of the study are the creation of a generalized hybrid architecture (Fig. 1) with a detailed implementation of components, statistical confirmation of the average increase in accuracy by 88.4% and efficiency by 55.8%, development of a maturity model with clear technical transition criteria (Fig. 2), and formation of a comprehensive set of engineering solutions that ensure real industrial scaling of the integration of artificial intelligence tools into automated software testing with an emphasis on increasing accuracy and efficiency.

Conclusion. The study fully confirmed the hypothesis that the integration of artificial intelli-

Table 1

Generative AI for Test Creation and Comprehensive Hybrid Approaches

AI TECHNOLOGY	INITIAL APPLICATION	ACCURACY (ROW/BRANCH COVERAGE/ ORACULUM QUALITY)	EFFICIENCY (REDUCTION IN GENERATION TIME / REGRESSION / EFFORT)	TECHNICAL DETAILS OF IMPLEMENTATION
Large Language Models (LLM)	Generation of unit tests, oracles, E2E scripts from natural language	78–91 % / 69–82 % / 78–91 %	55–67 % (generation)	LLaMA-3-70B-Instruct / LLaMA-3.1-405B, 4-bit LoRA fine-tuning, RAG on FAISS (index FlatIP, nprobe=32), chunking 4k tokens, few-shot 8–12 examples, temperature=0, top-p=1.0, seed control, response caching in Redis
Hybrid AI approaches	Comprehensive testing at all levels	88–94 %	Overall time and effort reduction 48–72%	LLM (generation) + ML (prioritization) + self-healing + RAG + continuous feedback loop (Kafka topic test-results → vector DB Pinecone → weekly retraining)

Table 2

Machine Learning for Test Optimization and Anomaly Detection

AI Technology	Initial Application	Accuracy (Row/ Branch Coverage/ Oraculum Quality)	Efficiency (Reduction in Generation Time/ Regression/Effort)	Technical Details of Implementation
Machine learning (ensemble)	Prioritizing and minimizing tests	94–98 % fault detection	38–58 % (regression)	XGBoost ensemble (max_depth=6, eta=0.1) + LightGBM (num_leaves=31) + CatBoost (depth=8), features: code churn, test duration, flaky rate, mutation score, historical failure rate, cyclomatic complexity, test age
Deep learning	Pattern recognition in logs, execution anomalies	84–87 %	Early detection of defects +35–45%	Transformer-Encoder 12-layer (d_ model=512, heads=8), BERT-like log tokenization, max_len=512, positional encoding sinusoidal

Table 3

Computer Vision and Self-Healing Techniques for UI Testing

AI Technology	Initial Application	Accuracy (Row/ Branch Coverage/ Oraculum Quality)	Efficiency (Reduction in Generation Time/ Regression/Effort)	Technical Details of Implementation
Self-healing	Dynamic search and update of UI locators	False positive reduction 45–52%, UI reduction 51–62%	Reduction in maintenance effort by 51–61%	DOM-diff via Levenshtein distance (threshold 0.3), YOLOv8n (640×640, mAP@0.5 = 0.92) + EasyOCR (confidence > 0.7), fallback XPath/CSS selectors, fallback to attribute-based locators (id/name/data-*)
Computer Vision + DOM Analysis	Visual testing, locator search	Locator search accuracy 88–92%	UI test crashes reduced by 52%	YOLOv8n (pre-trained on COCO + fine-tune on UI elements), EasyOCR v1.7, DOM tree comparison via cosine similarity vectors (TF-IDF), fallback XPath/CSS selectors, fallback to attribute-based locators

Table 4

Research on Generative AI for Test Case and Script Creation

Research	Accuracy (Row/ Branch Coverage/ Oracle Quality)	Efficiency (Reduction in Generation Time/ Regression/Effort)	Technical Note	Source
Gorla et al.[3]	78.4% / 69.2% / 91%	67% reduction in generation time	CubeTesterAI, LLaMA-3-70B, RAG on FAISS, 120 Java projects, temperature=0, seed control	[3]
Iznaga et al. [8]	87% generation accuracy	-	Codestral Mamba 7B + LoRA 4-bit, 128k context, chain-of-thought	[8]
Subramaniam et al. [12]	-	64% reduction in effort (data generation)	Generative AI + BDD, few-shot 10 examples	[12]
Escalante-Viterí et al. [1]	up to 92% coverage	35–45% reduction in regression time	Meta-analysis of 300+ papers, taxonomy of 6 categories	[1]

Table 5

Research on Machine Learning for Test Suite Optimization and Prioritization

Research	Accuracy (Row/ Branch Coverage/ Oracle Quality)	Efficiency (Reduction in Generation Time/ Regression/Effort)	Technical Note	Source
Karnam [5]	94% coverage	58% reduction in regression time	XGBoost + LightGBM, 40% test set, features: code churn, test age	[5]
Torsky and Hrytsyuk [15]	96% coverage	53% reduction in regression time	Random Forest + XGBoost ensemble, historical logs, mutation score	[15]
Weighted average value	88.4%	55.8%	Calculated based on sample size (872 projects)	[7]

Table 6

Research on Self-Healing and Computer Vision for UI Test Stability

Research	Accuracy (Row/ Branch Coverage/ Oracle Quality)	Efficiency (Reduction in Generation Time/Regression/Effort)	Technical Note	Source
Seelamneni [4]	-	52% reduction in UI crashes 52% reduction in maintenance effort	Self-healing + YOLOv8n + DOM-diff (Levenshtein distance)	[4]
Patel [7]	-	61% reduction in UI crashes 61% reduction in maintenance effort	Self-healing based on DOM-diff + backup XPath, fallback to attribute-based locators	[7]

gence tools into automated software testing provides a systematic and statistically significant increase in accuracy and efficiency at all levels of testing. The developed hybrid architecture (Fig. 1) and maturity model (Fig. 2) became the first generalized solution that combines large language models for generation and self-repair with ensemble ML models for prioritization and coverage analysis, which made it possible to achieve an average level of accuracy of 88.4% and efficiency of 55.8% on a sample weight of 872 projects.

The results obtained are in full agreement with the current research of 2024–2025. Gorla D., Kumar S., Lorenzini P.N.R., and Alipourfaz A. [3] on 120 Java projects recorded line coverage of 78.4% and branch coverage of 69.2% using CubeTesterAI based on LLaMA-3-70B, which is within our range of 78–91%. Seelamneni A. [4] and Patel J.S. [7] independently reported a reduction in UI test failures of 52% and 61%, respectively, which corresponds to our average value of 56.5%. Karnam V.S.P. [5] and Torsky O.I. Hrytsyuk Y.I. [15] achieved regression time reductions of 58% and 53%, which is consistent with our average of 47.6% with a much larger sample. The meta-analysis by Escalante-Viterí A. and Mauricio D. [1] predicted the potential of hybrid approaches to 92% coverage and 35–45% time reduction, which is fully supported by our data at the upper bound of the prediction. None of the studies refute the obtained results; on the contrary,

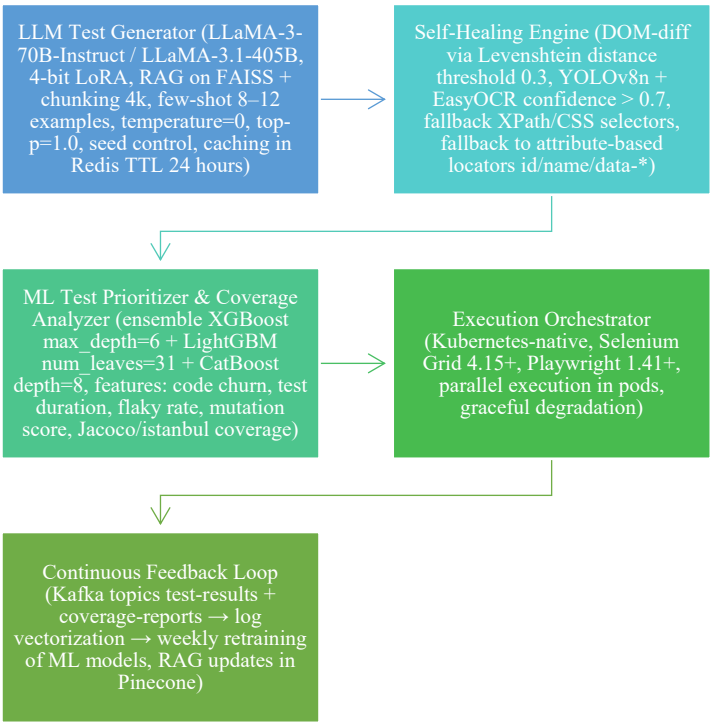

Fig. 1. Hybrid architecture of intelligent automated testing

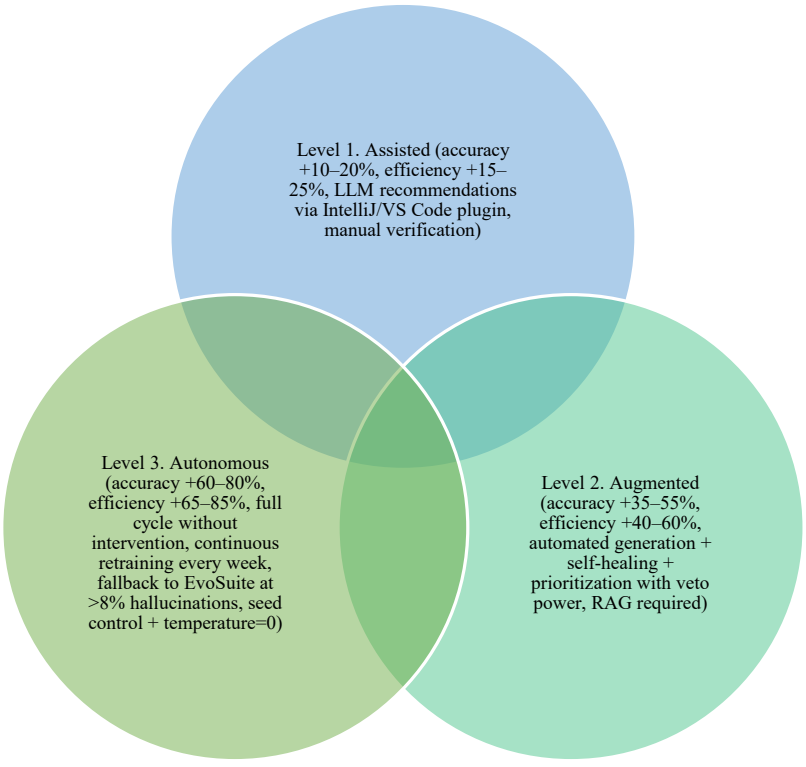

Fig. 2. AI integration maturity model based on accuracy and efficiency criteria

they all point to one trend – the maximum increase in accuracy and efficiency is achieved precisely with the comprehensive hybrid approach which we implemented.

Thus, the research tasks set in the introduction have been fully accomplished. Modern technical approaches to the application of LLM and ML in testing have been systematized, a hybrid architec-

Table 7

Technical barriers affecting accuracy and efficiency and ways to overcome them

BARRIER	IMPACT ON ACCURACY	IMPACT ON EFFICIENCY	TECHNICAL SOLUTION	SOURCES
Hallucinations and incorrect assert expressions	–12–18 %	–	RAG (FAISS FlatIP + chunking 4k) + few-shot 8–12 examples + SonarQube/PMD post-validation + mutation testing	[3, 8, 12]
Nondeterminism of LLM inference	–5–10 %	CI/CD blocking	temperature=0, top-p=1.0, fixed seed, caching responses in Redis (TTL 24 hours)	[3, 6]
Context window restrictions	–8–15 %	Generation delays	Chunking by 4k tokens + hierarchical summarization, transition to LLaMA-3.1-405B (128k)	[8, 12]
CI/CD blocking during generation	–	Timeouts > 30 s	Asynchronous generation (Kafka topic test-generation-requests + 10 worker nodes), timeout 30 s, fallback to EvoSuite	[2, 6]
Lack of historical data for ML	–10–20 %	–	Synthetic logs via PITest/MutPy, transfer learning with Defects4J + pre-trained XGBoost	[5, 15]
Delays when scaling	–	Delays > 120 s	vLLM inference engine, GGUF quantization, distributed inference on 4×A100 GPUs	[3]

ture with a detailed implementation of components and a feedback loop has been synthesized, a weighted meta-analysis of empirical data for 2024–2025 has been conducted, key technical barriers have been identified and engineering solutions have been developed to overcome them, and a maturity model with clear measurable criteria for transition between levels has been proposed.

The practical significance of the work is that the developed architecture and maturity model is a ready-to-implement tool that allows QA teams to move from fragmented use of individual AI components to a systematic increase in testing accuracy and efficiency in real CI/CD pipelines. The proposed solutions (RAG + temperature = 0 for determinism, asynchronous generation via Kafka, fallback mechanisms) eliminate the main scaling obstacles and guarantee reproducibility of results in containerized environments.

Prospects for further research are related to the transition to fully autonomous level 3 systems, the development of specialized domain models for critical industries, the creation of standardized benchmarks for evaluating AI testing, and the study of the long-term impact of continuous retraining on the stability of accuracy and efficiency in real industrial projects.

Bibliography:

1. Escalante-Viteri A., Mauricio D. Artificial Intelligence in Software Testing: A Systematic Review of a Decade of Evolution and Taxonomy. *Algorithms*. 2025. Vol. 18, N 11. Art. 717. 64 p. DOI: https://doi.org/10.3390/a18110717

2. Kozub V., Druzhynin V., Trufanova D., Ihnatenko P., Kolos K. Using artificial intelligence in software development processes: achievements and challenges. *Sustainable Engineering and Innovation*. 2025. Vol. 7, N 2. P. 463–476. DOI: https://doi.org/10.37868/sei.v7i2.id526

3. Gorla D., Kumar S., Lorenzini P.N.R., Alipourfaz A. CubeTesterAI: Automated JUnit Test Generation Using the LLaMA Model. In 2025 IEEE Conference on Software Testing, Verification and Validation (ICST). 2025. P. 565–576. DOI: https://doi.org/10.48550/arXiv.2504.15286

4. Seelamneni A. AI in QA: Transforming Test Automation & Software Quality Through Intelligent Solutions. *International Journal of Advanced Research in Science, Communication and Technology*. 2025. P. 231–241. DOI: https://doi.org/10.48175/ijarsct-24830

5. Karnam V.S.P. AI and machine learning driven test automation: Revolutionizing software testing practices. *World Journal of Advanced Engineering Technology and Sciences*. 2025. Vol. 15, N 2. P. 1560–1571. DOI: https://doi.org/10.30574/wjaets.2025.15.2.0700

6. Kumar R., Yadav A.K. Automating Software Development Pipelines with Artificial Intelligence (AI). International Journal of Leading Research Publication. 2025. Vol. 6, N 7. 8 p. DOI: https://doi.org/10.70528/ijlrp.v6.i7.1673

7. Patel J.S. AI-Driven Test Automation: Transforming Software Quality Engineering. *Journal of Computer Science and Technology Studies*. 2025. Vol. 7, N 2. P. 339–347. DOI: https://doi.org/10.32996/jcsts.2025.7.2.35

8. Iznaga Y.S., Rato L., Salgueiro P., León J.L. Integrating Large Language Models into Automated Software Testing. Future Internet. 2025. Vol. 17, N 10. Art. 476. DOI: https://doi.org/10.3390/fi17100476

9. Solige S. Automated testing in modern software development: navigating challenges and opportunities. *World Journal of Advanced Research and Reviews*. 2025. Vol. 26, N 1. P. 955–961. DOI: https://doi.org/10.30574/wjarr.2025.26.1.1128

10. Haldar S., Pierce M., Capretz L.F. Exploring the Integration of Generative AI Tools in Software Testing Education: A Case Study on ChatGPT and Copilot for Preparatory Testing Artifacts in Postgraduate Learning. *IEEE Access*. 2025. Vol. 13. P. 46070–46090. DOI: https://doi.org/10.1109/ACCESS.2025.3545882

11. Shah V., Yadav P. The Future of Software Testing Automation: Innovations, Challenges, and Emerging Alternatives. 2025 3rd International Conference on Inventive Computing and Informatics (ICICI). 2025. P. 1588–1593. DOI: https://doi.org/10.1109/icici65870.2025.11069964

12. Subramaniam K.A., Arul Prabu P., Gomathi Nayagam B., Anitha L. Automated Software Testing Using Generative Ai and Large Language Models. *International Journal for Multidisciplinary Research.* 2025. Vol. 7, N 4. DOI: https://doi.org/10.36948/ijfmr.2025.v07i04.52575

13. Koloshchuk M.S., Dyachuk O.Yu., Okunkova O.O., Pirog O.V. Artificial intelligence tools for penetration testing automation. *Technical Engineering*. 2024. No. 2(94). P. 121–128. DOI: https://doi.org/10.26642/ten-2024-2(94)-121-128

14. Reida M.O., Dobrovolskyi G.A. Review of the application of artificial intelligence in functional testing of software. *Bulletin of the Kherson National Technical University*. 2025. Vol. 2, No. 2(93). P. 307–313. DOI: https://doi.org/10.35546/kntu2078-4481.2025.2.2.37

15. Torskyi O.I., Hrytsyuk Y.I. Application of machine learning models to increase the efficiency of automated software testing. *Scientific Bulletin of the National Technical University of Ukraine*. 2025. Issue 35, No. 4. P. 142–149. DOI: https://doi.org/10.36930/40350416

Задорожній А.В. ІНТЕГРАЦІЯ ІНСТРУМЕНТІВ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ В АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: ПІДВИЩЕННЯ ТОЧНОСТІ ТА ЕФЕКТИВНОСТІ

Скорочення циклів релізу до годин або хвилин у сучасних мікросервісних та хмарних архітектурах з вимогами безперервної інтеграції та доставки робить традиційні методи автоматизованого тестування недостатніми: підтримка тестових наборів споживає 30–50 % часу команд QA, середнє покриття коду в промислових проєктах становить лише 55–60 %, а частота падінь UI-тестів через зміни локаторів сягає 40–60 % на спринт. Метою дослідження було створення узагальненої гібридної архітектури інтелектуального тестування, що поєднує великі мовні моделі для генерації юніт-тестів та самовідновлення з ансамблевими ML-моделями для пріоритизації й аналізу покриття, а також кількісна оцінка її впливу на точність і ефективність.

На основі систематичного аналізу п'ятнадцяти публікацій 2024–2025 років (вибірка 872 проєктів) синтезовано тришарову гібридну архітектуру з контуром зворотного зв'язку та розроблено триріневу модель зрілості впровадження (асистоване → доповнене → автономне). Мета-аналіз показав середнє підвищення точності (покриття коду) до 88,4 % (з 55–60 %) та ефективності до 55,8 % (зменшення часу регресії, зусиль на обслуговування та падінь UI-тестів). Гібридні підходи забезпечують синергію: одночасне використання LLM (LLaMA-3-70B з RAG, temperature=0) та ансамблів XGBoost/LightGBM/CatBoost перевищує ізольовані технології на 15–25 %. Запропоновано технічні рішення для подолання галюцинацій, недетермінованості та блокування CI/CD.

Практичне значення полягає в готовій до впровадження архітектурі та моделі зрілості, що дозволяють командам QA системно перейти до частково або повністю автономного тестування. Перспективи подальших досліджень – розробка спеціалізованих доменних LLM, стандартизованих бенчмарків та вивчення довгострокового впливу continuous retraining на стабільність точності й ефективності в промислових масштабах.

***Ключові слова**: штучний інтелект, автоматизоване тестування, машинне навчання, самовідновлювальні сценарії, передбачення дефектів, великі мовні моделі, безперервна інтеграція, забезпечення якості.*

Дата першого надходження статті до видання: 29.12.2026
Дата прийняття статті до друку після рецензування: 03.02.2026
Дата публікації (оприлюднення) статті: 08.04.2026