УДК 004.05:004.8 **DOI:** 10.31673/2412-9070.2026.017403
А. Ю. БУРАЧИНСЬКИЙ, аспірант, архітектор програмних рішень,
ORCID: 0009-0003-7913-2152
Державний університет інформаційно-комунікаційних технологій, Київ

ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІДЕНТИФІКАЦІЇ ТА АВТОМАТИЗАЦІЇ ВИПРАВЛЕННЯ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

У зв'язку з тенденціями глобального технологічного зростання та збільшенням складності та обсягів програмного забезпечення, все більш гостро постають проблеми підтримки та оптимізації відповідних процесів уже існуючого програмного забезпечення. У статті розглянуто актуальні методи і підходи ефективного застосування штучного інтелекту (ШІ) для ідентифікації та автоматизації процесу аналізу і виправлення програмних дефектів та збоїв у програмному забезпеченні (ПЗ). Значна увага приділена аналізу сучасних моделей машинного навчання та генеративних алгоритмів, які дозволяють ефективно виявляти помилки у програмному коді, оцінювати їх критичність та виконувати автоматизовану корекцію виявлених помилок з мінімальним залученням розробників та інженерів. Обґрунтовано доцільність використання ШІ у реальних проєктах, особливо в умовах масштабної розробки та підтримки ПЗ, де враховуються фактори точності, швидкості роботи та скорочення затрат на тестування, та підтримку. Зокрема, проаналізовано можливість практичного застосування генеративних реалізацій великих мовних моделей, таких як, наприклад, ChatGPT для автоматизації аналізу та подальшого виправлення дефектів ПЗ. У результаті дослідження запропоновано практичний підхід ефективної інтеграції ШІ у робочі процеси розробки ПЗ та підтримки уже працюючих софтверних систем, визначено перспективи його впровадження в індустрії та ідентифіковано основні складнощі, пов'язані із використанням таких технологій. Практичні результати дослідження підтверджують ефективність застосування ШІ для покращення якості ПЗ, зниження рівня людського фактору та підвищення продуктивності команди розробників.

Ключові слова: штучний інтелект; оптимізація розробки програмного забезпечення; оптимізація підтримки ПЗ; виправлення дефектів; аналіз збоїв і помилок; машинне навчання; генеративні моделі ШІ; великі мовні моделі.

Постановка задачі

У зв'язку з тенденціями глобального технологічного зростання та збільшення обсягів даних, дедалі зростає складність та обсяги коду програмного забезпечення (ПЗ). З огляду на зростання складності та обсягів коду ПЗ, критично важливими аспектами розробки є оптимізація процесу підвищення якості ПЗ та мінімізація витрат шляхом ідентифікації та автоматизації виправлення дефектів і збоїв. Традиційні підходи до тестування та виправлення помилок вимагають значних затрат інженерних ресурсів і часу, що збільшує фінансові витрати та ризики для проєктів. Крім того, людський фактор доволі часто привносить помилки та додає затримки в часі у процесі аналізу дефектів та відповідного програмного коду. Враховучи всі ці аспекти, постає завдання розробки та впровадження ефективних методів автоматизованого виявлення і виправлення дефектів у ПЗ, які можуть значно оптимізувати роботу команди розробників та зменшити затрати часу інженерів на підтриму ПЗ. У даній роботі пропонується проаналізувати та дослідити можливості використання штучного інтелекту (ШІ) як основного інструменту для аналізу дефектів і збоїв, і відповідного програмного коду, виявлення критичних дефектів і можливостей автоматизації їх виправлення за допомогою засобів ШІ для підвищення ефективності та зниження вартості розробки та підтримки ПЗ.

Аналіз останніх досліджень і публікацій

Аналіз останніх досліджень і публікацій у сфері застосування штучного інтелекту, яка займається ідентифікацією та автоматизацією виправлення дефектів у програмному забезпеченні, дозволили спроєктувати систему на основі агента ШІ у поєднанні з генеративною реалізацію великої мовної моделі, що дозволяє автоматизувати та оптимізувати процеси підтримки ПЗ. В ході дослідження, проробленого під час написання статті, проаналізовано дослідницькі роботи Bin, Y., & Zhong, L. (2017), які у своїй статті "Deep learning-based root cause analysis for logs in large systems" досліджували можливості інтеграції методів глибокого навчання для аналізу журналів логів системи і допомоги в ідентифікації першопричин помилок; проаналізовано публікації Gupta et al. (2021), який у своїй роботі "Autonomous debugging agents using reinforcement learning" запропонував підхід із використанням reinforcement learning для створення автономних агентів, які не лише знаходять дефекти, але й розробляють пропозиції оптимальних змін у коді для їх виправлення; опрацьовано публікацію Wang, J., Zhang, Y., & Huang, G. (2020) під назвою "Predicting software defect patterns using deep neural networks" на предмет прогнозування закономірностей виникнення дефектів, що дозволяє виявляти їх на ранніх етапах інтеграції ПЗ, опрацьовано джерела та публікації стосовно алгоритмів виявлення аномалій у логах ПЗ та методів їх практичної реалізації.

У цій статті ми розглянемо теоретичні засади та практичні методи побудови системи для ідентифікації та автоматизованого виправлення програмних дефектів та збоїв ПЗ у поєднанні із генеративними моделями штучного інтелекту як ефективний спосіб автоматизації і здешевлення підтримки софтверних платформ за допомогою агента штучного інтелекту. Новизна запропонованого у цій статті підходу полягає у дослідженні та побудові ефективного алгоритму аналізу дефектів та автоматизації їх виправлення за допомогою самописного агента, інтегрованого з генеративною моделлю штучного інтелекту, що дозволяє автоматизувати і оптимізувати процес підтримки ПЗ за рахунок часткової або повної заміни інженерів агентом штучного інтелекту.

Результати дослідження

У сучасних системах програмного забезпечення логи є критичним джерелом інформації, що дозволяє відстежувати роботу систем, аналізувати їхній стан та виявляти потенційні проблеми. З огляду на зростання масштабів інформації, з якими працюють сучасні програми, та складність сучасних програмних систем, інструменти для автоматизації обробки і аналізу логів стали обов'язковим елементом підтримки ПЗ. У цій частині статті розглядаються наявні механізми, які пропонують агрегатори логів для автоматизованої ідентифікації помилок, дефектів та виявлення аномалій.

Оскільки в процесі виправленні програмних збоїв та дефектів першочерговим завданням є ідентифікація збоїв, помилок та дефектів у програмному забезпеченні, то у першій частині дослідження пропонується проаналізувати наявні механізми, які пропонують широко використовуємі агрегатори логів для автоматизованої ідентифікації помилок та дефектів у ПЗ.

У сучасних комплексах програмного забезпечення для ефективної агрегації, упорядкування, аналізу логів використовують так звані лог-агрегатори, наприклад, Splunk, ELK Stack (Elasticsearch, Logstash, Kibana), які дозволяють ефективно аналізувати великі обсяги даних логів. Адже в сучасних складних платформах ПЗ обсяги логів можуть сягати кількох терабайт за день. Лог-агрегатори інтегрують методи обробки великих даних, пропонують вбудовані засоби машинного навчання (ML) та аналітики для забезпечення просунутої функціональності, включаючи автоматичну ідентифікацію аномалій у логах та виявлення помилок. Під аномаліями логів у даному контексті розуміються нетипові відхилення від норми та загальної закономірності при кластеризації (або класифікації) логів, які найчастіше обумовлені збоями та дефектами. У даному дослідженні ми фокусуємося на виявленні та автоматизації виправлення саме програмних дефектів, ажде у сучасних софтверних платформах збої та відмови також можуть бути обумовлені інфраструктурними проблемами.

Агрегатори логів застосовуються не лише для аналізу інцидентів і виявлення помилок, але і для побудови довгострокової стратегії вдосконалення архітектури ПЗ через глибокий аналіз логів.

Розглянемо основні стратегії та підходи до аналізу логів:

1. Структуризація та синтаксичний аналіз логів. Сучасні лог агрегатори дозволяють структурувати логи для спрощення подальшої роботи з ними та їхнього аналізу.

2. Аналіз на основі сигнатур та шаблонів. Це метод, який базується на пошуку заздалегідь визначених фрагментів тексту або ключових слів, таких як "ERROR" (з англ. “помилка”), "WARN" (з англ. “попередження”) або "EXCEPTION" (з англ. “виняткова ситуація”).

3. Ідентифікація аномалій у логах. Інтеграція алгоритмів машинного навчання (від англ. ML) в агрегатори логів дала змогу започаткувати новітні підходи до аналізу логів, зокрема алгоритми ідентифікації аномалій при аналізі логів. Моделі машинного навчання зазвичай навчаються на терабайтах історичних даних і створюють профіль нормальної поведінки даних. Аномалії визначаються як відхилення від типової поведінки, які зазвичай трапляються при збоях та помилках.

4. Кластеризація та сегментація логів. Це дозволяє окреслити схожі типи повідомлень у групи, що значно спрощує пошук та подальший аналіз потенційних проблем. Автоматична сегментація логів дає змогу швидко зрозуміти типову поведінку системи і виявити незвичні послідовності подій.

5. Причинно-наслідковий аналіз. Більшість сучасних лог-агрегаторів пропонують вбудовані механізми виявлення ланцюгів подій, які призвели до виникнення збою або дефекту.

Найпоширенішими викликами, які виникають при аналізі інцидентів, пов’язаних із відповідними програмними збоями і помилками, є:

- висока складність аналізу і кореляції подій у розподілених платформах (з метою забезпечення високої доступності системи) на великих обсягах даних, які генерують сучасні софтверні системи у вигляді логів;
- проблема "шуму" в логах – це велика кількість несуттєвої (або вторинної) інформації, яка ускладнює автоматичну інтерпретацію логів;
- проблема вирізнення “суттєвих” від “несуттєвих” проблем. Це може бути особливо гострою проблемою у великих софтверних платформах, де щоденно генерується кілька тисяч повідомлень про помилки у журналах логів. У такому випадку справді складно проаналізувати всі помилки та вирізнити суттєві, тобто такі, що істотно впливають на працездатність системи та основний функціонал;
- затратність з точки зору інженерних ресурсів та часу на проведення аналізу і виявлення першопричин дефекту та потрібних правок.

У результаті проведеного дослідження було проаналізовано основні підходи та стратегії до аналізу збоїв та помилок у ПЗ і встановлено, що метод "виявлення аномалій" при аналізі логів допомагає істотно спростити та вирішити ряд проблем, які виникають під час моніторингу, аналізу та управління програмним забезпеченням, інфраструктурою та сервісами.

У цій частині дослідження розглянемо, як працює виявлення аномалій в OpenSearch. OpenSearch – це розробка на базі ElasticSearch, як його відкритий форк (відгалуження), і відповідно OpenSearch можна використовувати безкоштовно. По цій причині OpenSearch є одним із найбільш широко використовуємих агрегаторів логів. OpenSearch пропонує готові вбудовані функції для виявлення аномалій. Метод виявлення аномалій в OpenSearch базується на використанні машинного навчання та обробки потокових даних у реальному часі. Цей фунціонал доступний як компонент "Anomaly Detection Plugin" і реалізований через такі основні етапи:

1. Вибір цільових даних. На цьому етапі визначаються поля логів або індексів, які будуть моніторитись та аналізуватись на предмет наявнсоті аномалій.

2. Налаштування детектора аномалій. На цьому етапі адміністратор системи налаштовує детектор аномалій і поля для аналізу (наприклад, "error_rate", "latency" і т.д.).

3. Обробка потокових даних та ідентифікація аномалій у часовому ряду (від англ. time series).

Для виявлення аномалій OpenSearch використовує неконтрольований метод навчання, заснований на принципах кластеризації (або класифікації) повідомлень та використання статистичних моделей. Основний алгоритм, на якому базується виявлення аномалій в OpenSearch - це Random Cut Forest (RCF). Random Cut Forest (з англ. «випадково зрізаний ліс») – ансамблевий метод машинного навчання для класифікації, регресії та інших завдань, який працює за допомогою побудови численних дерев прийняття рішень під час тренування моделі й продукує моду для класів (класифікацій) або усереднений прогноз (регресія) побудованих дерев.

Розглянемо основні особливості RCF алгоритму:

1. Ідентифікація аномалій: алгоритм працює шляхом створення дерев випадкових поділів даних, щоб оцінити "відхилення" зразків у багатовимірному просторі.

2. Оцінка аномалій: обчислюється аномальний бал (anomaly score), який вказує на відхилення певної точки від типового патерну.

3. Робота з великими обсягами даних: RCF особливо ефективний для аналізу великих потоків даних у реальному часі завдяки його масштабованості.

Приклад реалізації агента для автоматизації виправлення дефектів із застосуванням генеративної моделі штучного інтелекту

У цій частині статті розглянемо теоретичні аспекти та практичний приклад побудови агента для ідентифікації та автоматизованого виправлення дефектів із застосуванням генеративної моделі штучного інтелекту.

В якості провайдера ШІ можна використовувати будь-яку із широко відомих генеративних моделей, зокрема, таких як ChatGPT. В якості базової моделі ШІ (з англ. Foundation Model) пропонується використовувати модель, натреновану для задач генерації коду, наприклад:

- GPT-4 від OpenAI;
- Claude від Anthropic;
- CodeWhisperer від Amazon;
- AlphaCode від Deepmind.

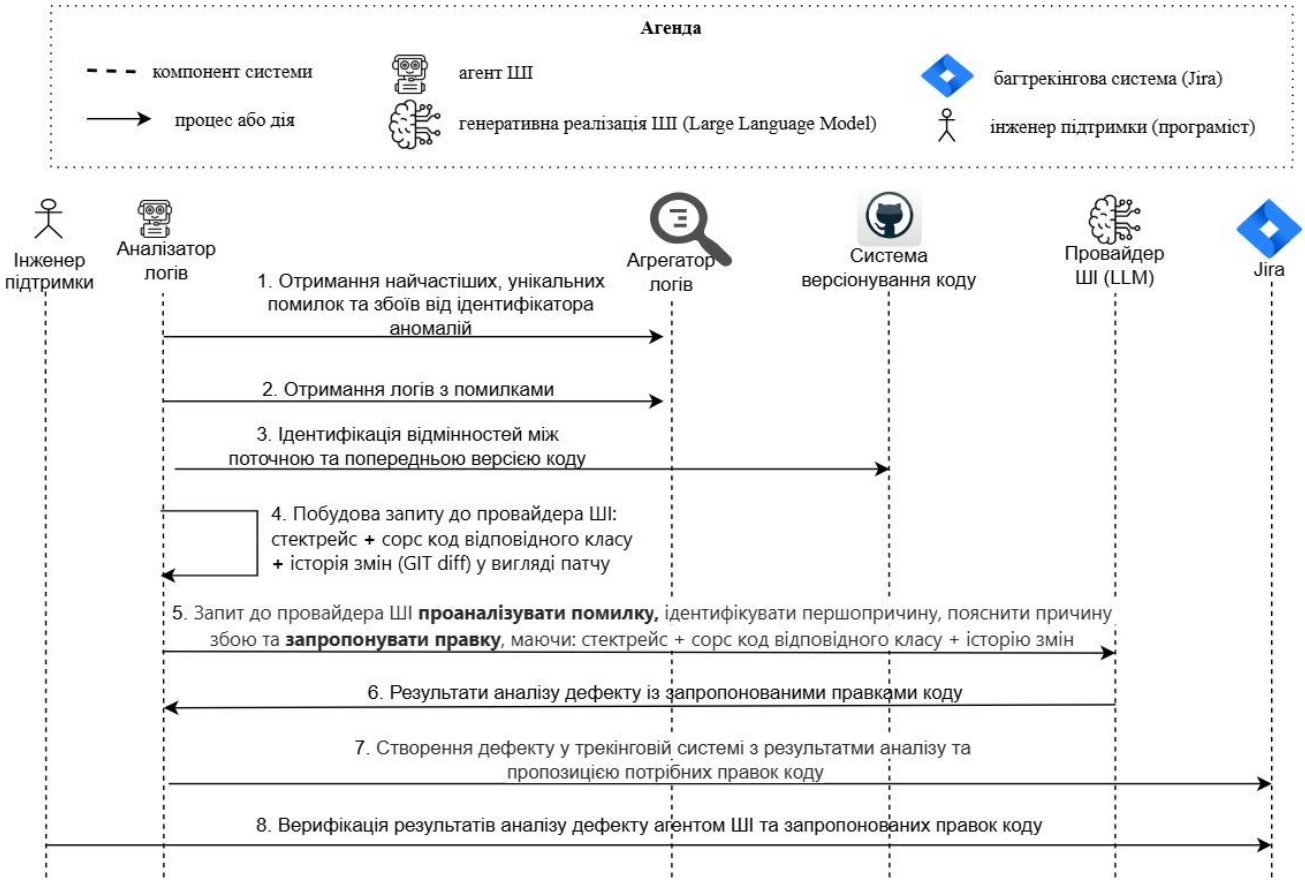

Рис. 1. Варіант побудови агента для аналізу логів у поєднанні з генеративною реалізацією ШІ

У запропонованому варіанті реалізації системи з використанням агента для автоматизованого аналізу і виправлення дефектів алгоритм роботи системи наступний:

1. Отримання найчастіших, унікальних помилок ідентифікатора аномалій. Для здешевлення вартості аналізатора логів варто вибирати лише унікальні помилки із агрегатора логів, що суттєво зменшує кількість запитів до провайдера ШІ та операційні витрати для функціонування системи.

2. Отримання логів з помилками.

3. Ідентифікація відмінностей між поточною та попередньою версією коду за допомогою запиту до системи версіонування коду (GitHub або Bitbucket).

4. Побудова запиту до провайдера ШІ:
стектрейс + сорс код відповідного класу + історія змін (GIT diff) у вигляді патчу

5. Запит до провайдера ШІ проаналізувати помилку**,** ідентифікувати першопричину, пояснити причину збою та запропонувати правку, маючи: стектрейс + сорс код відповідного класу + історію змін.

6. Провайдер ШІ генерує результати аналізу дефекту із запропонованими правками коду.

7. Агент – аналізатор логів створює дефект у трекінговій системі з результатми аналізу та пропозицією потрібних правок коду.

8. Інженер підтримки верифікує (перевіряє) результати аналізу дефекту агентом ШІ та запропоновані правки коду.

Практичні результати використання розробленого алгоритму для аналізу простих та нетривіальних дефетів і збоїв

У результаті практичних експериментів було встановлено, що запропонований алгоритм побудови системи для автоматизованого аналізу та виправлення дефектів добре працює для виправлення простих дефектів, таких як, наприклад NullPointerException, ClassCastException і т.д. Запропонована реалізація алгоритму вірно ідентифікує першопричину проблеми, в тому числі вірно ідентифікує зміни в коді (на основі гіт логу), які стали причиною дефекту, і пропонує прийнятні варіанти виправленння дефектів.

Щодо нетривіальних дефектів (спричинених некоректними станами даних), таких як наприклад IllegalStateException, генеративна модель ШІ (LLM) надає логічно обґрунтовані рекомендації, проте в більшості випадків не пропонує правильних виправлень, оскільки визначення справжньої першопричини й відповідного рішення або необхідних правок коду вимагає розуміння відповідних процесів і змін у станах даних в межах застосунку.

Виклики та обмеження запропонованого алгоритму автоматизованого виправлення програмних дефектів

Попри очевидні переваги ШІ для ідентифікації та виправлення дефектів існує низка викликів:

- Складність інтелектуального виправлення: деякі дефекти потребують глибокого розуміння доменної області або специфіки бізнес-логіки, що складно реалізувати навіть за допомогою найбільш досконалих реалізацій ШІ.
- Обмежені дані для навчання: для правильного функціонування ШІ-алгоритмів потрібна велика кількість структурованих даних для навчання, які можуть бути недоступними.
- Можливість генерації нових дефектів: автоматизоване виправлення може вносити зміни, які викликають нові несподівані помилки.
- Безпека: інтеграція ШІ у процес виправлення коду може створити потенційні ризики для безпеки систем, особливо якщо ШІ некоректно налаштований.

Отже, незважаючи на очевидні переваги використання ШІ для автоматизованого виправлення програмних дефектів, необхідно верифікувати запропоновані штуним інтелектом правки коду для зниження бекзпекових ризиків та ризиків привнесення нових дефектів.

Для вдосконалення запропонованого алгоритму з метою аналізу нетривіальних дефектів логічною оптимізацією була б інтеграція запропонованого агента для аналізу дефектів із доменною базою знань, що дало б агенту ШІ контекст щодо зміни стану даних в межах відповідних сценаріїв функціоналу у застосунку.

Висновки

Застосування штучного інтелекту для ідентифікації та автоматизації виправлення дефектів у програмному забезпеченні вже змінює підходи до розробки, тестування та супроводу ПЗ шляхом оптимізації та повної або часткової автоматизації. Технології ШІ допомагають знизити кількість помилок, підвищити ефективність роботи розробників і зменшити витрати, роблячи програмне забезпечення більш надійним і безпечним. Однак, необхідно пам'ятати, що ШІ – це не заміна людської креативності, а інструмент для її посилення. Успішне використання ШІ у цій сфері вимагає грамотної інтеграції, інтелектуальної стратегії контролю та зниження відповідних ризиків. З часом ці технології отримають більшу популярність, а їх вплив почне охоплювати ще ширші аспекти цифрового світу.

Список літератури

1. Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, Michael R. Lyu. Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection. – 2021. –Режим доступу: https://doi.org/10.48550/arXiv.2107.05908

2. Northrop, Jett Reinforcement Learning-Based Debugging Agents for Real-Time Bug Identification and Self-Healing Software Systems. – 2023. – Режим доступу: https://www.researchgate.net/publication/390887348_Reinforcement_Learning - Based_Debugging_Agents_for_Real-Time_Bug_Identification_and_Self-Healing_Software_Systems

3. Wang, J., Zhang, Y., & Huang, G. Predicting software defect patterns using deep neural networks. – 2020. – Режим доступу: https://doi.org/10.35940/ijitee.D1858.039520

4. Гасан, І. М., Кузьменко, П. М. Використання штучного інтелекту для автоматизації програмного забезпечення // Технології програмування. – 2022. – № 4. – С. 56–63.

5. Anthropic. Claude AI: Advancing Safe and Reliable AI Systems. – 2024. – Режим доступу: https://www.anthropic.com/claude

6. OpenAI. GPT-4 Technical Report // OpenAI Documentation. – 2023. – Режим доступу: https://openai.com/research/gpt-4

7. DeepMind. AlphaCode: AI for Competitive Coding // DeepMind Research Papers. – 2021. – Режим доступу: https://www.deepmind.com

8. Amazon. CodeWhisperer: AI-Powered Code Suggestions // Amazon AWS AI. – 2022. – Режим доступу: https://aws.amazon.com/en/q/developer

9. Mingjie Liu, Nathaniel Pinckney, Brucek Khailany. VerilogEval: Evaluating Large Language Models for Verilog Code Generation // NVIDIA Research. – 2023. – Режим доступу: https://research.nvidia.com/publication/2023-09_verilogeval-evaluating-large-language-models-verilog-code-generation

10. Hugging Face. CodexGLUE: Machine Learning Models for Code Generation and Completion // Hugging Face Blog. 2023. – Режим доступу: https://huggingface.co/docs

11. Хромов, А. В. Автоматизація тестування програмного забезпечення за допомогою штучного інтелекту // Системи управління і захист інформації. – 2020. – № 3. – С. 17–24.

12. Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In Proc. of ACL. – 2010. – Режим доступу: https://aclanthology.org/P10-1040.pdf

A. Burachynskyi

APPLICATION OF ARTIFICIAL INTELLIGENCE FOR IDENTIFICATION AND AUTOMATION OF SOFTWARE DEFECT CORRECTION

In the light of global technological growth trends and increasing complexity and scale of software, the challenges of maintaining and optimizing processes for existing software are becoming increasingly pressing. This article examines current methods and approaches of effective applying of artificial intelligence (AI) for the identification and automation of the processes of analyzing and cor-

recting software defects and failures. Significant attention is given to the analysis of modern machine learning models and generative algorithms, which enable effective error detection in program code, assessment of their severity, and automated correction of identified issues with minimal involvement of developers and engineers. The appropriateness of using AI in real-world projects is substantiated, especially in the context of large-scale software development and maintenance projects, where factors such as accuracy, operational speed, and cost reduction for testing and support are taken into account. In particular, the potential practical application of generative realizations of large language models (LLM), such as ChatGPT, is analyzed for automating the analysis and subsequent correction of software defects. As a result of the research, practical approach to effective integration of AI into the workflows of software development is proposed, the prospects for its implementation in the industry are identified, and the main challenges associated with the use of such technologies are highlyghted. The practical results of the accomplished research confirm the effectiveness of using AI to improve software quality, reduce the impact of the human factor, and enhance the productivity of development teams.

Keywords: artificial intelligence; software development optimization; software support optimization; defect correction; crash and error analysis; machine learning; generative AI models; large language models.

Надійшла до редакції: 05.01.2026
Прийнята до друку: 13.02.2026
Опубліковано: 27.02.2026