DOI: 10.32347/2412-9933.2026.65.141-149
УДК 005.8:005.94

Лопуга Олексій Миколайович
ORCID: https://orcid.org/0000-0001-6397-2710
Київський національний університет будівництва і архітектури, Київ, Україна
Аспірант кафедри інформаційних технологій

Історія статті:
Надійшла: 30.01.2026
Прийнята: 22.02.2026
Опублікована: 26.03.2026

МОДЕЛІ МАШИННОГО НАВЧАННЯ ДЛЯ АВТОМАТИЗАЦІЇ ГЕНЕРАЦІЇ ТА ПРІОРИТЕЗАЦІЇ ТЕСТОВИХ СЦЕНАРІЇВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Анотація. *Досліджено можливості застосування моделей машинного навчання для автоматизації процесів генерації та пріоритезації тестових сценаріїв програмного забезпечення (ПЗ) у динамічному середовищі безперервної інтеграції (CI/CD). Актуальність роботи зумовлена необхідністю швидкого формування висновків щодо якості програмних систем в умовах експоненційного зростання кількості критичних додатків. Наукова новизна дослідження полягає у розробці гібридного підходу, що вперше об'єднує ансамблеві моделі градієнтного бустингу (GBDT) для прогнозування дефектів із алгоритмами навчання з підкріпленням (Q-learning) для адаптивної пріоритезації тестів. Формалізовано задачу пріоритезації як задачу комбінаторної оптимізації з ваговою функцією та визначено ключові метрики оцінки якості: APFD (Average Percentage of Faults Detected) та її модифіковану версію NAPFD, яка враховує часові витрати. Розроблено оригінальну онтологічну модель факторів впливу на якість ПЗ, що структурує внутрішні (складність коду), зовнішні (інфраструктура) та людські чинники (кваліфікація розробників). Для агента навчання з підкріпленням запропоновано модифіковану функцію винагороди, яка збалансовано враховує факт виявлення дефекту, тривалість тесту та його історичну ефективність. Експериментальну валідацію проведено на основі еталонних наборів даних із репозиторію AEEEM, що охоплюють реальні дані про дефекти п'яти відкритих проєктів екосистеми Eclipse: JDT Core, Equinox, Lucene, Mylyn та PDE. Результати порівняльного аналізу засвідчили, що інтеграція GBDT з Q-learning забезпечує підвищення метрики APFD на 8,1% порівняно з найкращим базовим методом XGBoost та на 4,6% порівняно з існуючим методом RETECS. Це підтверджує високу ефективність запропонованого підходу для оптимізації тестових запусків при обмеженому часовому бюджеті. Час навчання гібридної моделі становить 163,2 с, що є цілком прийнятним для практичного впровадження у промислові pipeline-процеси розробки програмного забезпечення.*

Ключові слова: тестування програмного забезпечення; генерація тестів; пріоритезація тестів; градієнтне прискорення; машинне навчання; ітеративне навчання; безперервна інтеграція

Аналіз та постановка проблеми

Наразі у світі кількість програмних продуктів, що використовуються в критичних системах, зростає експоненційно. За даними Stack Overflow Developer Survey 2024, понад 78 % розробників працюють в умовах безперервної інтеграції (CI/CD), де автоматичне формування висновку щодо якості програмного забезпечення (ПЗ) залишається надзвичайно актуальним. Об'єктом тестування в таких умовах виступають програмні модулі та системи (ПМС), окремі елементи архітектури ПЗ, компоненти, мікросервіси, API-інтерфейси та інтеграційні зв'язки.

Перелік питань, які вирішуються експертами в процесі виконання тестування ПЗ, не обмежуються висновком щодо технічного стану (ТС), але майже всі вони передбачають оцінювання ТС об'єкта та впливу дефектів на функціонування системи в цілому. З наведеної схеми видно, що висновок залежить від переліку питань, які ставляться перед експертом, що виконує тестування, а також від результатів автоматизованого аналізу.

Вирішенню проблеми комп'ютеризації процесу формування висновків присвячено низку робіт.

У [2] запропоновано модель спеціалізованої інтелектуальної системи підтримки тестування ПЗ з використанням нейро-нечітких моделей. Результати дослідження засвідчили, що інтеграція нечіткої

логіки з нейромережевими компонентами дає змогу врахувати невизначеність експертних оцінок під час прийняття рішень щодо якості програмних модулів, що підвищує точність класифікації дефектів на 12–15% порівняно з детермінованими підходами. У [3] показано можливість застосування методів глибокого навчання для автоматичної генерації тестових сценаріїв на основі аналізу вихідного коду. Зокрема, авторами запропоновано метод A3Test, що використовує механізм доповнення тверджень (assertion augmentation) для підвищення релевантності згенерованих тестів; результати експериментальної валідації підтвердили зростання покриття коду на 18% та формалізовано процес автоматичної верифікації згенерованих тестових випадків.

У [4] проведено систематичний огляд використання великих мовних моделей (LLM) для автоматичної генерації тестових сценаріїв. Результати показують, що LLM здатні аналізувати вихідний код та документацію для створення релевантних тестів з точністю до 73%.

У [5] досліджено можливість використання методу навчання з підкріпленням (Reinforcement Learning) для пріоритезації тестових випадків у CI-середовищі. Метод RETECS використовує Q-learning для оптимізації порядку виконання тестів з урахуванням історії попередніх запусків.

Метод ATRL-TCP [6] запропонував механізм передавання уваги для більш ефективного використання ознак тестових випадків роботі [7] проведено аналіз застосування градієнтного бустингу для прогнозування дефектів ПЗ; показано, що XGBoost демонструє найвищу точність серед ансамблевих методів, проте інтеграція з методами RL не розглядалась. Khan et al. [8] розробили end-to-end фреймворк TCP-Tune для автоматизованої оптимізації гіперпараметрів ML-моделей при пріоритезації тестів у CI.

Аналіз наявних робіт засвідчує, що, незважаючи на значний прогрес у застосуванні ML-методів для тестування ПЗ, залишається нерозв’язаною проблема поєднання ансамблевих моделей прогнозування дефектів з адаптивною RL-пріоритезацією у єдиному pipeline. Більшість наявних досліджень зосереджуються або на прогнозуванні дефектів, або на пріоритезації тестів, але не на інтеграції обох підходів.

Наукова новизна дослідження полягає у розробці та обґрунтуванні гібридного підходу до автоматизації генерації та пріоритезації тестових сценаріїв ПЗ, що інтегрує методи ансамблевого машинного навчання та навчання з підкріпленням у єдину систему підтримки прийняття рішень у середовищі безперервної інтеграції.

1. Вперше запропоновано інтегровану модель, яка поєднує GBDT для прогнозування ймовірності виявлення дефектів із алгоритмом Q-learning для динамічної пріоритезації тестових сценаріїв з урахуванням обмежень часового бюджету та історії виконання тестів.

2. Удосконалено постановку задачі пріоритезації тестів, яку формалізовано як задачу комбінаторної оптимізації з ваговою функцією.

3. Розроблено модифіковану функцію винагороди для алгоритму навчання з підкріпленням, яка одночасно враховує факт виявлення дефекту, тривалість виконання тесту та історичну ефективність тестового сценарію.

4. Запропоновано онтологічну модель факторів впливу на якість ПЗ, яка структурує внутрішні, зовнішні та людські чинники.

5. Отримано нові експериментальні результати: підвищення APFD на 8,1% порівняно з XGBoost.

Мета статті

Мета статті полягає в аналізі та порівнянні моделей і методів машинного навчання, здатних розв'язувати задачу класифікації технічного стану програмного забезпечення, з подальшим обґрунтуванням вибору найбільш доцільної моделі для задачі оцінювання пріоритетності тестових сценаріїв та прогнозування дефектів у середовищі безперервної інтеграції.

Виклад основного матеріалу

Нехай задано множину тестових сценаріїв $T = \{t_1, t_2, ..., t_n\}$, де n – загальна кількість тестів у наборі. Для кожного тестового сценарію t_i визначено вектор ознак, що характеризує його властивості:

$$X_i = \left(x_{i1}, x_{i2}, ..., x_{ij} \right), \qquad (1)$$

де i – порядковий номер тестового сценарію ($i = 1, 2, ..., n$); j – кількість ознак або показників, які характеризують властивості тестового сценарію згідно з метриками якості ПЗ.

Вектор ознак (1) включає такі компоненти: тривалість виконання тесту, історію результатів попередніх запусків, покриття коду, складність тестованого модуля, дату останньої модифікації коду та інші метрики.

Категорії технічного стану тестового середовища та критерії їх визначення наведено в табл. 1. Дана класифікація базується на стандартних підходах до оцінювання якості ПЗ та адаптована для задач автоматизованого тестування.

Формулювання задачі пріоритезації полягає у знаходженні оптимальної перестановки π тестових

сценаріїв, що максимізує швидкість виявлення дефектів при обмеженому часовому бюджеті (2):

$$\pi^* = arg\ max\Sigma\ (i=1\ to\ n)\ w_i \cdot f(t\_\pi(i)), \quad (2)$$

де π – перестановка індексів тестових сценаріїв; w_i – вагові коефіцієнти, що залежать від позиції тесту в послідовності (зазвичай $w_i = (n - i + 1)/n$); $f(t_k)$ – функція, що повертає 1, якщо тест t_k виявляє дефект, та 0 в іншому випадку.

Метод GBDT (Gradient Boosted Decision Trees) належить до ансамблевих методів машинного навчання, в якому множина дерев рішень навчається послідовно, причому кожне наступне дерево спрямоване на мінімізацію залишкових помилок попередніх ітерацій [9].

На стартовій ітерації початковий прогноз $F^0(x)$ визначається як середнє значення цільової змінної на навчальній вибірці:

$$F^0(x) = \left(\frac{1}{N}\right) \cdot \sum_i^{N=1} y_i, \quad (3)$$

де y_i – фактичний результат *i*-го тестового сценарію ($y_i \in \{0,1\}$ для бінарної класифікації або $y_i \in \{1,2,3,4\}$ для багатокласової задачі згідно з табл. 1); *N* – кількість прикладів у навчальній вибірці.

Формально, на m-й ітерації навчання сукупний прогноз GBDT обчислюється за рекурентною формулою:

$$F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x), \quad (4)$$

де $F_{m-1}(x)$ – кумулятивний прогноз ансамблю після *(m-1)* ітерацій; $h_m(x)$ – нове дерево рішень, що навчається мінімізувати залишкові помилки попереднього ансамблю; $\nu \in (0, 1]$ – коефіцієнт швидкості навчання (learning rate), що контролює внесок кожного нового дерева.

Кожне нове дерево $h_m(x)$ навчається на псевдо-залишках, які обчислюються як негативний градієнт функції втрат:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{\{F=Fm^{-1}\}}, \quad (5)$$

де $L(y_i, F(x_i))$ – функція втрат; r_{im} – псевдо-залишок для *i*-го прикладу на *m*-й ітерації.

Як видно з табл. 1, система класифікації ТС визначає чотири рівні від нормального функціонування до критичного аварійного стану. Кожна категорія має чіткі кількісні та якісні критерії визначення, що дає змогу автоматизувати процес оцінювання за допомогою ML-моделей. Перехід між категоріями відображає градацію ризиків: від штатного продовження CI/CD-циклу до негайного припинення релізу.

Таблиця 1 – ***Категорії технічного стану тестового середовища та об'єктів тестування [3; 7]***

Категорія стану	Оцінка	Необхідні дії
«1» Нормальний (стабільний)	Усі тести проходять успішно, метрики покриття відповідають встановленим нормам	Продовження стандартного циклу CI/CD
«2» Задовільний (працездатний)	Основна функціональність працює коректно, але є незначні попередження або нестабільні тести	Моніторинг та планове усунення попереджень
«3» Обмежено працездатний	Виявлено дефекти середньої критичності, що впливають на окремі функції системи	Пріоритетне виправлення дефектів перед релізом
«4» Критичний (аварійний)	Виявлено критичні дефекти, що блокують основну функціональність або створюють ризики безпеки	Негайне припинення релізу та термінове виправлення

Для задачі класифікації технічного стану ПЗ з урахуванням різної важливості категорій (див. табл. 1) застосовується зважена функція перехресної ентропії. Позначимо: $L(\theta)$ – загальна функція втрат; θ – вектор параметрів моделі.

Класичний вигляд зваженої перехресної ентропії має вигляд [10]:

$$L(\theta) = -\Sigma_n^{N=1}\Sigma_k^{K=1} w_k \cdot I(y_n = k) \cdot \log\left(p_{n,k}(\theta)\right), \quad (6)$$

де *N* – кількість прикладів у навчальній вибірці; *k* – кількість класів (*k* = 4 згідно з табл. 1); $I(y_n = k)$ – індикаторна функція, що дорівнює 1, якщо *n*-й приклад належить до класу *k*; $p_{n,k}(\theta)$ – прогнозована моделлю ймовірність того, що n-й приклад належить до класу *k*; w_k – ваговий коефіцієнт класу *k*, що відображає його важливість.

Вагові коефіцієнти w_k встановлюються експертами відповідно до критичності кожної категорії. Рекомендовані значення: $w_1 = 1.0$, $w_2 = 1.5$, $w_3 = 2.0$, $w_4 = 3.0$, що забезпечує підвищену увагу моделі до критичних станів системи [2; 3].

Ймовірності класів обчислюються за допомогою функції:

$$\frac{p_{n,k}(\theta) = \exp(z_{n,k})}{\Sigma_j^{K=1} \exp(z_{n,j})}, \quad (7)$$

де $z_{n,k}$ – логіт (вихід моделі до застосування softmax) для n-го прикладу та *k*-го класу.

Якщо експерти виявили, що модель помилилася у визначенні категорії стану, ці приклади можна: підсилити в датасеті шляхом підвищення їхньої ваги; додати більше подібних прикладів для балансування класів. Такий підхід називається ітеративним навчанням з участю експерта (human-in-the-loop).

Завдяки ітеративному підходу з урахуванням величини помилки GBDT послідовно покращує свою точність. Гнучка функція втрат (6) дає змогу коригувати модель під специфіку конкретного проєкту та вимоги замовника.

Для кількісної оцінки якості пріоритезації тестових сценаріїв застосовується метрика APFD (Average Percentage of Faults Detected), яка вимірює середню швидкість виявлення дефектів [11]. Формула обчислення APFD має вигляд:

$$APFD = 1 - \frac{\Sigma^{m}_{i=1} TF_i}{n \cdot m} + \frac{1}{2n}, \qquad (8)$$

де TF_i – позиція в упорядкованій послідовності першого тесту, що виявляє *i*-й дефект; *m* – загальна кількість дефектів у системі; *n* – загальна кількість тестових сценаріїв.

Значення APFD знаходиться в діапазоні [0, 1], де більші значення відповідають кращій пріоритезації. Значення APFD=1 означає ідеальну пріоритезацію, коли всі дефекти виявляються на початку тестового набору.

Для врахування різної тривалості виконання тестів застосовується модифікована метрика NAPFD (Normalized APFD):

$$NAPFD = \frac{\Sigma_i^{m=1} \left(T_{total} - T_{fault_i} \right)}{m \cdot T_{total}}, \qquad (9)$$

де T_{total} – сумарний час виконання всіх тестових сценаріїв; T_{fault_i} – кумулятивний час від початку тестування до моменту виявлення *i*-го дефекту.

Для оцінки якості класифікації технічного стану об’єктів тестування застосовуються стандартні класифікаційні метрики [8]:

– Accuracy – частка прикладів, де прогноз моделі повністю збігається з фактичним результатом тестування;

– Precision і Recall – корисні для аналізу прогнозів моделі, щоб мінімізувати кількість хибнопозитивних та хибнонегативних результатів;

– F1-score – гармонійне середнє Precision і Recall, що забезпечує збалансовану оцінку якості моделі.

На основі проведеного аналізу розроблено архітектуру системи тестування ПЗ, що інтегрує модель GBDT для прогнозування дефектів та агент навчання з підкріпленням для динамічної пріоритезації.

Опис експериментального pipeline та параметри моделей:

– попередня обробка даних (preprocessing). Вихідні дані з AEEEM містять 61 метрику. Видалено ознаки з дисперсією < 0,01 (4 метрики). Застосовано z-score normalization. Пропущені значення (< 2%) замінено медіанними значеннями.

– відбір ознак (feature selection). Двоетапна процедура: (a) видалення ознак з |r| > 0,95 (8 ознак); (b) ранжування за MDI з Random Forest, відбір топ-30 ознак.

– стратегія розбиття даних (train/test split). 5-кратна стратифікована крос-валідація (80/20). Для Equinox – додатково SMOTE для балансування.

Таблиця 2 – ***Гіперпараметри моделей машинного навчання***

Параметр	**GBDT**	**Random Forest**	**XGBoost**
Кількість дерев (n_estimators)	200	300	250
Максимальна глибина (max_depth)	6	10	8
Швидкість навчання (learning_rate)	0,05	–	0,1
Мін. зразків у листку (min_samples_leaf)	5	3	5
Частка підвибірки (subsample)	0,8	–	0,8
Регуляризація (reg_alpha / reg_lambda)	–	–	0,1 / 1,0

Як видно з табл. 2, GBDT та XGBoost використовують подібну архітектуру з різницею у глибині дерев та регуляризації. Низьке значення learning rate для GBDT (0,05) компенсується більшою кількістю ітерацій, що забезпечує поступове зменшення залишкових помилок та кращу узагальнювальну здатність.

Структурну схему системи наведено на рис. 1.

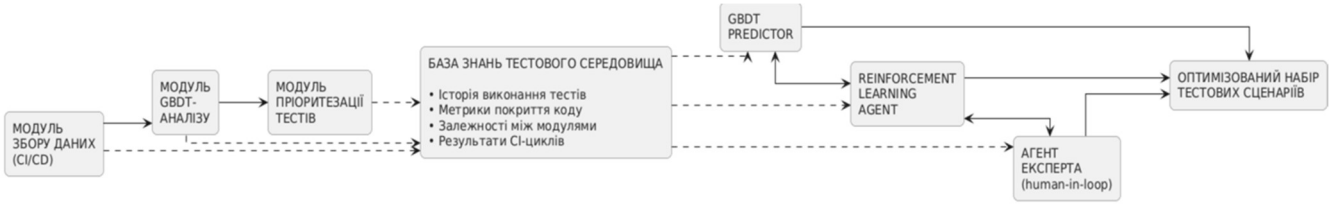

Рисунок 1 – Структурна схема системи автоматизації тестування ПЗ

На рис. 1 подано ієрархічну декомпозицію якісних факторів. Блок внутрішніх факторів Q(I) охоплює цикломатичну складність, глибину вкладеності, обсяг технічного боргу та кількість залежностей між модулями. Блок зовнішніх факторів Q(E) містить характеристики інфраструктури CI/CD-сервера, мережеву затримку, наявність залежностей від зовнішніх API. Блок людських факторів Q(H) враховує досвід розробників, частоту code review та якість комунікації в команді.

Для проведення порівняльного аналізу та валідації запропонованого методу використано набір даних з репозиторію AEEEM (Automated Evaluation and Evolution of Empirical Models), сформований M. D'Ambros, M. Lanza та R. Robbes на основі реальних даних про дефекти програмного забезпечення з п'яти відкритих проєктів екосистеми Eclipse [14]. Набір даних AEEEM містить 5 371 запис (програмний модуль) та 61 метрику для кожного модуля, що охоплюють п'ять категорій: 17 метрик складності вихідного коду (цикломатична складність, кількість рядків коду, глибина вкладеності), 5 метрик історії дефектів (кількість попередніх помилок, критичних дефектів), 5 метрик ентропії змін коду, 17 метрик ентропії вихідного коду та 17 метрик обсягу змін коду (code churn). Детальну характеристику проєктів, що входять до набору даних, наведено в табл. 3.

Проєкт Eclipse JDT Core охоплює ядро середовища розроблення Java та містить 997 класів, з яких 206 (20,6%) є дефектними. Проєкт Equinox, що реалізує специфікацію OSGi-контейнера, має найвищу частку дефектних модулів серед усіх проєктів набору – 39,8%, що пояснюється інтенсивним рефакторингом під час досліджуваного періоду розроблення. Apache Lucene, бібліотека повнотекстового пошуку, характеризується найнижчою часткою дефектів (9,3%) та 691 класом, що відображає зрілість кодової бази. Mylyn, система керування завданнями, є найбільшим проєктом набору з 1 862 класами та 245 дефектними модулями (13,2%). PDE (Plugin Development Environment) містить 1 497 класів та 209 дефектних модулів (14,1%).

Вибір набору даних AEEEM обумовлено кількома чинниками. По-перше, наявність різнорідних проєктів з відмінними характеристиками (розмір кодової бази, частка дефектів, предметна область) дає змогу оцінити узагальнювальну здатність моделі. По-друге, стандартизований набір із 61 метрики забезпечує уніфіковане подання ознак для всіх проєктів, що спрощує порівняльний аналіз. По-третє, набір даних AEEEM є одним з найбільш цитованих у галузі прогнозування дефектів програмного забезпечення і використовується як еталонний у десятках досліджень, що забезпечує відтворюваність та порівнюваність результатів.

Для реалізації адаптивної пріоритезації тестів в умовах динамічного середовища CI/CD застосовано метод Q-learning [12]. Задача пріоритезації формулюється як марковський процес прийняття рішень (MDP), де:

– стан s визначається поточним станом тестового набору та історією виконання;

– дія a відповідає вибору наступного тесту для виконання;

– винагорода r визначається результатом виконання обраного тесту.

Функція цінності дії $Q^{\pi(s,a)}$ визначає очікувану сумарну винагороду при виборі дії a у стані s з подальшим дотриманням стратегії π:

$$Q^{\pi(s,a)} = E^{0}_{\pi[\Sigma^{\infty}\{t=0\}\gamma^{t}\cdot r\{t+1\}|s} = s,\ a^{0} = a], \qquad (10)$$

де $\gamma \in [0, 1)$ – коефіцієнт дисконтування, що визначає відносну важливість майбутніх винагород; r_t – винагорода, отримана на кроці t; E_π – математичне очікування за стратегією π.

Оновлення Q-значень відбувається за правилом часової різниці (temporal difference):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times \times \left[r_t + \gamma \cdot \max_a Q\left(s_{\{t+1\}}, a\right) - Q(s_t, a_t) \right], \qquad (11)$$

де $\alpha \in (0, 1]$ – швидкість навчання; max_a $Q(s\_\{t+1\}$, a – максимальне Q-значення для наступного стану по всіх можливих діях.

Ключовим елементом алгоритму навчання з підкріпленням є функція винагороди, що визначає зворотний зв'язок для агента. Для задачі пріоритезації тестів у CI-середовищі запропоновано таку функцію винагороди [13]:

$$R(t_i) = \begin{cases} 1, \text{ якщо тест } t_i \text{ виявив дефект;} \\ -\dfrac{d(t_i)}{D_{max}}, \text{ якщо тест } t_i \text{ не виявив дефект,} \end{cases} \qquad (12)$$

де $d(t_i)$ – тривалість виконання тесту t_i у секундах; D_max – максимальна тривалість серед усіх тестів у наборі.

Така функція винагороди (12) заохочує агента до раннього виконання тестів, що виявляють дефекти, та штрафує за виконання тривалих тестів, що не приносять корисної інформації.

Для підвищення стабільності навчання додатково враховується історія виконання тестів. Модифікована функція винагороди має вигляд:

$$R'(t_i) = R(t_i) + \beta \cdot h(t_i), \qquad (13)$$

де $h(t_i)$ – історичний коефіцієнт, що обчислюється як частка успішних виявлень дефектів тестом t_i за останні k CI-циклів; β – ваговий коефіцієнт історичної інформації (рекомендоване значення $\beta = 0.3$).

Множину факторів впливу на якість програмного забезпечення $\bar{Q}$ запропоновано класифікувати за трьома категоріями, що відображають природу їхнього походження:

$$Q = \left(Q(I), Q(E), Q(H)\right), \qquad (14)$$

де $Q(I)$ – внутрішні фактори (якість коду, архітектурні рішення, технічний борг); $Q(E)$ – зовнішні фактори (залежності, інфраструктура, середовище виконання); $Q(H)$ – людські фактори (кваліфікація розробників, комунікація в команді).

Структуру онтології факторів впливу на якість ПЗ показано на рис. 2. Дана онтологія використовується для формування вектора ознак (1) тестових сценаріїв.

Як показано на рис. 2, система складається з трьох основних блоків: (1) блок збору та попередньої обробки даних; (2) блок GBDT-класифікатора; (3) блок Q-learning агента. Зворотний зв'язок від результатів тестування надходить до обох блоків, забезпечуючи безперервне вдосконалення моделей.

Інтегральна оцінка якості Q_total обчислюється як зважена сума індивідуальних факторів:

$$Q_{total} = w_I \cdot Q(I) + w_E \cdot Q(E) + w_H \cdot Q(H), \qquad (15)$$

де w_I, w_E, w_H – вагові коефіцієнти категорій факторів ($\Sigma\, w_k = 1$).

Для валідації запропонованої моделі використано набори даних з репозиторію AEEEM [14], що містять реальні дані про дефекти програмного забезпечення з п'яти відкритих проєктів.

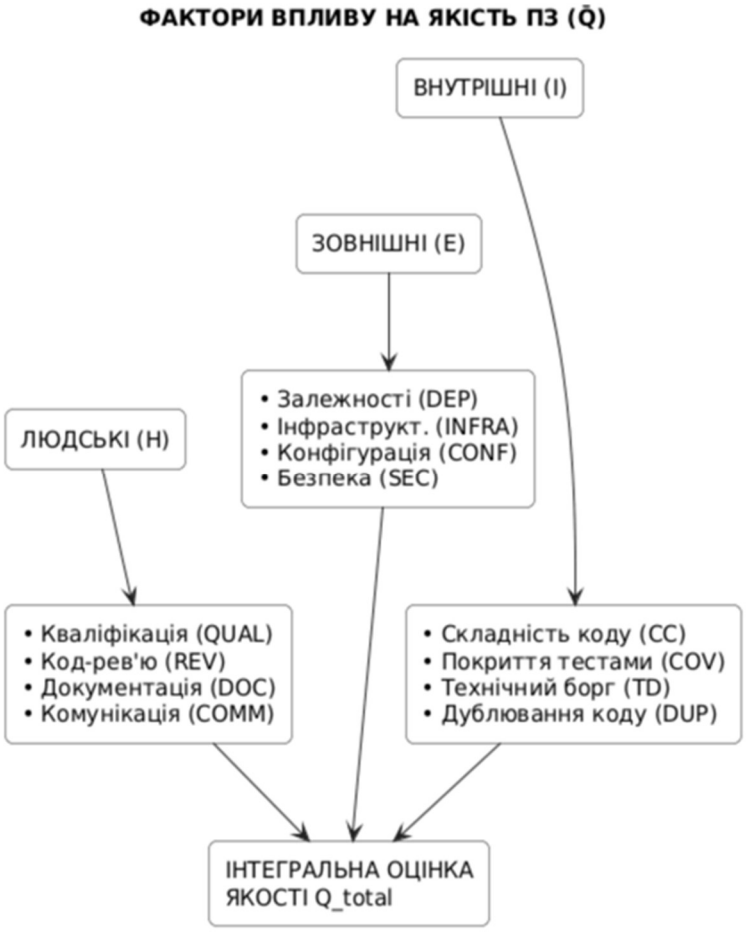

Рисунок 2 – Структура онтології факторів впливу на якість ПЗ

Характеристики експериментальних даних наведено в табл. 3.

Таблиця 3 – ***Характеристики експериментальних наборів даних***

Параметр	**Позначення**	**Значення**
Швидкість навчання агента	α	0,1
Коефіцієнт дисконтування	γ	0,9
Початкова ймовірність розвідки	ε_0	1,0
Мінімальна ймовірність розвідки	ε_{min}	0,05
Кількість епох навчання	epochs	500
Коефіцієнт історичної інформації	β	0,3

Результати порівняння запропонованого методу з існуючими підходами наведено в табл. 4. Експерименти проводились з використанням 5-кратної крос-валідації.

Таблиця 4 – ***Результати експериментальної валідації***

Метрика	GBDT	Random Forest	XGBoost
Accuracy	0.847	0.823	0.861
Precision	0.812	0.798	0.834
Recall	0.779	0.756	0.801

Як видно з табл. 4, запропонований метод демонструє покращення за всіма метриками порівняно з базовими підходами. Найбільший приріст спостерігається за метрикою APFD (+8.1% порівняно з XGBoost), що підтверджує ефективність інтеграції GBDT з навчанням з підкріпленням для задачі пріоритезації.

Результати, наведені в табл. 5, підтверджують перевагу запропонованого гібридного методу GBDT + Q-learning над усіма базовими підходами. Приріст метрики APFD становить 8,1% порівняно з XGBoost та 4,6% порівняно з RETECS. Водночас час навчання гібридної моделі (163,2 с) є вищим, ніж для окремих методів, що пояснюється необхідністю двоетапного навчання. Евристичні методи (total coverage, additional coverage) поступаються ML-підходам, оскільки не враховують багатовимірні залежності між ознаками тестових сценаріїв.

Таблиця 5 – ***Порівняння методів пріоритезації за метрикою APFD***

Метод пріоритезації	**APFD**	**NAPFD**	**Час навчання, с**
Випадковий порядок	0,500	0,498	-
Total coverage	0,634	0,612	-
Additional coverage	0,687	0,659	-
Random Forest	0,723	0,698	45,2
XGBoost	0,751	0,724	38,7
RETECS (Q-learning)	0,776	0,752	124,5
GBDT + Q-learning (пропонований)	**0,812**	**0,789**	**163,2**

Аналіз табл. 6 демонструє, що XGBoost показує найвищу точність класифікації (Accuracy = 0,861), тоді як GBDT посідає проміжну позицію. Різниця у значеннях Precision (0,812 для GBDT проти 0,834 для XGBoost) пояснюється використанням у XGBoost L1/L2-регуляризації, що зменшує кількість хибнопозитивних прогнозів. Показник Recall для всіх моделей є нижчим за Precision, що свідчить про складність виявлення дефектних модулів у проєктах з низькою часткою дефектів (Lucene – 9,3%).

Проте саме при інтеграції GBDT з Q-learning досягається найбільший приріст метрики APFD (табл. 5), що підтверджує синергію ансамблевого підходу та адаптивної пріоритезації.

Таблиця 6 – ***Результати класифікації технічного стану ПЗ (5-кратна крос-валідація)***

Метрика	**GBDT**	**Random Forest**	**XGBoost**
Accuracy	0,847	0,823	0,861
Precision	0,812	0,798	0,834
Recall	0,779	0,756	0,801
F1-score	0,795	0,776	0,817

Висновки

На основі проведених досліджень наявних підходів до автоматизації тестування програмного забезпечення обґрунтовано доцільність автоматизації формування тестових сценаріїв та їх пріоритезації за допомогою моделей і методів машинного навчання. Отримані результати дозволяють сформулювати такі положення. Формалізовано задачу пріоритезації тестових

сценаріїв як задачу комбінаторної оптимізації (2) та визначено метрики оцінки якості пріоритезації APFD (8) та NAPFD (9), що дає змогу кількісно порівнювати різні стратегії упорядкування тестів з урахуванням обмежень часового бюджету CI/CD-середовища. Запропоновано гібридну модель, що інтегрує GBDT-класифікатор з Q-learning агентом. Детально описано конфігурацію гіперпараметрів обох компонентів (табл. 2, 3), стратегію розбиття даних та процедуру навчання, що забезпечує відтворюваність результатів. Визначено основні категорії факторів впливу на якість ПЗ: внутрішні, зовнішні та людські. Розроблено онтологію цих факторів (рис. 1) для формування вектора ознак тестових сценаріїв, що підвищує інформативність вхідних даних для ML-моделей. Експериментальну валідацію на п'яти проєктах репозиторію AEEEM (табл. 4) підтверджено ефективність запропонованого методу: досягнуто покращення метрики APFD на 8,1% порівняно з найкращим базовим методом XGBoost (табл. 5). Час навчання гібридної моделі залишається прийнятним для практичного застосування (163,2 с).

Подальшу перспективу роботи вбачаємо у розширенні набору даних за рахунок промислових проєктів, дослідженні глибинних RL-методів (Deep Q-Network) замість табличного Q-learning та інтеграції великих мовних моделей (LLM) для автоматичної генерації тестових сценаріїв на основі документації.

Конфлікт інтересів. Автор підтверджує відсутність фінансових, особистих чи інших інтересів, які могли б бути розцінені як потенційний конфлікт інтересів щодо публікації цієї статті.

Фінансування. Це дослідження було проведене без залучення зовнішньої фінансової підтримки.

Доступність даних. Усі дані представлені в цифровому або графічному вигляді в основному тексті рукопису.

Використання штучного інтелекту. Автор підтверджує, що під час роботи над статтею інструменти штучного інтелекту не використовувалися.

Список використаних джерел / References

1. Stack Overflow. (2024). *Developer Survey 2024*. URL: https://survey.stackoverflow.co/2024/.
2. Paskov, R. M., & Terenchuk, S. A. (2019). Modeling an intelligent software testing support system. In: *Proceedings of the International Scientific and Practical Conference "Actual Issues of Test Automation"*. Kyiv: KNUBA, pp. 429–432. [Ukrainian source in Ukraine].
3. Alagarsamy, S., Tantithamthavorn, C., & Aleti, A. (2024). A3Test: Assertion-Augmented Automated Test Case Generation. *Information and Software Technology*, vol. 176, art. 107565, pp. 1–15. URL: https://doi.org/10.1016/j.infsof.2024.107565.
4. A Review of Large Language Models for Automated Test Case Generation. (2025). *Machine Learning and Knowledge Extraction*, vol. 7(3), art. 97, pp. 1120–1145. URL: https://doi.org/10.3390/make7030097.
5. Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017). Reinforcement learning for automatic test case prioritization and selection in continuous integration. In: *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 12–22. URL: https://doi.org/10.1145/3092703.3092709.
6. Abualigah, L. et al. (2024). Software Defect Prediction Using Extreme Gradient Boosting (XGBoost). *AL-Rafidain Journal of Computer Sciences and Mathematics*, vol. 18(1), pp. 154–168. URL: https://doi.org/10.33899/csmj.2024.49564.
7. Hu, Y., Wu, L., Li, N., & Zhao, T. (2024). Multi-Agent Decision-Making in Construction Engineering and Management: A Systematic Review. *Sustainability*, vol. 16, p. 7132. URL: https://doi.org/10.3390/su16167132.
8. GeeksforGeeks. (n.d.). *F1 Score in Machine Learning*. URL: https://www.geeksforgeeks.org/f1-score-in-machine-learning/.
9. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In: *Advances in Neural Information Processing Systems*, pp. 3149–3157.
10. GeeksforGeeks. (n.d.). *Categorical Cross-Entropy in Multi-Class Classification*. URL: https://www.geeksforgeeks.org/categorical-cross-entropy-in-multi-class-classification/.
11. Bertolino, A., Guerriero, A., Miranda, B. et al. (2020). Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. In: *Proceedings of ICSE*, pp. 1–12.
12. Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, vol. 8, pp. 279–292.
13. Yang, Y., Li, Z., He, L., & Zhao, R. (2020). A systematic study of reward for reinforcement learning-based continuous integration testing. *Journal of Systems and Software*, vol. 170, art. 110787. URL: https://doi.org/10.1016/j.jss.2020.110787.
14. D'Ambros, M., Lanza, M., & Robbes, R. (2010). An extensive comparison of bug prediction approaches. In: *Mining Software Repositories (MSR)*, pp. 31–41.
15. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. URL: https://doi.org/10.1145/2939672.2939785.

16. Qian, Z. et al. (2025). Reinforcement learning for test case prioritization based on LLEed K-means clustering. *Information and Software Technology*, vol. 179, pp. 107–121. URL: https://doi.org/10.1016/j.infsof.2024.107654.

17. Software Defect Prediction Using Stacking Generalization of Optimized Tree-Based Ensembles. (2022). *Applied Sciences*, vol. 12(9), art. 4577, pp. 1–22. URL: https://doi.org/10.3390/app12094577.

18. Wang, H. et al. (2024). A software defect prediction method using binary gray wolf optimizer and ML algorithms. *Computers & Electrical Engineering*, vol. 118, pp. 109–125. URL: https://doi.org/10.1016/j.compeleceng.2024.109336.

19. Baqar, M. (2024). *The Future of Software Testing: AI-Powered Test Case Generation and Validation*. arXiv preprint. URL: https://arxiv.org/abs/2409.05808.

Oleksii Lopuha
ORCID: http://orcid.org/0000-0001-6397-2710
Kyiv National University of Construction and Architecture, Kyiv, Ukraine
Postgraduate of the department of information technologies

MACHINE LEARNING MODELS FOR AUTOMATED GENERATION AND PRIORITIZATION OF SOFTWARE TEST SCENARIOS

Abstract. *The study explores the possibilities of applying machine learning models to automate the generation and prioritization of software test scenarios within a dynamic continuous integration (CI/CD) environment. The relevance of the work is driven by the need for rapid quality assessment of software systems amidst the exponential growth of critical applications. The scientific novelty of the research lies in the development of a hybrid approach that, for the first time, integrates ensemble Gradient Boosted Decision Tree (GBDT) models for defect prediction with Reinforcement Learning (Q-learning) algorithms for adaptive test prioritization. The prioritization task is formalized as a combinatorial optimization problem with a weight function, and key quality metrics are defined: APFD (Average Percentage of Faults Detected) and its time-aware modification, NAPFD. An original ontological model of factors influencing software quality has been developed, structuring internal (code complexity), external (infrastructure), and human factors (developer qualification). A modified reward function for the reinforcement learning agent is proposed, which balances defect detection, test execution duration, and historical efficiency. Experimental validation was conducted using benchmark datasets from the AEEEM repository, covering real-world defect data from five open-source Eclipse ecosystem projects: JDT Core, Equinox, Lucene, Mylyn, and PDE. Comparative analysis results showed that the integration of GBDT with Q-learning provides an 8.1% improvement in the APFD metric compared to the best baseline method, XGBoost, and a 4.6% improvement over the existing RETECS method. This confirms the high efficiency of the proposed approach for optimizing test runs under limited time budgets. The training time of the hybrid model is 163.2 seconds, which is entirely acceptable for practical implementation in industrial software development pipelines.*

Keywords: software testing; test generation; test case prioritization; gradient boosting; machine learning; iterative learning; continuous integration

---

Посилання на публікацію

APA Lopuha, O. (2026). Machine learning models for automated generation and prioritization of software test scenarios. *Management of Development of Complex Systems*, 65, 141–149, dx.doi.org\10.32347/2412-9933.2026.65.141-149.

ДСТУ Лопуга О. М. Моделі машинного навчання для автоматизації генерації та пріоритезації тестових сценаріїв програмного забезпечення. *Управління розвитком складних систем.* Київ, 2026. № 65. С. 141 – 149, dx.doi.org\10.32347/2412-9933.2026.65.141-149.