**УДК 004.415.5:004.8:519**
**DOI https://doi.org/10.32782/tnv-tech.2025.5.1.36**

# ВИКОРИСТАННЯ МАТЕМАТИЧНИХ І ШІ-МОДЕЛЕЙ В ТЕСТУВАННІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

***Шкурко В. В.*** *– аспірант кафедри прикладної математики*
*факультету інформаційно-аналітичних технологій та менеджменту*
*Харківського національного університету радіоелектроніки*
*ORCID ID: 0009-0003-3274-3941*

***Поляков А. О.*** *– кандидат технічних наук, доцент кафедри інформаційних систем*
*Навчально-наукового інституту інформаційних технологій*
*Харківського національного економічного університету імені Семена Кузнеця,*
*доцент кафедри прикладної математики*
*факультету інформаційно-аналітичних технологій та менеджменту*
*Харківського національного університету радіоелектроніки*
*ORCID ID: 0000-0003-1805-9011*

*В статті представлено комплексне дослідження та аналіз застосування математичних моделей т а моделей штучного інтелекту в тестуванні програмного забезпечення, яке спрямоване на підвищення ефективності, та автономності процесів контролі якості. Призведена інтеграція ключових напрямів інтелектуального моделювання – великі мовні моделі (LLM), еволюційні алгоритми, штучні нейронні мережі та три трактування трьох RM (Reinforcement Models, Risk Management Models, Regression Models), демонструючи їх потенціал для оптимізації сучасних підходів до тестування.*

*Узагальнені аналітичні підходи сформовано цілісні теоретико-методологічні засади побудови інтелектуальних систем тестування ПЗ. Також проаналізовано математичні моделі формального доведення коректності, ймовірнісні моделі помилок, регресійні RM-моделі передбачення дефектності та алгоритмічні структури для аналізу складності й оптимізації тестових наборів. Еволюційні алгоритми розглянуто як метод оптимізації шляхів проходження графів станів, формування мінімальних, але покривних тестових сценаріїв та генерування тестів у середовищах із великою кількістю варіантів конфігурацій. Нейронні мережі були представлені як основа для побудови моделей прогнозування ризиків, автоматизації регресійного тестування та виявлення прихованих закономірностей у телеметричних даних. Безпосередньо обґрунтовано можливість інтеграції RL-моделей (як частини RM) для створення самонавчальних тестувальних агентів, здатних адаптувати стратегії перевірки до поведінки системи.*

*Запропонована концептуальна архітектура інтелектуальної системи тестування поєднує математичні формальні підходи та сучасні AI-моделі, забезпечуючи багаторівневий аналіз ризиків, автоматичне удосконалення тестових наборів і прийняття рішень на основі RM-методів управління ризиками. Отримані результати демонструють значний потенціал впровадження LLM, еволюційних алгоритмів, нейронних мереж та RM-моделей у практику тестування ПЗ, що дозволяє скоротити витрати часу, підвищити точність та забезпечити адаптивність до змінних умов розробки. Дослідження формує теоретичну основу для подальших досліджень у сфері інтелектуалізації тестування та відкриває перспективи для створення нових поколінь автономних тестувальних систем.*

***Ключові слова:*** *LLM, еволюційні алгоритми, Reinforcement Models, Risk Management Models, Regression Models, нейромережі.*

***Shkurko V. V., Poliakov A. O. Using mathematical and AI-models in software testing***

*The article presents a comprehensive study and analysis of the application of mathematical models and artificial intelligence models in software testing, which is aimed at increasing the*

*efficiency and autonomy of quality control processes. The integration of key areas of intelligent modeling is brought about - large language models (LLM), evolutionary algorithms, artificial neural networks and three interpretations of the three RM (Reinforcement Models, Risk Management Models, Regression Models), demonstrating their potential for optimizing modern approaches to testing.*

*Generalized analytical approaches have formed holistic theoretical and methodological principles for building intelligent software testing systems. Also analyzed are mathematical models of formal proof of correctness, probabilistic error models, regression RM-models of defect prediction and algorithmic structures for analyzing the complexity and optimizing test sets. Evolutionary algorithms are considered as a method for optimizing state graph traversal paths, forming minimal but covering test scenarios and generating tests in environments with a large number of configuration options. Neural networks were presented as a basis for building risk prediction models, automating regression testing, and detecting hidden patterns in telemetric data. The possibility of integrating RL models (as part of RM) to create self-learning test agents capable of adapting testing strategies to system behavior was directly substantiated.*

*The proposed conceptual architecture of an intelligent testing system combines mathematical formal approaches and modern AI models, providing multi-level risk analysis, automatic improvement of test sets, and decision-making based on RM risk management methods. The results obtained demonstrate the significant potential of implementing LLM, evolutionary algorithms, neural networks, and RM models in software testing practice, which allows reducing time costs, increasing accuracy, and ensuring adaptability to changing development conditions. The study forms a theoretical basis for further research in the field of testing intellectualization and opens up prospects for creating new generations of autonomous testing systems.*

***Key words:*** *LLM, evolutionary algorithms, Reinforcement Models, Risk Management Models, Regression Models, neural networks.*

**Постановка проблеми.** Розвиток інформаційних технологій, зростання вимог до якості програмного забезпечення в контексті появи складних багатокомпонентних середовищ з прогресивним масштабуванням програмних систем зумовлюють системи змін у підходах до тестування. Традиційні практики, які проводились в ручному режимі втрачають ефективність у середовищі, де превалює швидкість результатів, де міжмодульні зв'язки ставляться все більш складнішими, архітектури можуть змінюватись в надшвидких заходах не відповідають вимогам сучасності і вимагають автоматизації з можливістю адаптації, що вимагає використання інтелектуальних механізмів аналізу.

За таких умов стає ключовим напрямом розвитку математичне моделювання та штучний інтелект (ШІ), великі мовні моделі (LLM), еволюційні алгоритми та нейромережі різних типів RM-моделей – моделей з підкріпленням (Reinforcement Models), моделей з управління ризиками (Risk Management) та регресивних моделей (Regression Models).

У сучасності, підходи що забезпечують якість ПЗ (Quality Assurance) спостерігається напрям до зменшення в бік інтелектуального тестування (Intelegent Testing), що характеризується поєднанням формальних математичних методів, статистичних та оптимізаційних моделей з глибинними формуванням ШІ, які можуть не тільки оптимізувати ручні операції, алей адаптувати своє функціонування на основі власної діяльності. Особливої уваги потребують моделі, що здатні обробляти великі масиви, виявляти приховані закономірності та прогнозувати можливі дефекти і забезпечувати глибоке розуміння складних програмних систем.

Згідно викликів сучасності щодо оптимізації тестування ПЗ, нами була сформована мета дослідження – зробити комплексний аналіз використання математичних моделей та моделей штучного інтелекту з використанням нейронних мереж у тестуванні програмного забезпечення.

Для досягнення запланованої мети необхідно вирішити ряд завдань, а саме:

1. Провести огляд теоретичних і практичних аспектів застосування цих підходів.

2. Сформувати інтегровану архітектуру інтелектуального тестування.

3. Визначити потенціал поєднання формальних та інтелектуальних методів для створення високопродуктивних , адаптивних і самонавчальних систем тестування.

**Методологія дослідження** ґрунтується на послідовному аналізі теоретичних і практичних підходів до використання математичних моделей та моделей штучного інтелекту в тестуванні програмного забезпечення. Спочатку проводився огляд сучасних літературних джерел, щоб визначити актуальні концепції й окреслити їхню корисність на практиці. Після цього досліджувались математичні методи, що застосовуються в системах тестування, включно з тими, які забезпечують формалізований аналіз, оцінку ризиків та моделювання поведінки програмних систем. Тестування розглядалось як багаторівнева структура, що охоплює формальні моделі, алгоритмічні процеси, інформаційні потоки та поведінкові аспекти роботи програмного забезпечення. У межах такого підходу аналізувалась можливість поєднання математичних методів і моделей штучного інтелекту для підвищення рівня автоматизації та адаптивності, тобто визначається їхня інтегративність. Завершальним етапом була формалізація алгоритмів, яка забезпечує стабільні й повторювані результати під час тестування.

**Аналіз останніх досліджень і публікацій.** Необхідно зазначити, що значну частину у програмній інженерії відводиться машинному навчанню та ШІ. Машинне навчання застосовується для прогнозування дефектів (fault prediction), пріоритизації тестів, статистичного та динамічного аналізу коду, оцінки ризику модулів [1]. Malhotra [2] у своєму систематичному огляді показав, що ансамблеві моделі, дерева рішень та глибокі нейронні мережі демонструють найвищу точність у прогнозуванні дефектів.

Тестування нейронних мереж є особливо складним через їхню стохастичну та нечітку природу. Zhang і Li [3] зазначають, що для нейронних мереж класичні критерії покриття неефективні, тому потрібні підходи на основі аналізу внутрішніх активацій та оцінку стійкості моделей (adversarial testing, coverage).

Xiao et al. [4] розробили ANN-моделі для прогнозу виявлення помилок та зусиль тестування. Suman і Khan [5] запропонували оптимізовану нейромережу для прогнозування загроз безпеки ПЗ. Batool і Khan [6] підкреслює, що складність ПЗ напряму впливає на тестованість, тому оцінка метрик є критичною для побудови точних моделей прогнозу дефектів.

Також підкреслюється в літературі, значення пріоритизації тестів при обмежених ресурсах та часових лімітах. Згідно цього можна виділити переваги еволюційного алгоритму (ЕА) та метаевистик, які застосовуються для генерації тестів, оптимізації покриття та кластеризації коду. Arasteh et al. [7] розробили метод Traxtor, натхненний імперіалістичною конкуренцією, для автоматичного створення тестових наборів. Arasteh et al. [8] застосували модифікований алгоритм horse herd optimization для кластеризації вихідного коду, показавши ефективність у складних системах. Chen et al. [9] пропонують адаптивний метод пріоритизації тестів на основі кластеризації об’єктно-орієнтованого ПЗ.

Окремим стовпом стоять великі мовні моделі (LLM). LLM змінюють парадигму тестування ПЗ, вони можуть виконувати reasoning та багатостадійний аналіз, інтегруючи дані з історії змін, логів та CI/CD-систем. LLM з 2020-х років зробили революцію в підходах до генерації та аналізу тестів. Моделі GPT (4–5) та інші здатні аналізувати код, формувати тестові сценарії, проводити статичну верифікацію і навіть моделювати поведінку користувачів [10].

Також в літературі зазначені і моделі трьох RM. (Requirement Modeling, Risk Management, Reinforcement Learning). Requirement Modeling (RM1) дозволяє

автоматично виділяти ключові функціональні вимоги, визначати залежності між модулями та генерувати тест-кейси.

Risk Management (RM-2) дозволяє прогнозувати ризики, забезпечувати пріоритизацію тестів. Nikravan і Haghi Kashani [11] показують, що моделі ризику можна використовувати для класифікації модулів за ймовірністю відмови. Графи, теплові карти, діаграми покриття допомагають ідентифікувати модулі з високим ризиком.

Reinforcement Learning (RM-3) дозволяє агенту навчатись взаємодіяти з ПЗ як з "чорною скринькою" (Aboeleneen et al., 2023) [12]. Це забезпечує: адаптивну навігацію в просторі станів, оптимізацію вибору тестових дій,автономне самонавчання стратегій тестування.

Data-driven підходи та прогнозування дефектів також передбачають використання великих наборів даних, логів, історії комітів та метрик коду для прогнозування дефектів.

Огляд сучасних джерел з питання тестування програмного забезпечення показує, що дедалі більше іде використання штучного інтелекту (AI), включно з великими мовними моделями (LLM), еволюційними алгоритмами, нейронними мережами та гібридними методами. Завдяки цьому можна виділити основні задачі і підходи в тестуванні, які можна побачити в таблиці 1.

Таблиця 1

**Задачі і підходи в тестуванні з урахуванням переваг і обмежень**

| Підхід | Основні задачі | Переваги | Обмеження |
|---|---|---|---|
| Еволюційні алгоритми | Оптимізація покриття | Скорочення тестів, глобальний пошук | Вимагає багато ітерацій |
| Нейронні мережі | Прогнозування відмов | Висока точність | Погана інтерпретованість |
| RM (Reinforcement Models) | Трасування, генерація тестів | Формалізація вимог | Помилки в моделюванні |
| RM (Risk Management) | Пріоритизація тестів | Оцінка ризику | Чутливість до помилок |
| RM (Regression Models) | Адаптивне тестування | Автономія, самонавчання | Складність налаштування винагород |
| LLM (Large Language Model) | Генерація тестів, аналіз логів | Автоматизація міркування | Вимоги до даних, "чорний ящик" |

Отже, можна стверджувати, що є значний потенціал поєднання математичних моделей і моделей штучного інтелекту для розвитку інтелектуального тестування.

**Виклад основного матеріалу.** Згідно вищесказаного ми можемо побудувати схему "Класифікація AI-підходів у тестуванні ПЗ (ASCII)" (рис. 1).

Ця класифікація відокремлює чотири основні категорії:

– Еволюційні алгоритми та метаевристики – для оптимізації тест-кейсів, покриття коду та кластеризації.

– Нейронні мережі – для прогнозування дефектів, оцінки ризиків та класифікації помилок.

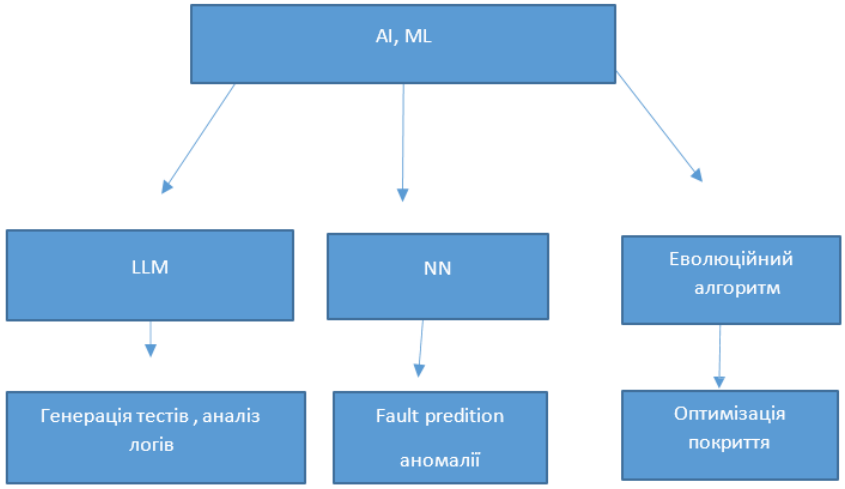


*Рис. 1. AI підходи у тестуванні ПЗ та їх класифікація*

– Великі мовні моделі (LLM) – для генерації тестів з природномовних вимог, аналізу логів та статичної верифікації коду.

– RM-моделі (тріада):

Regression Models (RM-1) – для моделювання вимог та передбачення дефектності. Risk Management Models (RM-2) – для пріоритизації тестів на основі оцінки ймовірності відмов. Reinforcement Models (RM-3) – для створення самонавчальних агентів, що адаптують стратегії тестування.

Ця класифікація служить основою для вибору та комбінування методів під час конкретних завдань тестування. На основі аналізу запропоновано багаторівневу архітектуру системи тестування, що інтегрує формальні математичні моделі та сучасні AI-підходи. Архітектура включає наступні рівні:

1. Рівень даних.

Збір та зберігання метрик коду, історії комітів, логів виконання, тест-кейсів та вимог.

2. Рівень аналітики та ML.

На цьому рівні функціонують прогностичні моделі AI-моделі (Нейромережі, Регресійні моделі), аналізують дані для прогнозування дефектів та оцінки ризиків.

3. Генеративні моделі.

LLM, EA відповідають за створення та оптимізацію тестових сценаріїв.

4. Оптимізаційні моделі.

EA, RL займаються мінімізацією тест-наборів, пріоритизацією та покриттям складних шляхів.

5. Рівень прийняття рішень (Risk Management).

Отримання даних від аналітичного рівня для стратегічного планування тестування, розподілу ресурсів та оцінки загального ризику.

6. Рівень виконання та зворотного зв'язку.

Автоматизоване виконання тестів, моніторинг поведінки системи та збір даних для закриття циклу навчання AI-моделей (особливо Reinforcement Models).

Дослідження показало, що поєднання різних підходів призводить до синергетичного ефекту, значно підвищуючи ефективність тестування:

1. LLM + Еволюційні алгоритми.

LLM можуть генерувати початкові популяції тестових сценаріїв на основі вимог, а еволюційні алгоритми – оптимізувати їх для максимального покриття.

2. Нейромережі + Risk Management Models.

Прогнози нейромереж щодо дефектності модулів стають вхідними даними для моделей управління ризиками, що дозволяє об'єктивно пріоритизувати тести.

3.Reinforcement Models + Формальні методи.

Агенти з підкріпленням можуть експериментально досліджувати поведінку системи ("чорний ящик"), а формальні методи забезпечують коректність і повноту виявлених шляхів.

Головним результатом цього дослідження є констатація зміни парадигми від простої автоматизації до повноцінної автономії. Традиційні інструменти автоматизації тестування виконують заздалегідь написані сценарії. Запропонована ж інтелектуальна система, що поєднує математичні моделі та ШІ, здатна до самостійного навчання, адаптації та прийняття рішень.

Наприклад, модель з підкріпленням (Reinforcement Models), інтегрована в архітектуру, не лише виконує тести, але й експериментує, щоб знайти нові, більш ефективні шляхи тестування або виявити неочікувані стани системи, які не були передбачені людиною. Це перетворює тестувальну систему з пасивного виконавця на активного дослідника.

Як видно з аналізу, кожен окремий підхід має свої обмеження: еволюційні алгоритми можуть бути обчислювально-витратними, нейромережі – неінтерпретованими, а LLM – неперевірними "чорними скриньками". Однак ключовим висновком є те, що їхнє комбіноване використання дозволяє компенсувати ці недоліки.

Формальні математичні моделі (як моделі формального доведення коректності) забезпечують детермінізм, точність і можливість верифікації, виступаючи "стовпом" надійності. Інтелектуальні моделі (ШІ) надають гнучкість, здатність до апроксимації складних нелінійних залежностей та роботи в умовах неповноти даних.

Таким чином, формальні методи можуть задавати "коридор безпеки" для експериментів ШІ, а ШІ-моделі, своєю чергою, можуть розширювати охоплення тестами для систем, занадто складних для повної формальної верифікації. Наприклад, LLM може згенерувати тисячі тестових варіантів, а формальна модель – відфільтрувати та валідувати їх на відповідність базовим інваріантам системи.

Поява великих мовних моделей значно змінює роль інженера з QA. Замість рутинного написання тест-кейсів, фахівець все більше перетворюється на "архітектора якості" або "куратора AI-систем". Його завданнями стають формулювання точних запитів (prompts) для LLM, валідація та інтерпретація результатів, згенерованих AI, налаштування та вибір моделей для конкретних завдань.

Аналіз та усунення "помилок" в логіці AI, що вимагає глибокого розуміння як предметної області, так і принципів роботи моделей. Це підвищує вимоги до кваліфікації, але водночас значно підвищує продуктивність та дозволяє сконцентруватися на найскладніших та найкритичніших аспектах якості ПЗ.

Інтеграція AI в тестування породжує низку серйозних викликів, які потребують подальшого вирішення:

1. Відповідальність за помилки.

Хто несе відповідальність, якщо AI-система пропустила критичний дефект: розробник моделі, тестувальник-куратор чи власник даних?

2. Прозорість та довіра.

Проблема “чорного ящика” особливо актуальна для безпек-критичних та медичних застосунків. Необхідний розвиток методів Explainable AI (XAI) для тестування, щоб людина могла розуміти, чому модель прийняла те чи інше рішення.

3.Вплив на безпеку.

AI-моделі, особливо LLM, самі можуть стати вектором атаки (наприклад, через adversarial attacks або “ін'єкції запитів”). Таким чином, система тестування повинна включати механізми тестування безпеки власних AI-компонентів.

**Висновки і перспективи подальших досліджень.** Проведене дослідження систематизує та класифікує сучасні AI-підходи в тестуванні ПЗ, виділивши чотири ключові категорії: еволюційні алгоритми та метаевристики для оптимізації; нейронні мережі для прогнозування; великі мовні моделі для генерації та аналізу; та RM-моделі (Reinforcement Models, Risk Management Models, Regression Models) для адаптивного управління. Ця класифікація забезпечує методологічну основу для вибору та комбінування методів відповідно до специфіки завдань тестування. Було запропоновано інтегровану архітектуру інтелектуальної системи тестування, яка поєднує формальні математичні підходи з AI-моделями в єдиний цикл автоматизації та навчання. Ця багаторівнева архітектура забезпечує перехід від простої автоматизації до повноцінної автономності, де система здатна не лише виконувати тести, але й адаптувати стратегії тестування, прогнозувати ризики та оптимізувати власну продуктивність.

Також виявлено значний синергетичний ефект від інтеграції різних підходів. Комбінування формальних методів з інтелектуальними моделями дозволяє компенсувати обмеження окремих технологій: формальні методи забезпечують детермінізм і точність, тоді як ШІ-моделі надають гнучкість та здатність до роботи в умовах неповноти даних.

Інтелектуальні системи тестування на основі математичних моделей та ШІ дозволяють значно скоротити витрати часу за рахунок автоматизації рутинних операцій та оптимізації тест-наборів. Вони підвищують точність виявлення дефектів через здатність аналізувати складні закономірності та прогнозувати ризикові зони. Крім того, такі системи забезпечують адаптивність до змінних умов розробки, здатність самостійно оновлювати тестові сценарії у відповідь на зміни в архітектурі ПЗ.

Незважаючи на значний потенціал, масштабне впровадження інтелектуальних методів тестування стикається з низкою викликів. Серед них – проблема “чорного ящика” у роботі складних AI-моделей, необхідність у великих обсягах якісних даних для навчання, обчислювальна складність ітераційних алгоритмів, а також етичні аспекти відповідальності за помилки, допущені AI-системами.

Майбутні дослідження мають бути спрямовані на розвиток методів інтерпретованого ШІ для тестування, створення стандартизованих бенчмарків для оцінки AI-моделей, розробку етичних рамок використання ШІ в QA, а також дослідження потенціалу квантових алгоритмів для прискорення оптимізаційних процесів.

У цілому, дослідження підтверджує, що поєднання математичної строгості з гнучкістю штучного інтелекту формує нову парадигму в тестуванні ПЗ – парадигму інтелектуального, проактивного та автономного контролю якості. Ця трансформація дозволяє перетворити тестування з затратної та часто “відстаючої” фази розробки на стратегічний актив, здатний впоратися зі складністю сучасних програмних систем. Подальший розвиток у цьому напрямку є не просто бажаним, а необхідним для майбутньої програмної інженерії, що забезпечить створення

більш надійного, безпечного та якісного програмного забезпечення в умовах стрімкої цифровізації суспільства.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Kotti Z., Galanopoulou R., Spinellis D. Machine learning for software engineering: A tertiary study. *ACM Computing Surveys*. 2023. Vol. 55, No. 12. Article 256. DOI: 10.1145/3572905

2. Malhotra R. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*. 2015. Vol. 27. P. 504–518. DOI: 10.1016/j.asoc.2014.11.023

3. Zhang J., Li L. Testing and verification of neural-network-based safety-critical control software: A systematic literature review. *Information & Software Technology.* 2020. Vol. 123. Article 106296. DOI: 10.1016/j.infsof.2019.106231

4. Xiao H., Cao M., Peng R. Artificial neural network based software fault detection and correction prediction models considering testing effort. *Applied Soft Computing*. 2020. Vol. 94. Article 106491. DOI: 10.1016/j.asoc.2020.106491

5. Suman S., Khan R. A. An optimized neural network for prediction of security threats on software testing. *Computers & Security*. 2024. Vol. 136. Article 103626. DOI: 10.1016/j.cose.2023.103626

6. Batool I., Khan T. A. Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review. Computers & Electrical Engineering. 2022. Vol. 100. Article 107886. DOI: 10.1016/j.compeleceng.2022.107886

7. Arasteh B., Hosseini S. M. J. Traxtor: An automatic software test suit generation method inspired by imperialist competitive optimization algorithms. *Journal of Electronic Testing*. 2022. Vol. 38, No. 2. P. 205–215. DOI: 10.1007/s10836-022-05999-9

8. Arasteh B., Gunes P., Bouyer A., Soleimanian Gharehchopogh F., Alipour Banaei H., Ghanbarzadeh R., Natella R. A modified horse herd optimization algorithm and its application in the program source code clustering. *Complexity*. 2023. Vol. 2023. P. 1–16. DOI: 10.1155/2023/3988288

9. Chen J., Zhu L., Chen T. Y., Towey D., Kuo F.-C., Huang R., Guo Y. Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering. Journal of Systems and Software. 2018. Vol. 135. P. 107–125. DOI: 10.1016/j.jss.2017.09.031

10. PrajnaAI. LLM Trends 2025: A Deep Dive into the Future of Large Language Models. 2025. URL: https://prajnaaiwisdom.medium.com/llm-trends-2025-a-deep-dive-into-the-future-of-large-language-models-bff23aa7cdbc (дата звернення: 10.11.2025).

11. Nikravan M., Haghi Kashani M. A review on trust management in fog/edge computing: Techniques, trends, and challenges. *Journal of Network and Computer Applications*. 2022. Vol. 204. Article 103402. DOI: 10.1016/j.jnca.2022.103402

12. Abdellatif A. A., Abo-Eleneen A., Mohamed A., Erbad A., Navkar N. V., Guizani M. Intelligent-slicing: An AI-assisted network slicing framework for 5G-and-beyond networks. *IEEE Transactions on Network and Service Management*. 2023. Vol. 20, No. 2. P. 1024–1039. DOI: 10.1109/TNSM.2023.3274236

## REFERENCES:

1. Kotti, Z., Galanopoulou, R., & Spinellis, D. (2023). Machine learning for software engineering: A tertiary study. *ACM Computing Surveys, 55*(12), Article 256. https://doi.org/10.1145/3572905

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

3. Zhang, J., & Li, L. (2020). Testing and verification of neural-network-based safety-critical control software: A systematic literature review. *Information & Software Technology, 123*, 106296. https://doi.org/10.1016/j.infsof.2019.106231

4. Xiao, H., Cao, M., & Peng, R. (2020). Artificial neural network based software fault detection and correction prediction models considering testing effort. *Applied Soft Computing, 94*, 106491. https://doi.org/10.1016/j.asoc.2020.106491

5. Suman, & Khan, R. A. (2024). An optimized neural network for prediction of security threats on software testing. *Computers & Security, 137*, 103626. https://doi.org/10.1016/j.cose.2023.103626

6. Batool, I., & Khan, T. A. (2022). Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review. *Computers & Electrical Engineering, 100*, 107886. https://doi.org/10.1016/j.compeleceng.2022.107886

7. Arasteh, B., & Hosseini, S. M. J. (2022). Traxtor: An automatic software test suite generation method inspired by imperialist competitive optimization algorithms. *Journal of Electronic Testing, 38*(2), 205–215. https://doi.org/10.1007/s10836-022-05999-9

8. Arasteh, B., Gunes, P., Bouyer, A., Soleimanian Gharehchopogh, F., Alipour Banaei, H., Ghanbarzadeh, R., & Natella, R. (2023). A modified horse herd optimization algorithm and its application in the program source code clustering. *Complexity, 2023*, 1–16. https://doi.org/10.1155/2023/3988288

9. Chen, J., Zhu, L., Chen, T. Y., Towey, D., Kuo, F.-C., Huang, R., & Guo, Y. (2018). Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering. *Journal of Systems and Software, 135*, 107–125. https://doi.org/10.1016/j.jss.2017.09.031

10. PrajnaAI. (2025). *LLM trends 2025: A deep dive into the future of large language models*. Retrieved from https://prajnaaiwisdom.medium.com/llm-trends-2025-a-deep-dive-into-the-future-of-large-language-models-bff23aa7cdbc

11. Nikravan, M., & Haghi Kashani, M. (2022). A review on trust management in fog/edge computing: Techniques, trends, and challenges. *Journal of Network and Computer Applications, 204*, 103402. https://doi.org/10.1016/j.jnca.2022.103402

12. Abdellatif, A. A., Abo-Eleneen, A., Mohamed, A., Erbad, A., Navkar, N. V., & Guizani, M. (2023). Intelligent-slicing: An AI-assisted network slicing framework for 5G-and-beyond networks. *IEEE Transactions on Network and Service Management, 20*(2), 1024–1039. https://doi.org/10.1109/TNSM.2023.3274236

*Дата першого надходження рукопису до видання: 23.10.2025*
*Дата прийнятого до друку рукопису після рецензування: 20.11.2025*
*Дата публікації: 30.12.2025*