doi: 10.32620/oikit.2025.105.15

УДК 004.4:004.8:004.9

С. Ю. Манаков,
О. Г. Трофименко,
П. О. Чикунов,
В. І. Гура

ШІ-керовані системи розробки кросплатформних застосунків

Національний університет «Одеська юридична академія»

Дослідження присвячене аналізу сучасного стану та перспектив розвитку штучного інтелекту (ШІ) в системах розробки кросплатформних застосунків. Розглянуто ключові технології машинного навчання, які застосовуються у процесах автоматизації програмної інженерії, включаючи системи автоматичної генерації коду на основі великих мовних моделей, інтелектуальні середовища розробки та ШІ-керовані методології тестування. Проаналізовано архітектурні рішення сучасних кросплатформних фреймворків та їхню інтеграцію з технологіями штучного інтелекту. Досліджено застосування трансформерних моделей, зокрема GPT-4/Codex, Claude, CodeT5 та CodeBERT, у задачах розуміння та генерації програмного коду. Аналіз показав, що наразі GPT-4/Codex є найточнішою та найпотужнішою ШІ-моделлю, яка підходить для складної генерації коду. CodeT5 тримає баланс між розміром і продуктивністю, а тому добре підходить для завдань трансформації коду. InCoder спеціалізується на заповненні шаблонів у коді, але має нижчу точність. CodeBERT більше підходить для аналітики коду, ніж для генерації. Розглянуто методи оцінювання якості ШІ-генерованого коду, включаючи метрики функціональної коректності та структурної якості. Висвітлено виклики безпеки та надійності автоматично згенерованого програмного коду, включаючи проблеми вразливостей та потребу у додатковій верифікації. Представлено аналіз ефективності різних підходів до кросплатформної розробки з використанням інструментів штучного інтелекту. Наукова новизна роботи полягає у комплексному аналізі взаємодії технологій штучного інтелекту з кросплатформними фреймворками розробки, систематизації сучасних підходів до ШІ-керованої генерації коду та дослідженні специфічних викликів інтеграції машинного навчання у процеси багатоплатформної розробки програмного забезпечення. Результати дослідження показують значний потенціал інтеграції технологій штучного інтелекту для підвищення продуктивності розробників та покращення якості програмного забезпечення.
Ключові слова: кросплатформна розробка, штучний інтелект, генерація коду, великі мовні моделі, інженерія програмного забезпечення, тестування, інтегроване середовище розробки.

Вступ

Сучасна програмна інженерія переживає кардинальну трансформацію під впливом технологій штучного інтелекту (ШІ). За останні роки спостерігається експоненційне зростання впровадження ШІ-керованих систем у процеси розробки програмного забезпечення, що відображається у статистичних показниках. Так, за даними опитувань 64% розробників активно використовують ШІ-інструменти у своїй повсякденній роботі [1]. Особливо актуальною є інтеграція технологій машинного навчання у кросплатформну розробку застосунків, де складність управління множинними цільовими платформами створює потребу в інтелектуальній автоматизації.

Актуальність дослідження обумовлена кількома ключовими факторами. По-перше, зростання складності сучасних програмних систем вимагає нових підходів до автоматизації рутинних завдань розробки. По-друге, розвиток великих мовних моделей (Large Language Model, LLM), як-от: GPT-4, Claude та

спеціалізованих систем генерації коду, відкриває нові можливості для автоматичного створення якісного програмного коду. По-третє, кросплатформна розробка стає домінантним підходом у мобільній та десктопній розробці, що створює потребу в ШІ-системах, здатних оптимізувати процеси адаптації коду для різних платформ.

1. Аналіз останніх досліджень і публікацій

Огляд сучасних досліджень у сфері ШІ-керованих систем розробки програмного забезпечення демонструє інтенсивний розвиток галузі та розмаїття підходів до розв'язання ключових завдань автоматизації програмної інженерії. Так, у дослідженні архітектур нейронних мереж для розуміння та генерації коду представлені в роботі [2] запропоновано модель CodeT5 з архітектурою кодувальник-декодувальник та завданнями попереднього навчання, включно з Identifier Tagging для кращого розуміння структури коду. У роботі [3] ШІ-модель досягла значних покращень у завданнях генерації та трансляції коду і показала гарні результати на CodeXGLUE. У статті [4] для розуміння коду та виявлення клонів використано модель CodeBERT, яка має BERT-подібну архітектуру. У дослідженні [5] представлено підхід до генеративного моделювання коду на моделі InCoder, яка реалізує можливість заповнення коду без попереднього навчання з двонаправленим контекстом. Це досягається через архітектуру з причинно-наслідковим маскуванням, що дозволяє моделі ефективно обробляти як лівий, так і правий контекст при генерації коду. У роботі [6] проведено системний літературний огляд впливу ШІ-моделей на безпеку генерованого коду та досліджено безпеку ШІ-генерованого коду. Це дослідження виявило проблеми з тим, що ШІ-моделі схильні генерувати код із вразливостями, які входять до переліку MITRE CWE Top 25, що підкреслює потребу додаткових механізмів верифікації безпеки. Стаття [7] присвячена аналізу впливу ШІ-інструментів на якість коду та досліджено задоволеність розробників при використанні ШІ-інструментів. Це дослідження виявило високий рівень задоволеності (понад 75%) при правильному впровадженні технологій, однак його автори наголошують на важливості збалансованого підходу до автоматизації та людського контролю при використанні ШІ-інструментів. Дослідження команди Machine Learning Lab Українського католицького університету під керівництвом професора Остапа Добосевича зосереджено на алгоритмах глибокого навчання для задач візуального розпізнавання [8]. Робота [9] демонструє високий рівень сучасних досліджень у сфері емпіричного аналізу ШІ-технологій у мобільних застосунках, що охоплює аналіз 56682 реальних ШІ-застосунків з акцентом на кросплатформні рішення та їх оптимізацію. У статті [10] запропоновано холістичний підхід до підвищення рівня абстракції в кросплатформній розробці через автоматичні трансформації для генерації виконуваного коду. Методологічні аспекти інтеграції ШІ в програмну інженерію розглянуто в роботі [11], в якій проаналізовано переваги, виклики та майбутні напрямки розвитку галузі. Дослідження підкреслює, що ШІ підвищує продуктивність розробки, покращує якість коду та прискорює цикли розробки, але вимагає стратегічного підходу до впровадження.

Аналіз останніх досліджень і публікацій свідчить про потребу формування комплексної екосистеми ШІ-керованих інструментів розробки, виявив прогалини в дослідженнях специфічних аспектів інтеграції машинного навчання з кросплатформними технологіями, що обумовлює актуальність даного дослідження.

2. Мета роботи

Мета дослідження – системний аналіз сучасного стану та перспектив розвитку ШІ-керованих систем розробки кросплатформних застосунків з виявленням ключових технологічних рішень, методологічних підходів та оцінкою ефективності їх практичного застосування.

Завдання дослідження:

1. проаналізувати архітектурні рішення та технічні характеристики сучасних ШІ-систем генерації програмного коду, включаючи великі мовні моделі та спеціалізовані трансформерні архітектури;
2. дослідити методи інтеграції технологій ШІ з провідними кросплатформними фреймворками розробки застосунків;
3. систематизувати підходи до автоматизації процесів розробки програмного забезпечення засобами машинного навчання, включаючи інтелектуальні середовища розробки та ШІ-керовані системи тестування;
4. оцінити ефективність та якість ШІ-генерованого коду через аналіз наявних метрик та методів верифікації;
5. виявити основні виклики безпеки та надійності при використанні ШІ-керованих систем розробки.

3. Архітектурні рішення ШІ-систем генерації коду

Сучасні ШІ-системи генерації програмного коду базуються на архітектурі трансформерних нейронних мереж, що забезпечує ефективне моделювання довгих послідовностей коду та розуміння складних залежностей між програмними конструкціями. Більшість систем заснована на архітектурі encoder-decoder з механізмом самоуваги (self-attention), що дозволяє моделям одночасно обробляти контекст коду та природномовні описи завдань.

Модель ШІ OpenAI Codex, яка є базою GitHub Copilot, використовує архітектуру GPT-4o, навчену на гігабайтах Python-коду з мільйонів публічних репозиторіїв. Модель демонструє гарні результати: 67.0% точності на бенчмарку HumanEval для завдань кодування мовою Python [12].

Альтернативний підхід реалізовано в моделі CodeT5, яка використовує dual-encoder архітектуру зі спеціалізованими завданнями попереднього навчання. Введення механізму розрізнення ідентифікаторів у коді від звичайних токенів Identifier Tagging значно покращило розуміння семантики програм. Бімодальна подвійна генерація забезпечила ефективність злагодження між природною мовою та програмним кодом [2].

Модель InCoder від Meta має можливості заповнення пропусків у коді з урахуванням як лівого, так і правого контексту. Це досягається через архітектуру з маскуванням причинно-наслідкових зв'язків, в якій замасковані регіони переміщуються в кінець послідовності, чим дозволяють моделі ефективно використовувати двонапрямлений контекст [5].

Проведений порівняльний аналіз провідних ШІ-моделей з різними нейронними архітектурами (табл. 1) дозволив узагальнити їхні характеристики. Показник HumanEval Pass@1 у табл. 1 є метрикою, яка використовується для оцінювання якості програмного коду, згенерованого ШІ, зокрема в задачах на кшталт автодоповнення коду чи розв'язання задач програмування. Pass@1 широко застосовується для оцінки моделей програмування, позаяк відображає відсоток задач, у яких перша згенерована відповідь моделі правильно проходить

тести з бенчмарку HumanEval [13]. Якщо модель не генерує правильний код з першої спроби, вона не проходить тест.

Таблиця 1

Порівняльні характеристики провідних ШІ-моделей генерації коду

Модель	Архітектура	Параметри	HumanEval Pass@1	Ключові особливості
GPT-4/ Codex	Декодувальний трансформер	понад 12 млрд [3]	84.9%	Мультимодальність, контекстне навчання
CodeT5	Кодувальник-декодувальник	770 млн [2]	35.0%	Identifier Tagging, Bimodal Generation
InCoder	Причинно-наслідкова мовна модель	6.7 млрд [5]	~15%	Заповнення коду, двонапрямений контекст
CodeBERT	BERT-подібна	125 млн [4]	~12%	Розуміння коду, виявлення клонів

Аналіз показав, що наразі GPT-4/Codex є найточнішою та найпотужнішою ШІ-моделлю, яка підходить для складної генерації коду. CodeT5 тримає баланс між розміром і продуктивністю, а тому добре підходить для завдань трансформації коду. InCoder спеціалізується на заповненні шаблонів у коді, але має нижчу точність. CodeBERT більше підходить для аналітики коду, ніж для генерації.

4. Інтеграція ШІ з кросплатформними фреймворками

Сучасні фреймворки кросплатформної розробки зазнають фундаментальних змін під впливом технологій ШІ. Flutter, React Native та .NET MAUI впроваджують ШІ-керовані можливості для автоматизації процесів адаптації коду до різних платформ та оптимізації продуктивності застосунків (рис. 2).

Екосистема Flutter активно інтегрує ШІ-інструменти через плагіни для Android Studio та VS Code, зокрема FlutterGPT – інструмент для генерації віджетів на основі природних мов. Його система аналізує семантику UI-елементів та автоматично генерує відповідний Dart-код з дотриманням архітектурних патернів Flutter.

Платформа React Native використовує можливості екосистеми JavaScript для інтеграції з ШІ-сервісами. Так, GitHub Copilot при роботі з React Native компонентами досягає 91.5% успішних генерацій валідного коду завдяки великій кількості прикладів коду JavaScript у тренувальних даних [14].

.NET MAUI має інтеграцію з Visual Studio IntelliCode та GitHub Copilot для аналізу коду, специфічного для платформи, та пропонує оптимальні рішення для різних цільових платформ, зокрема автоматичну генерацію модулів візуалізації та обробників елементів керування.

Також зростає значення федеративного навчання (federative learning) для оптимізації кросплатформних рішень, що дозволяє тренувати моделі машинного навчання на пристроях користувачів без порушення їх приватності. Системи на кшталт FedKit дозволяють пристроям навчати моделі локально та надсилати

лише знеособлені оновлення, які покращують глобальну модель без передачі особистих даних. Це дозволяє покращувати моделі без передачі приватних даних, що важливо для забезпечення конфіденційності користувачів.

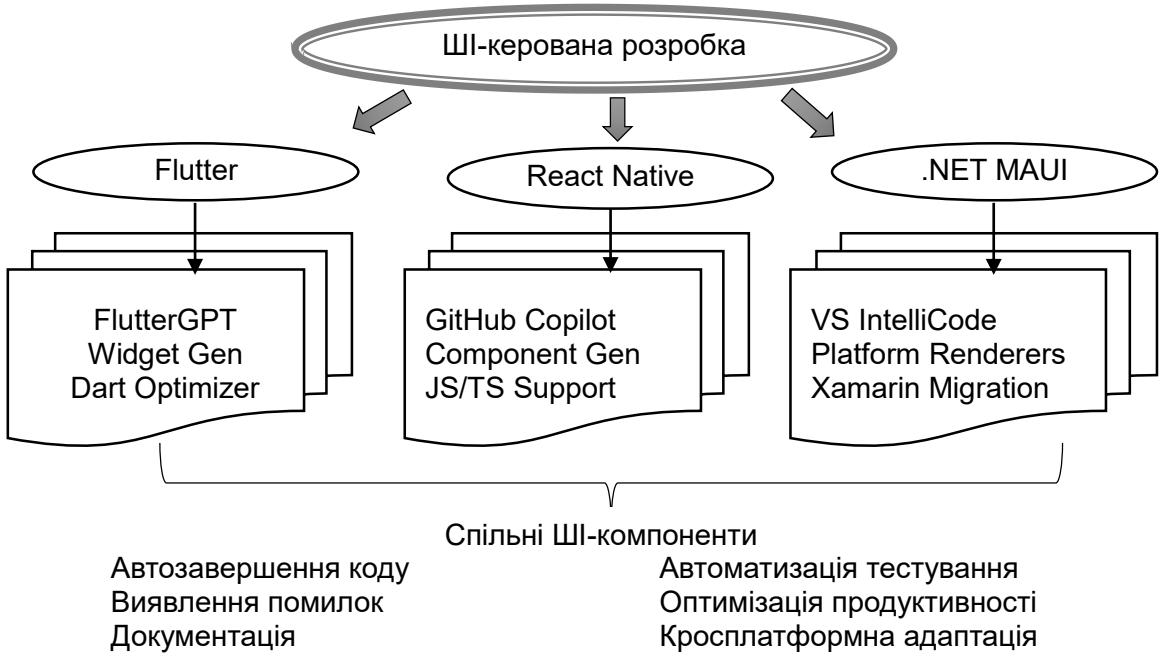

Рис. 1. Інтеграція ШІ з кросплатформними фреймворками

Інтеграція ШІ в кросплатформну розробку відкриває нові можливості для швидкої та ефективної розробки, але також ставить нові вимоги до безпеки даних, що особливо важливо в контексті федеративного навчання, яке забезпечує конфіденційність користувацьких даних.

5. Системи автоматичної генерації та оптимізації коду

Автоматична генерація програмного коду засобами ШІ базується на складних алгоритмах обробки природної мови та глибокому розумінні семантики програмування. Сучасні системи використовують кілька базових підходів: генерацію на основі шаблонів, нейромережевий синтез програм та генерацію з підтримкою пошуку. Узагальнений алгоритм показаний на рис. 2.

Генерація на основі шаблонів залишається ефективним підходом для стандартизованих завдань. Системи, як-от OpenRewrite, використовують трансформації на основі AST (Abstract Syntax Trees, абстрактні синтаксичні дерева) для автоматичного рефакторингу коду та міграції між версіями фреймворків. Алгоритми зіставлення зразків дозволяють ідентифікувати типові конструкції коду та замінювати їх оптимізованими варіантами.

Нейромережевий синтез програм є найбільш перспективним напрямком розвитку. Такі системи використовують наскрізне навчання для генерації програм безпосередньо з природномовних специфікацій. CodeT5 демонструє гарні результати у завданнях узагальнення коду та трансляції, досягаючи значення оцінки BLEU 24.41 для Python-Java трансляції [2].

Генерація з підтримкою пошуку поєднує переваги попередньо навчених моделей з можливістю доступу до зовнішніх баз знань. Система CodeRetriever

індексує мільйони фрагментів коду з публічних репозиторіїв та використовує семантичний пошук для знаходження релевантних прикладів при генерації нового коду.

Рис. 2. Узагальнений алгоритм ШІ-керованої генерації коду

Автоматичний рефакторинг коду досяг нового рівня завдяки системам на кшталт CodeScene ACE з dual-AI архітектурою. Система поєднує generation model для створення рефакторованого коду та модель перевірки фактів для верифікації коректності змін [15].

6. Інтелектуальні середовища розробки та ШІ-асистенти

Сучасні інтегровані середовища розробки (Integrated Development Environment, IDE) трансформуються в інтелектуальні платформи, які використовують технології машинного навчання для підвищення продуктивності програмістів. Лідерами цього сегменту є Visual Studio Code з GitHub Copilot, JetBrains IDE з AI Assistant та нові ШІ-орієнтовані редактори Cursor і Windsurf. У табл. 2. зведені результати порівняльного аналізу перелічених інтелектуальних середовищ розробки.

GitHub Copilot інтегрується з понад 50 мовами програмування та демонструє високі показники ефективності: 55% підвищення швидкості кодування та 91.5% успішних генерацій валідного коду [1]. Система використовує контекстно-усвідомлене автодоповнення, яке аналізує не лише поточний файл, а й пов'язані модулі та залежності проєкту.

Таблиця 2

Порівняльний аналіз інтелектуальних IDE

IDE/Platform	ШІ-компоненти	Основні переваги	Обмеження
VS Code + Copilot	GPT-4o, Claude 3.5-4	Широка підтримка мов, безкоштовний базовий функціонал	Залежність від хмарних сервісів
JetBrains + AI	Mellum, Junie	Локальні моделі, глибока інтеграція	Вартість ліцензування
Cursor	Claude 3.5-4, Agent Mode	ШІ-орієнтований дизайн, прозорий контроль	Обмежена кількість безкоштовних запитів
Windsurf	Cascade Agent	Агентний підхід, AI Flows	Новизна платформи

JetBrains AI Assistant 2025.1 використовує LLM-модель Mellum для автодоповнення коду та ШІ-агента Junie для складних завдань програмування. Інтеграція з локальними моделями через Ollama та LM Studio забезпечує приватність даних та контроль над ними, а нові безкоштовні тарифні опції підвищують доступність цих технологій для широкої аудиторії [16].

Cursor IDE позиціонується як перше повноцінне ШІ-орієнтоване середовище розробки з унікальними можливостями Agent Mode та Composer для багатофайлових редагувань. Система забезпечує прозоре управління змінами внесеними ШІ з можливістю їх детального перегляду та контролю всіх модифікацій коду.

В свою чергу, Windsurf від Codeium впроваджує концепцію "агентного" середовища розробки з Cascade Agent – системою для виконання складних мультифайлових редагувань. Система AI Flows синхронізує роботу між розробником та ШІ, оптимізуючи продуктивність команди.

Інтеграція ШІ в середовища розробки значно полегшує процес програмування, автоматизуючи рутинні завдання та підвищуючи продуктивність розробників. Кожна з платформ має свої переваги та обмеження, що дозволяє вибирати інструмент залежно від конкретних вимог до розробки та приватності даних.

7. ШІ-керовані системи тестування та забезпечення якості

Автоматизація тестування програмного забезпечення з використанням технологій ШІ є одним з найперспективніших напрямків практичного застосування машинного навчання в програмній інженерії [17]. Сучасні системи забезпечують автоматичну генерацію тестових випадків, інтелектуальне виявлення дефектів та тести, які адаптуються до змін у програмному коді [18].

Зокрема, система TestRigor використовує природну мову для опису тестових сценаріїв, перетворюючи англомовні описи на виконувані тести. Це дозволяє автоматично генерувати релевантні тестові випадки на основі аналізу поведінки користувачів у реальних умовах, а також використовувати самовідновлювані (self-healing) тести, які адаптуються до змін у користувацькому інтерфейсі. Такий підхід значно знижує витрати на підтримку тестової інфраструктури.

Система Code Intelligence реалізує підхід "білої скриньки" з використанням

самонавчання ШІ для автономного виявлення помилок та вразливостей. Вона поєднує символічне виконання та техніки розмивання (fuzzing), посилені машинним навчанням для глибокого аналізу прихованих дефектів без хибних спрацьовувань.

Дослідження з оцінювання впливу ШІ на продуктивність тестування, показали значні покращення. Так, контрольований експеримент з GitHub Copilot показав підвищення швидкості кодування на 55,8%, зменшення часу на виявлення багів на 54%, збільшення ранньої ідентифікації дефектів на 67% та 49% зменшення кількості проблем після розгортання [12]. Ці результати підтверджують ефективність використання машинного навчання в процесах забезпечення якості програмного забезпечення.

Інноваційний підхід до автоматизації тестування пропонує також BrowserStack Low-Code Automation, що включає self-healing тести з автоматичним виправленням, автоматичну генерацію тестових даних та інтелектуальне очікування завантаження елементів інтерфейсу. Інтеграція з конвеєрами CI/CD дозволяє здійснювати неперервну верифікацію якості коду на всіх етапах розробки.

Окремим напрямком є мутаційне тестування, яке через використання ШІ еволюціонує від простих синтаксичних змін до інтелектуальної генерації семантично значущих варіантів програм. Цей процес використовує генетичні алгоритми та навчання з підкріпленням для оптимізації генерації мутантів, що дозволяє максимізувати покриття тестами при мінімізації обчислювальних витрат.

Загалом інтеграція ШІ в автоматизацію тестування програмного забезпечення значно підвищує ефективність виявлення дефектів, знижує витрати на підтримку тестової інфраструктури та оптимізує процеси забезпечення якості. Використання таких підходів, як самовідновлювані тести, автоматична генерація тестових випадків та мутаційне тестування, дозволяє покращити якість програмного забезпечення на всіх етапах його розробки.

8. Методи оцінювання якості ШІ-генерованого коду

Оцінка якості автоматично згенерованого програмного коду є багатогранною задачею, що охоплює аналіз таких аспектів, як функціональна коректність, структурна якість, безпека та супроводжуваність коду. Для цієї мети застосовуються сучасні метрики та методології, що дозволяють забезпечити комплексну оцінку ефективності систем ШІ для генерації коду..

Функціональна коректність оцінюється за допомогою метрики Pass@k, яка визначає ймовірність генерації принаймні одного коректного рішення серед k спроб [13]:

$$\text{Pass@k} = 1 - \frac{C(n-c,k)}{C(n,k)}, \quad (1)$$

де n – загальна кількість генерованих рішень; c – кількість коректних рішень;

$C(n, k)$ – загальна кількість способів вибору k зразків з n;

$C(n\text{-}c, k)$ – загальна кількість способів вибрати k зразків з $n\text{-}c$ невірних зразків.

Загальновизнаним стандартом галузі стає бенчмарк HumanEval, який складається з 164 ретельно підібраних завдань програмування різної складності.

Структурну якість коду вимірюють за допомогою метрики CodeBLEU, яка поєднує традиційну оцінку BLEU з подібністю на основі абстрактних синтаксичних дерев (AST) та зіставленням потоків даних. Метрика враховує не лише лексичну подібність, а й коректність структури синтаксичного дерева та семантичну еквівалентність потоків даних [19].

Безпеку ШІ-генерованого коду оцінюють через відповідність до стандартів CWE (Common Weakness Enumeration) – списку поширених слабких місць в програмному забезпеченні, а також через інтеграцію з інструментами SAST (Static Application Security Testing) для статичного тестування безпеки застосунків.

Супроводжуваність коду оцінюють за допомогою метрик, які вимірюють читабельність, модулярність та наявність документації. Системи аналізу коду, як-от SonarQube з розширеннями на базі ШІ, автоматично оцінюють технічний борг і надають рекомендації щодо покращення архітектури програмного забезпечення.

Для всебічної оцінки якості автоматично згенерованого коду необхідно використовувати комплекс метрик, які охоплюють функціональну коректність, структурну якість, безпеку та супроводжуваність. Застосування цих методів дозволяє отримати точну оцінку ефективності ШІ-систем у контексті реальних розробницьких процесів.

9. Виклики безпеки та надійності

Впровадження ШІ-керованих систем розробки програмного забезпечення ставить перед галуззю нові виклики у сфері безпеки та надійності. Оскільки ШІ-генерований код часто автоматизує значну частину процесу написання програм, він може містити приховані вразливості, які є неочевидними навіть для досвідчених розробників. Це створює додаткові ризики для інформаційної безпеки, оскільки такі вразливості можуть бути не поміченими на етапах тестування та верифікації, але виявляються лише на етапі продакшн-запуску.

Одним з ключових аспектів є феномен "naive bugs" (наївних помилок), які часто виникають у результаті роботи ШІ-моделей [6]. Ці помилки можуть бути настільки простими, що досвідчені програмісти могли б легко їх виявити і виправити. Проте для автоматичних ШІ-систем вони можуть бути непомітними, що призводить до генерації коду з серйозними уразливостями. Такі помилки можуть призвести до критичних проблем у продакшн-середовищах, де надійність і безпека є першочерговими вимогами.

Крім того, особливу небезпеку для безпеки програмного забезпечення представляє безпека ланцюга поставок. ШІ-моделі часто навчаються на відкритих публічних репозиторіях, які можуть містити не лише корисний код, а й приховані бекдори або зловмисний код. Таке навчання може непомітно включити в генерацію шкідливі патерни, які з часом автоматично поширюються через різноманітні програмні проєкти. Це створює нові ризики для масштабованих атак на програмну інфраструктуру, де зловмисники можуть не лише використовувати вразливості, а й маніпулювати процесами розробки та доставки програмного забезпечення.

Дослідження Meta CyberSecEval 2 виявило, що приблизно 30% ШІ-генерованого коду містить потенційні вразливості [20]. Це підтверджує важливість впровадження додаткових механізмів верифікації безпеки на всіх етапах життєвого циклу розробки програмного забезпечення, особливо при

використанні ШІ для автоматичного написання коду. Без належних заходів контролю й аналізу, ризик використання шкідливих або ненадійних фрагментів коду значно зростає.

Діаграма розподілу типів кіберризиків у великих мовних моделях представлена на рис. 3.

Рис. 3. Розподіл типів кіберризиків у LLM

Крім технічних аспектів, проблеми оцінювання безпеки ШІ-генерованого коду також пов'язані з обмеженістю наявних бенчмарків. Традиційні метрики й тести, зокрема такі, як HumanEval, часто не враховують специфіку реального застосування програмного забезпечення, тому вони не завжди можуть бути точними індикаторами якості або безпеки коду. Крім того, є проблема забруднення тренувальних даних – якщо моделі навчалися на коді, який містить вразливості чи зловмисні патерни, це може призвести до автоматичного виведення таких вразливостей у новому коді.

Ще однією проблемою є довгострокова супроводжуваність коду, оскільки метрики, що оцінюють це питання, часто є неповними або не можуть адекватно відобразити всі аспекти супроводу. Вимірювання таких параметрів, як читабельність, підтримка та змінюваність коду, залишається складною задачею, яка потребує нових підходів для об'єктивної оцінки.

Зростання популярності ШІ-керованих систем розробки програмного забезпечення створює нові виклики в сфері безпеки та надійності програм. Важливою проблемою є можливість виникнення прихованих вразливостей у згенерованому коді, а також зростаючий ризик поширення зловмисного коду через відкриті репозиторії. Для мінімізації цих ризиків необхідно розвивати додаткові методи верифікації безпеки, удосконалювати бенчмарки і враховувати специфіку реальних умов застосування програмного забезпечення. Тому безпека та надійність ШІ-генерованого коду вимагають комплексного підходу та впровадження нових механізмів захисту на кожному етапі життєвого циклу програмного продукту.

10. Результати та обговорення

Проведене дослідження дозволило систематизувати сучасний стан ШІ-керованих систем розробки кросплатформних застосунків та виявити ключові

тенденції розвитку галузі. Сучасні досягнення у сфері генерації коду на основі штучного інтелекту продемонстрували значний прогрес в аспектах функціональної коректності. Наприклад, модель GPT-4/Codex досягла точності 67% на бенчмарку HumanEval, що є важливим проривом порівняно з попередніми поколіннями моделей. Це свідчить про прогрес в здатності ШІ до генерації коду, що відповідає вимогам якості, що в свою чергу відкриває нові можливості для інтеграції таких технологій в розробку програмного забезпечення.

Аналіз інтеграції ШІ-інструментів у кросплатформні фреймворки показав, що Flutter забезпечує найкращий баланс між продуктивністю та зручністю розробки завдяки власному механізму рендерингу та інтеграції з потужними ШІ-асистентами. Таке поєднання підвищує ефективність розробників і дозволяє оптимізувати час, необхідний для створення кросплатформних застосунків. React Native надає найширшу екосистему ШІ-інструментів завдяки популярності JavaScript, що дає можливість інтегрувати різноманітні зовнішні сервіси та моделі. Водночас .NET MAUI забезпечує глибоку інтеграцію з екосистемою Microsoft, пропонуючи ідеальні рішення для корпоративних застосунків, де важлива стабільність та безпека.

Дослідження ефективності впровадження ШІ-інструментів у робочі процеси розробників показало вражаючі результати. Зокрема, контрольований експеримент з GitHub Copilot показав підвищення швидкості кодування на 55,8%, зменшення часу виявлення помилок на 54%, та збільшення ранньої ідентифікації дефектів на 67% [12]. Ці цифри свідчать про трансформаційний потенціал технологій ШІ, які не тільки підвищують ефективність програмістів, а й значно покращують якість програмного забезпечення на етапах розробки [21].

Проте, поряд з потенціалом, використання ШІ в розробці програмного забезпечення створює нові виклики в галузі безпеки. Дослідження виявило, що 30% автоматично згенерованого коду може містити потенційні вразливості. Це підкреслює важливість інтеграції додаткових механізмів для забезпечення безпеки коду, таких як верифікація на етапах генерування та автоматичне тестування на наявність загроз. У цьому контексті важливо розвивати нові методології навчання, які враховують аспекти безпеки та відповідність стандартам на всіх етапах розробки.

Порівняльний аналіз інтелектуальних середовищ розробки (IDE) виявив значну диверсифікацію підходів. Традиційні редактори VS Code і JetBrains інтегрують ШІ в свої функціональні можливості, пропонуючи основні інструменти для автодоповнення та покращення якості коду. Водночас нові, повністю ШІ-орієнтовані платформи Cursor і Windsurf пропонують автономне виконання більш складних завдань, включаючи багатофайлові редагування та інтеграцію з агентовими системами. Останнє дає змогу створювати більш складні, інтегровані рішення для командної розробки.

Тренд до демократизації доступу до ШІ-технологій набирає обертів, зокрема завдяки безплатним базовим функціям інструментів GitHub Copilot у VS Code або JetBrains AI Assistant. Це значно прискорює впровадження ШІ-систем в різні рівні програмної інженерії, робить їх доступними для широкого загалу розробників.

Майбутні тенденції в розвитку ШІ-керованих систем стосуються еволюції до агентних ШІ-систем, здатних автономно виконувати складні проєктні завдання. Крім того, планується інтеграція мультимодальних моделей, які поєднують код з візуальними елементами, що сприятиме більш інтерактивній та інтуїтивно

зрозумілій розробці програм. Перспективними також є новітні протоколи агентної взаємодії, такі як MCP від Anthropic та A2A від Google, які мають потенціал значно вдосконалити комунікацію між різними агентами та системами, що, в свою чергу, дозволить досягти більшої інтеграції та ефективності в розробці програмного забезпечення.

Висновки

Штучний інтелект відіграє важливу роль у розвитку сучасних інструментів для розробки програмного забезпечення, зокрема кросплатформних застосунків. Прогрес у генерації коду, інтеграція з популярними фреймворками та інтелектуальні середовища розробки демонструють значний потенціал для покращення продуктивності та зручності роботи розробників. Однак, питання безпеки та верифікації ШІ-генерованого коду залишаються важливими для подальшого розвитку цієї технології. Інвестування в дослідження та розвиток інноваційних методів верифікації, а також інтеграція новітніх агентних систем і мультимодальних моделей є важливими напрямками для подальшого вдосконалення програмної інженерії в умовах швидкого розвитку технологій ШІ.

Трансформація ролі розробника від написання коду до оркестування ШІ-екосистем вимагає адаптації освітніх програм та професійних компетенцій. Майбутнє програмної інженерії лежить у симбіотичному партнерстві між людською креативністю та ШІ-автоматизацією, де технології штучного інтелекту звільняють розробників для зосередження на стратегічних та архітектурних рішеннях.

Список літератури

1. AI Trends Report 2024: AI’s Growing Role in Software Development. URL: https://www.docker.com/blog/ai-trends-report-2024/

2. Wang Y., Wang W., Joty S., Hoi St. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 2021. P. 8696-8708. DOI: https://doi.org/10.18653/v1/2021.emnlp-main.685

3. Lu S., Guo D., Ren S., et al. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. arXiv. 2021. Vol. 2102.04664. DOI: https://doi.org/10.48550/arXiv.2102.04664

4. Feng Z., Guo D., Tang D., et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. Findings of the Association for Computational Linguistics: EMNLP 2020. P. 1536-1547. DOI: https://doi.org/10.18653/v1/2020.findings-emnlp.139

5. Fried D., Aghajanyan A., Lin J., et al. InCoder: A Generative Model for Code Infilling and Synthesis. arXiv. 2022. Vol. 2204.05999. DOI: https://doi.org/10.48550/arXiv.2204.05999

6. Negri-Ribalta C., Geraud-Stewart R., Sergeeva A., Lenzini G. A systematic literature review on the impact of AI models on the security of code generation. Frontiers in Big Data. 2024. Vol. 7. Article 1386720. P. 1-20. DOI: https://doi.org/10.3389/fdata.2024.1386720

7. Martinović B., Rozić R. Impact of AI Tools on Software Development Code Quality. Communications in Computer and Information Science. 2024. Vol. 2124. P. 202-216. DOI: https://doi.org/10.1007/978-3-031-62058-4_15

8. Ukrainian Catholic University Machine Learning Lab. Research Portfolio 2020-2025. Lviv: UCU, 2025. URL: https://apps.ucu.edu.ua/en/mllab/

9. Ma L., Wang H., Yang K, et al. An empirical study of AI techniques in mobile applications. Journal of Systems and Software. 2024. Vol. 220. Article 112233. DOI: https://doi.org/10.1016/j.jss.2024.112233

10. Blanco J. Z., Lucrédio D. A holistic approach for cross-platform software development. Journal of Computer Languages. 2021. Vol. 65. Article 101049. DOI: https://doi.org/10.48550/arXiv.2104.14614

11. Alenezi M., Akour M. AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions. Applied Sciences. 2025. Vol. 15, No. 3. Article 1344. DOI: https://doi.org/10.3390/app15031344

12. Peng S., Kalliamvakou E., Cihon P., Demirer M. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv. 2023. Vol. 2302.06590. DOI: https://doi.org/10.48550/arXiv.2302.06590

13. Chen M., Tworek J., Jun H., et al. Evaluating Large Language Models Trained on Code. arXiv. 2021. Vol. 2107.03374. DOI: https://doi.org/10.48550/arXiv.2107.03374

14. Fu Ju., Liang P., Tahir A., et al. Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study. arXiv. 2025. Vol. 2310:02059v4. DOI: https://doi.org/10.48550/arXiv.2310.02059

15. CodeScene ACE: Auto-Refactor Code. Technical Documentation. 2025. URL: https://codescene.io/docs/auto-refactor/index.html#codescene-ace-auto-refactor-code

16. Maltseva A. JetBrains AI Assistant: Smarter, More Capable, and a New Free Tier. 2025. URL: https://blog.jetbrains.com/ai/2025/04/jetbrains-ai-assistant-2025-1/

17. Трофименко О. Г., Дика А. І., Лобода Ю. Г. Аналіз інструментів тестування вебзастосунків. Кібербезпека: освіта, наука, техніка. 2023. № 4(20). С. 62-71. DOI: https://doi.org/10.28925/2663-4023.2023.20.6271

18. Ramchand S., Shaikh S., Alam I. Role of Artificial Intelligence in Software Quality Assurance. Lecture Notes in Networks and Systems. 2022. Vol. 295. P. 89-102. DOI: https://doi.org/10.1007/978-3-030-82196-8_10

19. Ren S., Guo D., Lu S., et al. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. arXiv. 2020. Vol. 2009.10297. DOI: https://doi.org/10.48550/arXiv.2009.10297

20. Bhatt M. et al. CyberSecEval 2: A Wide-Ranging Cybersecurity Evaluation Suite for Large Language Models. arXiv. 2024. Vol. 2404.13161. DOI: https://doi.org/10.48550/arXiv.2404.13161

21. Трофименко, О. Г., Лобода, Ю. Г., Гура, В. І., Дика, А. І., Стрілець, М. І. Інструменти штучного інтелекту для системного аналізу. Вісник Херсонського національного технічного університету. 2024. № 4. С. 349–357. DOI: https://doi.org/10.35546/kntu2078-4481.2024.4.46

References

1. AI Trends Report 2024: AI’s Growing Role in Software Development. https://www.docker.com/blog/ai-trends-report-2024/

2. Wang, Y., Wang, W., Joty, S., Hoi, St. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. Proceedings of the 2021 Conference on Empirical Methods in Natural Language

Processing,2021, 8696-8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

3. Lu, S., Guo, D., Ren, S., et al. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. arXiv, 2021, 2102.04664. https://doi.org/10.48550/arXiv.2102.04664

4. Feng, Z., Guo, D., Tang, D., et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. Findings of the Association for Computational Linguistics: EMNLP 2020, 1536-1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

5. Fried, D., Aghajanyan, A., Lin, J., et al. InCoder: A Generative Model for Code Infilling and Synthesis. arXiv, 2022, 2204.05999. https://doi.org/10.48550/arXiv.2204.05999

6. Negri-Ribalta, C., Geraud-Stewart, R., Sergeeva, A., Lenzini, G. A systematic literature review on the impact of AI models on the security of code generation. Frontiers in Big Data, 2024, 7, 1-20. https://doi.org/10.3389/fdata.2024.1386720

7. Martinović, B., Rozić, R. Impact of AI Tools on Software Development Code Quality. Communications in Computer and Information Science, 2024, 2124, 202-216. DOI: https://doi.org/10.1007/978-3-031-62058-4_15

8. Ukrainian Catholic University Machine Learning Lab. Research Portfolio 2020-2025. Lviv: UCU, 2025. https://apps.ucu.edu.ua/en/mllab/

9. Ma L., Wang H., Yang K, et al. An empirical study of AI techniques in mobile applications. Journal of Systems and Software, 2024, 220, 112233. https://doi.org/10.1016/j.jss.2024.112233

10. Blanco J. Z., Lucrédio D. A holistic approach for cross-platform software development. Journal of Computer Languages. 2021. Vol. 65. Article 101049. https://doi.org/10.48550/arXiv.2104.14614

11. Alenezi, M., Akour, M. AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions. Applied Sciences, 2025, 15(3). https://doi.org/10.3390/app15031344

12. Peng, S., Kalliamvakou, E., Cihon, P., Demirer, M. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv, 2023, 2302.06590. https://doi.org/10.48550/arXiv.2302.06590

13. Chen, M., Tworek, J., Jun, H., et al. Evaluating Large Language Models Trained on Code. arXiv, 2021, 2107.03374. https://doi.org/10.48550/arXiv.2107.03374

14. Fu Ju., Liang P., Tahir A., et al. Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study. arXiv. 2025. Vol. 2310:02059v4. DOI: https://doi.org/10.48550/arXiv.2310.02059

15. CodeScene ACE: Auto-Refactor Code. Technical Documentation. 2025. URL: https://codescene.io/docs/auto-refactor/index.html#codescene-ace-auto-refactor-code

16. Maltseva A. JetBrains AI Assistant: Smarter, More Capable, and a New Free Tier. 2025. URL: https://blog.jetbrains.com/ai/2025/04/jetbrains-ai-assistant-2025-1/

17. Trofymenko, O., Dyka, A., & Loboda, Y. (2023). Analysis of web application testing tools. Electronic Professional Scientific Journal «Cybersecurity: Education, Science, Technique», 4(20), 62–71. https://doi.org/10.28925/2663-4023.2023.20.6271

18. Ramchand S., Shaikh S., Alam I. Role of Artificial Intelligence in Software Quality Assurance. Lecture Notes in Networks and Systems. 2022. Vol. 295.

P. 89-102. DOI: https://doi.org/10.1007/978-3-030-82196-8_10.

19. Ren S., Guo D., Lu S., et al. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. arXiv. 2020. Vol. 2009.10297. DOI: https://doi.org/10.48550/arXiv.2009.10297.

20. Bhatt M. et al. CyberSecEval 2: A Wide-Ranging Cybersecurity Evaluation Suite for Large Language Models. arXiv. 2024. Vol. 2404.13161. DOI: https://doi.org/10.48550/arXiv.2404.13161

21. Trofymenko, O. G., Loboda, Yu. G., Hura, V. I., Dyka, A. I., Strilets, M. I. artificial intelligence tools for systems analysis. Visnyk of Kherson National Technical University. 2024, 4, 349-357. https://doi.org/10.35546/kntu2078-4481.2024.4.46

Надійшла до редакції 15.08.2025, розглянута на редколегії 15.08.2025

AI-driven cross-platform application development systems

The study is devoted to the analysis of the current state and prospects for the development of artificial intelligence (AI) in cross-platform application development systems. The key machine learning technologies used in software engineering automation processes are considered, including automatic code generation systems based on large language models, intelligent development environments, and AI-driven testing methodologies. The architectural solutions of modern cross-platform frameworks and their integration with artificial intelligence technologies are analyzed. The application of transformer models, in particular GPT-4/Codex, Claude, CodeT5, and CodeBERT, in the tasks of understanding and generating software code is studied. The analysis showed that currently GPT-4/Codex is the most accurate and powerful AI model suitable for complex code generation. CodeT5 maintains a balance between size and performance and therefore is well suited for code transformation tasks. InCoder specializes in filling in code patterns but has lower accuracy. CodeBERT is more suitable for code analytics than for generation. Methods for assessing the quality of AI-generated code are considered, including metrics of functional correctness and structural quality. Challenges to the security and reliability of automatically generated software code are highlighted, including vulnerability issues and the need for additional verification. An analysis of the effectiveness of various approaches to cross-platform development using artificial intelligence tools is presented. The scientific novelty of the work lies in the comprehensive analysis of the interaction of artificial intelligence technologies with cross-platform development frameworks, the systematization of modern approaches to AI-driven code generation, and the study of specific challenges in integrating machine learning into multi-platform software development processes. The results of the study show the significant potential of integrating artificial intelligence technologies to increase developer productivity and improve software quality.

Keywords: cross-platform development, artificial intelligence, code generation, large language models, software engineering, testing, integrated development environment.

Відомості про авторів:

Манаков Сергій Юрійович – доцент, канд. техн. наук, доцент кафедри інженерії програмного забезпечення Національного університету «Одеська

юридична академія», Одеса, Україна, ел. пошта: manakov_serhii@onua.edu.ua, ORCID 0000-0001-5930-4592.

Трофименко Олена Григорівна – доцент, канд. техн. наук, доцент кафедри інженерії програмного забезпечення Національного університету «Одеська юридична академія», Одеса, Україна, ел. пошта: trofymenko@onua.edu.ua, ORCID 0000-0001-7626-0886.

Чикунов Павло Олександрович – доцент, канд. техн. наук, доцент кафедри інженерії програмного забезпечення Національного університету «Одеська юридична академія», Одеса, Україна, ел. пошта: pavel@onua.edu.ua, ORCID 0000-0003-4959-774412:27

Гура Володимир Ігорович – доцент, канд. техн. наук, доцент кафедри інженерії програмного забезпечення Національного університету «Одеська юридична академія», Одеса, Україна, ел. пошта: ihu@ukr.net, ORCID 0009-0001-2166-5410

About the authors:

Manakov Serhii – associate professor, candidate of technical sciences, associate professor of the department of Software Engineering of the National University “Odesa Law Academy”, Odesa, Ukraine, e-mail: manakov_serhii@onua.edu.ua, ORCID: 0000-0001-5930-4592.

Trofymenko Olena – associate professor, candidate of technical sciences, associate professor of the department of Software Engineering of the National University “Odesa Law Academy”, Odesa, Ukraine, e-mail: trofymenko@onua.edu.ua, ORCID: 0000-0001-7626-0886.

Chykunov Pavlo – associate professor, candidate of technical sciences, associate professor of the department of Software Engineering of the National University “Odesa Law Academy”, Odesa, Ukraine, e-mail: pavel@onua.edu.ua, ORCID: 0000-0003-4959-7744

Hura Volodymyr – associate professor, candidate of technical sciences, associate professor of the department of Software Engineering of the National University “Odesa Law Academy”, Odesa, Ukraine, e-mail: ihu@ukr.net, ORCID: 0009-0001-2166-5410