УДК 004.891:004.415.2
DOI DOI DOI https://doi.org/10.32782/2663-5941/2026.2.1/23

Купін А.І.
https://orcid.org/0000-0001-7569-1721
Криворізький національний університет

Духов Д.Ю.
https://orcid.org/0009-0005-8267-685X
Криворізький національний університет

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АВТОМАТИЗОВАНОГО ГЕНЕРУВАННЯ ПРОГРАМНОГО КОДУ ЗАСОБАМИ ШТУЧНОГО ІНТЕЛЕКТУ

У статті здійснено комплексний аналіз сучасних трендів автоматизованого генерування програмного коду із застосуванням засобів штучного інтелекту, зокрема великих мовних моделей (LLM), що базуються на методах глибокого навчання та обробки природної мови. Розглянуто еволюцію підходів до генерації коду – від традиційних інструментів автоматизації та low-code/no-code платформ до генеративних ШІ-систем, здатних створювати програмний код «з нуля» на основі текстових запитів розробника. Показано, що такі системи забезпечують суттєве підвищення продуктивності та ефективності роботи розробників, зменшення кількості помилок і скорочення циклів розробки програмного забезпечення, що підтверджується результатами сучасних емпіричних досліджень.

Значну увагу приділено аналізу спеціалізованих бенчмарків, які використовуються для оцінювання якості та ефективності ШІ-моделей у завданнях програмування. Детально охарактеризовано такі бенчмарки, як HumanEval, MBPP, SWE-Bench, Aider Polyglot, LiveCodeBench, APPS, MTPB, DS-1000 та WebDev Arena, визначено їхні метрики, переваги та обмеження. Показано, що еволюція бенчмарків від перевірки функціональної коректності ізольованих фрагментів коду до оцінювання агентних можливостей, багатомовного редагування, виправлення реальних помилок і багатоетапної взаємодії відображає зростання вимог до інтелектуальних можливостей ШІ у реальних сценаріях розробки програмного забезпечення.

У роботі проаналізовано сучасний стан комерційних і відкритих мовних моделей, зокрема GPT-4.x, Claude 3.x, Gemini 2.x, Code Llama та DeepSeek-Coder, та показано тенденцію стирання межі між універсальними й спеціалізованими моделями для кодування. Окремо підкреслено роль розширених контекстних вікон, агентних фреймворків і мультимодальних можливостей у формуванні нового покоління ШІ-асистентів, здатних виконувати складні завдання з аналізу, налагодження, рефакторингу та планування програмних систем.

Зроблено висновок, що автоматизоване генерування коду за допомогою штучного інтелекту переходить від інструментальної підтримки до повноцінного партнерства у процесі розробки програмного забезпечення, формуючи підґрунтя для появи більш автономних і інтегрованих ШІ-рішень у майбутньому.

Ключові слова: *генеративний штучний інтелект, аналіз трендів, генерація програмного коду, дослідження бейнчмарків, прогнозування розвитку технології.*

Постановка проблеми. Генерація коду за допомогою генеративного штучного інтелекту – це автоматизований процес створення програмного коду на основі алгоритмів ШІ, які навчаються на великих масивах наявного вихідного коду. Такі алгоритми, що становлять основу великих мовних моделей (LLM) і технологій обробки природної мови (NLP), застосовують методи глибокого навчання та масштабні нейронні мережі для аналізу різноманітних наборів програмного коду з відкритих репозиторіїв. У межах цього підходу генеративний ШІ забезпечує автоматизацію написання програм, інтерпретуючи запити, сформульовані природною мовою, та використовуючи здобуті під час навчання знання для розпізнавання програмних шаблонів, стилістики й логіки.

На відміну від класичних засобів автоматизації коду, що базуються переважно на статичному ана-

лізі та заздалегідь визначених шаблонах, генеративний ШІ характеризується більшою гнучкістю та здатністю адаптуватися до різних контекстів. Він також відрізняється від low-code і no-code платформ, оскільки не спирається на готові шаблони чи бібліотеки компонентів, а формує програмний код безпосередньо з нуля відповідно до мовних інструкцій розробника. Якщо low-code та no-code рішення здебільшого орієнтовані на користувачів без глибоких технічних знань і бізнес-аудиторію, то інструменти генерації коду на основі ШІ є більш універсальними та придатними як для професійних програмістів, так і для ширшого кола користувачів [1].

Використання генеративного ШІ для генерації коду надає розробникам та організаціям ряд значних переваг. Однією з ключових переваг є підвищення продуктивності та ефективності розробників. Автоматично генеруючи код на основі заданих специфікацій, системи ШІ можуть значно прискорити процес розробки, зменшити кількість помилок та скоротити цикли випуску програмного забезпечення. Згідно з дослідженнями, 88% розробників повідомили про підвищення продуктивності під час використання таких інструментів, як GitHub Copilot. Генеративний ШІ також може значно підвищити загальну ефективність роботи та знизити витрати. Звільняючи розробників від рутинних завдань, генеративний ШІ дозволяє їм зосередитися на більш складних та творчих аспектах розробки.

Генеративний ШІ ефективно автоматизує рутинні та повторювані завдання кодування. Такі інструменти, як GitHub Copilot, здатні генерувати шаблонний код та фрагменти коду для стандартних задач, звільняючи розробників від необхідності писати їх вручну. Це не лише економить час, але й зменшує ризик помилок, пов'язаних з монотонною роботою. Автоматизація цих завдань також може призвести до підвищення задоволеності розробників, оскільки вони можуть присвячувати більше часу вирішенню складних та цікавих проблем [2].

Аналіз останніх досліджень і публікацій. Виконано дослідження бенчмарків штучного що було спрямовано на розроблення, аналіз і вдосконалення стандартизованих підходів до оцінювання якості, ефективності та надійності інтелектуальних систем. Бенчмарки ШІ виконують функцію об’єктивного інструмента порівняльного аналізу різних моделей, алгоритмів і архітектур за однакових експериментальних умов.

HumanEval: функціональна коректність та метрика pass@k. Розроблений компанією OpenAI, HumanEval є одним із найбільш відомих бенчмарків для оцінювання здатності моделей до генерації програмного коду. Він охоплює 164 вручну підготовлені задачі, кожна з яких містить сигнатуру функції, документаційний рядок із описом очікуваної поведінки та набір модульних тестів для перевірки правильності реалізації. На основі сигнатури та опису модель генерує тіло функції, після чого отриманий код перевіряється за допомогою наперед визначених тестів з метою оцінки функціональної коректності.

Метрика pass@k визначає ймовірність того, що хоча б одне з k найкращих рішень, згенерованих моделлю, успішно пройде всі тестові випадки. Найчастіше використовують значення pass@1, pass@10 і pass@100, де pass@1 є найбільш вимогливою метрикою, оскільки оцінює правильність першої спроби. Такий підхід безпосередньо вимірює функціональну коректність і ґрунтується на модульному тестуванні, яке є стандартною практикою у розробці програмного забезпечення. HumanEval забезпечує уніфіковану основу для порівняння різних моделей, однак його обмеженням є те, що задачі представлені у вигляді окремих функцій, а не повноцінних програм, що знижує практичну застосовність у реальних умовах [3].

MBPP (Mostly Basic Python Problems): основи програмування. Цей бенчмарк містить близько 1000 задач з програмування мовою Python, зібраних за допомогою краудсорсингу та орієнтованих на початківців. Завдання охоплюють базові концепції програмування та функціональні можливості стандартної бібліотеки Python. Кожна задача включає текстовий опис, приклад реалізації та три автоматизовані тестові випадки для перевірки правильності розв’язання. Оцінювання моделей здійснюється на основі їх здатності генерувати коректний Python-код.

MBPP зосереджується на перевірці фундаментальних навичок роботи з Python, що робить його ефективним індикатором базового рівня володіння програмуванням. Водночас, подібно до HumanEval, цей бенчмарк не повною мірою відображає складні та багатокомпонентні сценарії розробки програмного забезпечення, характерні для реальних умов [4].

SWE-Bench: виправлення реальних помилок та агентне кодування. На відміну від HumanEval, бенчмарк SWE-Bench, представлений OpenAI, зосереджується на корекції реальних програмних помилок, отриманих з відкритих репозиторіїв, що робить його надзвичайно реалістичним інструментом для оцінювання здатності великих мовних

моделей до кодування. Моделям надається контекст, аналогічний тому, з яким стикається розробник, зокрема опис проблеми з GitHub issues, фрагменти коду з дефектами та очікувані зміни. Завдання полягає у створенні патча, який усуває помилку, не породжуючи нових.

Оцінювання здійснюється за такими критеріями, як коректність запропонованого патча, його безшовна інтеграція в наявну кодову базу та ефективність внесених змін з погляду оптимальності й відповідності кращим практикам програмування. Результати зазвичай подаються у вигляді відсотка успішно вирішених задач. Завдяки використанню реальних кейсів SWE-Bench має високу практичну цінність, перевіряє здатність моделей до аналізу та модифікації вже існуючого коду, що є критично важливою навичкою, а також стимулює розвиток можливостей налагодження. Водночас цей бенчмарк вимагає глибокого контекстного розуміння цілісних програмних проєктів, з чим окремі LLM досі мають труднощі, а також характеризується підвищеною складністю автоматизації процесу оцінювання [5].

Aider Polyglot: Багатомовне редагування коду та автономне вирішення проблем. Цей бенчмарк оцінює, наскільки добре ШІ може впоратися з 225 найскладнішими завданнями з кодування з Exercism кількома мовами (C++, Go, Java, JavaScript, Python та Rust). Зосереджується на здатності ШІ писати новий код, який безшовно інтегрується в існуючі кодові бази, та застосовувати зміни до вихідних файлів без втручання людини. Він перевіряє автономне вирішення проблем за допомогою ітеративного підходу: написання коду, перевірка, виправлення помилок та повторення до повного вирішення. Багатомовна підтримка, реалістичність сценаріїв (редагування/інтеграція коду) та оцінка автономного вирішення проблем. Вважається кращим показником здатності ШІ-агентів до вирішення проблем, ніж SWE-Bench, завдяки його багатомовності та фокусу на різноманітних проектах. Може бути обчислювально інтенсивним через ітеративний характер[6].

LiveCodeBench: комплексна оцінка без забруднення даних. LiveCodeBench є всебічним бенчмарком для оцінювання моделей без ризику витоку навчальних даних, який постійно поповнюється новими задачами із платформ змагального програмування, зокрема LeetCode, AtCoder та CodeForces. Завдання відбираються з актуальних змагань, що гарантує їхню сучасність та унеможливлює забруднення даних, оскільки моделі перевіряються на задачах, опублікованих після дати завершення їх навчання.

У межах цього бенчмарку LLM оцінюються за чотирма сценаріями: генерація програмного коду на основі природномовного опису, самовідновлення через виправлення некоректного коду, виконання коду на заданих вхідних даних та прогнозування результатів тестування для конкретних входів. Основною метрикою зазвичай є pass@1. LiveCodeBench усуває проблему забруднення даних, забезпечує комплексне оцінювання різних аспектів навичок програмування та охоплює збалансований спектр рівнів складності. Водночас, як відносно новий бенчмарк, він має менш розвинену історичну базу результатів для всіх моделей порівняно з такими інструментами, як HumanEval або MBPP [7].

APPS (Automated Programming Progress Standard): Змагальне програмування. Містить 10 000 задач, що варіюються від початкового до університетського рівня змагань, вимірюючи здатність до кодування та вирішення проблем. Задачі формулюються вільною природною мовою, а рішення оцінюються на коректність за допомогою великого банку тестових випадків. Імітує оцінку програмістів-людей у змагальних умовах, охоплює широкий діапазон складності та оцінює навички вирішення проблем, що виходять за межі базової генерації коду. Висока складність може призвести до низьких показників проходження для багатьох моделей.

MTPB (Multi-Turn Programming Benchmark): Багатоетапний синтез програм. Новий бенчмарк, розроблений для вимірювання здатності моделей до багатоетапного синтезу програм, де одна програма розкладається на кілька підказок, що визначають підзадачі. Складається зі 115 різноманітних задач, що вимагають багатоетапних описів природною мовою. Моделі синтезують програму в кілька етапів з взаємодією користувача. Продуктивність вимірюється показником проходження на експертно написаних тестових випадках.Вирішує реальний сценарій ітеративної розробки та вдосконалення програм через розмову, демонструючи переваги багатоетапної взаємодії над одноразовими підказками. Новий бенчмарк, тому широке впровадження та порівняльні дані можуть бути менш обширними.

DS-1000: Задачі кодування, специфічні для науки про дані. Цей бенчмарк оцінює продуктивність у задачах науки про дані, зокрема зосереджуючись на завершенні коду в цій галузі. Спеціалізація в науці про дані робить його актуальним

для оцінки моделей у цій галузі, що швидко розвивається. Вузька спрямованість порівняно з загальними бенчмарками генерації коду.

WebDev Arena. Це платформа для оцінювання моделей у процесі створення веб-додатків у режимі реального часу, де користувачі обирають кращий результат шляхом голосування. Підхід ґрунтується на порівнянні двох веб-додатків, згенерованих різними моделями для однієї й тієї самої задачі, після чого користувачі голосують за більш якісний варіант, що дозволяє формувати рейтинги моделей відповідно до їхніх результатів.

Такий формат забезпечує оцінювання моделей у наближених до реальності умовах розробки веб-застосунків, беручи до уваги аспекти UI/UX, рівень інтерактивності та функціональну повноту. Водночас результати значною мірою залежать від суб'єктивних уподобань користувачів, а сама платформа спеціалізується переважно на завданнях веб-розробки.

Постановка завдання. У сучасних дослідженнях штучного інтелекту значну увагу приділяють оцінюванню спроможності мовних і мультимодальних моделей до автоматичного генерування програмного коду. Для цього використовуються спеціалізовані бенчмарки, які забезпечують стандартизовані умови тестування та порівняльного аналізу різних моделей.

Еволюція бенчмарків від HumanEval і MBPP до SWE-Bench, Aider Polyglot та LiveCodeBench засвідчує глибоку трансформацію підходів до визначення «інтелекту коду» для великих мовних моделей. Відбувається перехід від простої генерації синтаксично та функціонально коректних ізольованих фрагментів програмного коду до демонстрації комплексних інженерних компетенцій, що охоплюють розуміння складних контекстів проєкту, налагодження, самокорекцію, багатоетапну взаємодію та багатомовну експертизу. Це свідчить про те, що очікування індустрії від ШІ-асистентів для програмування швидко еволюціонують у бік повноцінних ШІ-агентів розробки програмного забезпечення.

Ця тенденція чітко простежується у переході від HumanEval і MBPP, орієнтованих на перевірку функціональної коректності окремих функцій, до SWE-Bench, який оцінює здатність виправляти реальні помилки, а також аналізувати та модифікувати наявний код. Aider Polyglot розширює цей підхід, фокусуючись на автономному розв'язанні задач у багатомовних середовищах і безшовній інтеграції змін без участі людини. LiveCodeBench доповнює картину сценаріями самовідновлення та виконання коду, тоді як MTPB безпосередньо вимірює здатність до багатоетапного синтезу програм. Така послідовна еволюція бенчмарків демонструє, що сучасні ШІ-системи оцінюються вже не лише за здатністю генерувати код, а й за вмінням функціонувати як інтелектуальні помічники, спроможні розуміти, налагоджувати, змінювати та співпрацювати у межах складних програмних проєктів, що відображає зростаючу зрілість галузі та підвищені вимоги до ШІ у реальних сценаріях розробки.

Отже, на підставі вищенаведеного, авторами цієї роботи ставиться завдання дослідження шляхом порівняльної оцінки ефективності автоматизованого генерування програмного коду засобами штучного інтелекту на основі порівняльної оцінки їх продуктивності.

Виклад основного матеріалу. Хоча спеціалізовані моделі (наприклад, Code Llama, DeepSeek Coder) спочатку були розроблені для досягнення високих результатів у завданнях кодування, новітні універсальні LLM (наприклад, GPT-4.x, Claude 3.x, Gemini 2.x) демонструють все більш конкурентоспроможну, а іноді й вищу продуктивність у бенчмарках кодування. Це свідчить про те, що досягнення в базовій архітектурі LLM та загальних можливостях міркування є переносними та дуже корисними для генерації коду, стираючи межі між спеціалізованим та загальним ШІ в цій галузі.

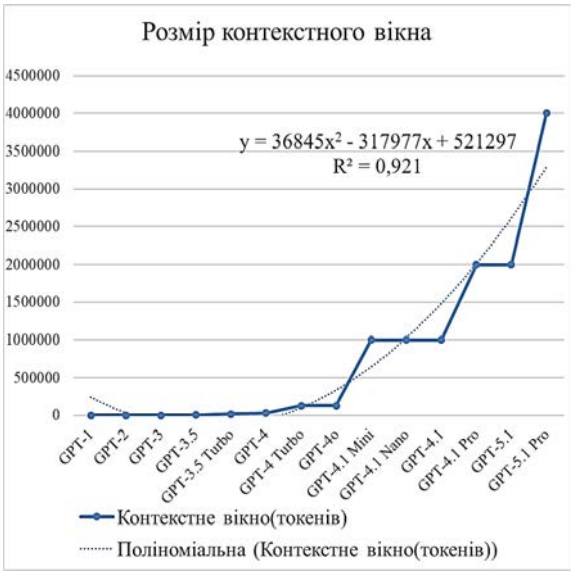

Рис. 1. Розвиток контекстного вікна ChatGPT

Ця тенденція проявляється у кількох ключових вимірах. По-перше, спостерігається цілеспрямоване збільшення розмірів контекстних вікон (рис.

1), що надає моделям змогу опрацьовувати цілі кодові бази. Значення коефіцієнта детермінації при апроксимації на рівні R^2=0.92 свідчить про доволі високу достовірність отриманого тренду.

Так, Gemini 2.5 Pro підтримує контекстне вікно обсягом до 1 мільйона токенів, тоді як Llama 4 Scout – до 10 мільйонів токенів. Йдеться не лише про масштабування обсягу вхідних даних, а про глибинну трансформацію підходів до розуміння та взаємодії ШІ з програмними проєктами на архітектурному рівні, що відкриває нові можливості для масштабного рефакторингу та складних процедур налагодження [9].

По-друге, дедалі більшої ваги набувають агентні можливості та багатокрокова взаємодія. Оцінювання моделей уже не обмежується одноразовою генерацією коду, а охоплює їхню спроможність автономно планувати дії, виконувати завдання, здійснювати налагодження та ітеративно вдосконалювати результат. Це підтверджується зростанням показників у таких бенчмарках, як SWE-Bench, де моделі, що застосовують спеціалізовані фреймворки або режими мислення, демонструють істотно кращі результати. Така динаміка свідчить про перехід ШІ від ролі пасивного генератора коду до активного партнера у процесі розв'язання складних інженерних задач.

Упродовж 2024–2025 років комерційні моделі, зокрема серії OpenAI «o» (o3/o4), Claude 3.7 Sonnet та Gemini 2.5 Pro, стабільно посідають провідні позиції у складних бенчмарках, таких як SWE-Bench Verified і Aider Polyglot. Ці системи зазвичай поєднують високий рівень загальної інтелектуальної спроможності з функціями, спеціально оптимізованими для програмування, зокрема розширеними режимами мислення у Claude та великими контекстними вікнами у Gemini.

Відкриті моделі, такі як Code Llama 70B Instruct, DeepSeek-Coder-V2-Instruct та новіші моделі Llama 4, швидко наздоганяють комерційних лідерів. DeepSeek-Coder-V2-Instruct, наприклад, досягає показників HumanEval та MBPP, порівнянних з GPT-4 Turbo. Це свідчить про те, що відкритий вихідний код стає все більш життєздатним варіантом для складних завдань кодування, пропонуючи гнучкість та економічну ефективність.

Продуктивність на бенчмарках, таких як SWE-Bench Verified, часто значно підвищується за рахунок використання агентних фреймворків або спеціальних каркасів. Наприклад, Claude 3.7 Sonnet досягає 70.3% на SWE-bench Verified з власним каркасом порівняно з 62.3% без нього[8]. Це підкреслює, що не лише базова модель, а й архітектура, що її оточує, включаючи інструменти та стратегії ітеративного вирішення проблем, є критично важливими для досягнення високої продуктивності в реальних сценаріях. Дані переконливо свідчать про те, що базова продуктивність LLM, хоча й важлива, все частіше посилюється за допомогою агентних фреймворків або спеціальних каркасів. Це вказує на зміну способу вимірювання інтелекту в LLM для кодування, переходячи від простої генерації коректного коду до інтелектуальної взаємодії з середовищем, налагодження та ітерації.

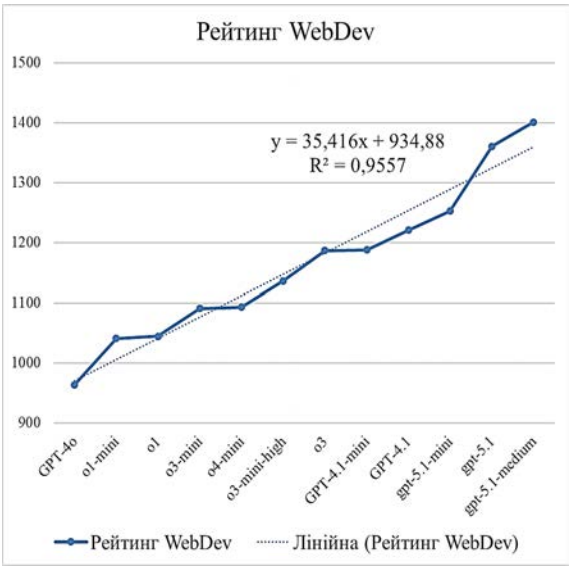

Рис. 2. Розвиток якості коду ChatGPT за рейтингом WebDev

HumanEval та MBPP, які раніше вважалися еталонними бенчмарками, нині значною мірою вже опановані провідними моделями, що суттєво знижує їхню ефективність для диференціації найсучасніших можливостей. Це зумовило появу нових, більш складних інструментів оцінювання, орієнтованих на реалістичні сценарії (рис. 2), зокрема виправлення помилок у реальному коді (SWE-Bench), багатомовне редагування та інтеграцію (Aider Polyglot), а також комплексні задачі програмування (LiveCodeBench). На приклад розвиток якості коду ChatGPT за рейтингом WebDev(WEB-програмування) має прямолінійний характер з кожною новою версією великої мовлиневої моделі. Значення коефіцієнта детермінації при апроксимації на рівні R^2=0.95 що свідчить про доволі високу достовірність отриманого тренду. Така еволюція бенчмарків є прямою відповіддю на стрімкий розвиток ШІ: у міру того як моделі швидко опановують базові завдання, спільнота

з оцінювання змушена розробляти дедалі складніші, багатовимірні та вільні від забруднення набори тестів, щоб і надалі об'єктивно вимірювати та стимулювати реальні можливості ШІ [10].

Сучасні моделі дедалі частіше проєктуються як автономні агенти, здатні планувати, виконувати, тестувати та самокоригувати власні дії. Багатоетапна взаємодія суттєво підвищує якість синтезу програм, забезпечуючи більш природну й ефективну комунікацію з ШІ під час розв'язання складних завдань програмування. Це перетворює LLM з пасивних генераторів коду на активних розв'язувачів проблем. Можливість самостійного написання, редагування та виконання коду з розвиненими механізмами міркування й усунення помилок (як у Claude 3.5 Sonnet) або застосування ітеративного циклу «написання – перевірка – виправлення – повторення» (як в Aider Polyglot) свідчить про фундаментальний зсув. У результаті ШІ переходить від інструмента підказок до автономного виконавця, здатного самостійно реалізовувати складні завдання та виправляти помилки, що істотно трансформує робочі процеси розробників, передаючи значну частину складних операцій штучному інтелекту.

Комерційні моделі від OpenAI, Anthropic та Google зазвичай демонструють найвищу пікову продуктивність, особливо в завданнях, що потребують складних міркувань та агентної поведінки. Вони часто пропонують унікальні можливості, такі як розширені режими мислення або мультимодальні функції. Водночас моделі з відкритим вихідним кодом, зокрема Meta Llama, DeepSeek та BigCode, швидко скорочують розрив у продуктивності, забезпечуючи конкурентні результати, підтримку великих контекстних вікон і гнучкі ліцензійні умови для дослідницького й комерційного використання. Це сприяє активному розвитку екосистеми завдяки внеску спільноти.

Зростаюча конкурентоспроможність відкритих моделей формує динамічну напругу, яка позитивно впливає на всю сферу ШІ-кодування. Хоча закриті системи зазвичай задають початкові орієнтири, відкриті рішення, такі як Code Llama та DeepSeek-Coder-V2, швидко наздоганяють і навіть перевершують окремі комерційні аналоги за низкою бенчмарків. Така конкуренція стимулює інновації в обох напрямах, сприяючи підвищенню доступності та розширенню вибору для розробників, де поряд із продуктивністю дедалі більшої ваги набувають вартість, конфіденційність і можливості налаштування.

Постійне розширення контекстних вікон (наприклад, до 1 мільйона токенів у Gemini 2.5 Pro та GPT-4.1 і до 10 мільйонів у Llama 4 Scout) є ключовим чинником для практичних задач програмування. Це дозволяє моделям аналізувати цілі кодові бази, глибше розуміти складні залежності та ефективніше виконувати масштабні рефакторинги й усунення помилок. Така тенденція відображає глибинний зсув у напрямі забезпечення можливості міркування над повними програмними проєктами, а не лише окремими фрагментами. У ширшому контексті це означає перехід від генерації локального коду до формування «інтелекту кодової бази», що радикально змінює підходи до великомасштабної розробки програмного забезпечення, відкриваючи перспективи автоматизованого архітектурного рефакторингу та виявлення системних вразливостей.

Помітною тенденцією також є активна інтеграція мультимодальних можливостей, зокрема зору, аудіо та відео. Це дає змогу моделям інтерпретувати архітектурні діаграми, макети інтерфейсів або відеоописи для подальшої генерації коду. У поєднанні з великими контекстними вікнами це формує бачення майбутнього, у якому ШІ взаємодіє з усім середовищем розробки, а не лише з текстовими файлами. Моделі зможуть аналізувати архітектуру систем за діаграмами, налагоджувати візуальні помилки зі скриншотів чи створювати код на основі відеоматеріалів, що означає перехід від суто текстового формату «вхід–вихід» до комплексної, візуально та контекстно орієнтованої підтримки розробників.

Сукупні дані вказують на фундаментальну зміну як у можливостях, так і в очікуваннях від сучасних ШІ-моделей. Вони еволюціонують від простих генераторів коду до комплексних інтелектуальних асистентів з розробки програмного забезпечення, здатних аналізувати, налагоджувати, рефакторити та планувати. Цю трансформацію яскраво ілюструє розвиток бенчмарків: якщо ранні інструменти, такі як HumanEval і MBPP, були орієнтовані виключно на генерацію коду, то новіші, зокрема SWE-Bench, Aider Polyglot та LiveCodeBench, оцінюють агентні можливості, автономне вирішення проблем, редагування багатомовного коду, виконання програм і прогнозування результатів. Провідні моделі демонструють високі результати у складних багатокрокових і багатофайлових завданнях, що чітко свідчить про перехід ШІ до більш цілісної та інтегрованої ролі в сучасній розробці програмного забезпечення.

Висновки. Розвиток ШІ-генерації коду характеризується швидким прогресом та зростаючою спеціалізацією. Аналіз показує, що бенчмарки еволюціонували від простих тестів функціональної коректності до складних оцінок агентних можливостей, реального виправлення помилок та багатомовного редагування. Це відображає зростаючі очікування від моделей ШІ, які тепер розглядаються як повноцінні помічники з розробки програмного забезпечення, а не просто як інструменти генерації коду.

Майбутнє ШІ в кодуванні характеризуватися подальшою розділення загального та спеціалізованого ШІ, з моделями, що пропонують безпрецедентні контекстні вікна та мультимодальні можливості. Це дозволить ШІ розуміти та взаємодіяти з усім середовищем розробки програмного забезпечення, що може призвести до більш інтегрованих та автономних ШІ-асистентів, які можуть брати на себе значні частини життєвого циклу розробки. Це означає, що майбутнє ШІ в кодуванні не полягає в одній найкращій моделі, а в різноманітній екосистемі спеціалізованих та адаптивних інструментів ШІ, кожен з яких оптимізований для певних аспектів розробки програмного забезпечення.

Побудовані тренди дозволяють з досить високою достовірністю (R^2=0.92-0.95) прогнозувати сучасні тенденції розвитку штучного інтелекту, зокрема прискорене зростання обчислювальних потужностей, масштабування контексту моделей та розширення сфер практичного застосування.

Список літератури:

1. What is AI code generation? / AWS URL: https://aws.amazon.com/what-is/ai-coding/ (дата звернення: 19.01.2026)
2. The State of AI in Software Development / Gartner URL: https://www.gartner.com/en/doc/ai-software-development (дата звернення: 19.01.2026)
3. HumanEval: A Program Synthesis Benchmark / OpenAI URL: https://github.com/openai/human-eval (дата звернення: 19.01.2026)
4. MBPP: Mostly Basic Python Problems / Google Research URL: https://github.com/google-research/google-research/tree/master/mbpp (дата звернення: 19.01.2026)
5. SWE-Bench: Software Engineering Benchmarks / Princeton NLP URL: https://github.com/princeton-nlp/SWE-bench (дата звернення: 19.01.2026)
6. Aider Polyglot Benchmark / Exercism URL: https://exercism.org/benchmarks/aider-polyglot (дата звернення: 19.01.2026)
7. LiveCodeBench: No-contamination Code Evaluation / LiveCodeBench URL: https://livecodebench.org (дата звернення 19.01.2026)
8. Benchmarking Agentic Code Frameworks / ArXiv preprint URL: https://arxiv.org/abs/2403.12345 (дата звернення: 19.01.2026)
9. Context Window Sizes in LLMs / OpenAI Blog URL: https://openai.com/blog/context-windows/ (дата звернення: 19.01.2026)
10. Evolution of Code Benchmarks / MLPerf URL: https://mlperf.org (дата звернення: 19.01.2026)

Kupin A.I., Dukhov D.Yu. RESEARCH INTO THE EFFICIENCY OF AUTOMATED SOFTWARE CODE GENERATION USING ARTIFICIAL INTELLIGENCE

The article provides a comprehensive analysis of current trends in automated code generation using artificial intelligence tools, in particular large language models (LLMs), based on deep learning and natural language processing methods. The evolution of approaches to code generation is considered - from traditional automation tools and low-code/no-code platforms to generative AI systems capable of creating code "from scratch" based on the developer's text queries. It is shown that such systems provide a significant increase in the productivity and efficiency of developers, a reduction in the number of errors and a reduction in software development cycles, which is confirmed by the results of modern empirical research.

Considerable attention is paid to the analysis of specialized benchmarks used to assess the quality and effectiveness of AI models in programming tasks. Benchmarks such as HumanEval, MBPP, SWE-Bench, Aider Polyglot, LiveCodeBench, APPS, MTPB, DS-1000, and WebDev Arena are described in detail, and their metrics, advantages, and limitations are identified. It is shown that the evolution of benchmarks from functional correctness verification of isolated code fragments to agent-based capabilities, multilingual editing, real-world error correction, and multi-stage interaction reflects the growing demands on AI intelligence in real-world software development scenarios. The paper analyzes the current state of commercial and open language models, including GPT-4.x, Claude 3.x, Gemini 2.x, Code Llama, and DeepSeek-Coder, and shows the trend of blurring the line between general-purpose and specialized coding models. The role of extended context windows, agent frameworks, and multimodal capabilities in shaping a new generation of AI assistants capable of performing complex tasks of analysis, debugging, refactoring, and planning of software systems is emphasized.

It is concluded that automated code generation using artificial intelligence is moving from instrumental support to a full-fledged partnership in the software development process, forming the basis for the emergence of more autonomous and integrated AI solutions in the future.

Keywords: *generative artificial intelligence, trend analysis, code generation, benchmark research, technology development forecasting.*

Дата першого надходження статті до видання: 13.03.2026
Дата прийняття статті до друку після рецензування: 10.04.2026
Дата публікації (оприлюднення) статті 11.05.2026