

УДК 004.8

[https://doi.org/10.52058/2786-6025-2024-12\(40\)-1314-1325](https://doi.org/10.52058/2786-6025-2024-12(40)-1314-1325)

Кривонос Олександр Миколайович кандидат педагогічних наук, доцент, доцент кафедри комп'ютерних наук та інформаційних технологій, Житомирський державний університет імені Івана Франка, м. Житомир, тел.: (098) 742-02-28, <https://orcid.org/0000-0002-4211-6541>

ВИКОРИСТАННЯ ГЕНЕРАТИВНОГО АІ ДЛЯ СТВОРЕННЯ ПРОГРАМНОГО КОДУ

Анотація. Стаття присвячена аналізу використання генеративного штучного інтелекту (ШІ) в програмуванні, що є актуальним напрямом у сучасній індустрії розробки програмного забезпечення. Розглядається, як інструменти на основі ШІ можуть підвищувати ефективність програмістів, автоматизувати рутинні завдання та сприяти генерації якісного коду.

Метою роботи є дослідження можливостей і обмежень генеративних АІ-технологій у програмуванні, аналіз сучасних інструментів і підходів до їх застосування, а також оцінка впливу ШІ на професійну діяльність розробників.

У статті наведено огляд популярних генеративних інструментів, таких як GitHub Copilot, Tabnine, OpenAI Codex, а також їх інтеграції з існуючими середовищами розробки. Особливу увагу приділено аналізу переваг цих інструментів, зокрема швидкості створення коду, зниження кількості помилок і покращення зручності командної роботи. Автор також висвітлює обмеження та ризики, пов'язані з їх використанням, серед яких: залежність від якості вхідних даних, потенційне створення небезпечного коду та етичні питання.

У роботі представлено реальні кейси застосування АІ для автоматизації задач, оптимізації існуючого коду та генерації тестових сценаріїв. Зроблено акцент на необхідності впровадження ШІ як доповнення, а не заміни людської праці, що зберігає творчий підхід до вирішення проблем.

Автор приходить до висновку, що використання генеративного ШІ у програмуванні є перспективним напрямком, який сприяє підвищенню продуктивності та інноваційності в індустрії розробки ПЗ. У подальших дослідженнях рекомендується зосередитися на інтеграції ШІ з low-code/no-code платформами, підвищенні його адаптивності до нових мов програмування та розробці алгоритмів, які враховують етичні аспекти.

Ключові слова: генеративний штучний інтелект; програмування; GPT-моделі; автоматизація написання коду.

Kryvonos Oleksandr Mykolaiovych Candidate of Pedagogical Sciences (PhD in Pedagogy) Docent Department of Computer Science and Information Technology, Zhytomyr Ivan Franko State University, Zhytomyr, tel.: (098) 742-02-28, <https://orcid.org/0000-0002-4211-6541>

THE USE OF GENERATIVE AI TO CREATE PROGRAM CODE

Abstract. The article is devoted to the analysis of the use of generative artificial intelligence (AI) in programming, a relevant trend in the modern software development industry. The article considers how AI-based tools can increase programmers' efficiency, automate routine tasks, and contribute to generating high-quality code.

The purpose of the study is researching the possibilities and limitations of generative AI technologies in programming, analysing modern tools and approaches to their application, and assessing the impact of AI on the professional activities of developers.

The article outlines popular generative tools such as GitHub Copilot, Tabnine, OpenAI Codex, and their integration with existing development environments. Particular attention is drawn to the analysis of the advantages of these tools, including the speed of code creation, reduction of errors, and improved teamwork. The author also highlights the limitations and risks associated with their use, including dependence on the quality of input data, potential creation of unsafe code, and ethical issues.

The study describes real cases of AI application for task automation, optimisation of existing code, and generation of test scenarios. The emphasis is placed on the need of AI implementation as a supplement, instead of a replacement of human labour, which preserves the creative approach to problem solving.

The author concludes the use of generative AI in programming is a promising area that contributes to increasing productivity and innovation in the software development industry. The author recommends that further research should focus on integrating AI with low-code/no-code platforms, increasing its adaptability to new programming languages, and developing algorithms that take into account ethical aspects.

Keywords: generative artificial intelligence; programming; GPT models; automation of code writing.

Постановка проблеми. У сучасних умовах, коли технології швидко розвиваються, штучний інтелект стає невід'ємною частиною багатьох галузей, зокрема й програмування. Використання штучного інтелекту у розробці програмного забезпечення відкриває перед програмістами унікальні можливості для покращення процесів створення, оптимізації коду, автоматизації завдань та навіть створення інноваційних рішень. Цей підхід до програмування не тільки прискорює розробку, а й дає змогу створювати більш інтелектуальні та адаптивні програми, здатні ефективно взаємодіяти з навколишнім світом. Одне з визначних досягнень – трансформація у сфері інформаційних технологій, яка стає можливою завдяки впровадженню штучного інтелекту. Це не

просто новий інструмент в арсеналі розробника, а повноцінний активний учасник у процесі створення та оптимізації програмного коду [1, 2, 3, 4].

Під «штучним інтелектом» (AI) прийнято розуміти машини, здатні виконувати завдання автономно. Водночас для розвитку штучного інтелекту в ньому потрібна активна участь людського розуму. Очевидно, що будь-який результат роботи штучного інтелекту потребує оцінки з боку людського розуму, включаючи людське сприйняття, міркування, аналіз, коригування, додаткове навчання тощо.

Зі збільшенням уваги до переваг AI, його стали повсюди використовувати для зменшення виконуваних робочих процесів, оскільки можливості штучного інтелекту дають підстави для автоматизації процесів та виконання рутинних процесів замість співробітників.

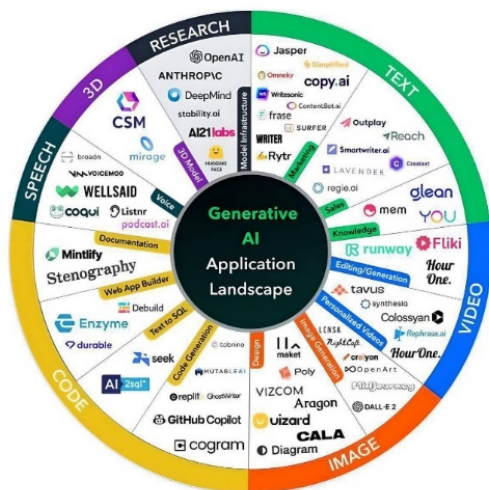


Рис. 1 Сфери застосування штучного інтелекту.

Джерело www.yarsavision.com/2023/06/exploring-generative-ai-applications.html

Аналіз останніх досліджень і публікацій свідчить про зростаючу увагу до використання генеративних систем штучного інтелекту (GenAI) у програмуванні. У роботі [9] розглядаються питання підвищення продуктивності та креативності за допомогою інструментів на основі штучного інтелекту, зокрема в контексті розробки програмного забезпечення. Здійснено аналіз переваг і викликів застосування таких інструментів, зокрема їх впливу на процес написання текстів. У статті [11] представлено таксономію швидких модифікаторів для перетворення тексту в зображення, що спрямована на вдосконалення якості й контрольованості згенерованого контенту, створюючи систематичну основу для дослідників та розробників. Робота [10] містить експериментальний аналіз впливу генеративного AI на продуктивність систем у різних галузях, що надає уявлення про потенційні переваги та можливі наслідки впровадження таких систем. У [12] досліджується досвід програмування з використанням штучного інтелекту, акцентуючи увагу на

перспективах програмістів та їхній взаємодії із системами генеративного AI, що відображає еволюцію підходів до програмування.

Мета статті – аналіз програмного забезпечення на базі генеративного штучного інтелекту, яке може бути використане в процесі створення програмного коду. Для досягнення мети використано наукові статті та результати, які отримали раніше вітчизняні та зарубіжні автори в контексті своїх досліджень.

Виклад основного матеріалу. Упровадження штучного інтелекту зумовило хвилю інновацій: якщо раніше комп'ютери суворо стежили за інструкціями, тепер вони здатні навчатися, адаптуватися й приймати рішення на основі даних. Часто це навіть можливе без безпосереднього втручання людини.

Сучасні технології штучного інтелекту, зокрема такі, як машинне навчання та нейромережі, здійснили революційний вплив на процес написання коду. Машинне навчання дає змогу комп'ютерам отримувати знання з даних без програмування кожного кроку. Нейромережі можуть виявляти закономірності та виконувати різні завдання: переклад тексту, розпізнавання образів.

Існує кілька сучасних систем на базі штучного інтелекту, які можуть генерувати, доповнювати та оптимізувати код, а також допомагати в автоматизації написання програмного забезпечення. Наведемо кілька основних платформ:

- GitHub Copilot;
- ChatGPT (зокрема, GPT-4 від OpenAI);
- Tabnine;
- Amazon CodeWhisperer;
- Replit Ghostwriter;
- Codex;
- CodeT5;
- IntelliCode від Microsoft [1, 2, 3, 4].

Сучасні AI-інструменти не лише допомагають писати код, а й оптимізують кожен етап розробки – від аналізу вимог до тестування й документування. Вони здатні пришвидшувати процес, зменшувати рутину й навіть розширювати можливості для менш досвідчених розробників, надаючи їм підтримку на кожному кроці. Крім того, AI активно вчиться, і з часом ці системи ставатимуть ще розумнішими, розвиваючи здатність до самовдосконалення, розпізнавання контексту та точнішого розуміння запитів. Це відкриває величезні перспективи як для професіоналів, так і для тих, хто тільки починає шлях у програмуванні. Водночас це далеко не всі можливості, адже сфера AI для написання коду швидко розвивається, і вже зараз існують ще більш вузькоспеціалізовані інструменти й експериментальні системи. Наведемо кілька додаткових напрямів та рішень.

AI для виправлення помилок та дебагінгу: DeepCode (тепер частина Snyk – це інструмент, що використовує AI для аналізу коду на наявність помилок, вразливостей та антипатернів. DeepCode може навіть пропонувати виправлення для типових помилок безпекового характеру); Ponicode – інструмент, що генерує тести на основі AI для виявлення помилок у різних випадках, його особливість – це здатність генерувати модульні тести з передбаченням можливих несподіваних результатів [2, 4].

Генерація документації та пояснень коду: Kite – AI-інструмент для написання коду, який також допомагає з документуванням функцій, пропонуючи приклади та пояснення, Kite стає корисним для великих команд і проектів, де потрібна єдина документація; DocGPT – експериментальна модель для створення пояснень і документації до складних функцій та модулів, особливо для коду, де коментарі відсутні або неповні [2, 4].

Налаштовані AI-моделі для специфічних потреб: AlphaCode від DeepMind – потужна модель для вирішення завдань змагань програмування, AlphaCode може розуміти умови конкурсних задач, писати рішення, а також перевіряти їхню ефективність; Codiga працює як “заготівник” кодових фрагментів, аналізуючи які модулі чи фрагменти часто використовують у проектах, і пропонує їх у швидкому доступі [8, 14].

Нейронні мережі для оптимізації коду: PrograMeter – інструмент для аналізу ефективності коду, який дає підстави зрозуміти, де є проблеми з оптимізацією, пропонуючи AI-оптимізовані рішення; Optimizer AI працює в інтеграції з Python і R, допомагає з налаштуванням параметрів та оптимізацією коду для високопродуктивних додатків у сфері Data Science [2, 4].

AI-помічники для конкретних фреймворків та технологій: FlutterFlow AI Builder – інструмент для генерації інтерфейсів на основі Flutter із використанням AI, що дає змогу швидко створювати UI-компоненти та перевіряти їхню адаптивність; JSXHub спеціалізується на створенні компонентів для React, він допомагає писати JSX-код для вебдодатків, пропонуючи адаптивні компоненти та шаблони для типових ситуацій [2, 14].

AI-платформи для Data Science та машинного навчання: DataRobot – платформа, яка автоматизує процес машинного навчання від підготовки даних до вибору моделі й оцінки результатів, DataRobot дає змогу легко створювати та вдосконалювати моделі без потреби в складному коді; AutoML інструменти (Google AutoML, H2O.ai) уможливають автоматичне створення ML-моделі, використовуючи AI для оптимізації коду та налаштування параметрів [7, 8].

Тестування безпеки за допомогою AI: Microsoft Security Copilot – інтеграція AI-алгоритмів з безпековими інструментами Microsoft для аналізу загроз, який перевіряє вразливості та пропонує патчі чи виправлення, коли знаходить проблеми; ShiftLeft – AI-інструмент, орієнтований на виявлення вразливостей ще на етапі написання коду, що зменшує потребу у виправленні безпекових помилок пізніше [2].

Ці інструменти показують, наскільки розширюються можливості автоматизації в різних аспектах розробки програмного забезпечення – від оптимізації й тестування до написання документації та безпеки.

Отже, вдалим прикладом генерації програмного коду на основі природної мови є нейромережа Chat GPT від OpenAI, яка здатна аналізувати та генерувати програмний код, даючи розробникам змогу автоматизувати та прискорити свою роботу.

Серед найбільш яскравих інновацій у цьому секторі виділяють два найважливіші проєкти: DeepCode та OpenAI Codex.

DeepCode, що зародився як стартап у Цюріху – приклад ефективного використання машинного навчання для аналізу та оптимізації коду. Цей інструмент, схожий на інтелектуальний код-рев'юєр, проводить сканування коду для виявлення помилок, уразливостей і неефективних методів програмування. На базі великої бази даних коду, зібраної з відкритих джерел, зокрема таких, як GitHub, DeepCode, навчається на мільйонах проєктів, стаючи ефективним інструментом із пропозицією корисних корекцій для оптимізації попереднього коду.

DeepCode надає плагіни для Visual Studio Code, Atom і Gitpod, а також інтерфейс командного рядка для виклику з робочого процесу розробки. Планується більше інтеграцій у типові інструменти розробника: вони мають відкритий вихідний код, тому, якщо у вас є ідеї, просто повідомте нам і приєднайтеся.

DeepCode пропонує безкоштовне сканування для проєктів із відкритим кодом і невеликих команд. Крім C і C++, DeepCode підтримує Java, Python, JavaScript і TypeScript. Ви можете спробувати це самостійно на DeepCode.ai [1, 2, 3].

Другий інструмент, OpenAI Codex, розроблений OpenAI – потужна система штучного інтелекту, здатна генерувати код на основі природної мови. Як просунута версія мовної моделі GPT-3, Codex вражає своєю здатністю розуміти текстові описи завдань та перетворювати їх на робочий програмний код.

OpenAI Codex – це передова модель штучного інтелекту, розроблена OpenAI, що спеціалізується на створенні коду у відповідь на команди природною мовою. Вона є основою GitHub Copilot, інструмента автозавершення для програмістів, інтегрованого з IDE, зокрема такими, як Visual Studio Code та Neovim. Codex базується на моделі GPT-3, адаптованій для програмування, і здатен генерувати код більш ніж дюжиною мов, найкраще працюючи з Python.

Модель була додатково навчена на 159 ГБ коду Python зі сховищ GitHub, що дає підстави їй вирішувати типові завдання, наприклад, генерувати код на основі коментаря. Codex демонструє високий рівень продуктивності,

створюючи коректні рішення приблизно для 70% тестових запитів. Основна її мета – допомога програмістам у прискоренні роботи, а не заміна людини в процесі програмування.

Хоча OpenAI закрав загальний доступ до Codex у 2023 році, дослідники все ще можуть користуватися ним у межах спеціальної програми. Інструмент також підтримує створення інтерактивних рішень, наприклад таких, як браузерні ігри чи наукові діаграми, завдяки інтеграції з бібліотеками на кшталт matplotlib. Codex найбільш ефективно застосовують для автоматизації рутинних завдань, аналізу коду та налагодження, залишаючи творчі аспекти розробки програмісту. Загалом Codex – потужний інструмент для оптимізації процесів розробки та навчання програмування [1, 2, 3, 4, 7, 14].

Автоматична генерація коду – це потужний засіб, який автоматизує написання коду різними мовами програмування: JavaScript, Python, SQL, JSON, HTML та інші. Цей інструмент стане в нагоді розробникам, які можуть бути незнайомими з певною мовою або потребують підтримки під час створення стандартного коду. Згенерований код легко модифікувати та адаптувати під конкретні вимоги користувача.

Для використання функцій автоматичного створення коду в різних областях, включаючи властивості компонента, редактори коду та конфігурацію кроку, просто введіть підказку або опис необхідного коду, і код буде створено автоматично. У процесі генерації SQL використовується базова структура підвищення якості відповіді.

Крім того, у процесі генерації штучний інтелект дає змогу перевірити та відредагувати згенерований код перед його вставкою. Такий підхід забезпечує гнучкість внесення будь-яких необхідних змін до коду відповідно до унікальних вимог вашого проекту.

Ще один представник генеративного штучного інтелекту – це AI асистент Tabnine. Tabnine – асистент для розробників програмного забезпечення, що використовує генеративний штучний інтелект для автоматичного доповнення коду. Цей інструмент аналізує контекст і синтаксис проекту, пропонуючи релевантні варіанти коду. Завдяки адаптивності Tabnine підлаштовується під стиль програмування користувача, що підвищує швидкість розробки та зменшує ймовірність помилок. Асистент підтримує різні мови програмування, зокрема Python, JavaScript, Go, Java, PHP, та ефективно інтегрується в популярні середовища розробки: Visual Studio Code і IntelliJ IDEA. Важливо, що Tabnine забезпечує конфіденційність даних: моделі тренуються тільки на відкритих даних із ліцензіями, а код користувачів не зберігається, якщо це не спеціально налаштовані приватні моделі. Додатково Tabnine може генерувати код на основі промптів, допомагаючи розробникам експериментувати з різними підходами та мовами, що значно оптимізує процес створення програмного забезпечення [2, 4, 8].

Codiga AI – це сучасний інструмент для розробки програмного забезпечення, що пропонує потужний набір функцій для написання, налагодження та оптимізації коду. Завдяки технології штучного інтелекту платформа забезпечує автоматичну перевірку коду, даючи змогу розробникам переконатися в його точності перед упровадженням. Codiga також допомагає швидко знаходити помилки, підвищувати продуктивність коду та дотримуватися найкращих практик програмування. Платформа має інтегроване середовище розробки (IDE), що підтримує роботу з різними мовами програмування, зокрема C++, Java та Python. Codiga не потребує додаткових плагінів, що спрощує роботу. Інструменти для налагодження дають змогу розробникам усіх рівнів досвіду ефективно вирішувати технічні проблеми та покращувати код. Codiga легко інтегрується в наявні робочі процеси й забезпечує високу гнучкість і швидкість розробки, що робить її надійною платформою для створення якісних програмних рішень [1, 2].

GitHub Copilot – це інструмент, створений у результаті співпраці GitHub, найбільшої платформи для хостингу коду, та OpenAI. Його розробили для покращення досвіду програмування та прискорення розробки [6, 7, 8].

Ідея Copilot бере свої витоки з розширення IntelliCode для Visual Studio, яке забезпечувало автозавершення коду за допомогою AI, орієнтуючись на шаблони кодування розробників та контекст проєктів. 2021 року GitHub об'єднався з OpenAI для створення нового помічника з кодування, заснованого на мовних моделях OpenAI. Це партнерство уможливило значне розширення можливостей генерації коду, зробивши Copilot потужним інструментом для розробників. У липні 2021 року інструмент став доступним у технічному попередньому перегляді для обмеженого кола розробників, які змогли протестувати можливості Copilot і надати зворотний зв'язок для його вдосконалення. Завдяки цьому періоду тестування GitHub Copilot став повноцінним інструментом, здатним значно підвищити ефективність написання коду [8].

Використання AI як джерела підказок і допоміжних інструментів під час програмування – це, мабуть, найкращий підхід, особливо для початківців і тих, хто вивчає нові технології. AI може швидко підказати синтаксис, шаблони чи загальні рішення для типових задач, звільняючи час для творчих і аналітичних частин розробки. Підказки AI часто пропонують зрозумілі фрагменти коду з поясненнями, які допомагають початківцям навчитися ефективно застосовувати різні структури та функції. Наприклад, ChatGPT або Copilot може пропонувати готові рішення, показуючи, як організувати код і які методи можна застосовувати для певних завдань.

Автозавершення, як у Tabnine або Kite, дає змогу швидко орієнтуватися в синтаксисі мови, використовуючи підказки для найчастіших структур, змінних і функцій. Це особливо зручно для роботи з довгими функціями чи

складними конструкціями. AI-системи часто здатні попереджати про помилки ще на етапі написання коду. Наприклад, підказки щодо можливих вразливостей чи неправильних обчислень допомагають уникнути помилок до моменту компіляції або виконання коду. Підказки уможливають збереження фокусу, адже початківцям часто буває важко продовжувати працювати, коли стикаються з труднощами. AI може надати корисні приклади й пояснення, що підвищує впевненість і мотивує вчитися далі.

Розглянемо на конкретному прикладі, як ChatGPT шукає розв'язок для текстової задачі. Ми навмисно використали тестову задачу з одного із сайтів для підготовки учнів до змагань із програмування (eolymp.com) з тієї причини, що AI досить непогано може впоратися з технічними завданнями [5].

Подорожі гномів (<https://basecamp.eolymp.com/uk/problems/400>)

Як відомо, гноми жили в лісі й подорожували рідко, пересуваючись в основному пішки. Проте вони набагато раніше за людей побудували своє метро й на далекі відстані переміщувалися лише використовуючи його.

На подорожі витрачали багато часу, та й квиток у метро коштував досить дорого. Проте вони домовилися, що всі гноми, яким випаде "щасливий" квиток, отримують його безкоштовно. Щасливим у розумінні гномів був той квиток, у якому у r -й системі числення сума цифр перших k розрядів дорівнювала сумі цифр останніх k розрядів. Відомо, що номери квитків у метро в гномів завжди містять $2k$ розрядів.

А скільки ж гномів зможе в цьому разі проїхати в метро безкоштовно? Урахуйте, що гноми дуже справедливі, і якщо в когось із них вже була безкоштовна поїздка, він завжди віддавав наступні щасливі квитки своїм товаришам.

Вхідні дані

У єдиному рядку задано через пропуск значення r та k .

Вихідні дані

Шукана кількість гномів, обрахована за модулем 18446744073709551616.

Обмеження

Гноми таємно повідомили, що проїздом до Бухареста через Львів вони дізналися про щасливу львівську цифру 4, і тому система числення u номери квитків у них тепер такі, що відповідають умові: $k(p-1) + 1 \leq 4444$ [13].

Далі наведемо те, що штучний інтелект відповів нам на запит:

Задачу про квитки гномів можна розв'язати так: треба порахувати кількість квитків, які вважають "щасливими" в системі числення з основою rrr і з кількістю розрядів $2k2k2k$, де сума перших kkk цифр дорівнює сумі останніх kkk цифр.

Пояснення задачі:

rrr — основа системи числення.

kkk — кількість розрядів у половині номера квитка. Загальна кількість розрядів у номері квитка — $2k2k2k$.

Квиток вважають "щасливим", якщо:

Номер квитка в r -й системі числення складається з $2k2k2k$ цифр.

Сума цифр перших kkk розрядів дорівнює сумі цифр останніх kkk розрядів.

Алгоритм:

Генеруємо всі можливі номери квитків у rrr -й системі числення з $2k2k2k$ розрядів.

Перевіряємо для кожного квитка, чи дорівнює сума цифр перших kkk розрядів сумі цифр останніх kkk розрядів.

Підраховуємо кількість таких квитків.

Покроковий план:

1. Генерація номерів квитків:

1. Квитки — це числа від 000 до $r^{2k}-1r^{2k} - 1r^{2k}-1$ у rrr -й системі числення.

2. Обчислення сум цифр:

1. Для кожного числа переводимо його в rrr -ву систему числення і ділимо на дві частини: перші kkk розрядів і останні kkk розрядів.

2. Підраховуємо суму цифр для обох частин і перевіряємо їх рівність.

3. Підрахунок результату:

1. Якщо суми збігаються, додаємо цей квиток до списку "щасливих".

2. У кінці виводимо кількість "щасливих" квитків. [<https://chatgpt.com/>]

Далі був запропонований код мовою Python, що реалізує описаний вище алгоритм:

```
def count_lucky_tickets(p, k):
    count = 0
    # Обмеження кількості цифр у квитку
    max_ticket = p ** (2 * k)
    for ticket in range(max_ticket):
        # Перетворюємо квиток у r-ву систему числення
        digits = []
        temp = ticket
        for i in range(2 * k):
            digits.append(temp % p)
            temp //= p
        # Сума перших k розрядів
        sum_first_k = sum(digits[:k])
        # Сума останніх k розрядів
        sum_last_k = sum(digits[k:])
        # Якщо суми рівні, квиток щасливий
        if sum_first_k == sum_last_k:
            count += 1
```



```
return count
```

```
# Читання вхідних даних
```

```
p, k = map(int, input().split())
```

```
# Перевірка умови задачі
```

```
if k * (p - 1) + 1 <= 4444:
```

```
    result = count_lucky_tickets(p, k)
```

```
    print(result)
```

```
else:
```

```
    print("Обмеження не виконуються") [5].
```

Наразі зазначимо, що представлений алгоритм пройшов лише 14% тестів.

AI не завжди дає оптимальні рішення для складних завдань, але він дуже добре допомагає заповнювати прогалини в знаннях, прискорює процес та зменшує стрес, особливо для початківців. Використання AI як помічника для підказок – це не просто спосіб пришвидшити написання коду, а й реальна можливість учитися, розширювати знання й отримувати більше задоволення від кодування.

Висновки. Отже, використання штучного інтелекту в програмуванні – значний крок у розвитку технологій. AI здатний покращити процеси розробки програмного забезпечення, забезпечуючи більш ефективно та швидко створення складних систем. Автоматизація рутинних завдань, аналіз даних, передбачувана аналітика та оптимізація коду – лише деякі з галузей, де AI виявляє свою силу. Водночас машинне навчання (нейронні мережі) все частіше застосовують для оптимізації коду, створеного людиною, а також для генерації свого коду та покращення, згенерованого ним. З урахуванням постійного розвитку AI потрібно очікувати на нові інновації та можливості для покращення процесів програмування.

Література:

- 12 Best AI Tools for Coding. Available: <https://www.taskade.com/blog/ai-coding-tools/>
- 31 Best AI Coding Tools in 2024. Available: <https://www.scaler.com/blog/coding-ai-tools/>
- AI That Writes Code: 7 Best & Popular AI Coding Tools 2024. Available: <https://www.bytracyjackson.com/ai-that-writes-code/>
- Best AI Coding Tools in 2024. Available: <https://www.qodo.ai/blog/best-ai-coding-assistant-tools/>
- ChatGPT. Available: <https://chatgpt.com/>
- Copilot for Xcode: Exploring AI-Assisted Programming by Prompting Cloud-based Large Language Models. Available: https://www.researchgate.net/publication/372684232_Copilot_for_Xcode_Exploring_AI-Assisted_Programming_by_Prompting_Cloud-based_Large_Language_Models.
- Guide to Using AI to Write Code. Available: <https://codeanywhere.com/blog/guide-to-using-ai-to-write-code>.
- How You Can Use AI to Improve Your Code Quality. Available: <https://www.freecodecamp.org/news/how-to-use-ai-to-improve-code-quality>.

9. Murillo A., D'Angelo S. An Engineering Perspective on Writing Assistants for Productivity and Creative Code. 2023. Available: https://cdn.glitch.global/d058c114-3406-43be-8a3c-d3afff35eda2/paper1_2023.pdf

10. Noy S., Zhang W. Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence. SSRN Electronic Journal. Elsevier BV. (March 1, 2023). 15 p. Available at SSRN: <https://ssrn.com/abstract=4375283> or <http://dx.doi.org/10.2139/ssrn.4375283>

11. Oppenlaender J. A Taxonomy of Prompt Modifiers for Text-To-Image Generation. arXiv. 2023. 18 p. <https://doi.org/10.48550/arXiv.2204.13988>

12. Sarkar A., Gordon A. D., Negreanu C., Poelitz C., Ragavan S. S., Zorn B. What is it like to program with artificial intelligence? 2022

13. The Travels of the Gnomes. Available: <https://basecamp.eolymp.com/uk/problems/4009>

14. Writing Code with AI. Available: <https://docs.superblocks.com/generative-ai/writing-code-with-ai>.

References:

1. 12 Best AI Tools for Coding. Available: <https://www.taskade.com/blog/ai-coding-tools/>
2. 31 Best AI Coding Tools in 2024. Available: <https://www.scaler.com/blog/coding-ai-tools/>

3. AI That Writes Code: 7 Best & Popular AI Coding Tools 2024. Available: <https://www.bytracyjackson.com/ai-that-writes-code/>

4. Best AI Coding Tools in 2024. Available: <https://www.qodo.ai/blog/best-ai-coding-assistant-tools/>

5. ChatGPT. Available: <https://chatgpt.com/>

6. Copilot for Xcode: Exploring AI-Assisted Programming by Prompting Cloud-based Large Language Models. Available: https://www.researchgate.net/publication/372684232_Copilot_for_Xcode_Exploring_AI-Assisted_Programming_by_Prompting_Cloud-based_Large_Language_Models.

7. Guide to Using AI to Write Code. Available: <https://codeanywhere.com/blog/guide-to-using-ai-to-write-code>.

8. How You Can Use AI to Improve Your Code Quality. Available: <https://www.freecodecamp.org/news/how-to-use-ai-to-improve-code-quality>.

9. Murillo A., D'Angelo S. An Engineering Perspective on Writing Assistants for Productivity and Creative Code. 2023. Available: https://cdn.glitch.global/d058c114-3406-43be-8a3c-d3afff35eda2/paper1_2023.pdf

10. Noy S., Zhang W. Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence. SSRN Electronic Journal. Elsevier BV. (March 1, 2023). 15 p. Available at SSRN: <https://ssrn.com/abstract=4375283> or <http://dx.doi.org/10.2139/ssrn.4375283>

11. Oppenlaender J. A Taxonomy of Prompt Modifiers for Text-To-Image Generation. arXiv. 2023. 18 p. <https://doi.org/10.48550/arXiv.2204.13988>

12. Sarkar A., Gordon A. D., Negreanu C., Poelitz C., Ragavan S. S., Zorn B. What is it like to program with artificial intelligence? 2022

13. The Travels of the Gnomes. Available: <https://basecamp.eolymp.com/uk/problems/4009>

14. Writing Code with AI. Available: <https://docs.superblocks.com/generative-ai/writing-code-with-ai>.