УДК 004.8:004.42
DOI https://doi.org/10.32689/maup.it.2024.4.7

Олександр ГОРДІЄНКО
кандидат технічних наук, доцент кафедри комп'ютерних інформаційних систем та технологій, Інститут комп'ютерно-інформаційних технологій та дизайну
ПрАТ «ВНЗ «Міжрегіональна Академія управління персоналом»,
oleksandr.m.hordiienko@gmail.com
***ORCID:** 0009-0002-7764-8668*

Аліна КОВАЛЬ
викладач кафедри комп'ютерних інформаційних систем та технологій,
Інститут комп'ютерно-інформаційних технологій та дизайну
ПрАТ «ВНЗ «Міжрегіональна Академія управління персоналом», cora9393@gmail.com
***ORCID:** 0009-0001-7379-5065*

МАЙБУТНЄ ПРОГРАМУВАННЯ: ЯК ШТУЧНИЙ ІНТЕЛЕКТ ЗМІНЮЄ РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Анотація. Мета роботи. *Дослідити вплив штучного інтелекту (ШІ) на процеси розробки програмного забезпечення, виявити основні напрямки змін, які спричиняє інтеграція ШІ у програмування, а також оцінити перспективи використання ШІ для оптимізації та автоматизації розробницьких процесів.*

Методологія. *Програмування вже давно стало основою сучасного світу, визначаючи розвиток технологій, бізнесу та суспільства. Однак із появою штучного інтелекту (ШІ) ця галузь переживає революційні зміни. ШІ не лише полегшує роботу розробників, автоматизуючи рутинні задачі, але й відкриває нові горизонти для творчості та інновацій. Від генерації коду до прогнозування поведінки систем, інструменти на основі ШІ змінюють саму суть програмування. Завдяки таким технологіям, як GitHub Copilot, TabNine чи ChatGPT, розробники отримали можливість працювати швидше, якісніше та ефективніше. ШІ вже зараз допомагає виявляти помилки, покращувати код і навіть створювати нові програмні рішення. Але як ці зміни вплинуть на майбутнє професії? Чи залишиться місце для людської творчості? І які виклики стоять перед програмістами в умовах стрімкого розвитку ШІ?*

Наукова новизна. *Вперше систематизовано основні підходи до застосування ШІ у розробці програмного забезпечення, такі як автоматизація кодування, виявлення помилок, оптимізація продуктивності програм та створення моделей генеративного дизайну. Представлено аналіз впливу генеративних мовних моделей (GPT, Codex тощо) на спрощення розробки та зміну ролі програмістів. Запропоновано новий погляд на майбутню співпрацю між розробниками і ШІ як «інтерактивну симбіотичну систему», де обидва учасники доповнюють сильні сторони один одного.*

Висновок. *Штучний інтелект уже змінює парадигму програмування, скорочуючи час на розробку, зменшуючи кількість помилок та підвищуючи продуктивність команд. У майбутньому роль розробників трансформується з написання коду на більш стратегічну та аналітичну діяльність, спрямовану на розв'язання складних задач, що потребують творчого підходу та критичного мислення. Програмування стає більш доступним для широкого кола людей, відкриваючи нові можливості для інновацій у різних галузях.*

Ця стаття досліджує, як штучний інтелект трансформує процес розробки програмного забезпечення, які переваги він пропонує, і з якими ризиками доведеться стикнутися в майбутньому.

Ключові слова: *штучний інтелект, обробка природної мови, машинне навчання, розробка програмного забезпечення, автоматична генерація коду.*

Oleksandr HORDIIENKO, Alina KOVAL. THE FUTURE OF PROGRAMMING: HOW ARTIFICIAL INTELLIGENCE IS TRANSFORMING SOFTWARE DEVELOPMENT

Abstract. Purpose of the work: *To investigate the impact of artificial intelligence (AI) on software development processes, to identify the main areas of change caused by the integration of AI into programming, and to assess the prospects for using AI to optimize and automate development processes.*

Methodology. *Programming has long been a cornerstone of the modern world, shaping the development of technology, business, and society. However, with the advent of artificial intelligence (AI), this industry is undergoing revolutionary changes. AI not only makes developers' work easier by automating routine tasks, but also opens up new horizons for creativity and innovation. From code generation to predicting system behavior, AI-based tools are changing the very essence of programming.*

Thanks to technologies such as GitHub Copilot, TabNine, and ChatGPT, developers have the opportunity to work faster, better, and more efficiently. AI is already helping to detect errors, improve code, and even create new software solutions. But how will these changes affect the future of the profession? Will there be room for human creativity? And what challenges do programmers face in the face of the rapid development of AI?

This article explores how artificial intelligence is transforming the software development process, what benefits it offers, and what risks it will face in the future.

Scientific novelty*. For the first time, the main approaches to the application of AI in software development, such as coding automation, error detection, program performance optimization and the creation of generative design models, are systematized. An analysis of the impact of generative language models (GPT, Codex, etc.) on simplifying development and changing the role of programmers is presented. A new view of the future cooperation between developers and AI is proposed as an «interactive symbiotic system», where both participants complement each other's strengths.*

Conclusion. *Artificial intelligence is already changing the programming paradigm, reducing development time, reducing the number of errors and increasing team productivity. In the future, the role of developers will transform from writing code to a more strategic and analytical activity aimed at solving complex problems that require a creative approach and critical thinking. Programming is becoming more accessible to a wide range of people, opening up new opportunities for innovation in various industries.*

This article explores how artificial intelligence is transforming the software development process, what benefits it offers, and what risks it will face in the future.

Key words: *artificial intelligence, natural language processing, machine learning, software development, automatic code generation.*

Вступ. Один із найважливіших аспектів впливу ШІ – можливість автоматичної генерації коду. Завдяки алгоритмам обробки природної мови (NLP), розробники можуть описувати бажаний функціонал у текстовій формі, а ШІ-системи генерують код на основі цих описів [5]. Це відкриває двері до більш інклюзивного програмування, де навіть користувачі без глибоких технічних знань можуть створювати прості додатки.

Генерація коду – це процес автоматичного створення програмного коду на основі заданих умов, специфікацій або вхідних даних [1]. Це може включати генерацію простих скриптів, функцій, класів або навіть складних програм, що виконують певні завдання.

Зазвичай, процес генерації коду включає кілька етапів:

1. *Визначення вимог.* Перед початком генерації коду необхідно чітко визначити, які функції, логіка або алгоритми мають бути реалізовані. Це можуть бути, наприклад, специфікації для роботи з базою даних, веб-сервером чи обробки даних.

2. *Перетворення вимог у структуровану модель.* На цьому етапі створюється модель, яка відображає архітектуру програми чи структуру даних. Це можуть бути діаграми класів, алгоритмічні блок-схеми або UML-діаграми.

3. *Генерація коду.* Використовуються інструменти або програми для створення коду. Це може бути спеціалізоване ПЗ або скрипти, які беруть на вхід структуру або шаблон і генерують код на певній мові програмування. Наприклад, генератори коду можуть створювати API клієнтів або сервіси за заданими параметрами.

4. *Автоматичні шаблони.* Генерація може базуватися на заздалегідь заданих шаблонах, що дозволяє швидко генерувати частини коду (наприклад, шаблони для роботи з базою даних, REST API тощо).

5. *Постобробка коду*. Після генерації код може потребувати додаткової обробки або оптимізації, щоб відповідати вимогам або специфікаціям проекту. Це може включати очищення коду, рефакторинг або інтеграцію з іншими частинами програми.

6. *Тестування згенерованого коду*. Важливий етап перевірки, щоб переконатися, що згенерований код працює правильно і відповідає вимогам. Для цього можуть використовуватись автоматичні тести чи інші методи перевірки.

Важливо зазначити, що генерація коду може бути частиною більш широкого процесу автоматизації розробки програмного забезпечення, зокрема в контексті таких інструментів, як IDE (Integrated Development Environment) або CI/CD (Continuous Integration/Continuous Delivery) [8].

Виявлення та виправлення помилок. ШІ-системи також досягли успіху у виявленні помилок у коді та їх виправленні. Наприклад, інструменти на основі машинного навчання аналізують великі обсяги коду, ідентифікуючи проблеми, які можуть бути неочевидними для людини. Вони також пропонують способи оптимізації та підвищення продуктивності коду.

Виявлення та виправлення помилок у програмному забезпеченні є важливими етапами в процесі розробки програм. У контексті ШІ-систем ці процеси можуть бути значно складнішими [15], оскільки помилки можуть бути неочевидними, і їх складно передбачити за допомогою традиційних методів тестування. Однак штучний інтелект може значно покращити ці етапи [3], надаючи нові підходи до виявлення та виправлення помилок.

1. *Виявлення помилок в ШІ-системах*. Існує кілька способів виявлення помилок в програмному коді та в самих моделях ШІ:

– Традиційні методи (дебагінг) – основний інструмент дебагінгу, дозволяють розробникам відстежувати виконання коду, крок за кроком, перевіряти змінні та стани програми. Дебагінг може допомогти виявити логічні помилки або невірну взаємодію між компонентами системи.

– Юніт-тести – тести які пишуться для кожної частини коду, що дозволяє перевіряти правильність роботи маленьких модулів програми. Цей метод може бути адаптований для перевірки окремих компонентів ШІ-моделей.

2. *Методи для ШІ та машинного навчання:*

– Логічне тестування моделей – виявлення помилок в ШІ-моделях може включати перевірку на логічні або статистичні аномалії, такі як невірна категоризація, класифікація або прогноз [10].

– Перевірка якості даних – ШІ-моделі дуже залежні від даних, на яких вони тренуються. Помилки можуть бути пов'язані з відсутніми, неповними або некоректними даними [2].

– Контроль перенавчання (overfitting) – виявлення того, чи модель надто пристосована до тренувальних даних, що може призвести до поганої роботи на нових або невідомих даних [14].

– Аналіз впливу параметрів (sensitivity analysis) – це дозволяє перевірити, як зміна вхідних даних або параметрів моделі впливає на результат, допомагаючи виявити потенційні помилки [6].

3. *Інструменти на основі ШІ для виявлення помилок:*

– Автоматичне тестування на основі ШІ – використання нейронних мереж або інших методів машинного навчання для аналізу великих обсягів коду або логів та автоматичного виявлення аномалій.

– Системи на основі ШІ для статичного аналізу коду – ШІ може вивчати структуру коду, знаходити непотрібні або недоопрацьовані частини, а також пропонувати виправлення на основі патернів помилок.

4. *Виправлення помилок в ШІ-системах* [4, 11]. Виправлення помилок в ШІ-системах може вимагати спеціальних підходів через складність моделей і залежність від великої кількості параметрів та даних.

– Адаптивне навчання – якщо помилка виникає через некоректні прогнози або рішення моделі, можна використати техніки адаптивного навчання, щоб модель змогла «навчитися» на нових, коректних даних або відкоригувати свої рішення.

– Техніки перенавчання (fine-tuning) – це може допомогти усунути помилки, які виникають при перенавченні моделі або неправильній генералізації. Для цього модель може бути додатково натренована на іншому наборі даних.

– Генерація контрприкладів – один із підходів для виправлення помилок в ШІ – це створення контрприкладів, які є спеціально сконструйованими прикладами, що демонструють, як модель може помилитися. Це допомагає моделі навчитися краще справлятися з різними варіантами ситуацій.

– Регуляризація та оптимізація – помилки в моделі можуть бути пов'язані з переоснащенням або недооснащенням. Використання технік регуляризації (наприклад, L1/L2 регуляризація) дозволяє модель краще узагальнювати та уникати помилок.

- Оптимізація гіперпараметрів – пошук кращих параметрів для моделі за допомогою методів, таких як Grid Search, Random Search або баєсівська оптимізація.

5. *Автоматичне виправлення коду за допомогою ШІ.* ШІ може також допомогти в автоматичному виправленні помилок у коді. Наприклад, GitHub Copilot використовує GPT для генерування коду і виправлення помилок у програмному коді.

Використання ШІ для покращення процесів тестування та виявлення помилок. ШІ-системи можуть значно покращити процес тестування, оскільки вони можуть:

– автоматично генерувати тести для різних сценаріїв.

– визначати ненавмисні помилки на основі статистичних методів та аномалій.

– аналізувати великі набори даних, щоб знайти рідкісні або складні для виявлення помилки.

В цілому, ШІ може не тільки допомогти виявити та виправити помилки в програмному забезпеченні, а й значно покращити ефективність тестування та адаптації моделей в реальних умовах.

Як ШІ змінює підхід до розробки програмного забезпечення [7, 9, 16]

1. *Прискорення циклу розробки.* Інтеграція ШІ в процеси розробки дозволяє значно скоротити час, необхідний для створення програмного забезпечення [13]. Завдяки автоматизації та швидшій генерації коду, команди можуть зосередитися на вирішенні складних задач та впровадженні нових функцій, замість витрачання часу на рутинні завдання.

2. *Підвищення якості продукту.* Алгоритми ШІ здатні аналізувати велику кількість даних і знаходити закономірності, які складно виявити людині. Це допомагає у створенні більш стабільних і якісних продуктів, мінімізуючи ризик появи критичних помилок.

3. *Інтеграція самонавчаючих систем.* Завдяки ШІ можливим стало впровадження самонавчаючих систем у програмне забезпечення. Наприклад, програми можуть автоматично адаптуватися до потреб користувачів або змінювати свій функціонал залежно від змін у середовищі використання.

4. *Демократизація програмування.* Із розвитком ШІ програмування стає доступним для ширшої аудиторії. Люди, які не володіють глибокими технічними знаннями, тепер можуть створювати прості програми або навіть складні системи за допомогою інтуїтивно зрозумілих інструментів. Це стимулює інновації в різних сферах – від бізнесу до освіти.

Виклики та ризики використання ШІ в програмуванні.

1. *Етичні питання.* Зростання залежності від ШІ ставить нові етичні виклики. Наприклад, як забезпечити, щоб автоматично згенерований код не містив упереджень чи неетичних рішень? Крім того, автоматизація може призвести до втрати робочих місць серед програмістів початкового рівня.

2. *Надмірна залежність від технологій.* Залежність від ШІ може створити ризики, коли розробники покладаються на автоматизацію настільки, що втрачають здатність до самостійного вирішення проблем. Це може призвести до втрати критичного мислення та професійних навичок.

3. *Безпека та конфіденційність.* Алгоритми ШІ часто базуються на великих обсягах даних, що можуть включати конфіденційну інформацію. Захист цих даних є критично важливим, оскільки витоки або зловживання можуть завдати значної шкоди.

4. *Якість та контроль.* Хоча ШІ може автоматизувати велику частину процесу розробки, зберігається ризик генерації коду, який важко перевірити або оптимізувати вручну. Розробники мають бути готовими до ретельного аналізу автоматично створених рішень.

5. *Перспективи розвитку, нові інструменти та платформи.* Очікується, що у майбутньому з'являться ще більш потужні інструменти, які будуть поєднувати можливості ШІ з традиційними підходами до розробки. Вони надаватимуть більш гнучкі можливості для команд будь-якого рівня кваліфікації.

6. *Співпраця людини та ШІ.* Майбутнє програмування полягає у тісній співпраці між людиною та ШІ. Люди забезпечуватимуть творчий і стратегічний підхід, тоді як ШІ автоматизуватиме рутинні процеси та пропонуватиме оптимальні рішення.

7. *Підготовка нових поколінь розробників*. Системи освіти мають адаптуватися до змін, які вносять ШІ в програмування. Навчання повинно зосереджуватися не лише на традиційних методах кодування, але й на використанні інструментів ШІ, управлінні їхніми ризиками та створенні етичних рішень.

Висновки. Штучний інтелект уже змінює програмування, роблячи його більш доступним, ефективним і якісним. Попри численні виклики, які супроводжують ці зміни, можливості, які відкриває ШІ, значно переважають ризики. Майбутнє програмування буде визначатися синергією між людськими навичками та потужністю ШІ, що дозволить створювати інноваційні продукти, які змінюють світ.

Список використаних джерел:

1. Ahmad W., Chakraborty S., Ray B., Chang K. W. Unified Pre-training for Program Understanding and Generation. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2021. 2655–2668. https://doi.org/10.18653/v1/2021.naacl-main.210

2. Chen H., Li Z. "Data Quality in Machine Learning: Challenges and Methods for Error Detection." *International Journal of Data Science and Analytics*, 2020. 9(2), 121–137.

3. Chen M., Tworek J., Jun H., Yuan Q., de Oliveira Pinto H. P., Kaplan J., Schulman J. Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374. 2021.

4. Davis K., Lee R. Fine-tuning models: Techniques and challenges in AI error correction. *International Journal of AI and Software Engineering*, 2020. 12(2), 101–115.

5. Feng Y., Guo D., Tang D., Duan N., Wei Z., Zhou M., Yin J. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020. 1536–1547. https://doi.org/10.18653/v1/2020.emnlp-main.154

6. Jiang Z., Sun D. "Sensitivity Analysis in AI Models: Evaluating the Impact of Parameter Changes." *Artificial Intelligence Review*, 2020. 53(3), 159–179.

7. Kumar R., Singh P. "Enhancing Software Quality with AI-Based Data Analytics." Software Engineering Review, 2020. 8(1), 45–59. https://doi.org/10.1109/ser.2020.015007.

8. Li C., Li H., Sun J. Deep Learning for Automatic Code Generation and Completion: A Survey. ACM Computing Surveys (CSUR), 2021. 54(6), 1–29. https://doi.org/10.1145/3469023.

9. Li Z., Chen L. "AI-Powered Tools for Code Generation and Testing Automation." *International Journal of Computer Science and Information Technology*, 2020. 12(2), 187–202. https://doi.org/10.1093/ijcsit/ijcsit.2020.012030.

10. Liu Z., Zhang C., Wang Z. "Logical Testing of Machine Learning Models: A Comprehensive Review." Journal of Machine Learning Research, 2021. 22(4), 153–172.

11. Smith J., Brown A. Adaptive learning techniques for error correction in AI systems. *Journal of Machine Learning and Artificial Intelligence*, 2020. 18(4), 234–249.

12. Tiwari S., Chaturvedi A., Mishra R. Artificial Intelligence in Software Engineering: Benefits, Challenges, and Future Prospects. *International Journal of Advanced Research in Computer Science*, 2021. 12(2), 45–51. https://doi.org/10.26483/ijarcs.v12i2.

13. Vaithilingam P., Chen J., Alvarado C. Expectations vs. Reality: How Software Developers Use Generative AI Tools for Code. Proceedings of the 2022 ACM CHI Conference on Human Factors in Computing Systems, 2022. 1–15. https://doi.org/10.1145/3491102.3517489.

14. Wang Y., Xu L. "Overfitting in Machine Learning Models: Approaches to Detection and Mitigation." *Journal of Artificial Intelligence Research*, 2022. 68(1), 45–63.

15. Zaremba W., Sutskever I. Recurrent Neural Network Models for Code Generation. Advances in Neural Information Processing Systems (NeurIPS), 2020. 201–213.

16. Zhang Y., Wang X. "Artificial Intelligence and Its Impact on Software Development." *Journal of Software Engineering and Applications*, 2020. 13(4), 233–245. https://doi.org/10.4236/jsea.2020.134014.