УДК 000.42
В. Кохан
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ОГЛЯД НАУКОВИХ ПІДХОДІВ ДО ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ

UDC 000.42
V. Kokhan

OVERVIEW OF SCIENTIFIC APPROACHES TO PROGRAM CODE GENERATION

У сучасному світі складно уявити своє життя без комп'ютера чи мобільного телефону, на які ми покладаємось для вирішення безлічі буденних проблем. Важливим аспектом такої взаємодії виступають додатки, програми та вебсайти, зазвичай створені однойменними компаніями та бізнесами для зручнішого надання послуг своїм клієнтам. До іншої категорії можна віднести програми загального вжитку, як калькулятори та календарі, а також програми написані ентузіастами для вирішення конкретної, менш поширеної проблеми. З появою великих мовних моделей та нового терміну, що описує їх використання для написання програмного коду - вайб-кодингу (vibecoding)[1], поріг входу для написання таких програм сильно знизився. Проте існує більше ніж один підхід до автоматизованого створення програмних продуктів.

Одним із таких підходів є генерація коду на базі шаблонів, TBCG (Template Based Code Generation). Основна ідея цього підходу, що зародився у середині 1990 років [2], полягає в одноразовому написанні шаблонів, які потім можуть генерувати багато програмних продуктів на виході, залежно від вхідних даних. Найширшого застосування такий підхід знайшов у середовищах розробки вебсайтів для генерування різних вебсторінок по одному шаблону та інструментах автоматизованої розробки програмного забезпечення, CASE (Computer-Aided Software Engineering) для генерації коду по наданих UML діаграмах [2].

Іншим підходом, що з'явився у 1964 році та був обґрунтований у 1985 роках [3, 4], є підхід на основі еволюційних алгоритмів - генетичне програмування. У такому підході основним є процес пошуку оптимальних рішень, код еволюціонує через мутації. Існує 3 способи написання програмного коду за підходом генетичного програмування. Це спосіб заснований на стеках (stack-based GP) [4], у якому використовуються окремі стеки для програмних інструкцій та даних. При виконанні інструкції, згідно з її опису, зі стека даних отримуються дані, якщо вони є, а потім відбувається перехід до наступної інструкції. Ідея способу що керується граматикою (grammar-guided GP)[4] полягає в застосуванні формальних граматик для обмеження допустимих конструкцій мови та встановлення правил формування інструкцій [5]. Третій спосіб застосування підходу генетичного програмування це лінійний спосіб. Такий спосіб подібний до написання програм мовою асемблера, де дані знаходяться у регістрах, а їх опрацюванням займаються прості функції [4].

Підхід архітектури керованої моделлю, MDA (Model Driven Architecture), запропонований у 2001 році організацією Object Management Group [6] дозволяє описати весь життєвий цикл розробки програмного продукту. Модель у даному контексті це формальна специфікація функцій, структури та поведінки системи в заданому контексті [7]. Така модель може бути представлена у вигляді зображень та текстів, написаних за стандартами до списку яких входять такі як: UML (Unified Modeling Language), MOF (Meta-Object Facility), CWM (Common Warehouse Metamodel) та інші [7, 8]. Опис починається з моделі незалежної від обчислень, CIM (Computation Independent Model) [7]. Ця модель описує логіку функціонування програми згідно з поставленим завданням приховуючи будь-які технічні реалізації. Ця модель використовується для подолання розриву між замовниками та виконавцями замовлення [7]. Наступною моделлю є платформонезалежна модель, PIM (Platform Independent Model) [7], у якій зібрані ключові моменти програми. Тут вже є певні технічні нюанси впровадження програми, але вони знаходяться на рівні абстракцій. Останньою формується платформозалежна модель, PSM (Platform Specific Model) [7]. У ній враховуються та описуються усі деталі необхідні для створення системи на обраній платформі, та генерується програмний код.

У 2017 році з'явився новий підхід, підхід глибокого навчання (Deep Learning) із застосуванням архітектури трансформерів (Transformers)[9]. Цей підхід було запропоновано у роботі під назвою «Увага - це все що вам треба» (Attention Is All You Need), написаній командою Google [9]. У запропонованій архітектурі вирішили замінити рекурентні нейронні мережі, RNN (Recurrent Neural Network) на механізм уваги (attention). Така зміна дозволила точніше обчислювати ваги для кожного слова так робити ці обчислення паралельно, надаючи однаковий доступ до будь-якої частини речення, на противагу RRN, які віддають перевагу новішим даним[9]. Основна ідея полягає в моделюванні через енкодер і декодер, де увага обчислює ваги взаємодій між елементами. Енкодер обробляє вхідні дані, як опис завдання, чи вже згенерована частина коду, а декодер поступово генерує вихідний код [10].

Незадовго після появи підходу з використанням трансформерів, з'явився підхід із використанням великих мовних моделей, LLM (Large Language Models) [11]. Такі моделі побудовані на основі моделей глибокого навчання з багатьма вхідними параметрами (мільйонами чи навіть мільярдами), натренованими на великій кількості даних. Також у контексті LLM можна почути абревіатуру GPT – generative pre-trained transformers (генеративний попередньо тренований трансформер) [12], що пояснює зв'язок цього підходу та підходу заснованого на трансформерах, а також дозволяє зрозуміти виникнення назви відомого чатбота ChatGPT від компанії OpenAI.

Останнім на сьогодні підходом до генерації програмних продуктів є підхід на базі агентних систем з LLM. Цей підхід поєднує LLM та так званих агентів, що можуть імітувати дії людини у контексті розробки програм. Цей підхід є відносно новим та стрімко розвивається, тому залежно від конкретної обраної реалізації агентивної системи - вона може самостійно, ітеративно аналізувати вимоги, писати код, проводити його тестування, та виправлення помилок [13]. Такі системи також можуть виконувати додаткові дії з інструментами у локальному середовищі розробки, де вони запущені, що значно збільшує їхню потенційну користь та зручність для кінцевого користувача.

Згідно із заявою гендиректора компанії Google у 3-му кварталі 2024 року, більше ніж 25% нового коду в компанії генерується за допомогою штучного інтелекту [14] (підхід із використанням великих мовних моделей та підхід на базі агентних систем), що допомагає прискорити та спростити цей процес. Оглянувши 6 підходів до генерації програмних продуктів стає очевидним що цей напрямок стрімко розвивається та за останні менш ніж 100 років еволюціонував від традиційних шаблонних та модельно орієнтованих підходів, що забезпечують контроль та структуру, до еволюційних алгоритмів оптимізації, і нарешті до сучасних технологій на базі глибокого навчання, великих мовних моделей та агентних систем. Ці підходи поступово знижують рівень входу, та роблять процес створення програмних продуктів доступнішим як для професійних розробників, так і для початківців у сфері.

Література

1. Вайб-кодинг – Вікіпедія. URL: https://uk.wikipedia.org/wiki/%D0%92%D0%B0%D0%B9%D0% B1-%D0%BA%D0%BE%D0%B4%D0%B8%D0%BD%D0%B3 (дата звернення: 01.12.2025).
2. Syriani E., Luhunu L., Sahraoui H. Systematic mapping study of template-based code generation. Computer Languages, Systems & Structures. 2018. Т. 52. С. 43–62. ISSN 1477-8424. URL: https://doi.org/10.1016/j.cl.2017.11.003.
3. Генетичне програмування – Вікіпедія. URL: https://uk.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F (дата звернення: 01.12.2025).
4. Sobania D., Schweim D., Rothlauf F. A Comprehensive Survey on Program Synthesis With Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation. 2023. Т. 27. С. 82–97. URL: https://doi.org/10.1109/TEVC.2022.3162324.
5. Ratle A., Sebag M. Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery. Parallel Problem Solving from Nature PPSN VI. 2000. Т. 1917 : Lecture Notes in Computer Science. URL: https://doi.org/10.1007/3-540-45356-3_21.

6. Sebastián G., Gallud J. A., Tesoriero R. Code generation using model driven architecture: A systematic mapping study. Journal of Computer Languages. 2020. T. 56. ISSN 2590-1184. URL: https://doi.org/10.1016/j.cola.2019.100935.
7. Truyen F. The Basics of Model Driven Architecture (MDA). Cephas Consulting Corp. 2006. The Fast Guide to Model Driven Architecture. URL: https://www.omg.org/mda/mda_files/Cephas_MDA_Fast_Guide.pdf.
8. Model-driven architecture – Wikipedia. URL: https://en.wikipedia.org/wiki/Model-driven_architecture (дата звернення: 01.12.2025).
9. Трансформер (архітектура глибокого навчання) – Вікіпедія. URL: https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80_(%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B3%D0%BB%D0%B8%D0%B1%D0%BE%D0%BA%D0%BE%D0%B3%D0%BE_%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F) (дата звернення: 01.12.2025).
10. Attention is All you Need / A. Vaswani та ін. Advances in Neural Information Processing Systems. 2017. T. 30. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd__053c1c4a845aa-Paper.pdf.
11. Велика мовна модель – Вікіпедія. URL: https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%BB%D0%B8%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C (дата звернення: 01.12.2025).
12. Generative pre-trained transformer – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Generative_pre-trained_transformer (дата звернення: 01.12.2025).
13. A Survey on Code Generation with LLM-based Agents / Y. Dong та ін. ArXiv. 2025. URL: https://doi.org/10.48550/arXiv.2508.00083.
14. Peters J. Oct 29, 2024. More than a quarter of new code at Google is generated by AI | The Verge. 29.10.2024. URL: https://www.theverge.com/2024/10/29/24282757/google-new-code-generated-ai-q3-2024 (дата звернення: 01.12.2025).