УДК 004.312.26 **DOI:** 10.31673/2412-9070.2024.013840

В. О. МИКОЛАЄНКО, аспірант;
К. П. СТОРЧАК, доктор техн. наук, професор,
Державний університет інформаційно-комунікаційних технологій, Київ

МЕТОДИ ОЦІНЮВАННЯ КОДОГЕНЕРАТОРІВ

Кодогенератори — це спеціалізовані програми або сервіси, які автоматично створюють код для різноманітних завдань. Вони сприяють підвищенню продуктивності програмування, спрощують розроблення та допомагають у вивченні нових мов програмування. У статті досліджено основні методи та підходи до оцінювання кодогенераторів. Визначено головні критерії для вибору найефективніших методів.

Ключові слова: кодогенератори; програмування; автоматизація; мови програмування; класифікація.

Вступ

У сучасному світі IT кодогенератори відіграють важливу роль, допомагаючи розробникам автоматизувати рутинні завдання та підвищити ефективність розроблення. У цій статті розглядаються основні методи та підходи до оцінювання кодогенераторів. Кодогенератори виступають у ролі спеціальних програм або сервісів, які автоматично створюють код для різних цілей. Вони можуть бути корисними для підвищення продуктивності програмування, спрощення розроблення або навчання нових мов програмування. Однак, що таке кодогенератор і як він працює? Кодогенератор — це інструмент або ресурс, який створює певний вид коду або мови програмування. Це має багато конкретних значень у світі IT, більшість з яких пов'язано зі складними процесами перетворення синтаксису людського програмування на машинну мову, яку може читати обчислювальна система.

Кодогенератор може виконувати різні завдання, зокрема аналізувати вихідний код та створювати проміжне подання (IR) з нього, здійснювати оптимізації на IR для створення цільового машинного коду та генерувати зовнішні подання (OR) для програм, що використовуються під час налагодження або тестування.

Основна частина

Аналіз дослідження. Кодогенератори можуть бути класифіковані за певними критеріями:

• *джерелом вхідних даних*: кодогенератори можуть приймати різні типи даних як вхід, зокрема природну мову, графічний інтерфейс, схему бази даних або наявний код. Наприклад, pix2code — це кодогенератор, який перетворює знімок екрана графічного інтерфейсу на код;

• *мовою вихідного коду*: кодогенератори можуть генерувати різні мови програмування або мови позначення. Наприклад, NL2SQL — це кодогенератор, який перетворює запит природною мовою на SQL-запит;

• *типом застосування*: кодогенератори можуть мати різну мету або сферу застосування. Наприклад, CodeBERT — це кодогенератор, який можна використовувати для перекладу коду між різними мовами програмування або для генерації коментарів до коду.

Одна з можливих класифікацій базується на трьох парадигмах кодогенерації: опис-до-коду, код-до-опису та код-до-коду. У парадигмі опис-до-коду кодогенератор отримує опис задачі природною або графічною мовою та створює вихідний код мови програмування, який виконує цю задачу. Опис може бути у формі текстового запиту, специфікації функції, сценарію використання або навіть ескізу екрана.

У парадигмі код-до-опису кодогенератор створює природномовний або графічний опис вихідного коду для документації або пояснення. Наприклад, кодогенератор може створювати коментарі до функцій Python з їх сигнатур.

У парадигмі код-до-коду кодогенератор перетворює вихідний код однієї мови програмування на вихідний код іншої мови програмування або модифікує вихідний код тієї самої мови програмування. Наприклад, кодогенератор може перекладати код Java у код C# або виправляти помилки у коді Python.

Метою цього дослідження є вивчення різних методів оцінювання кодогенераторів, їх переваг та недоліків, а також визначення критеріїв для вибору найефективніших методів.

Результати дослідження. Кодогенератори можуть мати різні цілі та застосування, наприклад:

• полегшення процесу розроблення програмного забезпечення та підвищення продуктивності програмістів;

• автоматизація рутинних та низькорівневих завдань із написання коду;

• підвищення якості та надійності коду завдяки використанню перевірених шаблонів та стандартів;

• забезпечення сумісності та портативності коду між різними платформами та мовами програмування;

• сприяння навчанню та освіті з програмування.

© В. О. Миколаєнко, К. П. Сторчак, 2024

Обговорення результатів проведеного дослідження. У процесі правильного використання кодогенератори можуть значно підвищити продуктивність розроблення та якість коду. Однак важливо розглянути та оцінити певні критерії під час вибору кодогенератора:

• *функціональна правильність*: наскільки точно та повно код виконує задану специфікацію або вимогу;

• *синтаксична правильність*: наскільки валідний та безпомилковий код з погляду синтаксису мови програмування;

• *стиль та читабельність*: наскільки зрозумілий та організований код для людей; чи використовуються відповідні імена змінних, коментарі, вирівнювання тощо;

• *ефективність та продуктивність*: наскільки швидко та економно код виконується на цільовому пристрої або в середовищі, чи оптимальний код щодо використання ресурсів;

• *адаптивність та гнучкість*: наскільки легко код може бути модифікований або розширений для задоволення нових потреб або умов.

Метод оцінювання кодогенератора, де на вхід надходить деякий контекст x, а на виході маємо потрібний кодогенератор y, x => codgen score method => codgen (some type of codegen), полягає в тому, що ми використовуємо функцію оцінювання codgen score method, яка приймає на вхід контекст x і потенційний кодогенератор y та повертає числову оцінку, яка відображає якість коду, що генерується y за допомогою x. Функція оцінювання може бути побудована на основі різних критеріїв, зокрема функціональній правильності, синтаксичній правильності, стилю та читабельності, ефективності та продуктивності, адаптивності та гнучкості тощо. Для реалізації функції оцінювання можуть бути використані такі методи:

• ***експертне оцінювання:*** залучаються люди-експерти зі сфери програмування, які мають достатній досвід та знання для оцінювання кодогенераторів. Їх просять переглянути код, що генерується y за допомогою x, та надати свою оцінку за певною шкалою або категоріями. Далі оцінювання різних експертів усереднюють або агрегують для здобуття кінцевого результату;

• ***автоматичне оцінювання:*** розробляється комп'ютерна програма, яка може автоматично аналізувати код, що генерується y за допомогою x, та надавати свою оцінку за певною формулою або метрикою. Є можливість використовувати різні техніки для реалізації програми, зокрема: статичний аналіз коду, динамічне виконання коду, машинне навчання тощо.

Мета цього методу полягає в тому, щоб знайти найкращий кодогенератор y для даного контексту x, тобто такий, котрий максимізує функцію оцінювання codgen score method. Тут можуть бути застосовні різноманітні стратегії для пошуку найкращого кодогенератора y, наприклад:

• ***порівняльний аналіз:*** вибираємо кілька наявних кодогенераторів y_1, y_2, ..., y_n та застосовуємо їх до контексту x. Для порівняння якості коду, що генерується кожним із них, використовуємо функцію оцінювання codgen score method. Потім вибираємо той кодогенератор, який має найвищу оцінку;

• ***генетичний алгоритм:*** ініціалізуємо випадкову популяцію кодогенераторів y_1, y_2, ..., y_n та застосовуємо їх до контексту x. Використовуємо функцію оцінювання codgen score method для визначення пристосованості кожного з них. Застосування генетичних операцій, зокрема схрещування, мутації, селекції тощо дає змогу створювати нові покоління кодогенераторів. Цей процес повторюємо доти, доки не буде досягнуто заданого критерію зупинки, наприклад максимальної кількості поколінь або максимальної оцінки. Далі вибираємо найкращий кодогенератор y з останнього покоління.

Висновки

Кодогенератори гравця виконують ключову роль у сучасному розробленні програмного забезпечення. Вони пропонують низку переваг, зокрема автоматизацію, підвищення продуктивності та якості коду. Однак важливо вибрати правильний кодогенератор на основі конкретних потреб та критеріїв.

Список використаної літератури

1. ***What*** *is a Code Generator? - Definition from Techopedia.*

2. ***Code*** *Generation Using Machine Learning: A Systematic Review // IEEE Journals & Magazine | IEEE Xplore.*

3. ***CodeBERT****: A Pre-Trained Model for Programming and Natural Languages / J. Zhang, Y. Wang, G. Li, Z. Jin // Proceedings of the 28th International Joint Conference on Artificial Intelligence. 2020. P. 4613–4619. AAAI Press.*

4. ***Codexglue:*** *A machine learning benchmark dataset collection for code understanding and generation / W. Lu, D. Chen, X. Wan [et al.] // 2020. arXiv preprint arXiv:2012.15644.*

5. ***Pangucoder:*** *A large-scale pretrained model for code generation / Z. Yang, S. Chen, C. Gao [et al.] // 2021. arXiv preprint arXiv:2109.06370.*

6. ***Deep*** *code comment generation / X. Hu, G. Li, X. Xia [et al.] // Proceedings of the 26th Conference on Program Comprehension. 2018. P. 200–210.*

7. ***code2seq:*** *Generating sequences from structured representations of code / U. Alon, M. Zilberstein, O. Levy, E. Yahav // Proceedings of the International Conference on Learning Representations. 2019.*

8. ***code2vec:*** *Learning distributed representations of code / U. Alon, S. Brody, O. Levy, E. Yahav // Proceedings of the ACM on Programming Languages, 2019. 3(POPL). P. 1–29.*

9. ***Transcoder:*** *Neural transcompiler system for source code migration / M. Chen, C. J. Maddison, M. Arjovsky, G. Lample // 2020. arXiv preprint arXiv:2006.03511.*

10. ***Yin P., Neubig G.*** *A syntactic neural model for general-purpose code generation // Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. 2018. Vol. 1: Long Papers. P. 440–450.*

11. ***Mapping*** *language to code in programmatic context / S. Iyer, I. Konstas, A. Cheung, L. Zettlemoyer // Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. 2018. P. 1643–1652.*

V. Mykolaienko, K. Storchak

METHODS OF EVALUATING CODE GENERATORS

In the ever-evolving landscape of software development, code generators have emerged as vital instruments, fostering innovation and efficiency. These automated maestros translate abstract requirements into functional code, significantly reducing development time and bridging gaps across varied programming environments. This article categorizes code generators based on their operational domain, input specificity, and the nature of the output they produce, unraveling a diverse spectrum from foundational scaffolding creators to sophisticated, language-specific assistants. Selecting an apt code generator is pivotal to the success of a project. To assist developers in this critical choice, the article presents an evaluation method that weighs essential factors such as adaptability, efficiency, and ease of integration with existing systems. This systematic approach ensures the selection of a tool that not only meets immediate project needs but also supports future scalability and complexity. The discussion extends beyond functionality to explore the transformative role of code generators. By automating routine coding tasks, they allow developers to focus on more creative aspects of software design, enhancing code quality and fostering a culture of innovation. The generators thus become indispensable allies in the creative process of software development. In essence, the article advocates for an informed selection of code generators, viewing them as cornerstones for modern coding practices. The proposed evaluation method serves as a strategic framework, empowering developers to navigate the myriad of options and align their choice with both current requirements and future aspirations. It is an invitation to embrace the synergy between human creativity and machine precision, crafting code that is both efficient and elegant.

Keywords: code generators; programming; automation; programming languages; classification.