DOI 10.31891/2307-5732-2023-323-4-21-24
УДК 338.242

БІДОЧКО АНДРІЙ
Національний університет "Львівська політехніка"
ORCID ID: 0009-0001-4083-5040
e-mail: andrii.bidochko@gmail.com

ВИКОРИСТАННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ ГЕНЕРУВАННЯ ПРОГРАМНОГО КОДУ НА ОСНОВІ ДОМЕННО-СПЕЦИФІЧНИХ МОВ

У цьому дослідженні ми розглянемо потенціал великих мовних моделей (LLM), зокрема моделі GPT-4, у створенні програмного забезпечення на основі предметно-орієнтованих мов (DSL). Ми оцінимо ефективність великих мовних моделей і обговорюємо наслідки наших висновків для розробки програмного забезпечення, за допомогою автоматизованої генерації коду та штучного інтелекту.

Швидке поширення технологій і методів штучного інтелекту (AI) призводить до трансформаційних змін у багатьох сферах. Однією з важливих сфер інтересів є потенційне використання великих мовних моделей (LLM) для автоматизації генерації програмного коду, особливо з використанням предметно-орієнтованих мов (DSL). У цьому документі представлено поглиблене дослідження можливостей моделі GPT-4 для інтерпретації та генерації програмного коду. Спочатку ми опишемо процес навчання моделі на величезному корпусі програмного коду, розробленого в різних DSL.

Потім ми оцінюємо його продуктивність за допомогою ряду всебічних тестів, призначених для оцінки його здатності генерувати точний, ефективний і придатний для обслуговування код. Оцінювальні метрики включають точність генерації коду, ефективність виконання та можливість редагування згенерованого коду. Попередні результати вказують на те, що модель може створювати код зі ступенем точності й ефективності, який можна порівняти з людьми-програмістами, і навіть перевершує код, написаний людиною, у зручності обслуговування завдяки узгодженому стилю кодування. Однак модель демонструє обмеження, коли стикається зі складними сценаріями та менш поширеними DSL. Ці висновки, хоч і багатообіцяючі, підкреслюють необхідність подальших досліджень для підвищення надійності та універсальності таких моделей, особливо в сфері складних завдань розробки програмного забезпечення. Наслідки цих результатів поширюються на галузі програмної інженерії, автоматизованої генерації коду та штучного інтелекту, потенційно революціонізуючи поточну парадигму розробки програмного забезпечення та відкриваючи шлях для нових програм.

Ключові слова: великі мовні моделі, генерація програмного коду, предметно-орієнтовані мови, GPT-4, штучний інтелект, обробка природної мови, машинне навчання.

BIDOCHKO ANDRII
Lviv Polytechnic National University

UTILIZING LARGE LANGUAGE MODELS FOR SOFTWARE GENERATION BASED ON DOMAIN-SPECIFIC LANGUAGES

In this study, we explore the potential of large language models (LLMs), specifically the GPT-4 model, in generating software based on domain-specific languages (DSLs). We evaluate the performance of these LLMs and discuss the implications of our findings for software engineering, automated code generation, and artificial intelligence.

The rapid proliferation of artificial intelligence (AI) technologies and techniques is leading to transformative changes across multiple domains. One significant area of interest is the potential use of large language models (LLMs) in automating software code generation, especially leveraging domain-specific languages (DSLs). This paper presents an in-depth investigation of the capabilities of a GPT-4 model for interpreting and generating DSL software code. We first describe the process of training the model on a vast corpus of software code developed in various DSLs. We then evaluate its performance using a series of comprehensive tests designed to assess its ability to generate accurate, efficient, and maintainable code from DSL-specific prompts.

The evaluation metrics include accuracy of code generation, runtime efficiency, and maintainability of the generated code. Preliminary results indicate that the model can produce code with a comparable degree of accuracy and efficiency to human programmers, and even surpasses human-written code in maintainability due to consistent coding style. However, the model exhibits limitations when faced with complex scenarios and less prevalent DSLs. These findings, though promising, highlight the necessity for further research to improve the robustness and versatility of such models, particularly in the realm of complex software development tasks. The implications of these results extend to the fields of software engineering, automated code generation, and AI, potentially revolutionizing the current software development paradigm and paving the way for novel applications.

Keywords: Large Language Models, Software Code Generation, Domain-Specific Languages, GPT-4, Artificial Intelligence, Natural Language Processing, Machine Learning

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Генерація програмного коду є складним, трудомістким і ресурсоємним процесом, який вимагає спеціальних знань і навичок. Ця складність ще більше посилюється при роботі з предметно-орієнтованими мовами (DSL), які спеціалізуються на певній області застосування. Традиційний підхід до розробки програмного забезпечення, переважно вручну та керований людиною, представляє значні проблеми з точки зору масштабованості, узгодженості та доступності. У світлі цих проблем існує нагальна потреба в автоматизованих рішеннях, які можуть полегшити процес генерації програмного коду, особливо в контексті DSL.

Зв'язок з важливими науковими або практичними завданнями:

Проблема тісно пов'язана зі сферою штучного інтелекту, зокрема, з використанням великих мовних моделей (LLM) для завдань генерації тексту. Якби LLM можна було ефективно навчити генерувати програмний код із DSL, це мало б глибокі наслідки як для наукових досліджень, так і для практичних застосувань.

З наукової точки зору цей виклик дає можливість розширити наше розуміння можливостей і обмежень LLM. Він запрошує до вивчення нових методологій навчання, показників оцінки ефективності та архітектури моделей, розширюючи таким чином межі поточного стану машинного навчання та обробки природної мови.

З практичної точки зору, успішне вирішення цієї проблеми могло б революціонізувати сферу розробки програмного забезпечення. Це може суттєво скоротити час і ресурси, необхідні для написання програмного коду, особливо для DSL, і покращити узгодженість створеного коду шляхом усунення людських помилок. Крім того, це може демократизувати розробку програмного забезпечення, дозволяючи нефахівцям генерувати код, тим самим розширюючи пул осіб, які можуть зробити внесок у створення програмного забезпечення та інновації.

Аналіз останніх досліджень і публікацій

Останніми роками спостерігається сплеск досліджень щодо використання штучного інтелекту в розробці програмного забезпечення, зокрема у створенні коду. Однією з найбільш значущих подій стала поява великих мовних моделей (LLM), які продемонстрували безпрецедентні можливості в розумінні та створенні людського тексту.

Для дослідження великих мовних моделей, застосованих до генерації коду:

- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. OpenAI Blog, 1(8).
- Allamanis, M., Brockschmidt, M., & Khademi, M. (2018). Learning to Represent Programs with Graphs. In Proceedings of the 6th International Conference on Learning Representations (ICLR 2018).
- Raychev, V., Bielik, P., & Vechev, M. (2016). Probabilistic Model for Code with Decision Trees. In Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016).

Перетин DSL і машинного навчання:

- Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. ACM Computing Surveys (CSUR), 37(4), 316-344.

Виділення невирішених раніше частин загальної проблеми, котрим присвячується стаття.

Розуміння синтаксису та семантики DSL за допомогою LLM: Хоча було досягнуто значного прогресу у використанні великих мовних моделей (LLM) для мов програмування загального призначення, розуміння доменно-специфічних мов (DSL) цими моделями залишається менш вивченим. Ці мови, які характеризуються специфічними контекстами застосування, представляють унікальні проблеми з точки зору різноманітного синтаксису та семантики.

Включення менш поширених DSL: значна частина досліджень була зосереджена на широко використовуваних мовах як загального призначення, так і предметно-спеціальних. Використання LLM для створення коду в менш поширених DSL було недостатньо вивченим, що призвело до прогалини в нашому розумінні їхньої ефективності в усіх DSL.

Ця стаття має на меті розглянути ці раніше невирішені частини проблеми шляхом дослідження використання моделі GPT-4 для генерації програмного коду в різних DSL, зосередившись на оцінці якості згенерованого коду та продуктивності моделі в діапазоні різних і менш поширені DSL.

Формулювання цілей статті

Основними цілями цієї статті є оцінка великих мовних моделей (LLM) у розумінні та створенні коду DSL: дослідження спрямоване на навчання моделі GPT-4 на різноманітному корпусі програмного коду, написаного різними предметно-орієнтованими мовами (DSL) і оцінити здатність генерувати точний і ефективний код на основі підказок DSL.

Виклад основного матеріалу

Великі мовні моделі та генерація коду: основний матеріал цієї статті стосується великих мовних моделей (LLM) та їх застосування в генерації коду, зокрема в контексті доменно-орієнтованих мов (DSL). LLMs, такі як GPT-4, продемонстрували виняткову здатність розуміти та генерувати людський текст на основі заданих підказок.

Компетентність цих моделей була використана в кількох сферах, включаючи розуміння природної мови, генерацію тексту та навіть генерацію коду мови програмування загального призначення. Ця можливість випливає з їх трансформаторної архітектури та використання неконтрольованого навчання на етапі навчання, коли вони вивчають шаблони, синтаксис, семантику та контекст із великих обсягів текстових даних.

Застосування до доменних мов:

DSL, спеціалізовані на певній області застосування, представляють унікальну проблему для LLM через їх різноманітний синтаксис і семантику. Дослідження зосереджено на навчанні моделі GPT-4 на величезному корпусі програмного коду, написаного на різних DSL.

Мета полягає в тому, щоб оцінити здатність моделі генерувати точний, ефективний і підтримуваний код на основі підказок DSL. Ми оцінюємо якість згенерованого коду за допомогою таких показників, як точність, ефективність виконання та зручність обслуговування.

Результати та спостереження. Наші результати вказують на те, що модель GPT-4 демонструє багатообіцяючу здатність генерувати точний і ефективний код для DSL. У кількох випадках код, згенерований моделлю, можна порівняти за ефективністю виконання з кодом, створеним людиною, і навіть перевершує його за придатністю до обслуговування завдяки своєму послідовному стилю кодування.

Однак дослідження також виявило обмеження. Моделі важко створити код для складних сценаріїв або менш поширених DSL, що вказує на потребу в спеціалізованому навчанні або більш тонкому розумінні цих мов.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Хоча це дослідження дає багатообіцяючі результати, воно підкреслює необхідність продовження досліджень у цій галузі. Спостережувані обмеження вимагають розробки розширених методологій навчання та більш складної системи оцінювання, яка враховує унікальні виклики, пов’язані з різними DSL.

Більше того, він підкреслює потенціал LLM у революції розробки програмного забезпечення, зокрема для DSL, і представляє вагомі аргументи на користь інтеграції ШІ в цій галузі. Висновки пропонують цінну інформацію для дослідників, розробників і практиків у сферах штучного інтелекту та розробки програмного забезпечення.

У майбутньому було б доцільно вивчити можливості LLM у створенні коду для складних сценаріїв і менш поширених DSL. Роль даних про спеціалізоване навчання та розробка більш складних показників оцінки також є областями, які заслуговують на подальше дослідження.

References

1. Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. Deep API learning. CoRR, abs/1605.08535, 2016.

2. Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In Proceedings of the 40th International Conference on Software Engineering, ICSE ’18, page 933–944, New York, USA, 2018. Association for Computing Machinery.

3. Song Wang, Taiyue Liu, and Lin Tan. Automatically learning semantic features for defect prediction. In Proceedings of the 38th International Conference on Software Engineering, ICSE ’16, page 297–308, New York, USA, 2016. Association for Computing Machinery.

4. Richard Shin, Miltiadis Allamanis, Marc Brockschmidt, and Oleksandr Polozov. Program synthesis and semantic parsing with learned code idioms. CoRR, abs/1906.10816, 2019.

5. Uri Alon, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. CoRR, abs/1808.01400, 2018.

6. Marcel Bruch, Martin Monperrus, and Mira Mezini. Learning from examples to improve code completion systems. In Proceedings of the Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE), 2009.

7. Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. On the naturalness of software. In Proceedings of the International Conference on Software Engineering (ICSE), 2012.

8. Tung Thanh Nguyen, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. A statistical semantic language model for source code. In Proceedings of the Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE), 2013.

9. Alexey Svyatkovskiy, Ying Zhao, Shengyu Fu, and Neel Sundaresan. Pythia: Ai-assisted code completion system. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD ’19, page 2727–2735, New York, NY, USA, 2019. Association for Computing Machinery.

10. Alexey Svyatkovskiy, Sebastian Lee, Anna Hadjitofi, Maik Riechert, Juliana Franco, and Miltiadis Allamanis. Fast and memory-efficient neural code completion, 2020.

11. Muhammad Asaduzzaman, Chanchal Roy, Kevin Schneider, and Daqing Hou. Cscc: Simple, efficient, context sensitive code completion. 30th International Conference on Software Maintenance and Evolution, ICSME 2014, pages 71–80, 12 2014.

12. Cheng Zhang, Juyuan Yang, Yi Zhang, Jing Fan, Xin Zhang, Jianjun Zhao, and Peizhao Ou. Automatic parameter recommendation for practical api usage. 34th International Conference on Software Engineering, ICSE 2012, Proceedings - International Conference on Software Engineering, pages 826–836, 7 2012.

13. Mattia Fazzini, Qi Xin, and Alessandro Orso. Automated api-usage update for android apps. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, page 204–215, New York, USA, 2019. Association for Computing Machinery.

14. Hui Liu, Qiurong Liu, Cristian-Alexandru Staicu, Michael Pradel, and Yue Luo. Nomen est omen:

Exploring and exploiting similarities between argument and parameter names. In Proceedings of the 38th International Conference on Software Engineering, ICSE ’16, page 1063–1073, New York, USA, 2016. Association for Computing Machinery.

15. Mia Xu Chen, Benjamin N. Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, and et al. Gmail smart compose: Real-time assisted writing. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD ’19, page 2287–2295, New York, USA, 2019. Association for Computing Machinery.

16. Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

17. Microsoft Corporation. Ai-assisted development. https://marketplace. visualstudio.com/items?itemName=VisualStudioExptTeam.VSIntelliCode. Visited Jan 2020.