DOI: 10.34185/1991-7848.itmm.2025.01.110

МЕТОДИ ПОКРАЩЕННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ ПОКРАЩЕННЯ ЯКОСТІ РЕФАКТОРИНГУ КОДУ

Сирота О.А.[1], Горячкін В.М.[2]

[1]Український державний університет науки і технологій, аспірантУкраїна

[2]Український державний університет науки і технологій,[2]к.т.н., доц., Україна

Анотація. *У даному досліджені розглядаються методи покращення великих мовних моделей (LLM) для рефакторингу програмного забезпечення. Використовуючи методи точного налаштування та індексації файлів вихідного коду, розкривається питання покращення результатів використання LLM для задачі рефакторингу кодових баз та покращеного використання контексту мовних моделей. Запропоновані підходи націлені на підвищення якості вихідного коду після рефакторингу, а також покращення різних великих мовних моделей шляхом змін в самій моделі чи інтеграції її з додатковим програмним забезпеченням. Оцінка результатів реалізації методів буде здійснюватися за допомогою показників Code Health та F1-метрики. Це дає змогу визначити ефективність запропонованих рішень. Результати досліджень відкривають нові перспективи для академічних досліджень та ефективного впровадження у проекти різного масштабу.*

Ключові слова: *рефакторинг, розробка, код, LLM, штучний інтелект, машинне навчання, велика мовна модель.*

Сучасний технологічний ландшафт характеризується стрімким розвитком програмного забезпечення, що призводить до зростання обсягів коду та складності проектів. Реалізація якісного програмного забезпечення (ПЗ) вимагає врахування багатьох факторів, таких як компетентність розробників, ефективність проектного менеджменту, доступність ресурсів та зміни вимог до системи. У цьому контексті рефакторинг коду виступає ключовим інструментом покращення внутрішньої структури ПЗ без зміни бізнес-логіки, сприяючи підвищенню читабельності, підтримуваності та зменшенню технічних заборгованостей. Водночас зростання складності проекту та використання застарілого коду підвищують ризики появи помилок та зниження якості продукту.

Одним із сучасних підходів до автоматизації процесу рефакторингу є застосування технологій штучного інтелекту (ШІ) та великих мовних моделей

(LLM), таких як GPT, Gemini, Llama та інших. Проте, наявні дослідження свідчать про те, що застосування стандартних LLM супроводжується низькою ймовірністю успіху, що обумовлено низкою обмежень, зокрема недостатнім розумінням повного контексту проекту та порушенням функціональної відповідності після рефакторингу [1]. Таким чином, важливим завданням є розробка та вдосконалення методів, що підвищать ефективність застосування LLM в контексті рефакторингу програмного коду.

Дослідження базується на комплексному аналізі можливостей застосування LLM у процесі рефакторингу, що включає розгляд як технічних аспектів, так і впливу на загальну архітектуру програмних систем. Основними завданнями які вирішуються є розробка методів, які можуть бути застосовані при тонкому налаштуванні великої мовної моделі та індексації файлів вихідного коду.

Тонке налаштування являє собою використання попередньо навченої моделі та її адаптацію до специфічних завдань рефакторингу шляхом додаткового навчання на спеціалізованих наборах даних. Окрім цього цей процес включає в себе підготовку даних, кероване навчання, навчання з підкріпленням, а також налаштування параметрів моделі, що дозволяє підвищити її ефективність при обробці конкретних кодових баз [2].

Підхід індексації контексту в свою чергу це впровадження механізму індексації файлів з вихідним кодом для збереження інформації про залежності між модулями без необхідності завантаження всього вмісту в контекст [3]. Таким чином це дає змогу LLM краще «розуміти» структуру проекту та забезпечувати ефективне використання доступного контексту, що є критичним для запобігання помилкам, пов’язаним із ізоляцією окремих функцій коду.

Обидва підходи спрямовані на вирішення основних проблем застосування ШІ для рефакторингу, зокрема на забезпечення збереження функціональної відповідності та підвищення якості коду після змін. Ретельна оцінка результатів реалізації методів може здійснюватися за допомогою показників Code Health та F1-метрики, що дозволяють кількісно визначити ефективність запропонованих рішень [4, 5].

Проведений аналіз дозволив виділити кілька ключових аспектів застосування LLM у процесі рефакторингу:

- проблематика контексту - LLM мають обмежений розмір контексту, що ускладнює розуміння повної архітектури проекту. Це призводить до ситуацій, коли ізольована рефакторинг-функція може порушувати роботу суміжних модулів, що критично впливає на функціональність ПЗ [1];
- функціональна відповідність – штучний інтелект часто не здатен гарантувати повну відповідність між початковим кодом та кодом після змін. Навіть незначні помилки, такі як використання неправильного оператора, можуть мати серйозні наслідки для роботи системи [1];
- методи покращення - тонке налаштування LLM дозволяє адаптувати модель до конкретних особливостей проекту, що сприяє підвищенню точності та зниженню ризиків порушення функціональної відповідності. Одночасно індексація контексту забезпечує ефективну обробку великих обсягів коду шляхом збереження важливих залежностей між модулями;
- обчислювальні витрати - основним викликом застосування додаткового навчання є високі обчислювальні вимоги, а також потреба у високоякісних даних для навчання, що визначає успішність адаптації моделі до специфічних завдань.

Висновки. Запропоновані методи вдосконалення LLM для рефакторингу програмного коду мають потенціал до розвитку з кількох причин. По-перше, адаптивність до специфічних проектів – розробка методів тонкого налаштування дозволяє створити вузькоспеціалізовані моделі, здатні ефективно працювати з конкретними кодовими базами, що значно підвищує якість коду та знижує ризик виникнення помилок, що є критичним для сучасних розробницьких процесів. По-друге, оптимізація використання контексту – впровадження механізму індексації контексту сприяє більш ефективному використанню пам’яті та дозволяє моделі враховувати складні залежності між модулями програмного продукту, що забезпечує створення більш стабільних та підтримуваних систем. Нарешті, практична реалізація – результати дослідження мають практичне значення для розробників програмного забезпечення, оскільки пропонують конкретні рекомендації щодо інтеграції ШІ-інструментів у процес розробки та підтримки ПЗ, що дозволяє зменшити технічний борг та підвищити продуктивність розробницьких команд. Таким чином, запропоновані підходи не лише сприяють підвищенню

ефективності процесу рефакторингу, але й розширюють можливості автоматизації складних етапів розробки програмного забезпечення, відкриваючи нові перспективи як для академічних досліджень у сфері застосування ШІ, так і для практичної імплементації у великих ІТ-проектах.

ЛІТЕРАТУРА

1. Codescene. Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements. URL: https://codescene.com/hubfs/whitepapers/Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements.pdf
2. Rishab Ramesh, Akarsh Thejasvi Raju M, Harsha Vardhan Reddy, Sandeep Varma N. Fine-Tuning Large Language Models for Task Specific Data. URL: https://ieeexplore.ieee.org/document/10730913 DOI: 10.1109/ICNEWS60873.2024.10730913
3. Fu Q., Cho M., Merth T., Mehta S., Rastegari M., Najibi M. LazyLLM: DYNAMIC TOKEN PRUNING FOR EFFICIENT LONG CONTEXT LLM INFERENCE. arXiv preprint arXiv:2407.14057, 2024. URL: https://arxiv.org/pdf/2407.14057
4. Codescene. Code Health – How easy is your code to maintain and evolve? URL: https://codescene.io/docs/guides/technical/code-health.html
5. ENCORD. F1 Score in Machine Learning. URL: https://encord.com/blog/f1-score-in-machine-learning/

METHODS FOR IMPROVING LARGE LANGUAGE MODELS TO IMPROVE THE QUALITY OF CODE REFACTORING

Oleksandr Syrota, Horiachkin Vadym,

Abstract. *This study discusses methods for improving large language models (LLMs) for software refactoring. Using the methods of fine-tuning and indexing source code files, the paper addresses the issue of improving the results of using LLM for the task of refactoring code bases and improving the use of the context of language models. The proposed approaches are aimed at improving the quality of the source code after refactoring, as well as improving various large language models by changing the model itself or integrating it with additional software. The results of the methods implementation will be evaluated using Code Health and F1 metrics. This allows us to determine the effectiveness of the proposed solutions. The research results open up new perspectives for academic research and effective implementation in projects of various sizes.*

Keywords: *refactoring, development, code, LLM, artificial intelligence, machine learning, large language model.*

REFERENCE

1. Codescene. Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements. URL: https://codescene.com/hubfs/whitepapers/Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements.pdf
2. Rishab Ramesh, Akarsh Thejasvi Raju M, Harsha Vardhan Reddy, Sandeep Varma N. Fine-Tuning Large Language Models for Task Specific Data. URL: https://ieeexplore.ieee.org/document/10730913 DOI: 10.1109/ICNEWS60873.2024.10730913
3. Fu Q., Cho M., Merth T., Mehta S., Rastegari M., Najibi M. LazyLLM: DYNAMIC TOKEN PRUNING FOR EFFICIENT LONG CONTEXT LLM INFERENCE. arXiv preprint arXiv:2407.14057, 2024. URL: https://arxiv.org/pdf/2407.14057
4. Codescene. Code Health – How easy is your code to maintain and evolve? URL: https://codescene.io/docs/guides/technical/code-health.html
5. ENCORD. F1 Score in Machine Learning. URL: https://encord.com/blog/f1-score-in-machine-learning/