ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)
Наука та прогрес транспорту, 2025, № 1 (109)

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ

УДК 04.4`2:004.8

О. А. СИРОТА[1*], В. М. ГОРЯЧКІН[2]

[1*]Каф. «Комп'ютерні інформаційні технології», Український державний університет науки і технологій, ННІ ДІІТ, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (098) 359 60 01, ел. пошта sirotaalexandr30@gmail.com, ORCID 0000-0001-7391-2471
[2]Каф. «Комп'ютерні інформаційні технології», Український державний університет науки і технологій, ННІ ДІІТ, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (097) 972 52 55, ел. пошта v.m.horiachkin@ust.edu.ua, ORCID 0000-0002-8952-952X

Проблеми рефакторингу програмного коду із застосуванням штучного інтелекту

Мета. Сучасний технологічний ландшафт характеризується стрімким розвитком програмного забезпечення, орієнтованого на різноманітні предметні галузі та платформи. Це зумовлює неперервне створення нових програмних продуктів, що складаються з величезної кількості рядків коду. Процес розробки якісного програмного забезпечення є багатоетапним і містить низку факторів, які впливають на кінцевий результат. Серед ключових аспектів виділяють компетентність розробників, ефективність проєктного менеджменту, доступність необхідних ресурсів та здатність адаптуватися до змінних вимог. Кожна платформа має свої специфічні особливості, які необхідно враховувати під час розробки, що додатково ускладнює процес створення універсальних та ефективних програмних рішень. Наше дослідження має на меті здійснити комплексний аналіз потенціалу та перспективних напрямів застосування великих мовних моделей у контексті рефакторингу програмного коду. Робота спрямована на розробку та вдосконалення методів, які сприятимуть підвищенню ефективності процесу рефакторингу за допомогою цих моделей. **Методика.** Для вирішення вищезазначених проблем запропоновано реалізувати комплекс методів, які можуть бути застосовані як окремо, так і в синергії, з метою оптимізації кінцевого результату. Ці методи, ретельно розроблені в контексті сучасних парадигм програмної інженерії, спрямовані на підвищення ефективності процесу рефакторингу, забезпечуючи при цьому збереження функціональності програмного забезпечення. Їх імплементація передбачає систематичний підхід до аналізу та модифікації кодової бази, враховуючи як технічні аспекти, так і потенційний вплив на загальну архітектуру системи. **Результати.** Проведено комплексний аналіз наявних мовних моделей та розроблено методи підвищення ефективності великих мовних моделей у контексті рефакторингу коду. Виявлено ключові фактори, що впливають на успішність застосування запропонованих методів, зокрема обсяг навчальних даних та обмеження контексту моделі. **Наукова новизна.** Розроблено підхід до підвищення ефективності великих мовних моделей у рефакторингу коду, що враховує специфіку різних проєктів та етапів розробки. Запропоновано інноваційні методи донавчання мовних моделей та оптимізації використання контексту, що розширюють можливості автоматизованого рефакторингу. **Практична значимість.** Результати дослідження дозволяють поліпшити ефективність рефакторингу коду із застосуванням великих мовних моделей.

Ключові слова: програмна інженерія; штучний інтелект; велика мовна модель; рефакторинг; програмний код; якість програмного коду

Вступ

У сучасному світі застосування програм, розроблених для різних операційних систем і платформ, є частиною нашого повсякденного життя. Щодня ми використовуємо застосунки, які підтримують такі операційні системи, як Web, iOS, Android, MacOS, Windows і Linux. Необхідність використання програмного забезпечення, орієнтованого на різні предметні галузі,

doi: https://doi.org/10.15802/stp2025/325888

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)
Наука та прогрес транспорту, 2025, № 1 (109)

стала причиною безперервної появи нових програмних продуктів, що являють собою незліченну кількість рядків коду.

Створення якісного програмного забезпечення (ПЗ) складається з декількох етапів. Різні етапи імплементації програмного продукту містять сукупність факторів, які можуть впливати на процес як позитивно, так і негативно [3]. До таких факторів належать:

– знання та досвід розробників;
– якість проєктного менеджменту;
– наявність необхідних ресурсів;
– часті зміни вимог до програмного забезпечення.

Важливо враховувати також те, що кожній платформі притаманні конкретні особливості ПЗ, які впливають на процес розробки, підтримку певних функцій. Крім того, наявність різних проблем, а також вплив факторів, описаних вище, може призводити до зниження якості вихідного коду, а в результаті – самого програмного засобу. Найбільш поширеними є такі проблеми [2, 7]:

– вихідний код поганої якості;
– недостатня продуктивності програми, яка виражається у повільній роботі інтерфейсу користувача: замалій кількості зображених кадрів за секунду, «зависання»;
– надлишкове використання оперативної пам’яті під час виконання, яке може призводити до екстреного завершення роботи;
– складнощі розуміння наявного коду, що негативно впливає на швидкість реалізації нового функціоналу;
– збільшення кількості розробників у команді недостатньо поліпшує або взагалі не поліпшує швидкість розробки;
– вартість реалізації продукту постійно зростає.

Одним із ефективних інструментів поліпшення якості та внутрішньої структури коду без зміни бізнес-логіки є рефакторинг. Завдяки йому код стає більш читабельним, обслуговування та розширення функцій має меншу складність, усуваються технічні заборгованості, а також підвищується продуктивність. Складність застосування такого інструменту може часто залежати від проєкту. Якщо проєкт досить великий і на додачу має застарілу кодову базу, то це суттєво впливає на складність проведення рефакторингу.

Штучний інтелект (ШІ) з його можливостями надшвидкого предиктивного аналізу та обробки великих обсягів даних можна використовувати, щоб значною мірою автоматизувати процес рефакторингу коду. Підхід на основі штучного інтелекту полягає в тому, що його інструменти проводять аналіз різного за обсягом коду, виявляють архітектурні помилки, порушення загальноприйнятих принципів програмування, такі як повторювані шаблони, та автоматично проводять рефакторинг або пропонують можливі рішення розробнику [7].

Проте, окрім значного приросту в продуктивності, ШІ показав, що має досить вагомі недоліки і труднощі, які можуть трапитись під час його використання.

Однією з найвагоміших проблем використання ШІ є обмежене розуміння повного контексту проєкту. Великі мовні моделі (LLM) мають контекст обмеженого розміру. Розмір контексту впливає на можливість «розуміння» мовною моделлю загальної архітектури проєкту, залежностей між програмними модулями і потенційних наслідків змін у кодовій базі, що робить процес менш ефективним. Рефакторинг ізольованої функції може здаватися цілком задовільним, але може порушити роботу інших модулів через те, що LLM не враховує всіх залежностей. Без цілісного уявлення про проєкт зміни, внесені штучним інтелектом, мають більшу ймовірність помилки або можуть порушити наявну функціональність у складній кодовій базі [7].

Ще одним недоліком є наявність проблеми із забезпеченням функціональної відповідності між початковим програмним кодом та кодом після рефакторингу з використанням LLM [7]. Часто функціональна відповідність порушується на перший погляд непомітною помилкою, наприклад, помилкою може бути використання оператором невідповідної мови програмування. Такі результати роботи LLM призводять до невідповідності роботи системи зазначеним вимогам.

Іншими, але не менш вагомими є такі проблеми [7]:

– ризик створення вразливостей у системі безпеки: інструменти штучного інтелекту можуть створювати вразливості в системі, якщо вони не розроблені з урахуванням певних вимог;

doi: https://doi.org/10.15802/stp2025/325888

– legacy code може бути складним та незрозумілим, що додає ймовірності до того, що розробник помилиться, перевіряючи запропоновані зміни ШІ-помічником.

Мета

Основною метою роботи є аналіз можливостей та перспектив застосування великих мовних моделей до рефакторингу програмного коду, а також розробка методик їх поліпшення для збільшення ефективності рефакторингу.

Методика

До найбільш поширених великих мовних моделей належать GPT, Gemini, Llama, PaLM. Ці мовні моделі мають велику кількість параметрів, різну продуктивність у контексті різних завдань та конкурують між собою. Проте наявні дослідження в галузі рефакторингу коду свідчать, що використання стандартних LLM є досить ризиковим, бо кращому випадку велика мовна модель дає лише 37 % ймовірності успіху (табл. 1).

Таблиця 1

Коректність рефакторингу для низки популярних LLM [4]

Table 1

Refactoring correctness for a number of popular LLMs [4]

Модель ШІ	Правильний код (перевірка синтаксису рефакторингованого коду)	Якість коду поліпшено (чи зміна коду штучним інтелектом поліпшила його)	Правильний рефакторинг (чи проходять тести після того, як ШІ змінив код)
PaLM 2 code	99,93 %	68,75 %	37,29 %
GPT–3.5	100 %	69,89 %	30,26 %
PaLM 2t	100 %	66,54 %	34,73 %
phind-codellama–34B–v2	100 %	78,76 %	18,14 %

Велика кількість помилок є неочевидною для людського ока під час перевірки. ШІ, засноваому на LLM, бракує сурового розуміння «коректності». Він може передбачати кожен наступний токен на основі ймовірності, але не виконує і не перевіряє коду. Враховуючи те, що програмування є набагато більш обмеженим, ніж людський діалог, це може бути проблемою. Так, наприклад, у природній мові вибір одного синоніма замість іншого зазвичай не впливає на суть тексту, однак у програмному коді кожен символ формує функціональність програми. Приклад подібної проблеми показано на рис. 1.

Крім того, наявні дослідження показують, що поліпшення використання контексту LLM має сенс [7].

Для вирішення описаних вище проблем можна реалізувати декілька методів, використаних окремо або в сукупності з іншими для досягнення кращого кінцевого результату.

Як відомо, обсяг вхідних даних під час застосування LLM впливає на якість вихідних даних, що є результатом роботи великої мовної моделі. Але це стосується не лише даних, які зберігаються в контексті, але й тих, на основі яких модель навчалась [5-6]. Одним із методів, який пропоновано використовувати для поліпшення роботи LLM у контексті рефакторингу, є тонке налаштування великої мовної моделі.

Тонке налаштування являє собою процес додаткового навчання попередньо навченої великої мовної моделі (LLM) на нових або додаткових даних із метою її адаптації для виконання специфічних завдань або роботи з наборами даних, що є характерним для певної предметної галузі [1, 5, 6] і відповідає передбачуваному застосуванню.

doi: https://doi.org/10.15802/stp2025/325888

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)
Наука та прогрес транспорту, 2025, № 1 (109)

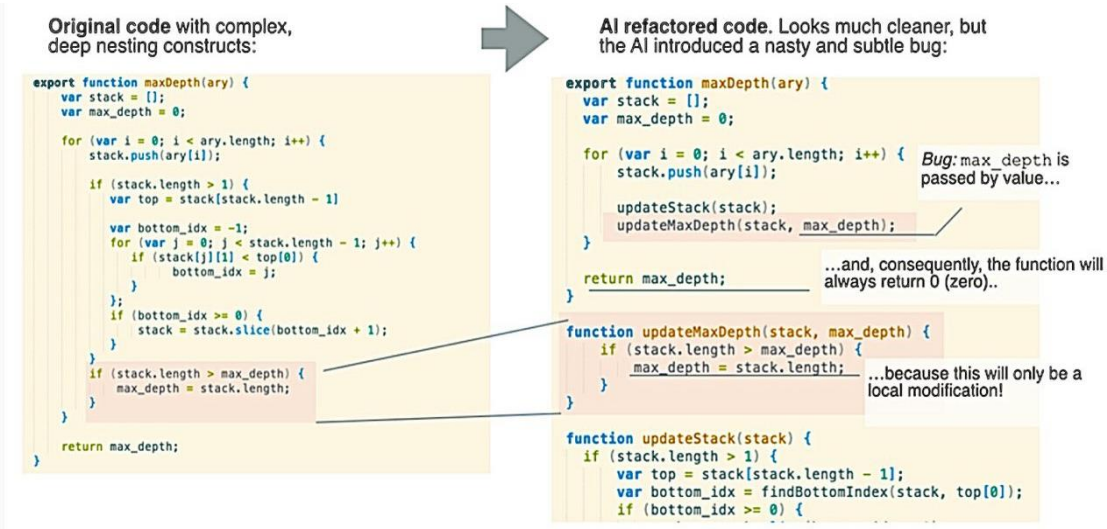

Рис. 1. Малопомітна помилка, створена штучним інтелектом

Fig. 1. A subtle error created by artificial intelligence

Основною метою тонкого налаштування моделі є підвищення її ефективності для виконання конкретних завдань, цього досягають використанням узагальнених знань, здобутих під час початкових етапів навчання, а також нових знань, отриманих на етапі додаткового навчання. Процес тонкого налаштування передбачає такі етапи (рис. 2):

1. Вибір попередньо навченої моделі: починають із використання попередньо навченої великої мовної моделі (LLM), такої як GPT-3 або BERT, яка вже освоїла розуміння мови на основі великого обсягу текстових даних.

2. Підготовка даних: здійснюють збір та підготовку набору даних, що є специфічним для певного завдання або галузі. Цей набір даних має бути ретельно очищений та анотований для забезпечення його репрезентативності для заданої програми.

3. Додаткове навчання, що містить:

3.1. Кероване навчання: модель навчається на основі маркованих даних, що дозволяє їй встановлювати відповідність між вхідними даними та бажаними виходами, при цьому мінімізуючи функцію втрат.

3.2. Навчання з підкріпленням: передбачає надання моделі зворотного зв’язку у вигляді винагороди за генерацію бажаних результатів, що дозволяє оптимізувати її продуктивність на основі отриманого зворотного зв’язку.

3.3. Налаштування параметрів моделі на основі нового набору даних: налаштування може передбачати повне точне налаштування, яке передбачає оновлення всіх шарів моделі. Такий підхід потребує значних обчислювальних ресурсів, але забезпечує всебічну адаптацію.

3.4. Перепрофілювання: під’єднання шару вбудовування моделі до класифікатора, спеціально призначеного для конкретного завдання. Хоча такий метод є менш ресурсоємним, він може не забезпечувати такого ж рівня продуктивності, як повне точне налаштування.

3.5. Оцінка та валідація: після завершення етапу додаткового навчання ефективність моделі підлягає ретельній оцінці з використанням валідаційного набору даних. Це потрібно для підтвердження, що модель досягає необхідної точності та функціональності відповідно до специфікацій конкретного завдання.

3.6. Розгортання: після досягнення задовільних показників роботи модель може бути інтегрована в реальні додатки, де вона здатна генерувати результати, адаптовані до специфічного контексту, для якого була проведена її оптимізація.

Однією з переваг точного налаштування є підвищення продуктивності моделі, оскільки таке налаштування здатне суттєво поліпшити точність і відповідність результатів для визначених завдань. Крім того, додатково навчена LLM стає більш вузькоспеціалізованою. Інакше ка-

doi: https://doi.org/10.15802/stp2025/325888

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)
Наука та прогрес транспорту, 2025, № 1 (109)

жучи, процес тонкого налаштування надає можливість моделі стати експертною у конкретній сфері, у нашому випадку в рефакторингу.

Важливо зазначити, що тонке налаштування потребує значно менше даних та обчислювальних ресурсів порівняно з навчанням моделі від початку, тому є більш ефективним шляхом, ніж навчання моделі від початку [5, 6, 8]. Завдяки тонкому налаштуванню на основі наявного вихідного коду, документації та інших відомостей про програмний засіб, який підлягає рефакторингу, якість результату роботи мовної моделі може значно зрости.

Результати

Запропонований вище алгоритм передбачає додаткове навчання моделі кожного разу, коли нові зміни вихідного коду потрапляють до системи контролю версій (рис. 3). Мовна модель навчається на вже написаному коді, документації тощо. Таким чином, з кожною ітерацією розробки модель починає краще «розуміти» відповідну програмну систему, що впливає на процес генерації коду, роблячи його більш коректним.

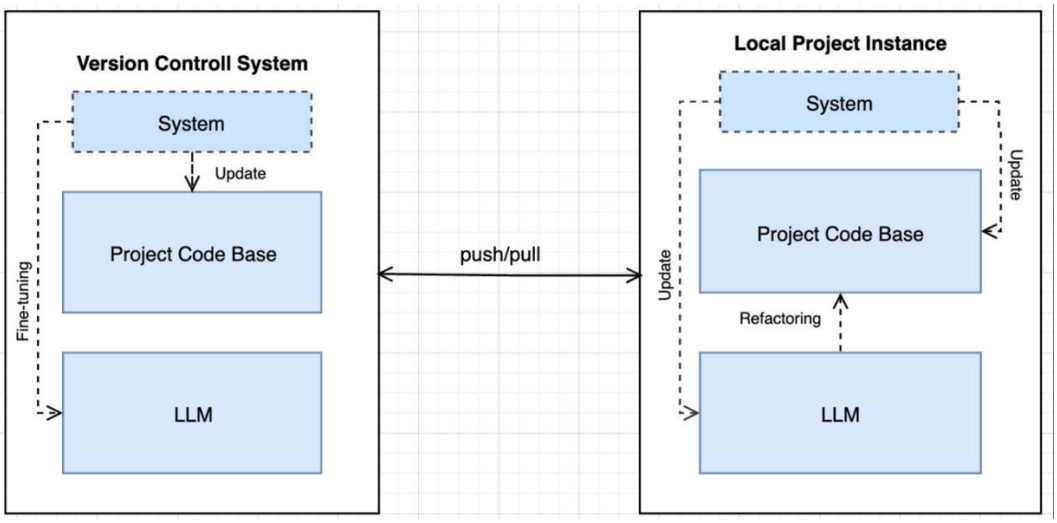

Рис. 2. Процес додаткового навчання LLM у процесі роботи над програмою

Fig. 2. The process of additional LLM training in the process of working on the program

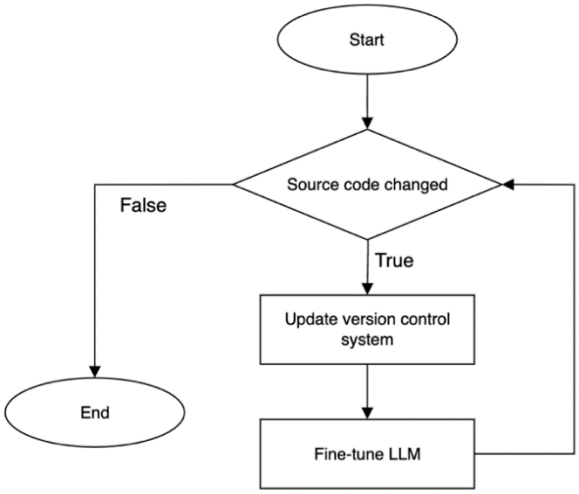

Рис. 3. Схема алгоритму додаткового навчання LLM

Fig. 3. Diagram of the LLM supplementary learning algorithm

Проте у процесу додаткового навчання великих мовних моделей є й певні виклики. Один із найважливіших – обчислювальні витрати, які можуть вимагати значних ресурсів. Іншим важливим фактором є якість даних, оскільки успішність точного налаштування значною мірою залежить від якості та репрезентативності набору даних, використаного для цього процесу. Також слід зауважити, що потенційний ризик надмірної спеціалізації моделі в результаті точного налаштування може призвести до зниження її здатності узагальнювати та інтерпретувати широкий спектр мовних конструкцій. Ще одним недоліком запропонованого методу додаткового навчання є те, що використання додатково навченої моделі для рефакторингу іншого програмного продукту є неможливим, оскільки дані, на яких LLM навчалась, не відповідають тим, що мають бути опрацьовані.

doi: https://doi.org/10.15802/stp2025/325888

Як ще один метод для поліпшення роботи великої мовної моделі можна застосовувати метод індексації контексту LLM (рис. 4).

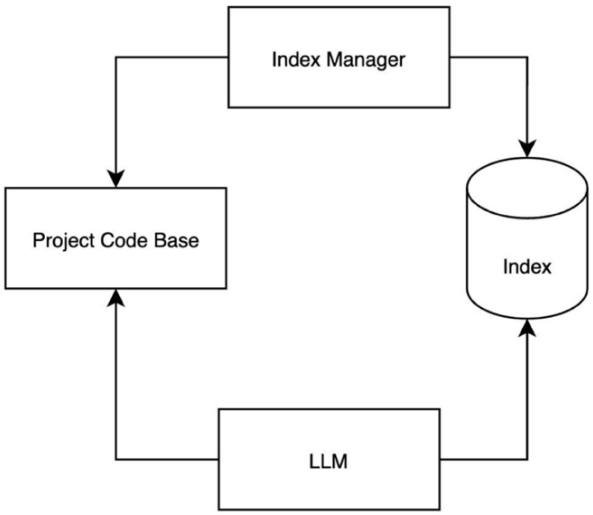

Рис. 4. Схема застосування індексації контексту LLM

Fig. 4. Scheme of application of LLM context indexing

Це дає змогу зберігати в контексті моделі велику кількість залежностей файлів із вихідним кодом за допомогою індексації, при цьому не завантажуючи сам вміст цих файлів у пам'ять ЕОМ без потреби. У разі застосування цього підходу кожному файлу з вихідним кодом присвоюють індекс і встановлюють залежності між індексами, що відповідають файлам, які зберігаються в базі даних. LLM під час рефакторингу, використовуючи індекси, «розуміє», від яких файлів залежить поточний файл, і завантажує в пам'ять вміст лише необхідних файлів. Такий підхід допомагає ефективно використовувати наявний контекст і не тримати в ньому непотрібний код.

Перевагою цього підходу є модульність: модулі кодової бази та індексу не залежать від будь-яких інших модулів, а значить можуть, бути замінені на інші, що дозволяє використовувати реалізацію наведеного методу для різних проєктів.

Схематичне зображення модулів та їх залежності подано на рис. 5.

Для оцінки результатів застосування кожного з методів можуть бути використані показники Code Health, що дозволяють дослідити якість коду після рефакторингу [7]. Оцінку ефективності великої мовної моделі можна провести шляхом вирахування показника F_1, який обчислюють як середнє гармонійне значення точності P та запам'ятовування R [8]:

$$F_1 = 2PR/(P+R).$$

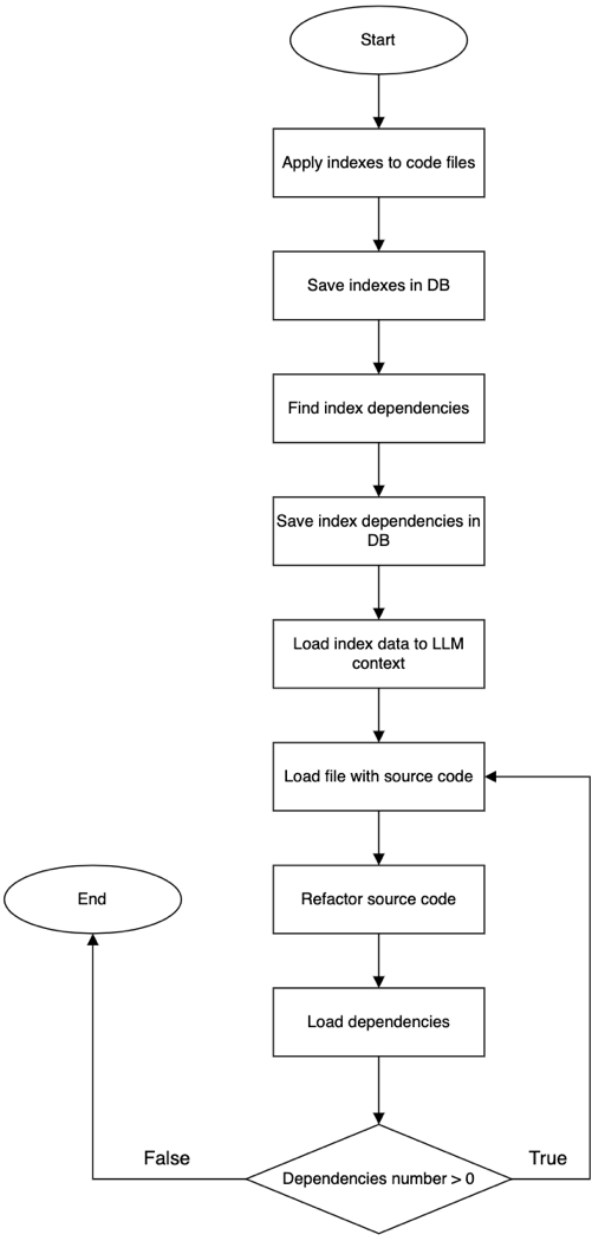

Рис. 5. Схема алгоритму застосування індексації контексту LLM

Fig. 5. Scheme of the algorithm for applying LLM context indexing

doi: https://doi.org/10.15802/stp2025/325888

На основі проведених досліджень та аналізу результатів рефакторингу з наявними мовними моделями було розроблено методи для поліпшення ефективності LLM. Кожен метод проаналізовано в контексті переваг і недоліків, які можуть вплинути на його реалізацію та застосування. Представлено схеми реалізації запропонованих методів.

Наукова новизна та практична значимість

Запропоновано методи поліпшення використання контексту великих мовних моделей для задачі рефакторингу коду програмного забезпечення, які дозволяють поліпшити ефективність рефакторингу із застосуванням великих мовних моделей.

Аналіз результатів проведених досліджень показує, що цей напрям є перспективним для подальших досліджень, а також корисним із наукової та практичної сторони науковцям та розробникам програмного забезпечення.

Висновки

Аналіз методу додаткового навчання великої мовної моделі показав, що мовна модель буде відповідати вимогам лише у випадку навчання на достатньому об’ємі даних, який для кожного проєкту є різним. На початкових етапах розробки це ускладнює задачу. Іншим важливим моментом є вибір методу додаткового навчання LLM, що має чималий вплив на кінцевий результат.

Метод поліпшення використання контексту мовної моделі можна застосовувати до проєктів, які мають достатній розмір програмного продукту, щоб не впливати на успішність його застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Іванов О. П., Шинкаренко В. І., Скалозуб В. В., Косолапов А. А. Визначення авторства художнього україномовного тексту засобами штучного інтелекту за надкороткими уривками. *Наука та прогрес транспорту*. 2023. № 2 (102). С. 45–53. DOI: https://doi.org/10.15802/stp2023/288289
2. Code Health – How easy is your code to maintain and evolve? *CodeScene*. URL: https://codescene.io/docs/guides/technical/code-health.html
3. Critical Problems in Software Development Projects and How to Address Them. *appit.* URL: https://appitventures.com/blog/8-issues-in-software-development-and-how-to-tackle-them
4. F1 Score in Machine Learning. *ENCORD*. URL: https://encord.com/blog/f1-score-in-machine-learning/
5. Fu Q., Cho M., Merth T., Mehta S., Rastegari M., Najibi M. LazyLLM: Dynamic Token Pruning for Efficient Long Context LLM Inference. *arXiv:2407.14057*. 2024. P. 1–12.
6. Ramesh R., M. A. T. R., Reddy H. V., N. S. V. (2024). Fine-Tuning Large Language Models for Task Specific Data. *2024 2nd International Conference on Networking, Embedded and Wireless Systems (ICNEWS)* (Bangalore, 22-23 Aug. 2024). Bangalore, 2024. P. 1–6. DOI: https://doi.org/10.1109/icnews60873.2024.10730913
7. Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements. CodeScene. URL: https://codescene.com/hubfs/whitepapers/Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements.pdf
8. Zhang Y., Li Y., Meredith G., Zheng K., Li X. Move method refactoring recommendation based on deep learning and LLM-generated information. *Information Sciences*. 2025. Vol. 697. P. 121753. DOI: https://doi.org/10.1016/j.ins.2024.121753

O. A. SYROTA[1*], V. M. HORIACHKIN[2]

[1*]Dep. «Computer and Information Technology», Ukrainian State University of Science and Technologies, SEI DIIT, Lazaryan St., 2, Dnipro, Ukraine, 49010, tel. +38 (098) 359 60 01, e-mail sirotaalexandr30@gmail.com, ORCID 0000-0001-7391-2471
[2]Dep. «Computer and Information Technology», Ukrainian State University of Science and Technologies, SEI DIIT, Lazaryana St., 2, Dnipro, Ukraine, 49010, tel. +38 (056) 373 15 35, e-mail vgora@ukr.net, ORCID 0000-0002-8952-952X

doi: https://doi.org/10.15802/stp2025/325888

Problems of Program Code Refactoring with the Use of Artificial Intelligence

Purpose. The modern technological landscape is characterized by the rapid development of software focused on various subject areas and platforms. This leads to the continuous creation of new software products consisting of a huge number of lines of code. The process of developing high-quality software is a multi-stage process that involves a number of factors that affect the final result. The key aspects include the competence of developers, the effectiveness of project management, the availability of necessary resources, and the ability to adapt to changing requirements. Each platform has its own specific features that must be taken into account during development, which further complicates the process of creating universal and effective software solutions. Our research aims to conduct a comprehensive analysis of the potential and promising areas of application of large language models in the context of program code refactoring. The work is aimed at developing and improving methods that will help to increase the efficiency of the refactoring process using these models. **Methodology.** To solve the above problems, it is proposed to implement a set of methods that can be used both separately and in synergy to optimize the final result. These methods, carefully developed in the context of modern software engineering paradigms, are aimed at increasing the efficiency of the refactoring process while ensuring that the software functionality is preserved. Their implementation involves a systematic approach to analyzing and modifying the code base, taking into account both technical aspects and the potential impact on the overall system architecture. **Findings.** A comprehensive analysis of existing language models has been conducted and methods for improving the efficiency of large language models in the context of code refactoring have been developed. The key factors that influence the success of the proposed methods, including the amount of training data and the limitations of the model context, are identified. **Originality.** An approach to improving the efficiency of large language models in code refactoring that takes into account the specifics of different projects and development stages is developed. Innovative methods for retraining language models and optimizing the use of context are proposed, which expand the capabilities of automated refactoring. **Practical value.** The results of the study allow to improve the efficiency of code refactoring using large language models.

Keywords: software engineering; artificial intelligence; large language model; refactoring; program code; program code quality

LIST OF REFERENCE LINKS

1. Ivanov, O. P., Shynkarenko, V. I., Skalozub, V. V., & Kosolapov, A. A. (2023). Determining the Authorship of a Ukrainian-Language Literary Text by Means of Artificial Intelligence from Ultra-Short Excerpts. *Science and Transport Progress, 2*(102), 45-53. DOI: https://doi.org/10.15802/stp2023/288289 (in Ukrainian)
2. Code Health - How easy is your code to maintain and evolve? *Codescene* Retrieved from https://codescene.io/docs/guides/technical/code-health.html (in English)
3. Critical Problems in Software Development Projects and How to Address Them. *appit.* Retrieved from https://appitventures.com/blog/8-issues-in-software-development-and-how-to-tackle-them (in English)
4. F1 Score in Machine Learning. *ENCORD*. Retrieved from https://encord.com/blog/f1-score-in-machine-learning/ (in English)
5. Fu Q., Cho M., Merth T., Mehta S., Rastegari M., Najibi M. (2024)LazyLLM: Dynamic Token Pruning for Efficient Long Context LLM Inference. *arXiv:2407.14057,* 1-12. (in English)
6. Ramesh, R., M, A. T. R., Reddy, H. V., & N, S. V. (2024, August). Fine-Tuning Large Language Models for Task Specific Data. In *2024 2nd International Conference on Networking, Embedded and Wireless Systems (ICNEWS)* (pp. 1-6). Bangalore, India. DOI: https://doi.org/10.1109/icnews60873.2024.10730913 (in English)
7. Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements. *Codescene*. Retrieved from https://codescene.com/hubfs/whitepapers/Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements.pdf (in English)
8. Zhang, Y., Li, Y., Meredith, G., Zheng, K., & Li, X. (2025). Move method refactoring recommendation based on deep learning and LLM-generated information. *Information Sciences, 697*, 121753. DOI: https://doi.org/10.1016/j.ins.2024.121753 (in English)

Надійшла до редколегії: 26.11.2024
Прийнята до друку: 28.03.2025

doi: https://doi.org/10.15802/stp2025/325888