https://doi.org/10.31891/2307-5732-2026-361-38
УДК 004.8

СКРИПНЮК ОЛЕКСАНДР
Хмельницький національний університет
https://orcid.org/0009-0006-1216-379X
e-mail: cawa.ckp@gmail.com
БАГРІЙ РУСЛАН
Хмельницький національний університет
https://orcid.org/0000-0001-5219-1185
e-mail: bahriiro@khmnu.edu.ua
МАНЗЮК ЕДУАРД
Хмельницький національний університет
https://orcid.org/0000-0002-7310-2126
e-mail: manziuk.e@khmnu.edu.ua
СКРИПНИК ТЕТЯНА
Хмельницький національний університет
https://orcid.org/0000-0002-8531-5348
e-mail: tkskripnik1970@gmail.com

АВТОМАТИЗОВАНЕ ВІДНОВЛЕННЯ ТРАСУВАЛЬНИХ ЗВ'ЯЗКІВ МІЖ ВИМОГАМИ ТА ПРОГРАМНИМ КОДОМ З ВИКОРИСТАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

Проблема забезпечення узгодженості між вимогами та програмним кодом набуває критичного значення зі зростанням масштабу та складності сучасних програмних систем, адже відсутність надійних трасувальних зв'язків часто призводить до неповної реалізації вимог, ускладнює супровід коду та перевірку коректності роботи системи. Ручне формування трасувальних матриць є трудомістким і схильним до помилок процесом, особливо у великих проєктах. Використання великих мовних моделей відкриває нові можливості для автоматизації цього процесу, оскільки такі моделі здатні відображати глибокі семантичні зв'язки між текстовими вимогами та фрагментами програмного коду.

У статті запропоновано метод виявлення трасувальних зв'язків між вимогами та програмним кодом із використанням трансформерних моделей великих мовних систем. Запропонований підхід ґрунтується на перетворенні текстових артефактів у векторні представлення за допомогою моделей CodeBERT, SBERT та TF-IDF з подальшим обчисленням семантичної подібності для автоматичного визначення потенційних зв'язків. Метод охоплює такі етапи як підготовки даних, генерації ембедингів, пошуку релевантних фрагментів і оцінювання отриманих результатів.

Експерименти проведено на датасеті MSR-2021, що містить реальні трасувальні зв'язки для кількох проєктів. Отримані результати засвідчили перевагу CodeBERT над традиційними підходами (TF-IDF, SBERT): метод забезпечує точність до 0.85 та F1-score до 0.50 (залежно від глибини пошуку), що є високими показниками для задач автоматизованого інформаційного пошуку та ранжування. Додатково підтверджено важливість урахування структурного контексту коду та продемонстровано вплив параметра Top-K на баланс між повнотою та точністю. Результати доводять, що інтеграція моделей на основі LLM істотно підвищує рівень автоматизації, узгодженості та якості трасування вимог у сучасних середовищах розробки програмного забезпечення.

Ключові слова: *трасування вимог, великі мовні моделі, CodeBERT, семантична подібність, програмний код, ембединги, автоматизація*

SKRYPNIUK OLEXANDR, BAHRII RUSLAN, MANZIUK EDUARD, SKRYPNYK TETIANA
Khmelnytskyi National University

AUTOMATED TRACEABILITY LINK RECOVERY BETWEEN REQUIREMENTS AND SOURCE CODE USING LARGE LANGUAGE MODELS

The problem of consistency between software requirements and their implementation in source code becomes critically important as the scale and complexity of modern software systems increase, since the absence of reliable traceability links often leads to incomplete requirement implementation, complicates code maintenance, and hinders the verification of system correctness. Manual generation of traceability matrices is a labour-intensive and error-prone process, especially in large projects. The use of large language models opens up new opportunities for automating this process, as such models are capable of reflecting deep semantic relationships between text requirements and software code fragments.

The article proposes a method for identifying traceability links between requirements and software code using transformer models of large language systems. The proposed approach is based on converting text artefacts into vector representations using CodeBERT, SBERT, and TF-IDF models, followed by calculating semantic similarity to automatically identify potential connections. The method covers such stages as data preparation, embedding generation, search for relevant fragments, and evaluation of the results obtained.

The experiments were conducted on the MSR-2021 dataset, which contains real traceability links for several projects. The obtained results demonstrated the advantage of CodeBERT over traditional approaches (TF-IDF, SBERT): the method achieves accuracy of up to 0.85 and an F1-score of up to 0.50, depending on the search depth, which represents strong performance for automated information retrieval and ranking tasks. The study additionally confirmed the importance of considering the structural context of the code and showed the influence of the Top-K parameter on the balance between recall and precision. The results indicate that integrating LLM-based models significantly improves the level of automation, consistency, and quality of requirements traceability in modern software development environments.

Keywords: *requirement tracing, large language models, CodeBERT, semantic similarity, program code, embeddings, automation*

Стаття надійшла до редакції / Received 27.11.2025
Прийнята до друку / Accepted 11.01.2026
Опубліковано / Published 29.01.2026

This is an Open Access article distributed under the terms of the Creative Commons CC-BY 4.0

© Скрипнюк Олександр, Багрій Руслан, Манзюк Едуард, Скрипник Тетяна

Постановка проблеми

У сучасних дослідженнях у сфері комп'ютерних наук проблема забезпечення узгодженості між вимогами та їх реалізацією у вихідному коді набуває особливої актуальності. Трасування вимог розглядається як процес встановлення та підтримання зв'язків між вимогами, проєктними артефактами, кодом і тестами, що забезпечує контроль за виконанням функціональних і нефункціональних вимог протягом усього життєвого циклу системи [1, 2]. Відсутність або низька якість трасувальних зв'язків призводить до утруднень під час супроводу програмного забезпечення, збільшення вартості розробки та ризику виникнення помилок, пов'язаних із невідповідністю реалізації початковим вимогам [3].

Ручне або напівавтоматизоване формування трасувальних матриць є трудомістким, суб'єктивним та схильним до людських помилок процесом, особливо у великих проєктах [4]. Класичні автоматизовані методи трасування, що базуються на статистичних або лінгвістичних підходах (TF-IDF, LSI, VSM), забезпечують лише поверхневе порівняння текстів і не враховують глибокі семантичні залежності між артефактами [5, 6].

Останні досягнення у сфері обробки природної мови, зокрема використання трансформерних архітектур (BERT, RoBERTa, CodeBERT), відкрили нові можливості для підвищення точності трасування вимог [7, 8]. Такі моделі дозволяють ефективно аналізувати контекст, синтаксичні структури та зміст текстових і кодових фрагментів, створюючи спільні семантичні простори для порівняння вимог і методів програмного коду [9, 10].

Разом із тим залишається невирішеним питання адаптації великих мовних моделей до специфіки програмного коду, вибору оптимальної архітектури для обчислення схожості та забезпечення відтворюваності результатів при роботі з різними наборами даних. Відсутність єдиних підходів до кількісного оцінювання якості трасування також ускладнює порівняння результатів різних досліджень [11, 12].

Таким чином, актуальною є науково-практична задача розроблення методу автоматичного виявлення трасувальних зв'язків між вимогами та програмним кодом, який поєднує переваги великих мовних моделей, трансформерних архітектур і сучасних метрик семантичної схожості з метою підвищення точності, узагальнюваності та масштабованості процесу трасування.

Аналіз останніх джерел

Проблематика трасування вимог була вперше детально описана в дослідженні, у якому сформульовано поняття вимог, що підлягають трасуванню, та визначено ключові труднощі встановлення й підтримання зв'язків між вимогами та іншими артефактами життєвого циклу програмних систем [13]. Подальший розвиток цієї тематики продемонстрував, що трасування необхідно розглядати як безперервний процес керування знаннями, спрямований на підтримання узгодженості між рівнями специфікації та реалізації [14].

Сучасні огляди підкреслюють, що трасування є важливим механізмом забезпечення верифікації, валідації, управління змінами та контролю ризиків упродовж розвитку програмних систем [15]. Окремі дослідження акцентують увагу на класифікації технік трасування та показують обмеження традиційних лінгвістичних підходів, які базуються на векторних моделях або латентно-семантичному аналізі й часто не здатні забезпечити глибинне розуміння змісту текстів [16].

Поява методів латентно-семантичного аналізу стала важливим етапом еволюції трасування, оскільки ці методи вперше продемонстрували можливість відновлення семантичних зв'язків між документацією та програмним кодом [17]. Подальші підходи, основані на інформаційному пошуку та TF-IDF, забезпечили певний рівень точності, проте виявилися недостатньо узагальнювальними у складних проєктах [18]. Додаткові дослідження показали, що комбінування текстових і структурних ознак може суттєво покращити результати трасування [19].

Суттєвий прогрес у цій галузі став можливим завдяки розвитку глибоких нейронних мереж і трансформерних моделей. У низці сучасних робіт наголошується на необхідності інтегрування трасування у процеси безперервної інтеграції, тестування та DevOps, оскільки це дозволяє забезпечити стабільність і прозорість змін у проєктах [20]. Систематичні огляди останніх років демонструють формування нової парадигми Deep Traceability, у межах якої нейронні моделі використовуються для автоматичного відновлення зв'язків між вимогами та кодом без ручної інженерії ознак [21].

Моделі на основі BERT відкрили можливість побудови спільного семантичного простору для вимог і програмних артефактів, що дало змогу суттєво підвищити точність встановлення відповідностей [22]. Подальші дослідження показали, що спеціалізовані трансформерні моделі для коду, такі як CodeBERT і GraphCodeBERT, здатні враховувати як текстову, так і структурну природу програмних фрагментів, що покращує якість трасування, виявлення дублювання та формування високоякісних embedding-представлень [23]. Результати цих робіт підтверджують ефективність трансформерних підходів у задачах, де необхідне глибоке семантичне розуміння змісту та структури програмного коду.

Незважаючи на досягнутий прогрес, низка проблем залишається відкритою. Зокрема, актуальними залишаються питання адаптації великих мовних моделей до різних мов програмування, інтеграції структурної інформації (AST-графів, потоків даних) у процес трасування та підвищення інтерпретованості результатів моделі. Відсутність уніфікованих підходів до кількісної оцінки якості трасування ускладнює порівняння різних методів і результатів експериментів.

Отже, сучасні дослідження демонструють перехід від класичних текстових методів до контекстно-залежних моделей на основі трансформерних архітектур. Це створює передумови для розроблення нових методів, що поєднують переваги великих мовних моделей, високопродуктивних алгоритмів пошуку та методів навчання з урахуванням доменної специфіки програмного коду.

Метою роботи є: розроблення та експериментальна перевірка методу автоматичного виявлення трасувальних зв'язків між вимогами та програмним кодом із використанням великих мовних моделей. Запропонований підхід спрямований на підвищення точності, масштабованості та відтворюваності процесу трасування завдяки глибшому урахуванню семантичного контексту програмних артефактів.

Виклад основного матеріалу

Запропонований метод базується на використанні векторних представлень текстових артефактів програмного забезпечення та обчисленні семантичної подібності між ними. Його концепція ґрунтується на припущенні, що вимоги та фрагменти програмного коду, які реалізують однакову функціональність, повинні мати близькі векторні репрезентації у спільному embedding-просторі [24]. Це дозволяє автоматично виявляти потенційні трасувальні зв'язки між вимогами та кодом без участі експерта.

Схема методу включає такі основні етапи як: підготовку даних, генерацію ембедингів, пошук схожих елементів, формування трасувальних зв'язків та оцінювання результатів. Кожен етап реалізує певні підпроцеси обробки природної мови, векторного аналізу та пошуку за семантичною схожістю [25]. Загальну схему архітектури подано на рисунку 1.

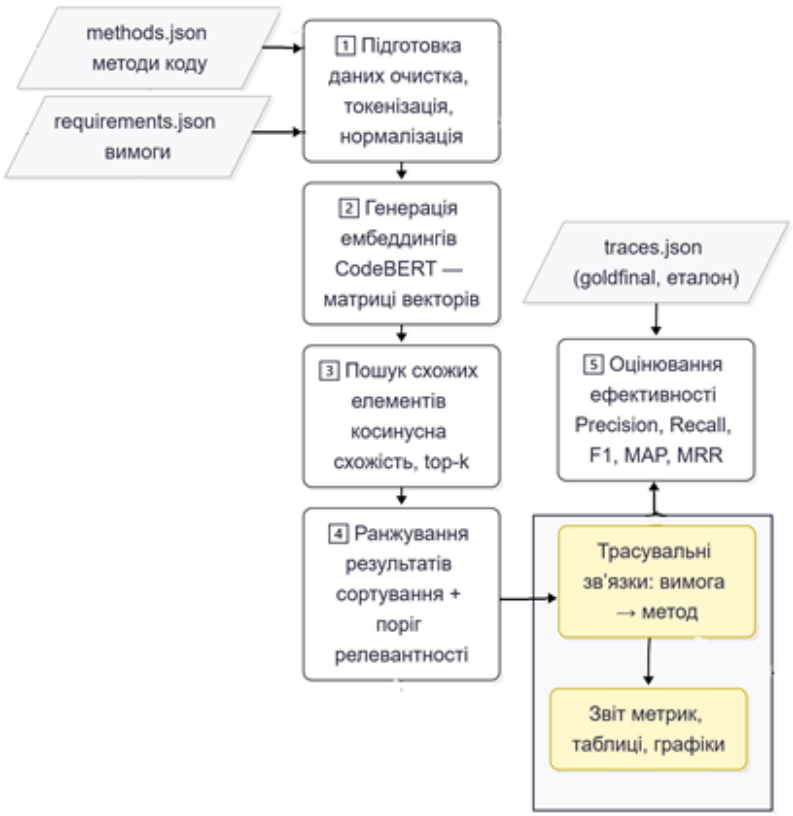

Рис. 1. Схема методу виявлення трасувальних зв'язків між вимогами та програмним кодом.

На етапі підготовки даних метод оперує трьома структурованими файлами у форматі JSON:

– requirements.json – тексти функціональних і нефункціональних вимог;

– methods.json – фрагменти програмного коду з назвами класів і методів;

– traces.json – еталонні трасувальні зв'язки що використовуються для подальшого оцінювання .

Перед побудовою векторних представлень дані проходять очищення, токенізацію, нормалізацію та лематизацію. Для коду додатково виконується видалення коментарів, уніфікація форматування та розбиття ідентифікаторів у стилі camelCase, що забезпечує лексичну узгодженість між артефактами .

Для побудови векторних представлень використовується попередньо натренована трансформерна модель CodeBERT, яка формує спільний семантичний простір для природної мови та програмного коду. Для порівняння результатів застосовуються також моделі SBERT[26] і TF-IDF[27]. Кожен текст вимоги та метод програмного коду перетворюються у вектор фіксованої розмірності, що відображає їх семантичні ознаки. Отримані ембедінги кешуються у вигляді матриць для забезпечення швидкої обробки великих обсягів даних.

Пошук релевантних методів здійснюється за допомогою бібліотеки FAISS[28] (Facebook AI Similarity Search), яка забезпечує ефективний пошук найближчих сусідів у багатовимірних векторних просторах. Для кожної вимоги обчислюється косинусна подібність із векторами методів коду, після чого формується впорядкований список найбільш семантично схожих кандидатів. Результати пошуку зберігаються для подальшого оцінювання.

Використання трансформерних моделей і векторного пошуку забезпечує автоматизацію процесу трасування, зменшує кількість ручних операцій та підвищує точність узгодження між специфікацією та реалізацією програмних систем.

Програмна реалізація методу

Для перевірки запропонованого методу було створено програмний прототип системи автоматичного виявлення трасувальних зв'язків між вимогами та програмним кодом. Реалізацію виконано мовою Python , яка забезпечує високу гнучкість та зручність інтеграції з бібліотеками машинного навчання.

Для побудови векторних представлень текстових даних використано бібліотеку Transformers , яка надає інтерфейс до попередньо натренованих моделей глибокого навчання, зокрема CodeBERT. Ця модель забезпечує спільне представлення тексту природної мови та програмного коду, що дозволяє порівнювати їх у єдиному семантичному просторі. Ембединги зберігаються у форматі .npy, що забезпечує швидке завантаження та обробку даних при повторному запуску системи.

Для виконання обчислень нейронних мереж і роботи з тензорними структурами застосовано бібліотеку PyTorch , яка підтримує апаратне прискорення на графічних процесорах (GPU) і забезпечує можливість гнучкого моделювання. Пошук найбільш схожих елементів реалізовано за допомогою бібліотеки FAISS (Facebook AI Similarity Search) , яка дозволяє ефективно здійснювати пошук найближчих сусідів у багатовимірних просторах, що особливо важливо при роботі з великими обсягами ембедингів.

Для аналізу результатів і статистичної обробки використано бібліотеки scikit-learn, pandas та numpy, які реалізують обчислення метрик Precision, Recall, F1-score, MAP і MRR, а також надають інструменти для візуалізації отриманих результатів.

Графічний інтерфейс користувача розроблено з використанням бібліотеки CustomTkinter, що забезпечує створення адаптивних інтерфейсів із підтримкою темної теми та сучасного дизайну. Інтерфейс дозволяє завантажувати вхідні файли (requirements.json, methods.json, traces.json), виконувати пошук трасувальних зв'язків, переглядати результати у табличній формі та будувати графіки метрик ефективності.

Реалізована система є модульною, що дозволяє легко змінювати або розширювати окремі компоненти – наприклад, інтегрувати нові моделі ембедингів або алгоритми пошуку. Така архітектура забезпечує масштабованість, відтворюваність і придатність до використання як у наукових дослідженнях, так і в промислових системах управління вимогами.

Аналіз ефективності запропонованого методу

Метою експериментальної частини є кількісна й якісна оцінка ефективності запропонованого методу трасування вимог до програмного коду з використанням великих мовних моделей. Основна увага приділяється оцінюванню здатності методу точно ідентифікувати семантичні зв'язки між вимогами та програмними артефактами, а також визначенню впливу структурного контексту коду, параметра Top-K і міжпроєктного узагальнення.

Дослідження проводилося на MSR-2021 Traceability Dataset[29], який містить вимоги, методи та еталонні зв'язки (goldfinal) для чотирьох проєктів: GanttProject, iTrust, JHotDraw і Synapses. Набір вважається еталонним у спільноті досліджень трасування та забезпечує можливість відтворення результатів.

Оцінювання виконувалося за метриками Precision, Recall, F1-score, MAP (Mean Average Precision) та MRR (Mean Reciprocal Rank).

Одним із ключових чинників, що впливають на якість трасування, є наявність структурного контексту – тобто врахування не лише коротких назв методів, а й їхнього розташування в класах та повного вихідного коду. Більшість класичних методів, таких як TF-IDF або SBERT, оперують лише лексичним рівнем опису, ігноруючи архітектурні залежності. Проте в реальних програмних системах логіка реалізації однієї вимоги часто розподілена між кількома класами, а сама семантика методу визначається взаємодією з іншими елементами програми .

Щоб оцінити вплив структурного контексту, модель CodeBERT була запущена у двох конфігураціях:

- без контексту – використовується лише коротка назва методу (fullmethod);
- з контекстом – під час векторизації враховуються ім'я класу та тіло методу.

Таким чином, у другому варіанті модель отримує більш насичене представлення коду, що дозволяє їй формувати глибше семантичне розуміння функціональності. Результати урахування структурного контексту на результати трасування наведено у таблиці 1.

Таблиця 1

Вплив урахування структурного контексту на результати трасування

Конфігурація CodeBERT	Precision	Recall	F1-score	MAP	MRR
Без контексту (fullmethod)	0.6955	0.1001	0.1627	0.0750	0,9546
З урахуванням classname + sourcecode	0.7200	0.1021	0.1788	0.0770	0.9722

Як видно із тезультати показали, що використання структурного контексту покращує точність, повноту та якість ранжування. Значення Precision зросло з 0.6955 до 0.7200, а F1-score - з 0.1627 до 0.1788. Показники MAP також покращилися (з 0.0750 до 0.0770), що свідчить про підняття релевантних методів на вищі позиції у списку результатів. MRR змінився з 0.9546 до 0.9722, що підтверджує здатність моделі стабільно виводити хоча б один релевантний результат у верхній частині ранжування.

Низькі значення F1-score є очікуваними для задач трасування на невеликих або структурно нерівномірних наборах даних. Простір пошуку містить тисячі нерелевантних методів, тоді як кількість правильних зв'язків для однієї вимоги зазвичай не перевищує кількох сотень. У таких умовах навіть невелике зниження Recall суттєво впливає на F1-score, оскільки ця метрика є гармонійним середнім. Додатково слід

зазначити, що всі наведені значення є середніми по 18 вимогах, включно з тими, які мають слабкий або неоднозначний семантичний сигнал. Усереднення по всіх випадках природно зменшує кінцевий F1-score, оскільки частина вимог є складними для трасування незалежно від обраної моделі.

Високе значення MRR, яке наближається до 1.0, пояснюється тим, що CodeBERT майже завжди знаходить хоча б один правильний зв'язок у верхніх позиціях списку кандидатів. Ця метрика враховує лише першу релевантну відповідність, тому навіть у випадках низького Recall або обмеженого F1-score вона залишається високою. Таким чином, модель здатна правильно ранжувати найбільш очевидний релевантний метод, навіть якщо інші правильні зв'язки виявляються рідше.

Отже, урахування структурного контексту є важливим елементом підвищення точності трасування, оскільки модель починає враховувати не лише лексичні збіги, а й логічну структуру коду. Це дозволяє точніше встановлювати відповідність між вимогами та їх реалізацією, що особливо важливо при аналізі результатів, усереднених по всіх 18 вимогах датасету.

На наступному етапі проведено порівняння трьох підходів: TF-IDF, SBERT і CodeBERT. Усі моделі були протестовані в єдиній експериментальній схемі, яка передбачала спільний набір даних, однакову процедуру оцінювання та однакові метрики порівняння. Порівняльні результати роботи моделей наведено в таблиці 2.

Таблиця 2

Порівняльні результати роботи моделей

Модель	Precision	Recall	F1-score	MAP	MRR
TF-IDF	0.6298	0.0890	0.1558	0.0619	0.8647
SBERT	0.7049	0.0907	0.1604	0.0729	0.7059
CodeBERT	0.7200	0.1021	0.1788	0.0770	0.9722

Як видно з таблиці, CodeBERT демонструє найвищі значення Precision(0.7200), Recall(0.1021), F1-score(0.1788), MAP(0.0770) та MRR(0.9722). Це свідчить, що модель краще відтворює семантичну відповідність між вимогами та програмним кодом порівняно з іншими підходами. TF-IDF дає найнижчі результати, оскільки працює лише зі статистичною частотою термінів, а SBERT, хоч і краще моделює контекст, не враховує структуру коду. CodeBERT формує спільний простір представлень для тексту та коду, що і забезпечує підвищення показників.

Низькі значення Recall і F1-score для всіх трьох методів пов'язані з тим, що загальна кількість правильних трасувальних зв'язків у наборі даних є дуже малою порівняно з кількістю всіх можливих пар «вимога–метод». Навіть невелика кількість пропущених відповідностей значно зменшує середні метрики.

Наступним етапом експериментального дослідження стало визначення впливу параметра Top-K на якість результатів трасування вимог до програмного коду. Параметр Top-K визначає кількість найбільш схожих методів, які повертаються для кожної вимоги після пошуку векторної подібності.

Іншими словами, Top-K задає глибину ранжування: чи аналізується лише невелика вибірка найближчих методів (наприклад, 10 або 20), чи модель враховує значно більший список кандидатів (до 2000). Збільшення значення K дає змогу охопити ширший простір пошуку та потенційно знайти більше правильних зв'язків, тоді як невеликі значення K зосереджуються лише на найсильніших відповідностях. Метою цього експерименту було оцінити, як глибина ранжування впливає на такі параметри, як точність, повнота, збалансованість результатів та якість ранжування релевантних методів.

Його вплив було оцінено для K = {10, 20, 50, 100, 200, 500, 1000, 2000}. Порівняльні результати залежності метрик від параметра K наведено у таблиці 3.

Таблиця 3

Залежність метрик від параметра K

K	Precision	Recall	F1-score	MAP	MRR
10	0,8471	0,0024	0,0048	0,0022	0,9722
20	0,8088	0,0046	0,0091	0,0041	0,9722
50	0,7871	0,0112	0,022	0,0094	0,9722
100	0,7547	0,0214	0,0416	0,0174	0,9722
200	0,7474	0,0424	0,0801	0,0332	0,9722
500	0,7200	0,1021	0,1788	0,0770	0,9722
1000	0,7011	0,1986	0,3091	0,146	0,9722
2000	0,6949	0,3937	0,5018	0,2821	0,9722

Зі збільшенням значення K спостерігається закономірне зростання показника Recall: від 0.0024 при K = 10 до 0.3937 при K = 2000. Це пояснюється тим, що розширення списку кандидатів дає змогу охопити більшу кількість релевантних методів, які за малих значень K просто не потрапляють у вибірку. Водночас Precision поступово зменшується: від 0.8471 (K = 10) до 0.6949 (K = 2000), оскільки разом із релевантними результатами

до списку додається дедалі більше нерелевантних збігів. Такий компроміс є типовим для задач інформаційного пошуку.

Показник F1-score зростає від 0.0048 до 0.5018, що демонструє покращення загальної збалансованості між точністю та повнотою при збільшенні глибини пошуку. Значення MAP також підвищується – від 0.0022 до 0.2821, що свідчить про появу релевантних методів на вищих позиціях ранжованого списку. При цьому MRR залишається стабільним на рівні 0.9722 для всіх значень K, що вказує на здатність моделі знаходити хоча б один правильний зв'язок у верхніх позиціях незалежно від глибини вибірки.

Графік на рисунку 2 відображає класичну поведінку метрик: при малих значеннях K система забезпечує високу точність, але має дуже низьке охоплення; при великих – навпаки, повнота стає значно вищою, хоча точність зменшується. Оптимальний баланс досягається приблизно при K = 500–1000, де F1-score (0.1788–0.3091) суттєво зростає, а Precision залишається на прийнятному рівні.

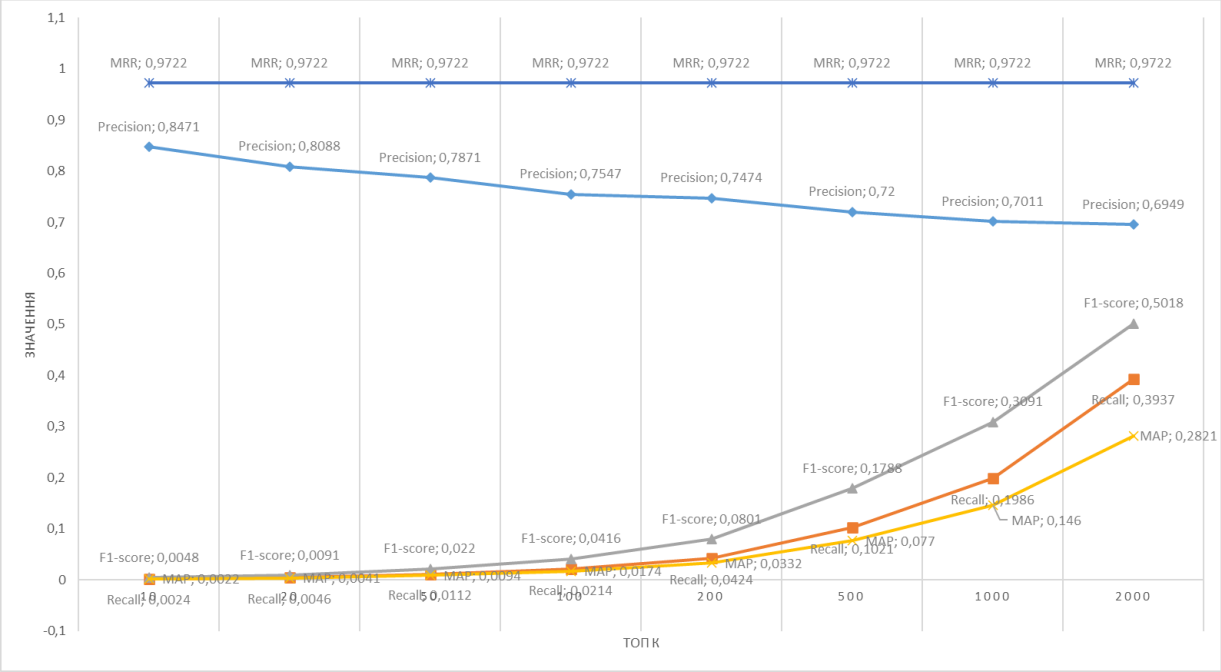

Рис. 2. – Динаміка зміни основних метрик (Precision, Recall, F1-score, MAP) залежно від параметра Top-K

Отже, результати дослідження впливу параметра Top-K підтверджують, що вибір глибини ранжування є критичним для побудови якісних трасувальних зв'язків. Занадто малі значення K призводять до втрати більшості релевантних відповідностей, тоді як надто великі – до накопичення шуму й зниження точності. Саме тому оптимальним є використання проміжних значень K, які забезпечують збалансоване поєднання точності, повноти та якості ранжування. Таке налаштування дозволяє підвищити ефективність процесу трасування та створює підґрунтя для подальшого вдосконалення моделі в реальних умовах розробки програмного забезпечення.

Висновки

У ході проведеного дослідження було розроблено, реалізовано та експериментально перевірено метод автоматизованого виявлення трасувальних зв'язків між вимогами та програмним кодом із використанням великих мовних моделей. Основою підходу стало побудування спільного семантичного простору, у якому вимоги та програмний код подаються у вигляді векторних представлень, здатних відображати їх змістову подібність. Модель CodeBERT продемонструвала здатність інтерпретувати природну мову та програмні конструкції єдиним узгодженим способом, що дозволило встановлювати смислові відповідності навіть за відсутності прямих лексичних збігів.

Експериментальні результати підтвердили перевагу трансформерних моделей над класичними методами. Порівняння з TF-IDF та SBERT показало, що CodeBERT забезпечує найкращі показники: Precision досягає 0.7200, Recall досяг 0,1021, а F1-score - 0.1788, перевищуючи результати інших моделей у тих самих умовах. Значення MAP (0.0770) та MRR (0.9722) свідчать про коректне ранжування релевантних результатів і здатність моделі стабільно знаходити правильні зв'язки серед перших позицій списку.

Особливе значення має урахування структурного контексту під час підготовки вхідних даних. Додавання до текстового подання імен класів і тіл методів забезпечило покращення як точності, так і збалансованості результатів: Precision зріс з 0. 0.6298 до 0.7200, F1-score - з 0.1627 до 0.1788, а MAP - з 0.0750 до 0.0772. Це підтверджує, що ефективне трасування потребує врахування не лише лінгвістичних, а й структурних характеристик програмних артефактів.

Проведений аналіз впливу параметра Top-K засвідчив типовий компроміс між точністю та повнотою. Зі збільшенням K Recall зріс від 0.0024 (K = 10) до 0.3937 (K = 2000), тоді як Precision зменшився з 0.8471 до

0.6949. Найбільш збалансовані значення F1-score (0.1788–0.3091) досягнуто в діапазоні K = 500–1000, що може вважатися оптимальним режимом для практичного використання моделі.

Загалом результати роботи підтверджують доцільність застосування великих мовних моделей у задачах трасування вимог. Запропонований метод забезпечує підвищену точність та стабільність, зменшує потребу у ручному встановленні зв'язків, полегшує аналіз відповідності між вимогами та реалізацією й створює основу для інтеграції більш просунутих підходів.

Література

1. Pauzi Z., Capiluppi A. Applications of natural language processing in software traceability: A systematic mapping study. Journal of Systems and Software. 2023. Vol. 198. DOI: 10.1016/j.jss.2023.111616
2. Ruiz M., Hu J., Dalpiaz F. Why don't we trace? A study on the barriers to software traceability in practice. Requirements Engineering. 2023. Vol. 28. No. 4. P. 619-637. DOI: 10.1007/s00766-023-00408-9
3. Charalampidou S. et al. Empirical studies on software traceability: A mapping study. Journal of Software: Evolution and Process. 2021. Vol. 33. No. 10. DOI: 10.1002/smr.2294
4. Ali S. J., Naganathan V., Bork D. Establishing traceability between natural language requirements and source code using large language models and RAG. In: Conceptual Modeling. ER 2024. Lecture Notes in Computer Science. 2024. Vol. 15238. P. 295-314
5. Rosado da Cruz A. M., Cruz A. M., Rocha Varela M. L. Artificial intelligence techniques for requirements engineering: A comprehensive literature review. Preprints. 2025. DOI: 10.20944/preprints202503.2259.v1
6. Mucha J., Kaufmann A., Riehle D. A systematic literature review of pre-requirements specification traceability. Requirements Engineering. 2024. Vol. 29. No. 2. P. 119-141. DOI: 10.1007/s00766-023-00412-z
7. Zhao W. X. et al. A survey of large language models. arXiv. 2023. arXiv: 2303.18223. URL: https://arxiv.org/abs/2303.18223
8. Ge C. et al. Cross-level requirements tracing based on large language models. IEEE Transactions on Software Engineering. 2025. Vol. 51. No. 7. P. 3404-3421. DOI: 10.1109/TSE.2025.3408781
9. Dai P. et al. Constructing traceability links between software requirements and source code based on neural networks. Mathematics. 2023. Vol. 11. No. 2. DOI: 10.3390/math11020315
10. Legesse H., Bicknell S. AI-enhanced traceability using MBSE and large language models. In: AI4SE Research Workshop. 2025. URL: https://sercuarc.org/wp-content/uploads/2025/09/Legesse_AI_Enhanced_Requirements_Traceability_Using_MBSE_LLM_Complex_Systems.pdf
11. Zhang J. et al. Enhancing requirement traceability through data augmentation using large language models. arXiv. 2025. arXiv: 2509.20149. URL: https://arxiv.org/abs/2509.20149
12. Rodriguez A. D., Dearstyne K. R., Cleland-Huang J. Prompts matter: Insights and strategies for prompt engineering in automated software traceability. In: IEEE 31st International Requirements Engineering Conference Workshops (REW). 2023. P. 455-464. URL: https://arxiv.org/abs/2308.00229
13. Li X. et al. Applications of machine learning in requirements traceability: A systematic mapping study. In: Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering (SEKE). 2023. P. 566-571. URL: https://www.researchgate.net/publication/373619969_Applications_of_Machine_Learning_in_Requirements_Traceability_A_Systematic_Mapping_Study_S
14. Zhang H. et al. A systematic literature review of traceability approaches. Journal of Systems and Software. 2022. Vol. 184. DOI: 10.1016/j.jss.2021.111113
15. Guo D. et al. GraphCodeBERT: Pre-training code representations with data flow. In: Proceedings of the 9th International Conference on Learning Representations (ICLR). 2021. URL: https://arxiv.org/abs/2009.08366
16. Wang Y. et al. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2021. P. 8696-8708. URL: https://arxiv.org/abs/2103.00390
17. Zhu Q. et al. A survey on natural language processing for programming. arXiv. 2022. arXiv: 2212.05773. URL: https://arxiv.org/abs/2212.05773
18. Brown T. B. et al. Language models are few-shot learners. In: Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS). 2020. URL: https://arxiv.org/abs/2005.14165
19. Lewis P. et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS). 2020. URL: https://arxiv.org/abs/2005.11401
20. Ouyang L. et al. Training language models to follow instructions with human feedback. arXiv. 2022. arXiv: 2203.02155. URL: https://arxiv.org/abs/2203.02155
21. Wang B. et al. Machine learning for requirements engineering: A systematic review. Information and Software Technology. 2024. Vol. 170. DOI: 10.1016/j.infsof.2024.107477
22. Hammoudi M. et al. A traceability dataset for open source systems. In: Proceedings of the 18th International Conference on Mining Software Repositories (MSR). 2021. P. 555-559. DOI: 10.1109/MSR52588.2021.00073
23. Guo D. et al. UniXcoder: Unified cross-modal pre-training for code representation. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL). 2022. Vol. 1. P. 7212-7225. URL: https://arxiv.org/abs/2109.08826

24. Feng Z. et al. CodeBERT: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020. 2020. P. 1536-1547. URL: https://arxiv.org/abs/2002.08155

25. Wang F., Vasilescu B., Devanbu P. Embedding traceability in large language model code generation: Towards trustworthy AI-augmented software engineering. In: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion). 2025. DOI: 10.1145/3696630.3730569

26. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Офіційна документація. URL: https://www.sbert.net/

27. Term frequency–inverse document frequency. Wikipedia. URL: https://en.wikipedia.org/wiki/Tf%E2%80%93idf

28. Facebook AI Similarity Search (FAISS). GitHub. 2017. URL: https://github.com/facebookresearch/faiss

29. A Traceability Dataset for Open Source Systems. MSR Conference. 2021. URL: https://2021.msrconf.org/details/msr-2021-data-showcase/5/A-Traceability-Dataset-for-Open-Source-Systems

References

1. Pauzi Z., Capiluppi A. Applications of natural language processing in software traceability: A systematic mapping study. Journal of Systems and Software. 2023. Vol. 198. DOI: 10.1016/j.jss.2023.111616

2. Ruiz M., Hu J., Dalpiaz F. Why don't we trace? A study on the barriers to software traceability in practice. Requirements Engineering. 2023. Vol. 28. No. 4. P. 619-637. DOI: 10.1007/s00766-023-00408-9

3. Charalampidou S. et al. Empirical studies on software traceability: A mapping study. Journal of Software: Evolution and Process. 2021. Vol. 33. No. 10. DOI: 10.1002/smr.2294

4. Ali S. J., Naganathan V., Bork D. Establishing traceability between natural language requirements and source code using large language models and RAG. In: Conceptual Modeling. ER 2024. Lecture Notes in Computer Science. 2024. Vol. 15238. P. 295-314

5. Rosado da Cruz A. M., Cruz A. M., Rocha Varela M. L. Artificial intelligence techniques for requirements engineering: A comprehensive literature review. Preprints. 2025. DOI: 10.20944/preprints202503.2259.v1

6. Mucha J., Kaufmann A., Riehle D. A systematic literature review of pre-requirements specification traceability. Requirements Engineering. 2024. Vol. 29. No. 2. P. 119-141. DOI: 10.1007/s00766-023-00412-z

7. Zhao W. X. et al. A survey of large language models. arXiv. 2023. arXiv: 2303.18223. URL: https://arxiv.org/abs/2303.18223

8. Ge C. et al. Cross-level requirements tracing based on large language models. IEEE Transactions on Software Engineering. 2025. Vol. 51. No. 7. P. 3404-3421. DOI: 10.1109/TSE.2025.3408781

9. Dai P. et al. Constructing traceability links between software requirements and source code based on neural networks. Mathematics. 2023. Vol. 11. No. 2. DOI: 10.3390/math11020315

10. Legesse H., Bicknell S. AI-enhanced traceability using MBSE and large language models. In: AI4SE Research Workshop. 2025. URL: https://sercuarc.org/wp-content/uploads/2025/09/Legesse_AI_Enhanced_Requirements_ Traceability_Using_MBSE_LLM_Complex_Systems.pdf

11. Zhang J. et al. Enhancing requirement traceability through data augmentation using large language models. arXiv. 2025. arXiv: 2509.20149. URL: https://arxiv.org/abs/2509.20149

12. Rodriguez A. D., Dearstyne K. R., Cleland-Huang J. Prompts matter: Insights and strategies for prompt engineering in automated software traceability. In: IEEE 31st International Requirements Engineering Conference Workshops (REW). 2023. P. 455-464. URL: https://arxiv.org/abs/2308.00229

13. Li X. et al. Applications of machine learning in requirements traceability: A systematic mapping study. In: Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering (SEKE). 2023. P. 566-571. URL: https://www.researchgate.net/publication/373619969_Applications_of_Machine_Learning_in _Requirements_Traceability_A_Systematic_Mapping_Study_S

14. Zhang H. et al. A systematic literature review of traceability approaches. Journal of Systems and Software. 2022. Vol. 184. DOI: 10.1016/j.jss.2021.111113

15. Guo D. et al. GraphCodeBERT: Pre-training code representations with data flow. In: Proceedings of the 9th International Conference on Learning Representations (ICLR). 2021. URL: https://arxiv.org/abs/2009.08366

16. Wang Y. et al. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2021. P. 8696-8708. URL: https://arxiv.org/abs/2103.00390

17. Zhu Q. et al. A survey on natural language processing for programming. arXiv. 2022. arXiv: 2212.05773. URL: https://arxiv.org/abs/2212.05773

18. Brown T. B. et al. Language models are few-shot learners. In: Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS). 2020. URL: https://arxiv.org/abs/2005.14165

19. Lewis P. et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS). 2020. URL: https://arxiv.org/abs/2005.11401

20. Ouyang L. et al. Training language models to follow instructions with human feedback. arXiv. 2022. arXiv: 2203.02155. URL: https://arxiv.org/abs/2203.02155

21. Wang B. et al. Machine learning for requirements engineering: A systematic review. Information and Software Technology. 2024. Vol. 170. DOI: 10.1016/j.infsof.2024.107477

22. Hammoudi M. et al. A traceability dataset for open source systems. In: Proceedings of the 18th International Conference on Mining Software Repositories (MSR). 2021. P. 555-559. DOI: 10.1109/MSR52588.2021.00073

23. Guo D. et al. UniXcoder: Unified cross-modal pre-training for code representation. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL). 2022. Vol. 1. P. 7212-7225. URL: https://arxiv.org/abs/2109.08826

24. Feng Z. et al. CodeBERT: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020. 2020. P. 1536-1547. URL: https://arxiv.org/abs/2002.08155

25. Wang F., Vasilescu B., Devanbu P. Embedding traceability in large language model code generation: Towards trustworthy AI-augmented software engineering. In: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion). 2025. DOI: 10.1145/3696630.3730569

26. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Офіційна документація. URL: https://www.sbert.net/

27. Term frequency–inverse document frequency. Wikipedia. URL: https://en.wikipedia.org/wiki/Tf%E2%80%93idf

28. Facebook AI Similarity Search (FAISS). GitHub. 2017. URL: https://github.com/facebookresearch/faiss

29. A Traceability Dataset for Open Source Systems. MSR Conference. 2021. URL: https://2021.msrconf.org/details/msr-2021-data-showcase/5/A-Traceability-Dataset-for-Open-Source-Systems