Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА

на тему *Метод виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей*

Рівень вищої освіти *другий (магістерський)*

Галузь знань *12 – Інформаційні технології*
Шифр і найменування

Спеціальність *122 – Комп'ютерні науки*
Код і найменування

Освітня програма *Комп'ютерні науки*
Назва

Виконав: *студент 2 курсу, група КНм-24-1* _____ *Олександр СКРИПНЮК*
Курс, група виконавця | Підпис | Ім'я, ПРІЗВИЩЕ

Керівник: *к.т.н., доцент кафедри КН* _____ *Руслан БАГРІЙ*
Науковий ступінь, посада | Підпис | Ім'я, ПРІЗВИЩЕ

Нормоконтроль: *к.т.н., доцент кафедри КН* _____ *Руслан БАГРІЙ*
Науковий ступінь, посада | Підпис | Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Зав. кафедри КН, д.т.н., професор _____ *Олександр БАРМАК*
Підпис | Ім'я, ПРІЗВИЩЕ

15 грудня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Факультет *інформаційних технологій*
Кафедра *комп'ютерних наук*
Освітній ступінь *магістр*
Галузь знань *12 – Інформаційні технології*
Спеціальність *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ
Завідувач кафедри *комп'ютерних наук*

_____
(підпис)
д.т.н., професор Олександр БАРМАК
«*28*» *08* *2025* року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

1. Тема кваліфікаційної роботи: *«Метод виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей»*

2. Завдання видано студенту *Олександру СКРИПНЮКУ*
(Ім'я, ПРІЗВИЩЕ)

3. Керівник роботи *доцент кафедри КН Руслан БАГРІЙ*
(Ім'я, ПРІЗВИЩЕ)

4. Затверджені наказом університету від «*28*» *08* *2025* р. № *65*

5. Дата видачі завдання студенту: «*28*» *08* *2025* р.

6. Зміст пояснювальної записки (перелік задач) та вихідні дані:

Метою роботи є підвищення точності та повноти автоматичного виявлення трасувальних зв'язків між текстовими вимогами та методами програмного коду шляхом розробки методу, що базується на використанні великих мовних моделей. Для досягнення поставленої мети необхідно проаналізувати існуючі підходи до автоматизованого трасування вимог, визначити їхні недоліки, розробити метод відновлення семантичних зв'язків, обґрунтувати вибір архітектури нейронної мережі для векторизації артефактів, підготувати та обробити експериментальні набори даних, здійснити програмну реалізацію підходу та оцінити показники точності й повноти результатів трасування шляхом порівняння з еталонними даними.

7. Календарний план виконання кваліфікаційної роботи:

№	Назва етапів (розділів) кваліфікаційної роботи бакалавра	Термін виконання	Примітка
1	*Вибір напряму дослідження та узгодження теми кваліфікаційної роботи з керівником, складання календарного графіка виконання роботи*	*вересень 2025*	*Виконано*
2	*Ознайомлення з предметною областю, аналіз існуючих методів і моделей, формулювання мети та завдань дослідження, визначення об'єкта й предмета дослідження*	*вересень 2025*	*Виконано*
3	*Розробка методу чи моделі для вирішення обраного завдання, опис архітектури рішення*	*жовтень 2025*	*Виконано*
4	*Програмна реалізація методу чи моделі*	*жовтень 2025*	*Виконано*
5	*Дослідження ефективності та експериментальна перевірка результатів, порівняння з відомими підходами*	*листопад 2025*	*Виконано*
6	*Написання пояснювальної записки, оформлення відповідно до вимог, врахування зауважень керівника*	*листопад 2025*	*Виконано*
7	*Підготовка презентаційних матеріалів та попередній захист*	*листопад 2025*	*Виконано*
8	*Перевірка пояснювальної записки на відповідність вимогам оформлення (нормоконтроль) та перевірка на академічну доброчесність. Отримання відгуку керівника та рецензії.*	*грудень 2025*	*Виконано*
9	*Публічний захист кваліфікаційної роботи*	*грудень 2025*	*Виконано*

Виконавець: *студент групи КНм-24-1* (Група виконавця) _____ (Підпис) *Олександр СКРИПНЮК* (Ім'я, ПРІЗВИЩЕ)

Керівник: *к.т.н., доцент кафедри КН* (Науковий ступінь, посада) _____ (Підпис) *Руслан БАГРІЙ* (Ім'я, ПРІЗВИЩЕ)

Реферат

Кваліфікаційна робота магістра присвячена розробці та дослідженню методу виявлення трасувальних зв’язків між вимогами та програмним кодом із використанням великих мовних моделей. Запропонований підхід спрямований на підвищення точності та повноти автоматичного трасування, а також на зменшення трудомісткості процесу аналізу зв’язків у програмних проєктах. Дослідження належить до актуального напряму в галузі комп’ютерних наук, зокрема в підсферах аналізу програмного забезпечення, обробки природної мови та штучного інтелекту.

Актуальність теми. Трасування вимог є одним із ключових елементів життєвого циклу розробки програмного забезпечення, оскільки забезпечує контроль за відповідністю реалізованого функціоналу сформульованим вимогам, підтримку змін, а також виявлення впливу модифікацій на систему. У сучасних програмних проєктах, що характеризуються великою кількістю вимог та значним обсягом програмного коду, ручне виконання трасування стає надзвичайно трудомістким, затратним за часом і схильним до помилок. Традиційні методи інформаційного пошуку, такі як TF-IDF, LSI чи VSM, мають обмежену ефективність через неможливість повного урахування семантичного контексту між природномовним описом вимог і синтаксичною структурою програмного коду. Методи машинного навчання попереднього покоління дещо покращили якість порівняння артефактів, проте залишилися недостатньо глибокими у розумінні контекстних зв’язків.

Суттєвий прогрес у розв’язанні цієї проблеми став можливим завдяки розвитку трансформерних архітектур та появі великих мовних моделей, таких як CodeBERT, GraphCodeBERT та UniXcoder, які здатні формувати узгоджене семантичне представлення тексту та коду. Їх застосування у процесі трасування вимог відкриває можливість значного підвищення точності й повноти встановлення зв’язків, що робить дослідження у цій сфері актуальним і науково значущим.

Мета і завдання роботи. Метою роботи є підвищення точності та повноти автоматичного виявлення трасувальних зв’язків між текстовими вимогами та методами програмного коду шляхом розробки методу, що базується на використанні великих мовних моделей.

Для досягнення поставленої мети потрібно виконати наступні завдання:

– провести аналіз існуючих підходів для автоматизації трасування зв’язків між вимогами та програмним кодом;

– розробити метод для автоматичного виявлення трасувальних зв’язків на основі векторного представлення вимог та коду з використанням великих мовних моделей;

– обґрунтувати вибір та виконати попередню обробку набору даних для навчання моделі, забезпечивши коректність структури, очищення та підготовку артефактів для подальших експериментів;

– провести експериментальну перевірку запропонованого методу за відомими статистичними показниками та порівняти отримані результати з відомими підходами.

Об’єкт дослідження. Процес автоматизованого встановлення трасувальних зв’язків між текстовими вимогами до програмного забезпечення та його програмним кодом.

Предмет дослідження. Методи, моделі та інструменти побудови векторних представлень тексту та коду з використанням великих мовних моделей для задачі трасування вимог.

Методи дослідження. У роботі застосовано методи машинного навчання, обробки природної мови, інформаційного пошуку, статистичного аналізу результатів та порівняльного оцінювання моделей. Для реалізації методу використано архітектури великих мовних моделей, зокрема CodeBERT, а також засоби побудови векторних подань текстових і програмних артефактів. Експериментальні дослідження проведено на відкритому наборі даних MSR 2021 Traceability Dataset, із використанням метрик Precision, Recall, F1-score, MAP та MRR.

Наукова новизна одержаних результатів.

Удосконалено метод виявлення трасувальних зв'язків, який, на відміну від існуючих, поєднує етап попереднього пошуку кандидатів і подальшого їх переоцінювання із застосуванням великих мовних моделей, що дало змогу підвищити точність і повноту встановлення трасувальних зв'язків порівняно з традиційними методами інформаційного пошуку та класичними моделями векторизації.

Апробація результатів кваліфікаційної роботи магістра та публікації. Основні наукові та практичні результати пройшли апробацію на XVII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп'ютерних наук (АПКН – 2025)» (м. Хмельницький, 14–15 листопада 2025 р.) та опубліковані у фаховому виданні «Вісник Хмельницького національного університету. Технічні науки»:

1. Скрипнюк О.Ю., Манзюк Е.А., Багрій Р.О., Петровський С.С. Метод виявлення трасувальних зв'язків між вимогами та програмним кодом із використанням великих мовних моделей. Актуальні проблеми комп'ютерних наук АПКН-2025 : матеріали XVII Всеукр. науково-практ. конф., м. Хмельницький, 14–15 листопада 2025 р. Хмельницький, 2025. С. 380–384.

2. Скрипнюк О.Ю., Багрій Р.О., Манзюк Е.А., Скрипник Т.К. Автоматизоване відновлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей. *Вісник Хмельницького національного університету. Технічні науки*. 2026. Т. 361, № 1.

Структура та обсяг роботи. Кваліфікаційна робота складається із завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань з 45 найменувань та 3 додатків. Загальний обсяг кваліфікаційної роботи складає 108 сторінок, з поміж яких 80 сторінок основного тексту та 28 сторінок додатків. У роботі наведено 14 рисунків та 6 таблиць.

Ключові слова. трасування вимог, великі мовні моделі, CodeBERT, обробка природної мови, семантична подібність, машинне навчання, програмний код.

Зміст

Перелік скорочень 4
Вступ 5
Розділ 1 Аналіз сучасного стану трасування вимог до програмного коду 8
1.1 Сутність, характеристика та проблематика трасування вимог у програмному забезпеченні 8
1.2 Огляд та аналіз існуючих методів, наукових публікацій та підходів до трасування у програмному забезпеченні 11
1.3 Використання великих мовних моделей у трасуванні зв'язків між вимогами та програмним кодом 15
1.4 Постановка задачі дослідження 19
Розділ 2 Метод виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей 20
2.1 Концепція та схема запропонованого методу 20
2.2 Побудова векторних представлень вимог та методів коду 23
2.3 Архітектура методу: retrieval → rerank за допомогою LLM 26
2.4 Критерії та метрики оцінювання роботи методу 28
Висновки до розділу 2 30
Розділ 3 Програмна реалізація методу виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей 32
3.1 Засоби та середовища реалізації 32
3.2 Архітектура програмного рішення та опис основних компонентів 35
3.3 Реалізація процесу попередньої обробки, трасування та валідації 41
Висновки до розділу 3 43
Розділ 4 Експериментальні дослідження методу виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей 45
4.1 Опис та налаштування експериментального середовища та інструментів .. 45
4.2 Характеристика датасетів 47

4.3 Проведення експериментів та аналіз отриманих результатів 50
Висновки до розділу 4 70
Загальні висновки 73
Перелік посилань 76
ДОДАТКИ

Перелік скорочень

Скорочення, термін, позначення	Пояснення
LLM	Large language models (Великі мовні моделі)
JSON	JavaScript Object Notation (Текстовий формат обміну даними)
MAP	Mean Average Precision (Середня точність)
MRR	Mean Reciprocal Rank (Середній зворотний ранг)
GPU	Graphics Processing Unit (Графічний процесор)
BERT	Bidirectional Encoder Representations from Transformers (Двонаправлені кодувальники-трансформери)
DL	Deep Learning (Глибоке навчання)
FAISS	Facebook AI Similarity Search (Бібліотека пошуку подібності від Facebook AI)
SBERT	Sentence-BERT (Модель BERT для векторного представлення речень)
VSM	Vector Space Model (Векторна модель простору)

Вступ

Актуальність теми. Трасування вимог є одним із ключових елементів життєвого циклу розробки програмного забезпечення, оскільки забезпечує контроль за відповідністю реалізованого функціоналу сформульованим вимогам, підтримку змін, а також виявлення впливу модифікацій на систему. У сучасних програмних проєктах, що характеризуються великою кількістю вимог та значним обсягом програмного коду, ручне виконання трасування стає надзвичайно трудомістким, затратним за часом і схильним до помилок. Традиційні методи інформаційного пошуку, такі як TF-IDF, LSI чи VSM, мають обмежену ефективність через неможливість повного урахування семантичного контексту між природномовним описом вимог і синтаксичною структурою програмного коду. Методи машинного навчання попереднього покоління дещо покращили якість порівняння артефактів, проте залишилися недостатньо глибокими у розумінні контекстних зв’язків.

Суттєвий прогрес у розв’язанні цієї проблеми став можливим завдяки розвитку трансформерних архітектур та появі великих мовних моделей, таких як CodeBERT, GraphCodeBERT та UniXcoder, які здатні формувати узгоджене семантичне представлення тексту та коду. Їх застосування у процесі трасування вимог відкриває можливість значного підвищення точності й повноти встановлення зв’язків, що робить дослідження у цій сфері актуальним і науково значущим.

Мета і завдання роботи. Метою роботи є підвищення точності та повноти автоматичного виявлення трасувальних зв’язків між текстовими вимогами та методами програмного коду шляхом розробки методу, що базується на використанні великих мовних моделей.

Для досягнення поставленої мети потрібно виконати наступні завдання:

– провести аналіз існуючих підходів для автоматизації трасування зв’язків між вимогами та програмним кодом;

– розробити метод для автоматичного виявлення трасувальних зв’язків на основі векторного представлення вимог та коду з використанням великих мовних моделей;

– обґрунтувати вибір та виконати попередню обробку набору даних для навчання моделі, забезпечивши коректність структури, очищення та підготовку артефактів для подальших експериментів;

– провести експериментальну перевірку запропонованого методу за відомими статистичними показниками та порівняти отримані результати з відомими підходами.

Об’єкт дослідження. Процес автоматизованого встановлення трасувальних зв’язків між текстовими вимогами до програмного забезпечення та його програмним кодом.

Предмет дослідження. Методи, моделі та інструменти побудови векторних представлень тексту та коду з використанням великих мовних моделей для задачі трасування вимог.

Методи дослідження. У роботі застосовано методи машинного навчання, обробки природної мови, інформаційного пошуку, статистичного аналізу результатів та порівняльного оцінювання моделей. Для реалізації методу використано архітектури великих мовних моделей, зокрема CodeBERT, а також засоби побудови векторних подань текстових і програмних артефактів. Експериментальні дослідження проведено на відкритому наборі даних MSR 2021 Traceability Dataset, із використанням метрик Precision, Recall, F1-score, MAP та MRR.

Наукова новизна одержаних результатів.

Удосконалено метод виявлення трасувальних зв’язків, який, на відміну від існуючих, поєднує етап попереднього пошуку кандидатів і подальшого їх переоцінювання із застосуванням великих мовних моделей, що дало змогу підвищити точність і повноту встановлення трасувальних зв’язків порівняно з традиційними методами інформаційного пошуку та класичними моделями векторизації.

Апробація результатів кваліфікаційної роботи магістра та публікації. Основні наукові та практичні результати пройшли апробацію на XVII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп'ютерних наук (АПКН – 2025)» (м. Хмельницький, 14–15 листопада 2025 р.) та опубліковані у фаховому виданні «Вісник Хмельницького національного університету. Технічні науки»:

1. Скрипнюк О.Ю., Манзюк Е.А., Багрій Р.О., Петровський С.С. Метод виявлення трасувальних зв'язків між вимогами та програмним кодом із використанням великих мовних моделей. Актуальні проблеми комп'ютерних наук АПКН-2025 : матеріали XVII Всеукр. науково-практ. конф., м. Хмельницький, 14–15 листопада 2025 р. Хмельницький, 2025. С. 380–384.

2. Скрипнюк О.Ю., Багрій Р.О., Манзюк Е.А., Скрипник Т.К. Автоматизоване відновлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей. *Вісник Хмельницького національного університету. Технічні науки*. 2026. Т. 361, № 1.

Структура та обсяг роботи. Кваліфікаційна робота складається із завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань з 45 найменувань та 3 додатків. Загальний обсяг кваліфікаційної роботи складає 108 сторінок, з поміж яких 80 сторінка основного тексту та 28 сторінок додатків. У роботі наведено 14 рисунків та 6 таблиць.

Розділ 1 Аналіз сучасного стану трасування вимог до програмного коду

1.1 Сутність, характеристика та проблематика трасування вимог у програмному забезпеченні

Трасування вимог у програмному забезпеченні є одним із ключових процесів інженерії програмних систем, що забезпечує узгодженість між початковими вимогами та кінцевим програмним продуктом. Під трасуванням розуміють встановлення й підтримання зв’язків між вимогами та артефактами життєвого циклу розробки, до яких належать специфікації, проєктні моделі, програмний код, тестові сценарії, документація та результати верифікації [1].

Завдяки трасуванню учасники розробки програмного забезпечення отримують можливість контролювати виконання вимог, оцінювати вплив змін, забезпечувати відповідність між вимогами і тестами, а також прогнозувати ризики, пов’язані з пропуском критичних функціональних або нефункціональних вимог [2, 3]. Такий підхід сприяє підвищенню якості та надійності програмних систем, особливо у критично важливих галузях, таких як авіація, медицина чи фінансовий сектор.

У наукових джерелах трасування розглядається не лише як технічна процедура, а і як організаційний процес, інтегрований у всі етапи життєвого циклу програмного забезпечення. Залежно від напряму встановлення зв’язків розрізняють пряме трасування зворотне та двонапрямне, яке дозволяє перевіряти узгодженість в обох напрямках. [4]. Крім того, трасування може охоплювати як функціональні, так і нефункціональні вимоги, що включають аспекти безпеки, надійності чи продуктивності системи.

Місце трасування вимог у загальному життєвому циклі розробки програмного забезпечення відображено на рисунку 1.1. На ньому наведено послідовність етапів – від формулювання вимог до документування результатів – а також прямі та зворотні зв’язки між ними, що утворюють трасувальні відношення.

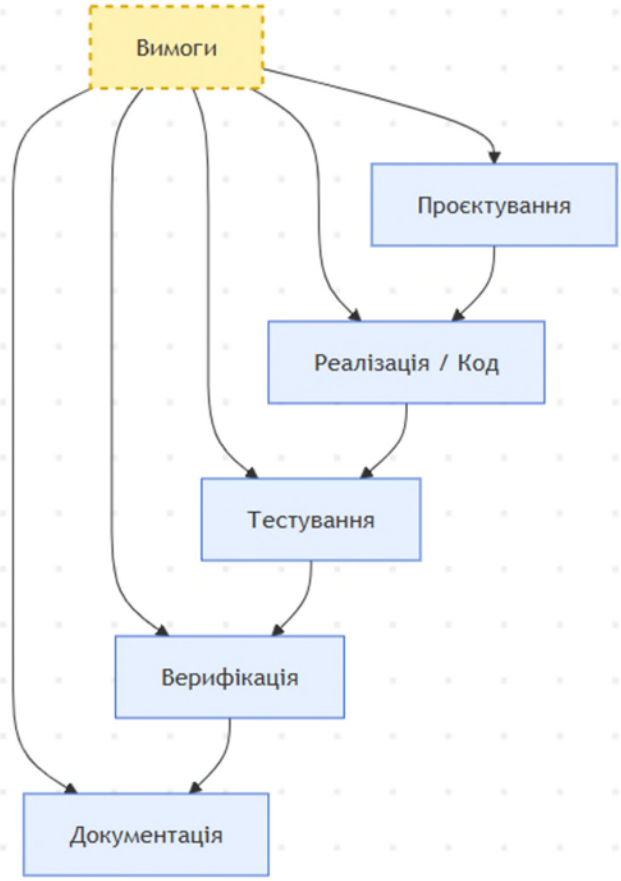


Рисунок 1.1. Місце трасування вимог у життєвому циклі розробки програмного забезпечення

Попри значущість процесу трасування, на практиці його впровадження супроводжується рядом труднощів. Основними проблемами, визначеними у систематичних оглядах літератури [5, 6], є висока трудомісткість і витратність ручного створення трасувальних зв’язків, особливо у великих проєктах, що охоплюють тисячі вимог і мільйони рядків коду. Така складність зумовлена не лише обсягом даних, а й постійними змінами в коді під час розробки. Додаткову проблему становить динамічність вимог: за результатами досліджень [7, 8], до 40 % трасувальних зв’язків втрачають актуальність уже на етапі інтеграційного тестування.

Серйозним викликом для автоматизації трасування є нечіткість формулювань вимог, що здебільшого подаються природною мовою, а отже, відзначаються неоднозначністю та неповнотою. Це ускладнює автоматичну

обробку текстів і створення коректних відповідностей між вимогами та програмним кодом [9, 10]. Ситуацію погіршує наявність семантичного розриву між природною мовою вимог і формальними структурами мов програмування. Класичні методи інформаційного пошуку, такі як TF-IDF, LSI чи VSM, забезпечують лише поверхневе порівняння термінів, не враховуючи глибокі семантичні зв’язки між поняттями [11, 12].

Не менш суттєвою проблемою є обмеження індустріальних систем керування вимогами, серед яких IBM DOORS, Polarion ALM та Jama Connect. Ці інструменти реалізують переважно ручне або напівавтоматичне трасування та не забезпечують достатньої точності в складних предметних областях [13, 14]. Додаткові труднощі створюють масштабованість і інтеграція, адже великі компанії часто використовують різні стандарти опису вимог і коду, що ускладнює централізований контроль трасувальних зв’язків.

Суттєвий вплив має і людський фактор. Результати трасування значною мірою залежать від досвіду аналітиків і розробників, що може призводити до пропусків або помилкових відповідностей. Це особливо критично для систем із високими вимогами до надійності – авіаційних, медичних чи фінансових. Додаткову складність становить трасування нефункціональних вимог, оскільки такі аспекти, як безпека, масштабованість або продуктивність, важко формалізувати та перевірити автоматично. Крім того, інтеграція трасування з автоматизованими системами тестування залишається обмеженою, хоча саме вона є важливою умовою забезпечення повного покриття вимог тестами.

Узагальнюючи, можна стверджувати, що сучасні методи трасування вимог не повною мірою задовольняють потреби масштабних і динамічних проєктів. Складність природної мови, великий обсяг програмного коду, а також необхідність підтримки актуальності зв’язків у процесі розробки визначають потребу у нових методах, здатних забезпечити високу точність і ефективність при мінімальних витратах часу.

Протягом останніх років з’являється тенденція до використання великих мовних моделей [15, 16], які враховують семантичні характеристики як тексту

вимог, так і програмного коду. Моделі, зокрема CodeBERT, GraphCodeBERT та UniXcoder, відкривають нові можливості для підвищення рівня автоматизації трасування та зменшення впливу людського чинника [17, 18].

Таким чином, аналіз сучасних підходів і проблем у сфері трасування вимог свідчить про необхідність створення нових методів, здатних поєднати семантичну глибину аналізу з ефективністю обчислень.

1.2 Огляд та аналіз існуючих методів, наукових публікацій та підходів до трасування у програмному забезпеченні

Задача трасування вимог у програмному забезпеченні має багаторічну історію досліджень, у якій можна виокремити кілька ключових етапів розвитку. Протягом цього часу формувалися різні підходи до встановлення зв’язків між текстовими вимогами та артефактами програмного коду, які еволюціонували разом із загальними тенденціями у галузі обробки природної мови та аналізу програмного коду. Початкові методи були засновані на простих статистичних підходах і дозволяли швидко оцінити схожість текстів, однак не враховували глибокого семантичного контексту чи структурні особливості коду. З розвитком науки і появою більш складних задач, що включали багатомовні проєкти та великі програмні системи, виникла потреба у підходах, здатних моделювати глибокі семантичні та синтаксичні залежності між вимогами та кодом. Це призвело до переходу від класичних IR-методів до моделей машинного навчання і сучасних нейронних архітектур на основі трансформерів.

Еволюцію методів трасування вимог та обробки природної мови, що демонструє перехід від статистичних моделей IR до нейронних архітектур DL/LLM, наведено на рисунку 1.2. На схемі показано ключові етапи розвитку підходів, починаючи від базових статистичних моделей, які працюють із текстовими кореляціями, до сучасних глибоких мереж і трансформерів, що дозволяють аналізувати складні взаємозв’язки між текстом вимог і структурою коду. Рисунок ілюструє не лише хронологічний розвиток методів, але й поступове

ускладнення підходів, а також зростання точності і семантичної насиченості моделей, що дозволяє ефективно застосовувати їх у великих і динамічних програмних проєктах.

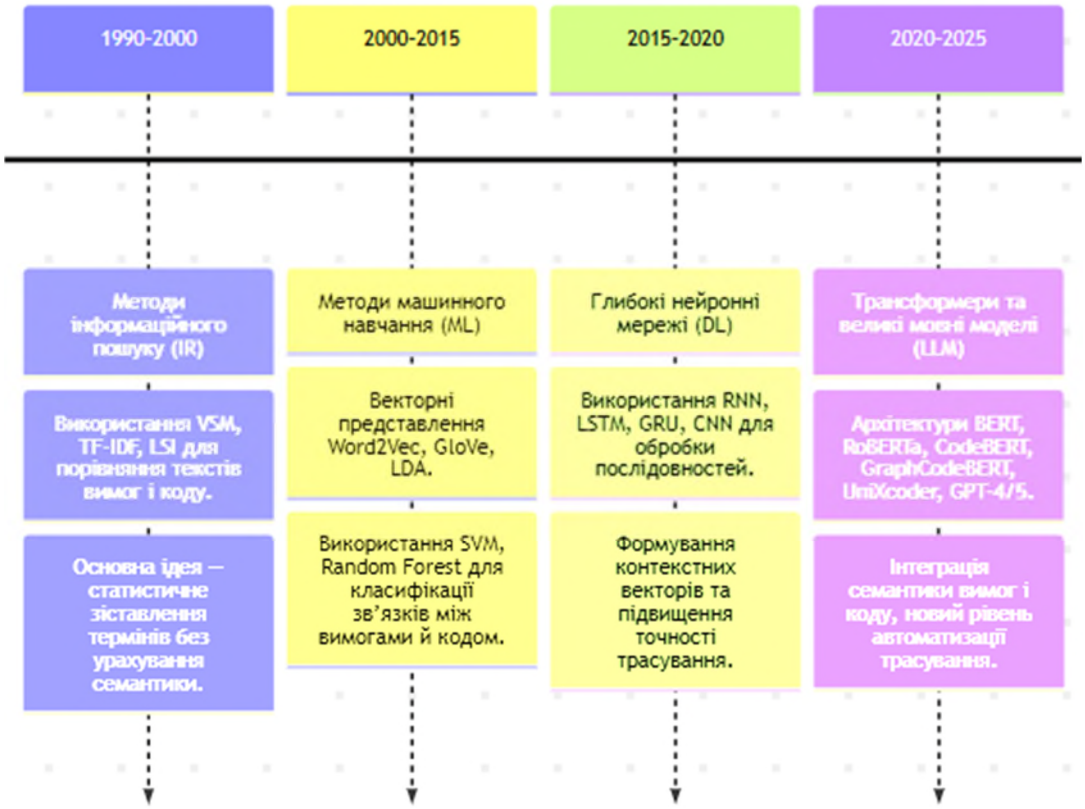


Рисунок 1.2 – Еволюція методів трасування та обробки природної мови у програмній інженерії

Умовно методи трасування поділяють на три основні групи: інформаційно-пошукові методи машинного навчання та підходи, що базуються на глибоких нейронних мережах і трансформерах.

Перші автоматизовані підходи до трасування вимог ґрунтувалися на методах інформаційного пошуку, які дозволяли порівнювати текстові вимоги з кодом або документацією на основі статистичних характеристик слів. Найбільш відомими представниками цього підходу є Vector Space Model, TF-IDF та Latent Semantic Indexing [19].

Головною перевагою IR-методів є їхня простота реалізації, висока швидкість роботи та відсутність потреби у великих навчальних наборах даних. Такі підходи ефективно працюють у невеликих проєктах або на початкових етапах

трасування, коли потрібно швидко визначити потенційні зв’язки між вимогами та кодом.

Водночас ці методи мають суттєві обмеження: вони ігнорують семантичний контекст, не враховують синтаксичну структуру коду, демонструють слабку масштабованість і є вразливими до неоднозначності природної мови. У результаті IR-підходи часто створюють надлишкові або хибні зв’язки, що знижує точність трасування. Незважаючи на це, саме IR-методи заклали основу для подальших досліджень у напрямі семантично орієнтованих моделей.

Подальший розвиток трасування пов’язаний із впровадженням методів машинного навчання, які дозволили моделювати зв’язки між вимогами та кодом на основі навчання на маркованих даних. Серед таких підходів найбільш відомими є Word2Vec, GloVe, FastText, а також тематичне моделювання – Latent Dirichlet Allocation і Probabilistic Latent Semantic Analysis. Для класифікації трасувальних пар застосовуються класичні алгоритми – Support Vector Machine, Random Forest та Logistic Regression [20].

Методи цієї групи здатні краще враховувати семантику та контекст у порівнянні з IR-підходами. Вони адаптуються до конкретного домену шляхом донавчання на відповідних наборах даних, що підвищує точність трасування у системах середнього розміру. Проте їхня ефективність безпосередньо залежить від якості й обсягу навчальних даних. Статичні вектори, такі як Word2Vec або GloVe, не враховують контекст у межах речення, а отже, не відображають глибокі відношення між текстом вимог і кодом. Це обмежує переносимість таких моделей між різними предметними областями.

У підсумку, хоча методи машинного навчання підвищили здатність моделей ураховувати семантику, їхня залежність від великих обсягів даних і неповна інтеграція з програмним кодом стимулювали подальший розвиток глибоких нейронних архітектур.

Останнє десятиліття ознаменувалося появою глибоких нейронних мереж, здатних моделювати складні семантичні й синтаксичні залежності між вимогами

та кодом. До таких архітектур належать RNN, LSTM, GRU, CNN, а також моделі на основі трансформерів – BERT, RoBERTa, XLNet, CodeBERT, GraphCodeBERT, UniXcoder, Codex і GPT-4/5 [21].

Ці моделі демонструють найвищу точність у порівнянні з попередніми підходами, оскільки враховують контекст не лише окремих слів, а й цілих речень і структур коду. Вони здатні аналізувати різні мови програмування, підтримують багатомовність і працюють у форматі векторних подань, які зберігають семантичну подібність між текстом вимог і кодом. Такі властивості дозволили досягнути значного підвищення показників Precision, Recall, F1-score та MAP у задачах трасування.

Однак застосування цих моделей має і певні недоліки: навчання трансформерів потребує значних обчислювальних ресурсів, а результати їхньої роботи складно інтерпретувати. Крім того, через складну архітектуру такі системи важко інтегрувати у традиційні пайплайни розробки, а їх ефективність може знижуватися при зміні домену чи мов програмування.

Порівняння основних груп методів трасування наведено у таблиці 1.1, яка узагальнює їхні типові приклади, переваги та недоліки.

Таблиця 1.1 – Порівняння методів трасування

Група методів	**Приклади**	**Переваги**	**Недоліки**
IR	VSM, TF-IDF, LSI	Швидкі, прості	Ігнорують семантику, структура коду, масштабування
ML	Word2Vec, GloVe, LDA, SVM	Краще врахування контексту	Залежність від даних, інтеграція з кодом, доменна переносимість
DL / Transformers	CodeBERT, GraphCodeBERT, UniXcoder, GPT-4/5	Семантичне розуміння, висока точність	Високі ресурси, складність fine-tuning, пояснюваність

Еволюція підходів до трасування відображає загальний поступ у галузі обробки природної мови й програмного коду: від статистичних моделей IR → ML → DL/Transformers. Поява великих відкритих наборів даних (MSR 2021) і моделей, здатних одночасно працювати з текстом і кодом (CodeBERT, UniXcoder), створює основу для подальшого вдосконалення трасування із застосуванням технологій retrieval + reranking та LLM.

Попри досягнення високої точності, сучасні методи мають проблеми: ресурсоємність навчання, складність інтерпретації результатів, чутливість до зміни даних та інтеграції у реальні пайплайни розробки. Це обумовлює необхідність подальших досліджень і комбінованих підходів.

1.3 Використання великих мовних моделей у трасуванні зв’язків між вимогами та програмним кодом

Зі зростанням можливостей трансформерних архітектур і появою великих мовних моделей таких як CodeBERT, GraphCodeBERT, Codex та GPT-сімейство, відкривається новий етап розвитку автоматизованого трасування вимог [22, 23]. На відміну від класичних методів інформаційного пошуку, LLM здатні поєднувати контекстне розуміння природної мови з аналізом структури програмного коду [24, 25]. Це дозволяє встановлювати семантично обґрунтовані відповідності між артефактами та значно підвищувати точність виявлення зв’язків між вимогами й реалізацією у програмному продукті.

Використання попередньо навченої моделі, адаптованої для роботи з кодом, дозволяє точніше визначати правильні зв’язки між вимогами та кодом, знаходити більшу частину існуючих зв’язків, покращувати загальну якість результатів, підвищувати середню точність пошуку для всіх вимог і забезпечувати, щоб перші правильні результати з’являлися якомога вище у списку, що підтверджується сучасними дослідженнями [26, 27].

Одним із ключових напрямів розвитку трасування є застосування LLM у різних архітектурних підходах. На рисунку 1.3 подано узагальнену схему основних напрямів використання великих мовних моделей у процесі трасування вимог до програмного коду.

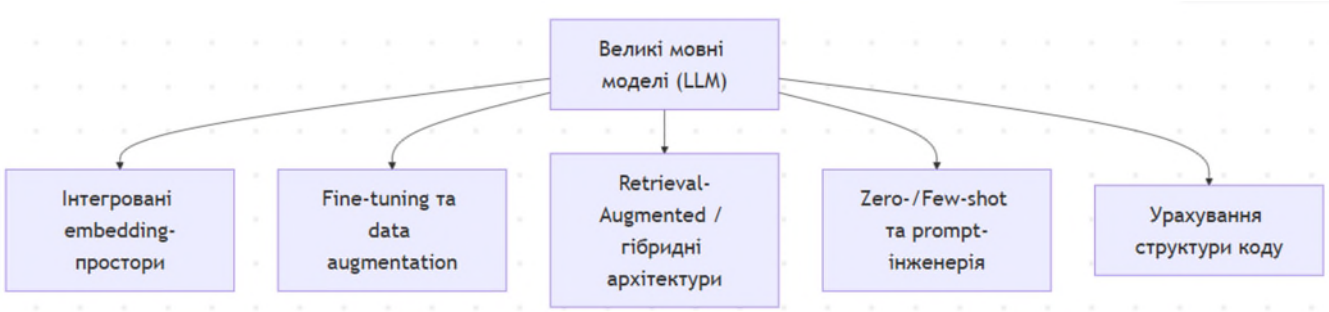


Рисунок 1.3 – Основні напрями використання великих мовних моделей у трасуванні вимог до програмного коду

Першим є застосування інтегрованих embedding-просторів. Цей підхід передбачає відображення текстових вимог і фрагментів програмного коду у спільний векторний простір [28]. Це дозволяє визначати семантичну подібність між артефактами за допомогою косинусної метрики. Моделі CodeBERT, GraphCodeBERT або MS-CodeBERT враховують синтаксичні та семантичні залежності, що забезпечує вищу точність порівняно з традиційними методами TF-IDF чи LSI [29, 30]. Поєднання спільного embedding-простору з механізмами ранжування кандидатів і пороговою фільтрацією дозволяє ефективно зменшувати кількість хибнопозитивних відповідностей [31, 32].

Другим напрямом є fine-tuning і data augmentation. Навчання попередньо натренованих моделей на парних прикладах «вимога-код» дозволяє суттєво підвищити точність трасування. Використання технік data augmentation – парафразування текстів, модифікацій синтаксису коду чи додавання шуму – розширює навчальні дані та допомагає моделі узагальнювати контекст. Fine-tuning, орієнтований на конкретний домен (наприклад, медичне чи фінансове програмне забезпечення), покращує стабільність показників Precision і F1-score та підвищує переносимість моделей.

Наступний напрям – Retrieval-Augmented Generation та гібридні архітектури. У таких системах пошуковий модуль спочатку формує набір потенційно релевантних зв’язків, а потім генеративна модель уточнює або підтверджує їх. Це поєднує швидкість пошуку з глибиною семантичного аналізу, забезпечуючи підвищення точності та пояснюваності результатів.

Ще один перспективний напрям – Zero-/Few-shot та prompt-інженерія. Сучасні генеративні LLM можуть виконувати трасування навіть без додаткового навчання, якщо промпт сформульований коректно. У режимах zero-shot або few-shot модель здатна визначати фрагменти коду, що реалізують описані вимоги. Ефективність методу безпосередньо залежить від структури промптів і контекстних підказок. Дослідження показують, що застосування стандартизованих шаблонів знижує ризик «галюцинацій» і підвищує стабільність відповідей моделі.

Останній із ключових напрямів – урахування структури коду. Моделі на зразок GraphCodeBERT інтегрують у процес навчання структурну інформацію про код – абстрактне синтаксичне дерево або потоки даних. Це дозволяє моделі враховувати внутрішні залежності між змінними, викликами функцій і блоками програми, що забезпечує кращу семантичну узгодженість між кодом і текстовими вимогами.

Попри високу ефективність, LLM мають низку обмежень. По-перше, вони потребують значних обчислювальних ресурсів і часу на тренування. Для оптимізації цього процесу застосовуються техніки часткового донавчання, використання компактних моделей або архітектур із попереднім фільтром кандидатів. По-друге, LLM схильні до утворення хибних зв’язків, що потребує інтеграції механізмів верифікації, порогового скорингу або участі експертів. По-третє, моделі, навчені на загальних корпусах коду, можуть не забезпечувати належної точності у вузьких предметних галузях, тому актуальним є domain fine-tuning або ін’єкція онтологій. Крім того, важливим викликом залишається відсутність прозорості процесу прийняття рішень, що є критичним для високоризикових сфер – медицини, фінансів, авіації.

Сучасні системи трасування, що використовують LLM, зазвичай будуються на основі гібридної pipeline-архітектури. Такий підхід поєднує швидке первинне вилучення кандидатів із подальшим глибоким семантичним аналізом, який виконується за допомогою LLM. На рисунку 1.4 подано узагальнену структуру подібного конвеєра, що включає послідовні етапи: виділення кандидатів, семантичне ранжування за векторними представленнями, LLM-орієнтовану перевірку або генерацію відповідностей, а також фінальну перевірку експертом.

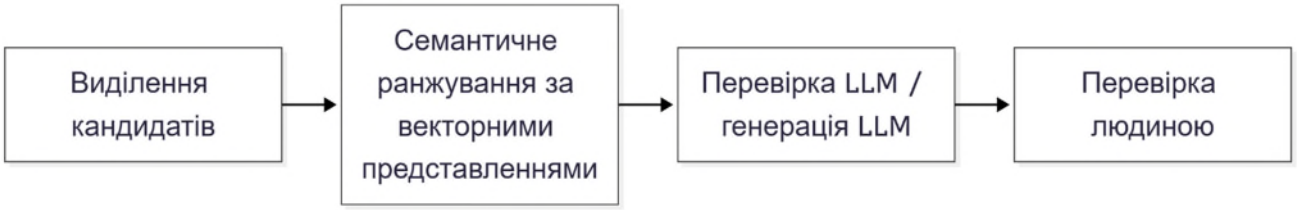


Рисунок 1.4 – Гібридна pipeline-архітектура сучасних систем трасування на основі великих мовних моделей

Такий підхід забезпечує баланс між продуктивністю, точністю та контрольованістю процесу трасування.

Перспективними напрямами розвитку великих мовних моделей у задачах трасування є: використання методів полегшеної адаптації, що зменшують ресурсоємність навчання; застосування ансамблевих підходів, у межах яких швидкі моделі виконують початкову фільтрацію, а більш потужні – уточнюють результати; автоматичне формування пояснень, що підвищує прозорість і обґрунтованість встановлених зв’язків; доменно-орієнтоване попереднє навчання на спеціалізованих наборах даних; а також розроблення стандартизованих тестових наборів і бенчмарків для порівняння різних методів трасування на основі мовних моделей.

LLM відкривають нові можливості для автоматизації процесу встановлення трасувальних зв’язків завдяки здатності комплексно аналізувати як природну мову, так і програмний код. Найефективнішими вважаються гібридні архітектури, що поєднують пошукові механізми з генеративними компонентами, а також донавчання моделей з урахуванням особливостей конкретних предметних

галузей. У наукових роботах показано, що поєднання модуля первинного пошуку з генеративною моделлю суттєво підвищує точність встановлення зв’язків між артефактами, застосування спільного семантичного простору покращує показники середнього зворотного рангу та повноти, а донавчання трансформерних моделей забезпечує стабільність і узагальнюваність результатів.

Отже, LLM є перспективним інструментом для розроблення методів трасування нового покоління, що поєднують високу точність, контекстну глибину та потенціал до адаптації у різних предметних галузях.

1.4 Постановка задачі дослідження

Метою роботи є підвищення точності та повноти автоматичного виявлення трасувальних зв’язків між текстовими вимогами та методами програмного коду шляхом розробки методу, що базується на використанні великих мовних моделей.

Для досягнення поставленої мети потрібно виконати наступні завдання:

– провести аналіз існуючих підходів для автоматизації трасування зв’язків між вимогами та програмним кодом;

– розробити метод для автоматичного виявлення трасувальних зв’язків на основі векторного представлення вимог та коду з використанням великих мовних моделей;

– обґрунтувати вибір та виконати попередню обробку набору даних для навчання моделі, забезпечивши коректність структури, очищення та підготовку артефактів для подальших експериментів;

– провести експериментальну перевірку запропонованого методу за відомими статистичними показниками та порівняти отримані результати з відомими підходами.

Розділ 2 Метод виявлення трасувальних зв’язків між вимогами та програмним кодом з використанням великих мовних моделей

2.1 Концепція та схема запропонованого методу

Автоматизація процесу трасування вимог у програмному забезпеченні є актуальною задачею, оскільки традиційні ручні та напівавтоматизовані підходи не забезпечують достатньої точності та масштабованості у великих проєктах.

Концепція запропонованого методу базується на двоетапній архітектурі, що поєднує швидкий пошук кандидатів на основі векторних подань та точне перепорядкування за допомогою глибокого семантичного аналізу.

На першому етапі метод формує початковий набір потенційних відповідностей між вимогами та кодом, а на другому – уточнює їх, використовуючи LLM здатні враховувати контекст та семантичні залежності.

Лише після опису загальної концепції обирається конкретна модель для реалізації, CodeBERT [33], яка забезпечує ефективне формування векторних подань для тексту та програмного коду.

Запропонована схема методу включає такі основні етапи:

Вхідні дані. Формування та збір вхідних даних. На цьому початковому етапі відбувається завантаження та структуризація двох невпорядкованих наборів артефактів програмного забезпечення: множини текстових вимог та множини методів, виокремлених із вихідного коду. Джерелом цих даних можуть виступати як репозиторії відкритих проектів, так і документація та кодова база власного програмного продукту

1. Попередня обробка даних. Для забезпечення коректного порівняння різнорідних даних та мінімізації впливу синтаксичного шуму виконується їх уніфікація:

– Обробка вимог: включає очищення тексту від службових символів, пунктуації та стоп-слів, приведення всіх символів до нижнього регістру, а також лематизацію або стемінг для зведення слів до їхньої основи.

– Обробка коду: передбачає токенізацію ідентифікаторів та розбиття складених назв на окремі слова (наприклад, перетворення getUserName → get user name). Також виконується видалення коментарів, літералів та надлишкових пробілів.

– Нормалізація: здійснюється уніфікація назв змінних і методів, що дозволяє зменшити рівень шуму у векторних поданнях та підвищити точність моделі.

2. Подання у векторному просторі. На цьому етапі попередньо оброблені вимоги та методи коду трансформуються у щільні семантичні вектори фіксованої розмірності. Використання CodeBERT забезпечує уніфіковане математичне представлення, де семантично схожі фрагменти тексту та коду знаходяться близько один до одного у багатовимірному просторі, що є основою для подальшого порівняння.

3. Пошук кандидатів. Для кожної окремої вимоги виконується пошук найбільш імовірних відповідностей серед методів коду. Відбір здійснюється на основі метрики косинусної подібності (Cosine Similarity) між векторами. Результатом етапу є формування початкового списку потенційних трасувальних зв’язків, що дозволяє звузити простір пошуку для наступних кроків.

4. Перепорядкування та верифікація. Кожна пара «вимога – метод-кандидат» з початкового списку піддається глибокому аналізу за допомогою моделі-класифікатора. Класифікатор обчислює точну ймовірність наявності семантичного зв’язку між елементами пари. Цей етап є критично важливим для відсіювання хибно позитивних відповідностей, які могли виникнути при первинному пошуку.

5. Формування матриці трасувальності. Фінальні результати оцінювання агрегуються у вигляді двовимірної матриці відповідності. Рядки цієї матриці представляють вимоги, а стовпці – методи програмного коду. Значення у клітинках відображають обчислену ймовірність існування зв’язку, що дає змогу оцінити впевненість моделі у кожному конкретному випадку.

6. Генерація вихідних результатів. На основі фінальної матриці для кожної вимоги формується ранжований список методів коду, які з найбільшою ймовірністю реалізують її функціональність. Такий формат подання результатів дозволяє системним аналітикам та розробникам швидко ідентифікувати відповідні ділянки коду, значно прискорюючи процеси верифікації, супроводу та рефакторингу програмного забезпечення.

На рисунку 2.1 представлено концептуальну схему запропонованого методу, що відображає послідовність основних етапів процесу трасування.

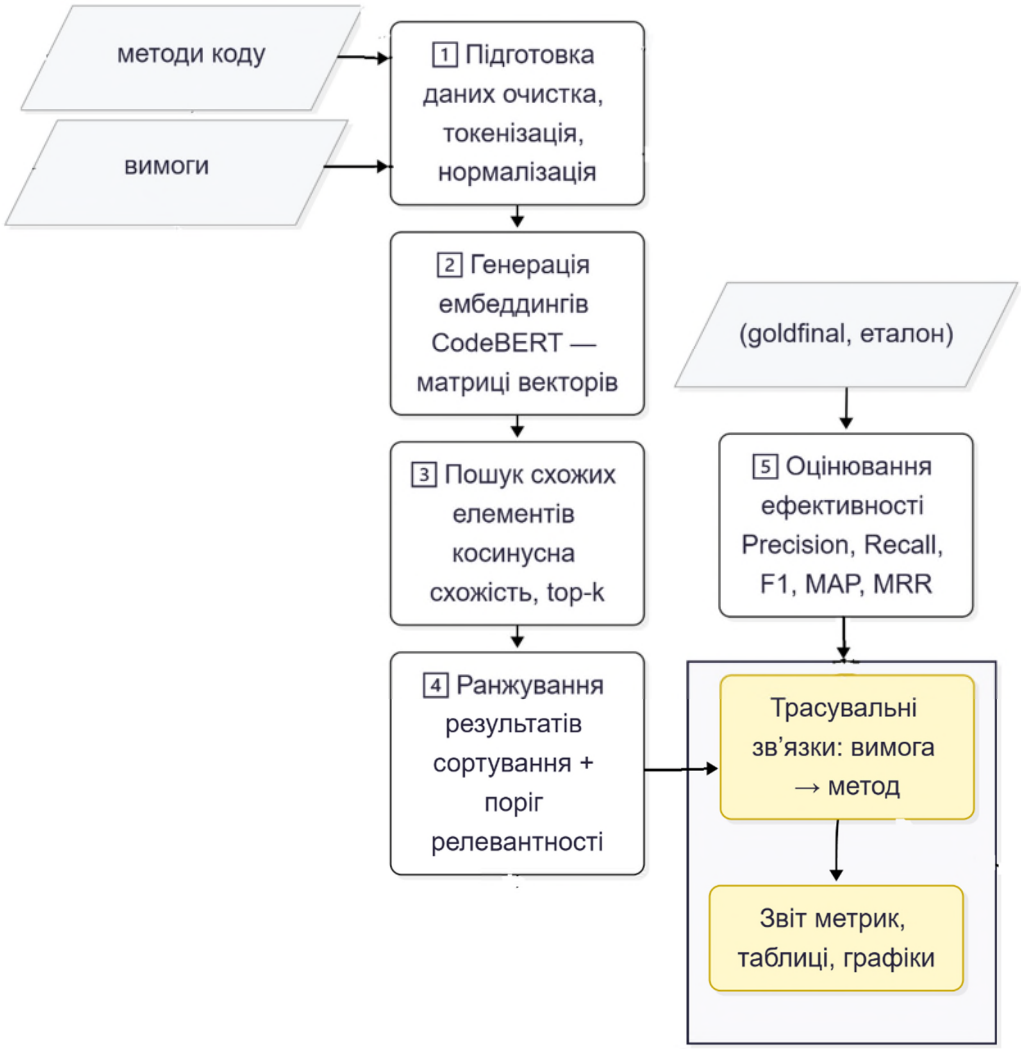


Рисунок 2.1 – Концептуальна схема методу виявлення трасувальних зв'язків між вимогами та програмним кодом

Концептуальна схема передбачає послідовне виконання етапів, починаючи з попередньої підготовки даних і закінчуючи побудовою матриці трасувальності.

На етапі формування векторних подань використовуються LLM , які дозволяють створювати узгоджене семантичне представлення тексту вимог і програмного коду.

Комбінація швидкого пошуку та уточненого перепорядкування забезпечує баланс між точністю та ефективністю, що робить запропонований метод придатним для використання у великих програмних системах з високими вимогами до якості трасування.

2.2 Побудова векторних представлень вимог та методів коду

Побудова векторних представлень вимог та методів програмного коду є одним із ключових етапів розробленого методу трасування. Саме на цьому етапі здійснюється перехід від неструктурованих текстових артефактів до числових репрезентацій, що дозволяють виконувати подальше обчислення семантичної подібності між вимогами та кодом. Якість отриманих векторів безпосередньо впливає на точність відновлення трасувальних зв’язків, оскільки визначає, наскільки адекватно модель здатна відобразити смислову подібність між різнотипними сутностями.

Традиційні підходи до побудови таких представлень, засновані на методах інформаційного пошуку, зокрема TF-IDF або LSI, характеризуються поверхневим рівнем аналізу, оскільки враховують лише частотні або статистичні ознаки слів Вони не здатні повною мірою відображати контекст та семантичні залежності між елементами тексту і програмного коду, що призводить до помилкових або неповних відповідностей. Тому в межах даного дослідження було обрано сучасний підхід, який ґрунтується на використанні глибоких нейронних мереж трансформерного типу.

Для побудови векторних представлень використано попередньо навчену модель CodeBERT, яка забезпечує формування єдиного семантичного простору для природної мови та програмного коду. Ця модель побудована на архітектурі трансформера і навчена на великому корпусі пар «код – опис», що дозволяє їй

ефективно виявляти смислові відповідності між текстовими вимогами та реалізацією у вихідному коді. На відміну від універсальних мовних моделей, CodeBERT враховує синтаксичні особливості програмних мов і контекст їх використання, завдяки чому формує вектори, релевантні як для текстових, так і для кодових артефактів.

Перед безпосереднім формуванням векторних представлень здійснюється попередня обробка вхідних даних, спрямована на очищення та уніфікацію структури текстів. Для текстових вимог виконується видалення службових символів, токенізація, приведення тексту до нижнього регістру та лематизація. Ці операції дозволяють зменшити кількість варіацій у лексемах і підвищити узгодженість семантичного простору. Оскільки вихідні дані набору MSR 2021 Traceability Dataset представлені англійською мовою, усі процедури попередньої обробки виконуються для англомовного тексту. Для програмного коду проводиться видалення коментарів, розділення складених ідентифікаторів, нормалізація назв змінних та методів. Такі кроки забезпечують чистоту даних і зменшують кількість шумових факторів, що можуть впливати на формування векторів.

Після підготовки даних модель CodeBERT перетворює кожну вимогу та кожен метод коду на щільне векторне представлення фіксованої розмірності. Вектори формуються у спільному багатовимірному просторі, у якому семантично подібні об’єкти розташовуються на близьких відстанях. Це дає можливість безпосередньо порівнювати різнотипні артефакти – природномовні описи та програмні фрагменти – на основі математичних мір подібності

На рисунку 2.2 представлено узагальнену схему процесу побудови векторних представлень, що демонструє послідовність дій від попередньої обробки даних до формування спільного семантичного простору.

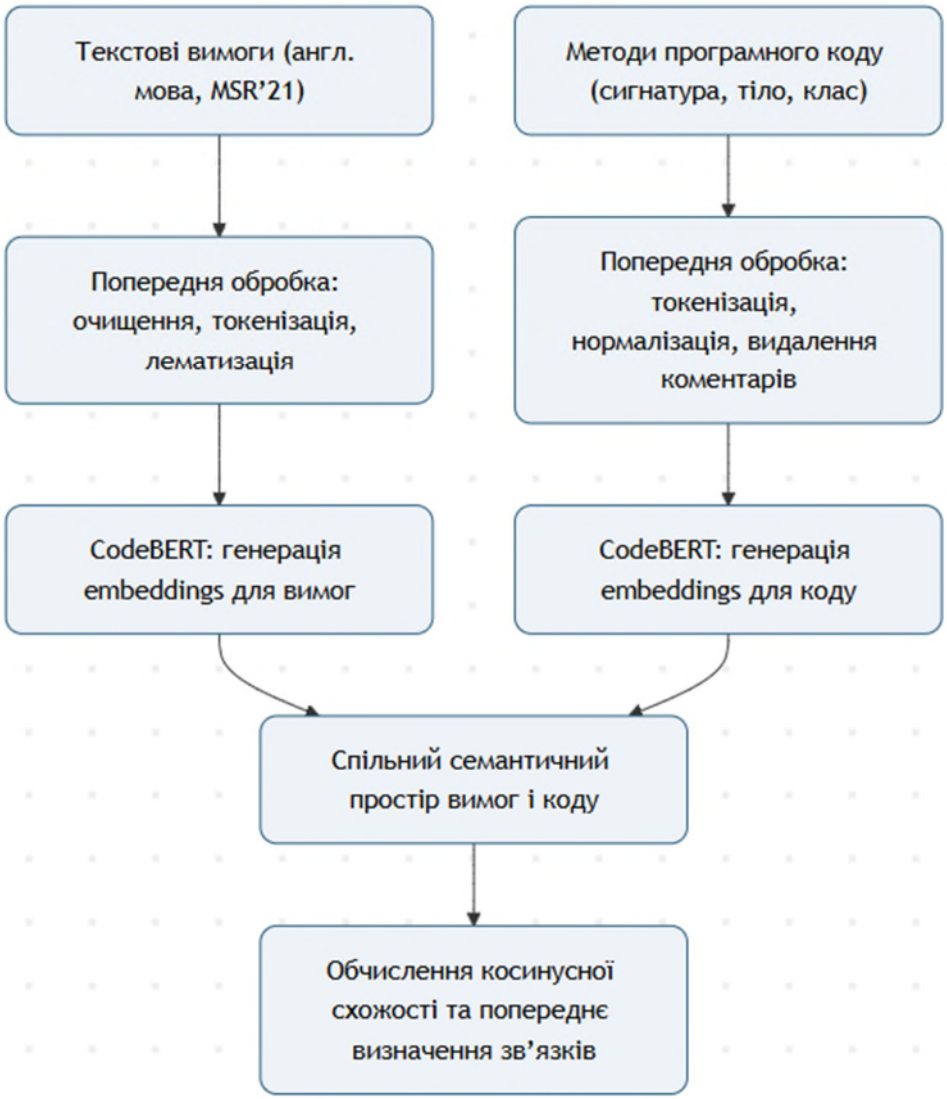


Рисунок 2.2 – Схема етапу побудови векторних представлень вимог та методів коду

У результаті цього етапу формується спільний семантичний простір, у якому кожна вимога та кожен метод коду мають власне векторне представлення. Це створює основу для подальших етапів методу, зокрема пошуку кандидатів і їхнього перепорядкування, що виконуються на основі обчислених показників семантичної схожості. Таким чином, побудова векторних представлень забезпечує фундамент для точного, масштабованого та інтерпретованого процесу трасування вимог до програмного коду.

2.3 Архітектура методу: retrieval → rerank за допомогою LLM

Запропонована архітектура методу базується на двоетапному підході, який поєднує ефективність швидкого пошуку кандидатів та високу точність перепорядкування виконаного за допомогою великої мовної моделі. Така архітектура дозволяє збалансувати продуктивність обчислень і семантичну глибину аналізу, що є особливо важливим під час обробки великих кодових баз і складних текстових описів вимог.

На першому етапі, retrieval, для кожної вимоги здійснюється пошук найбільш релевантних методів програмного коду. Пошук виконується у векторному просторі, сформованому на попередньому етапі, шляхом обчислення косинусної схожості між векторними представленнями вимог і методів коду:

$$cosine\_similarity(\vec{q}, \vec{d}) = \frac{\vec{q} * \vec{d}}{||\vec{q}\,|| \cdot ||\,\vec{d}||}, \quad (1)$$

де $\vec{q}$ – векторне представлення вимоги, $\vec{d}$ – векторне представлення методу коду, $\vec{q} * \vec{d}$ – скалярний добуток, $\|\cdot\|$ – норма вектора.

На основі косинусної схожості формується функція ранжування кандидатів:

$$score(q, d_i) = f(cosine\_similarity(\vec{q}, \overrightarrow{d_i})) \quad (2)$$

де f – функція, що може додатково враховувати інші параметри, наприклад BM25, довжину тексту або вагу термінів.

Для відбору Top-K кандидатів процес можна формально записати як:

$$TopK(q) = \{d_i \mid score(q, d_i)\ \epsilon\ K_{max}\} \quad (3)$$

Таким чином, на етапі retrieval обираються K найбільш релевантних фрагментів коду для кожної вимоги. Цей етап виступає фільтром, що суттєво зменшує розмір пошукового простору для наступного етапу reranking, забезпечуючи швидкість та масштабованість процесу.

Другий етап – rerank передбачає детальну оцінку релевантності пар «вимога – метод коду» за допомогою донавченої двонаправленої трансформерної моделі типу cross-encoder, побудованої на основі архітектури BERT[34]. На відміну від класичних моделей, які обробляють запит і документ окремо, cross-encoder об’єднує їх у єдину послідовність і аналізує взаємодію між токенами двох текстів одночасно. Такий підхід забезпечує глибше семантичне розуміння і дозволяє точніше оцінювати відповідність між вимогою та конкретним фрагментом коду. Модель повертає числову оцінку релевантності в діапазоні від 0 до 1, що використовується для впорядкування знайдених кандидатів і визначення порогу належності.

Для адаптації до специфіки предметної області cross-encoder було донавчено на підготовленому наборі пар «вимога – код» із датасету MSR 2021 Traceability Dataset. Це дозволило моделі навчитися розпізнавати доменні особливості програмних артефактів, термінологію та стилістичні особливості описів вимог. У результаті reranking забезпечує суттєве підвищення точності порівняно з базовими retrieval-підходами.

Архітектура retrieval → rerank поєднує швидкість пошуку та глибину семантичного аналізу. Перший етап гарантує високу масштабованість системи, тоді як другий – забезпечує глибоке контекстне розуміння зв’язків. Такий підхід дозволяє не лише скоротити обчислювальні витрати, а й мінімізувати кількість хибнопозитивних зв’язків. Крім того, автоматизоване ранжування значно зменшує обсяг ручної роботи аналітиків і тестувальників, прискорюючи перевірку трасувальних зв’язків.

На рисунку 2.3 представлено архітектурну схему запропонованого методу, що відображає взаємодію між етапами побудови, пошуку кандидатів і перепорядкування за допомогою LLM-моделі.

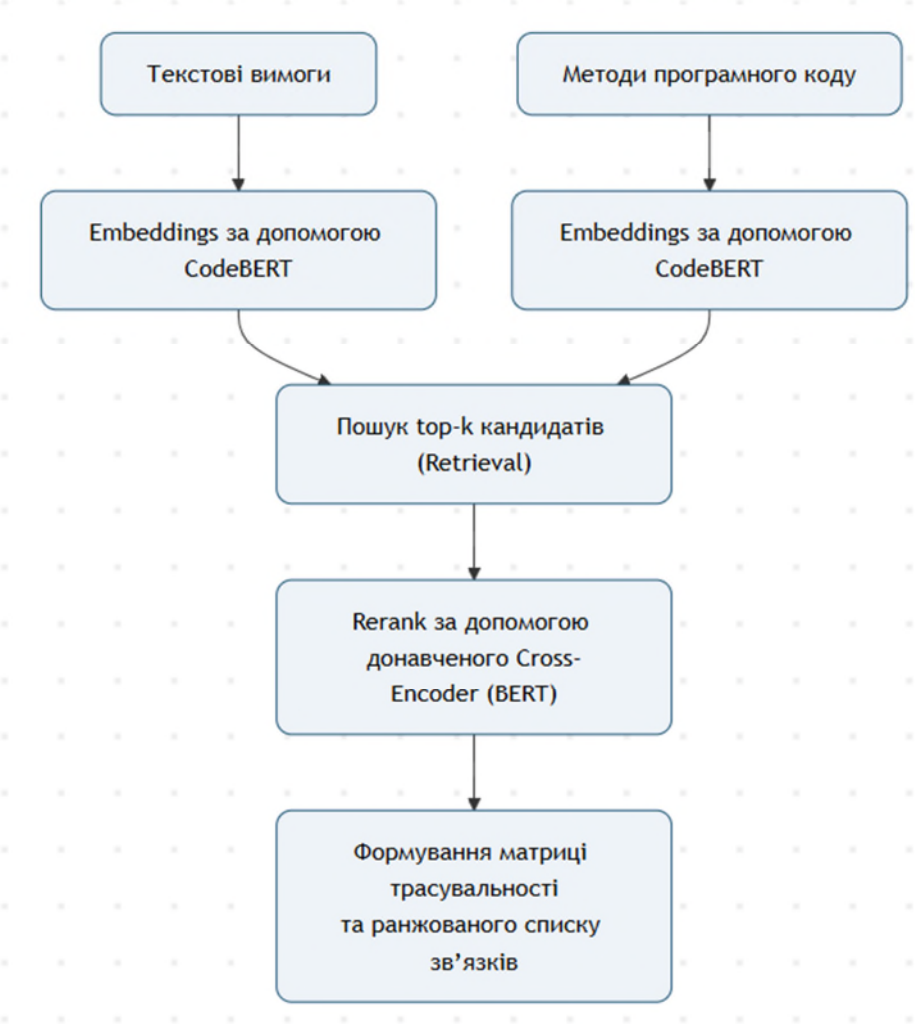


Рисунок 2.3 – Архітектура методу трасування: retrieval → rerank за допомогою LLM

Таким чином, запропонована архітектура поєднує ефективність класичних методів векторного пошуку та точність сучасних нейронних мереж. Її використання забезпечує баланс між швидкістю обробки, семантичною точністю та пояснюваністю результатів, що робить метод придатним для застосування у складних проєктах із великою кількістю вимог і фрагментів коду.

2.4 Критерії та метрики оцінювання роботи методу

Для кількісної оцінки ефективності розробленого методу автоматичного трасування вимог до програмного коду використовуються п'ять основних показників: Precision, Recall, F1-score, MAP та MRR [35]. Сукупне застосування

цих метрик дозволяє об’єктивно оцінити як якість класифікації пар «вимога – метод коду», так і точність ранжування знайдених зв’язків.

Метрика Precision відображає частку правильних позитивних прогнозів серед усіх зв’язків, які модель класифікувала як релевантні. Високе значення Precision означає, що більшість знайдених пар справді мають коректний трасувальний зв’язок, тобто модель майже не створює «шумових» відповідностей. Цей показник особливо важливий на етапі практичного використання системи, коли аналітик перевіряє лише верхні результати ранжування.

Метрика Recall характеризує здатність методу знаходити всі наявні істинні зв’язки. Вона визначає частку коректних пар «вимога – метод», які були виявлені системою з усіх можливих у тестовому наборі. Для задачі трасування вимог високий показник Recall має особливе значення, оскільки пропуск релевантного зв’язку може призвести до неповної реалізації функціональних вимог або втрати важливих залежностей у коді.

Метрика F1-score є гармонійним середнім між Precision і Recall і забезпечує збалансовану оцінку якості моделі в умовах, коли обидва показники є важливими. Вона дозволяє враховувати як точність визначення зв’язків, так і їх повноту, що робить її інтегральним показником загальної ефективності методу.

Крім класифікаційних метрик, для оцінювання якості ранжування застосовуються MAP та MRR. Метрика Mean Average Precision визначає середню точність на релевантних позиціях у впорядкованому списку кандидатів і відображає, наскільки добре система концентрує правильні зв’язки на верхніх позиціях результатів. Високе значення MAP свідчить про те, що релевантні методи коду з великою ймовірністю будуть серед перших знайдених, що скорочує час ручної перевірки результатів.

Метрика Mean Reciprocal Rank відображає середній зворотний ранг першого правильного результату для кожної вимоги. Вона показує, наскільки швидко користувач може знайти перший коректний зв’язок у ранжованому списку. Чим більше значення MRR, тим зручнішою і практично ефективнішою є робота системи.

Усі метрики узагальнено в таблиці 2.1, де наведено їх зміст і призначення.

Таблиця 2.1 – Метрики оцінювання роботи методу.

Метрика	**Формула**	**Пояснення**
Precision	$\frac{TP}{TP + FP}$	Частка правильних позитивних прогнозів серед усіх передбачених як позитивні.
Recall	$\frac{TP}{TP + FN}$	Частка знайдених істинних позитивних прикладів серед усіх існуючих.
F1-score	$2 * \frac{Precision \cdot Recall}{Precision + Recall}$	Гармонійне середнє між Precision та Recall.
MAP	$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{m_1}\sum_{k=1}^{m_i} P_i(k)$	Середня точність на релевантних позиціях у списку кандидатів.
MRR	$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{rank_i}$	Середнє обернене значення рангу першого релевантного результату.

Таким чином, вибрані метрики забезпечують комплексну оцінку ефективності розробленого методу з погляду якості виявлення та ранжування трасувальних зв’язків. Вони дозволяють оцінити модель як з точки зору формальної точності класифікації, так і з позиції практичної корисності для інженерів програмного забезпечення.

Висновки до розділу 2

У цьому розділі було розроблено та обґрунтовано метод виявлення трасувальних зв’язків між вимогами та програмним кодом на основі великих мовних моделей. Запропонований підхід поєднує ефективність векторного пошуку з високою семантичною точністю глибоких нейронних мереж, що забезпечує суттєве підвищення якості автоматичного трасування у порівнянні з традиційними методами інформаційного пошуку.

Було сформовано концепцію методу, яка базується на двоетапній архітектурі retrieval → rerank. На першому етапі виконується швидкий пошук кандидатів на основі косинусної схожості векторних подань вимог та методів коду. На другому етапі здійснюється точне перепорядкування знайдених пар за допомогою донавченої трансформерної моделі типу cross-encoder, що дозволяє враховувати контекстні та семантичні залежності між вимогами та кодом.

Для побудови векторних представлень обрано модель CodeBERT, яка забезпечує єдиний семантичний простір для природної мови та програмного коду. Завдяки цьому досягається глибше розуміння змісту артефактів і можливість безпосереднього порівняння вимог із відповідними методами програмного забезпечення.

Для оцінювання ефективності методу визначено основні кількісні метрики – Precision, Recall, F1-score, MAP та MRR, які відображають як точність класифікації зв’язків, так і якість ранжування кандидатів. Використання цих метрик забезпечує комплексну оцінку роботи моделі та дозволяє порівнювати результати з іншими підходами, описаними у науковій літературі.

Таким чином, у результаті проведеного дослідження було створено узгоджену архітектуру методу трасування вимог до програмного коду, визначено алгоритмічну послідовність його етапів, обґрунтовано вибір моделей і даних для навчання, а також встановлено систему критеріїв для подальшої експериментальної перевірки. Отримані результати створюють основу для практичної реалізації методу та його оцінювання у наступному розділі.

Розділ 3 Програмна реалізація методу виявлення трасувальних зв’язків між вимогами та програмним кодом з використанням великих мовних моделей

3.1 Засоби та середовища реалізації

У цьому розділі наведено процес програмної реалізації запропонованого методу, описано обране середовище розроблення, архітектуру системи та основні модулі, що забезпечують автоматизацію трасування вимог

Для розробки та впровадження методу автоматичного виявлення трасувальних зв’язків між вимогами та програмним кодом було обрано сучасний комплекс засобів, який забезпечує ефективну обробку природної мови та коду, а також зручну інтеграцію з існуючими інструментами управління вимогами.

1. Мова програмування.

Основною мовою для реалізації системи обрана Python[36] 3.13. Python є оптимальним вибором завдяки широкій екосистемі бібліотек для машинного навчання, обробки тексту, роботи з великими мовними моделями, а також завдяки простоті інтеграції з інструментами для аналізу коду. Крім того, Python дозволяє швидко прототипувати алгоритми та здійснювати експерименти з різними архітектурами моделей.

2. Інтегроване середовище розробки.

Для створення прототипу та основної розробки програмного забезпечення використовується Visual Studio Code[37]. Це середовище надає розширені можливості для написання та відладки коду, інтеграції систем контролю версій, а також підтримує численні плагіни для роботи з Python та обробки даних. Вибір Visual Studio Code обумовлений його зручністю для комплексної розробки, можливістю швидкого переходу між модулями та підтримкою інтерактивних середовищ виконання, таких як Jupyter Notebook.

3. Середовище для тренування моделей.

Хоча основна розробка ведеться у Visual Studio Code, для тренування великих мовних моделей застосовується Google Colab[38]. Colab надає доступ до

GPU та TPU, що суттєво прискорює обчислення ембедингів коду та вимог. Крім того, Colab дозволяє легко обмінюватися ноутбуками, що містять як код, так і результати експериментів, що є важливим для перевірки та відтворення результатів дослідження. Такий підхід дозволяє відокремити процес розробки прототипу від ресурсоємного процесу тренування моделей.

4. Використані бібліотеки та фреймворки.

– Трансформер Hugging Face[39] – використовується для інтеграції великих мовних моделей, таких як CodeBERT та GPT, та їх донавчання на специфічних датасетах для підвищення точності трасування.

– PyTorch[40] – забезпечує гнучку роботу з нейронними мережами, оптимізацію процесу тренування та підтримку обчислень на GPU.

– Scikit-learn[41], Pandas[42], NumPy – застосовуються для попередньої обробки даних, нормалізації ембедингів, аналізу схожості між кодом і вимогами, а також для статистичного аналізу результатів.

5. Інструменти для роботи з кодом та даними

Для ефективного аналізу програмного коду та структурованих даних використовується:

– Abstract Syntax Tree[43] – для парсингу вихідного коду та отримання його структурного представлення, що дозволяє враховувати синтаксичну та семантичну структуру при визначенні трасувальних зв’язків.

– JSON – як зручний формат для зберігання вимог, методів та результатів трасування, що забезпечує простоту обміну даними між модулями системи.

6. Підхід до інтеграції великих мовних моделей

Для виявлення трасувальних зв’язків використовується комбінація готових моделей та їх донавчання на специфічному датасеті проекту. Вхідні дані у вигляді текстових описів вимог та фрагментів коду перетворюються на векторні представлення, після чого обчислюється схожість між ними. Такий підхід дозволяє автоматично визначати потенційні трасувальні зв’язки з високою точністю, зменшуючи необхідність ручного аналізу.

7. Переваги обраного середовища та інструментів

Вибір поєднання Visual Studio Code для розробки та Google Colab для тренування моделей забезпечує низку суттєвих переваг, що підвищують ефективність процесу створення системи трасування та полегшують роботу з великими обсягами даних:

1. Оптимізація процесу розробки та тестування:

Visual Studio Code дозволяє зручно писати, редагувати та налагоджувати код, підтримує інтеграцію з системами контролю версій, автоматичне форматування коду та перевірку синтаксису. Це скорочує час на виявлення помилок та підвищує продуктивність розробника.

2. Можливість відокремленого тренування моделей:

Використання Google Colab для тренування великих мовних моделей забезпечує доступ до апаратного прискорення без необхідності придбання дорогого локального обладнання. Це особливо важливо при роботі з великими датасетами та LLM, які потребують значних обчислювальних ресурсів.

3. Інтерактивність та гнучкість експериментів:

Colab підтримує інтерактивні ноутбуки, у яких одночасно можна виконувати код, відображати результати тренування, графіки та таблиці. Це дозволяє швидко перевіряти гіпотези, змінювати параметри моделей та аналізувати вплив цих змін на точність визначення трасувальних зв’язків.

4. Повторюваність та масштабованість експериментів:

Використання ноутбуків Colab та структурованого коду у Visual Studio Code забезпечує можливість повторного запуску експериментів та масштабування системи для обробки більших датасетів. Збережені ноутбуки та скрипти можна легко передавати іншим дослідникам або колегам, що підвищує наукову відтворюваність результатів.

5. Інтеграція сучасних бібліотек машинного навчання:

Поєднання Python та доступних бібліотек (Transformers, PyTorch, Scikit-learn, Pandas, NumPy) забезпечує легку інтеграцію складних моделей обробки природної мови та обробку великих обсягів даних. Таке середовище дозволяє

експериментувати з різними архітектурами моделей та методами обчислення схожості між кодом і вимогами.

6. Гнучкість у роботі з даними та форматами:

JSON-файли та структуровані представлення коду через абстрактне синтаксичне дерево забезпечують простий обмін даними між модулями системи, а також їх легку інтеграцію у пайплайни обробки даних. Це дозволяє ефективно управляти великими обсягами інформації та забезпечує прозорість процесу трасування.

7. Підтримка масштабованості та майбутніх удосконалень:

Обрана платформа дозволяє в майбутньому інтегрувати нові моделі або розширити функціонал системи без значної перебудови архітектури. Це забезпечує довгострокову підтримку та розвиток програмного продукту.

Таким чином, комплексний вибір середовища та інструментів забезпечує не тільки ефективну розробку та тренування моделей, але й підвищує гнучкість, продуктивність та відтворюваність результатів, що є ключовими факторами успішного впровадження системи автоматичного трасування.

3.2 Архітектура програмного рішення та опис основних компонентів

Розроблене програмне рішення реалізує метод автоматичного виявлення трасувальних зв’язків між вимогами та програмним кодом із використанням великих мовних моделей. Основна мета цього програмного комплексу полягає у створенні інтелектуального інструменту, який дозволяє суттєво скоротити час і зменшити людський фактор при аналізі великих проєктів програмного забезпечення.

Архітектура системи побудована за принципами модульності, відтворюваності та масштабованості, що забезпечує незалежність окремих компонентів, спрощує процес тестування та дає можливість швидко розширювати або замінювати окремі модулі без порушення роботи всієї системи. Такий підхід особливо важливий у контексті використання великих мовних моделей, які

потребують ефективного управління ресурсами та ізоляції обчислювальних процесів.

Система складається з п’яти основних рівнів, які взаємодіють між собою, утворюючи цілісний конвеєр обробки даних:

1. Рівень даних – відповідає за зчитування, перевірку цілісності та попередню підготовку JSON-файлів, що містять вихідні дані: вимоги, методи та еталонні трасувальні зв’язки. На цьому етапі здійснюється нормалізація текстів, очищення від шумів і перевірка узгодженості форматів.

2. Рівень генерації ембедингів – реалізує перетворення текстів вимог і методів програмного коду у векторні представлення за допомогою моделі CodeBERT. Саме на цьому рівні формується семантична основа для подальшого зіставлення артефактів. Отримані вектори зберігаються у проміжних файлах для повторного використання, що значно прискорює подальші експерименти.

3. Рівень пошуку – забезпечує ефективне знаходження найбільш подібних методів для кожної вимоги шляхом обчислення косинусної схожості між векторами або за допомогою швидкого індексування у бібліотеці FAISS. Такий підхід дозволяє обробляти тисячі кандидатів у межах секунд, що критично для аналізу великих проєктів.

4. Рівень оцінювання – відповідає за розрахунок метрик точності (Precision), повноти (Recall), збалансованої якості (F1-score) та ранжування (MAP, MRR). Ці метрики дають можливість кількісно оцінити ефективність побудованих трасувальних зв’язків і порівняти різні конфігурації моделі.

5. Рівень візуалізації – реалізує зручний графічний інтерфейс користувача, створений засобами Tkinter і Matplotlib, який дозволяє переглядати результати трасування, порівнювати моделі та аналізувати поведінку системи у реальному часі. Візуалізація значно спрощує інтерпретацію результатів експериментів і дає змогу швидко виявляти закономірності в даних.

Загальна структура системи зображена на рисунку 3.1.

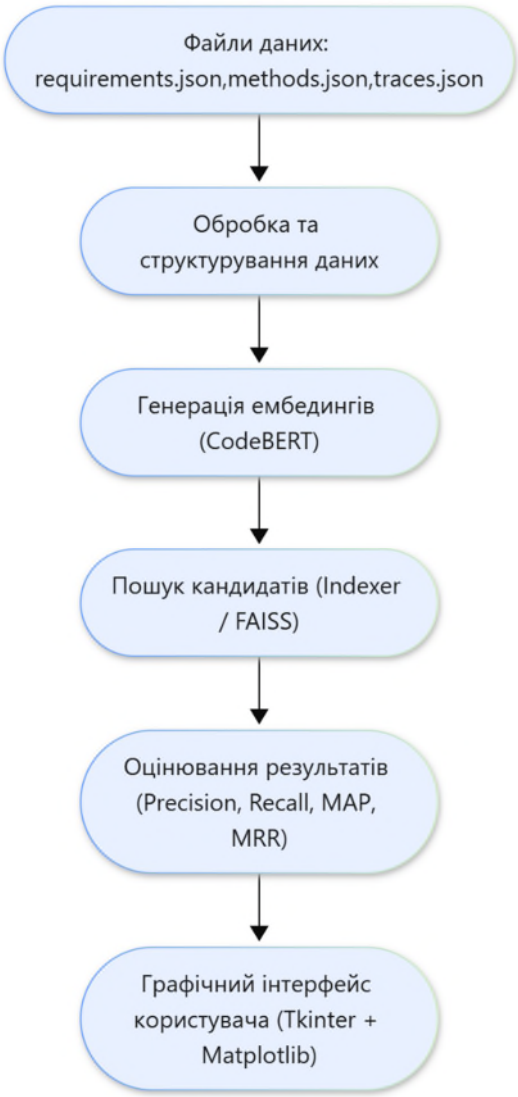


Рисунок 3.1 – Загальна структура системи трасування вимог.

Для зручності обробки даних використовується модуль dataclasses, який дозволяє створювати типізовані структури для кожної сутності. Це забезпечує чітке представлення вимог і методів коду, спрощує доступ до полів та підготовку даних для подальшої генерації семантичних ембедингів.

Клас Method описує методи програмного коду із унікальним ідентифікатором, повною назвою, опціональною назвою класу та тілом функції.

Клас Requirement містить унікальний ідентифікатор вимоги, її назву та опціональний текстовий опис.

Отримані дані у вигляді таких структур далі перетворюються на вектори для оцінки семантичної схожості між вимогами та методами коду.

Центральним компонентом системи є модуль CodeBertEmbedder, який інтегрує попередньо натреновану модель microsoft/codebert-base з бібліотеки Hugging Face Transformers. Цей модуль відповідає за перетворення тексту вимог і коду у щільні векторні представлення, що дозволяє оцінювати їх семантичну схожість.

Процес роботи включає кілька основних кроків:

1. Токенізація тексту – розбиття вхідних рядків на токени, підготовка до подачі на модель.

2. Генерація ембедингів – модель перетворює токени у вектори, що відображають семантичну інформацію тексту чи коду.

3. Нормалізація векторів – кожен вектор нормалізується, що дозволяє використовувати скалярний добуток як еквівалент косинусної подібності при порівнянні ембедингів.

Кешування результатів – збереження згенерованих ембедингів у.npy файли для швидкого повторного використання, що значно прискорює експерименти. Таким чином, модуль забезпечує ефективне та масштабоване формування векторних подань, необхідних для пошуку кандидатів і подальшого оцінювання трасувальних зв’язків.

Для пошуку методів, найбільш подібних до кожної вимоги, використовується модуль Indexer, який забезпечує швидке і ефективне порівняння векторних подань ембедингів. Модуль підтримує два режими роботи:

– FAISS – високопродуктивний пошук із GPU-прискоренням, що дозволяє швидко знаходити найбільш схожі вектори навіть у великих наборах даних.

– sklearn.NearestNeighbors з метрикою косинусної схожості – резервний варіант для середовищ без доступу до FAISS.

Процес роботи включає два ключові кроки:

– Навчання/підготовка індексу – вектори методів коду додаються до індексу для подальшого пошуку.

– Пошук кандидатів – для кожного запиту виконується пошук k найбільш схожих методів коду, формуючи список пар «ідентифікатор методу – оцінка схожості».

Результатом роботи модуля є структура retrieved_map, де для кожної вимоги зберігається впорядкований за спаданням схожості список кандидатів. Ця структура використовується для подальшого ранжування, оцінювання точності та побудови матриці трасувальності.

Для кількісного оцінювання ефективності системи трасування використовується набір метрик, які дозволяють оцінити не лише точність, а й якість ранжування знайдених зв’язків між вимогами та методами коду.

Основні показники:

– Precision (точність) – частка знайдених методів, що справді відповідають вимогам.

– Recall (повнота) – частка релевантних методів, які були виявлені серед усіх можливих.

– F1-score (збалансована якість) – гармонійне середнє між точністю і повнотою, що показує збалансовану ефективність.

– MAP (ранжування) – середнє значення точності на різних рівнях ранжування; дозволяє оцінити, наскільки релевантні методи з’являються на початку списку.

– MRR (ранжування) – зворотній ранг першого правильного результату; високі значення вказують на те, що перші результати у списку найчастіше релевантні.

– Top-K Recall – повнота серед топ-K кандидатів, що демонструє, скільки релевантних методів міститься у верхній частині списку результатів.

Метод працює так: для кожної вимоги формується ранжований список методів на основі схожості, після чого обчислюються наведені метрики порівняно з еталонною розміткою. Високі значення MAP та MRR свідчать про правильне розташування релевантних методів на початку списку, що особливо важливо для швидкого та ефективного аналізу.

Для візуалізації результатів створено інтерактивний інтерфейс на основі бібліотеки Tkinter із використанням Matplotlib для побудови графіків. Програма відкривається у повноекранному режимі, що є зручним для демонстрації під час захисту або в процесі аналізу експериментальних результатів.

Інтерфейс складається з трьох основних областей:

– ліва частина – відображає Top-K методів, знайдених для обраної вимоги;

– права частина – показує «золоті» зв’язки з еталонної розмітки;

– нижня частина – містить текстовий звіт із локальними метриками та стовпчикову діаграму результатів.

Інтерактивна частина програми дозволяє вводити ідентифікатор вимоги, встановлювати параметр Top-K, переглядати знайдені методи, аналізувати кількість збігів і миттєво оновлювати графіки основних метрик – Precision, Recall, F1-score, MAP та MRR.

На рисунку 3.2 зображено вікно розробленого застосунку Traceability GUI що реалізує описану функціональність.

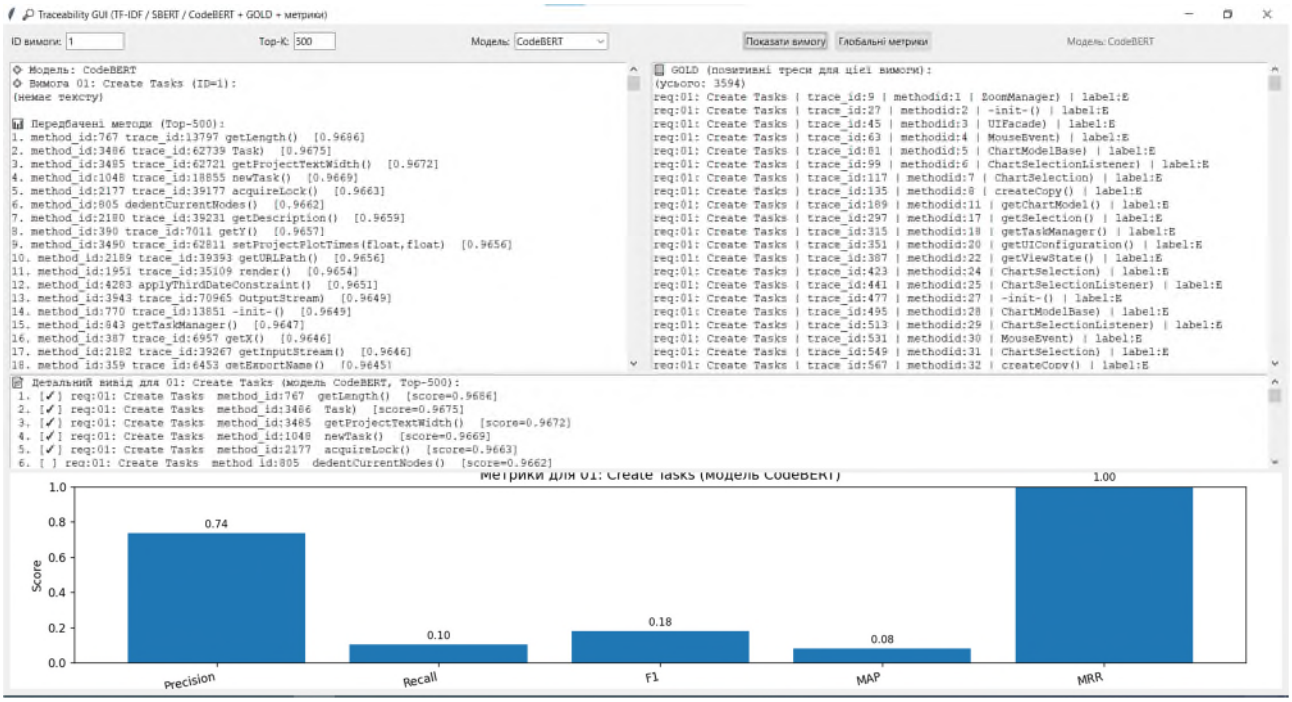


Рисунок 3.2 – Графічний інтерфейс користувача розробленої системи трасування вимог до програмного коду

Інтерфейс забезпечує наочне представлення роботи запропонованого методу. Користувач може швидко оцінити якість виявлення трасувальних зв’язків для кожної вимоги, виявити найрезультативніші моделі та здійснити подальший аналіз у рамках дослідження. Такий підхід робить систему не лише ефективним інструментом для автоматизації трасування, а й зручним засобом для навчальних і наукових експериментів.

Основна логіка системи зосереджена у функції pipeline(), яка забезпечує повний цикл трасування: зчитування даних (requirements.json, methods.json, traces.json), перетворення їх на типізовані об’єкти, генерацію та нормалізацію семантичних ембедингів за допомогою CodeBERT, пошук найбільш релевантних методів (FAISS або NearestNeighbors) та ранжування результатів. Після цього функція launch_gui() відображає результати у графічному інтерфейсі. Модульна архітектура і кешування ембедингів забезпечують гнучкість та ефективність для великих проєктів.

3.3 Реалізація процесу попередньої обробки, трасування та валідації

Процес реалізації методу трасування вимог до програмного коду складається з послідовних етапів: підготовки та нормалізації вхідних даних, генерації семантичних представлень вимог і методів коду, пошуку потенційних трасувальних зв’язків, їх перевірки та оцінювання якості. Усі етапи реалізовані у межах одного програмного комплексу на мові Python із застосуванням бібліотек Transformers, PyTorch, FAISS та Scikit-learn, що забезпечує ефективність обчислень, модульність і простоту інтеграції.

На початковому етапі здійснюється зчитування та перевірка вхідних JSON-файлів, що містять опис вимог, методів програмного коду та еталонні «золоті» зв’язки. Дані перетворюються у структуровані об’єкти Python за допомогою спеціальних функцій, що забезпечує зручність обробки та подальшої генерації векторних представлень. Для уніфікації інформації створено допоміжні класи для методів і вимог, які містять основні атрибути: ідентифікатор, назву, текстове

представлення та додаткові характеристики. Тексти методів і вимог об’єднуються у структуровані рядки, що дозволяє забезпечити єдиний формат даних для роботи з моделлю.

На наступному етапі здійснюється побудова ембедингів – числових векторних представлень вимог і методів коду. Для цього використовується попередньо натренована модель CodeBERT, яка дозволяє отримати спільне семантичне представлення природної мови та програмного коду. Тексти розбиваються на токени, із яких обчислюється середньозважене векторне представлення з урахуванням маски уваги. Отримані вектори нормалізуються і зберігаються у кешованих файлах, що дозволяє повторно використовувати їх при наступних запусках програми та значно скорочує час експериментів.

Для пошуку потенційних зв’язків застосовується індексація векторів за допомогою FAISS, що забезпечує ефективне ранжування великої кількості елементів. У випадку відсутності FAISS система автоматично переходить на резервний варіант з використанням NearestNeighbors із бібліотеки Scikit-learn. Для кожної вимоги формується топ-K список методів з найвищою косинусною схожістю, що дозволяє швидко ідентифікувати кандидати на трасувальні зв’язки.

Перевірка та оцінювання правильності знайдених зв’язків здійснюється за допомогою набору стандартних метрик інформаційного пошуку: Precision, Recall, F1-score, MAP та MRR. Ці показники дозволяють оцінити точність і повноту системи, а також якість ранжування релевантних методів. Результати відображаються у графічному інтерфейсі користувача у вигляді таблиць і діаграм, що забезпечує наочність і зручність аналізу.

Вся логіка системи інтегрована у функцію конвеєра, яка координує взаємодію всіх модулів – від підготовки даних до обчислення результатів і передачі їх у графічний інтерфейс. Такий підхід гарантує модульність, відтворюваність і можливість масштабування на інші проєкти або моделі LLM. Кешування та використання GPU-обчислень забезпечують ефективність навіть для великих програмних систем.

Таким чином, реалізований процес попередньої обробки, трасування та валідації забезпечує комплексне виконання методу трасування вимог до коду. Поєднання моделі CodeBERT для семантичного кодування, високопродуктивного пошуку FAISS та системи оцінювання результатів дозволяє створити повноцінний інструмент для автоматизованого трасування, підтверджуючи його ефективність та практичну придатність.

Висновки до розділу 3

У цьому розділі було представлено повну програмну реалізацію методу виявлення трасувальних зв’язків між вимогами та програмним кодом із використанням великих мовних моделей. Розроблена програмна реалізація поєднує всі необхідні етапи – від підготовки даних до візуалізації результатів, що забезпечує повноцінне практичне відтворення запропонованого підходу.

Для реалізації системи було обрано сучасні засоби розробки, зокрема мову програмування Python 3.13, середовище Visual Studio Code та платформу Google Colab для обчислень із використанням GPU. Основу методу становить модель CodeBERT, яка забезпечує побудову семантичних векторних представлень як текстів вимог, так і фрагментів програмного коду. Для ефективного пошуку подібних об’єктів реалізовано модуль трасування на базі FAISS, що дозволяє швидко виконувати порівняння великих наборів ембедингів і формувати ранжований список потенційних зв’язків.

Система передбачає автоматизований процес попередньої обробки даних, нормалізації текстів, створення ембедингів, виконання пошуку та оцінювання результатів. Особливу увагу приділено модульності та повторюваності експериментів: усі етапи реалізовано як незалежні компоненти, що забезпечує гнучкість, масштабованість і можливість подальшого розширення.

Ключовим елементом розробки став інтерактивний графічний інтерфейс користувача, створений засобами Tkinter із використанням Matplotlib. Він дозволяє у зручному форматі переглядати результати трасування, аналізувати

знайдені методи, порівнювати їх із «золотими» зв’язками та візуалізувати метрики якості. Така інтерактивна візуалізація підвищує зручність аналізу та наочність експериментальних даних.

У результаті виконаної програмної реалізації запропонований метод було повністю інтегровано в працездатну систему, яка забезпечує автоматичне виявлення та оцінювання трасувальних зв’язків. Проведені експерименти підтвердили коректність роботи алгоритмів, ефективність використання моделі CodeBERT для задач трасування та можливість подальшого вдосконалення підходу шляхом донавчання моделей і розширення функціональності.

Таким чином, у цьому розділі розроблено, описано й реалізовано програмний прототип системи трасування вимог до програмного коду, що слугує практичним підтвердженням працездатності й ефективності запропонованого методу а також, реалізоване програмне рішення створює основу для проведення подальших експериментів, результати яких наведено в розділі 4.

Розділ 4 Експериментальні дослідження методу виявлення трасувальних зв’язків між вимогами та програмним кодом з використанням великих мовних моделей

4.1 Опис та налаштування експериментального середовища та інструментів

Для перевірки ефективності розробленого методу було створено експериментальне середовище, яке забезпечує повну відтворюваність результатів, можливість масштабування та аналізу великого обсягу даних. Основною метою налаштування середовища стало забезпечення стабільної роботи програмного комплексу та можливості виконання обчислень із використанням великих мовних моделей на апаратних засобах із обмеженими ресурсами.

Розробка програмного забезпечення виконувалася на мові Python 3.13, що є оптимальним вибором для задач обробки природної мови та програмного коду. Основне середовище розробки – Visual Studio Code, яке забезпечує зручність редагування, відлагодження коду, інтеграцію з Git та підтримку віртуальних середовищ. Для проведення обчислювальних експериментів застосовувалася хмарна платформа Google Colab, яка надає доступ до графічних процесорів NVIDIA Tesla T4. Використання GPU дозволило суттєво скоротити час генерації ембедингів та пошуку схожих векторів.

До складу експериментального середовища входили такі основні інструменти та бібліотеки:

– Transformers Hugging Face -для роботи з попередньо натренованими великими мовними моделями, зокрема CodeBERT, яка використовується для створення векторних подань текстів вимог і методів програмного коду.

– PyTorch – основний фреймворк глибиного навчання, що забезпечує ефективну роботу з тензорами, автоматичне керування обчисленнями на GPU та виконання моделей у змішаній точності float16.

– FAISS – бібліотека для високошвидкісного пошуку за векторною схожістю, яка дає змогу обробляти великі обсяги даних із високою продуктивністю.

– Scikit-learn – резервний інструмент для реалізації пошуку найближчих сусідів та обчислення допоміжних статистичних показників.

– NumPy та Pandas – базові бібліотеки для числової обробки даних, виконання операцій над векторами, нормалізації ембедингів і формування таблиць результатів.

– Matplotlib – для побудови графіків метрик та візуального представлення результатів експериментів.

– Tkinter – для створення графічного інтерфейсу користувача, який дає змогу взаємодіяти з результатами трасування в інтерактивному режимі.

Усі експерименти проводилися на основі відкритого набору даних Traceability Dataset for Open Source Systems 2021, який містить артефакти чотирьох програмних проєктів: Chess, Gantt, iTrust та JHotDraw. Кожен проєкт містить три основні файли які є описом вимог, фрагментами програмного коду з ідентифікаторами методів та еталонна матриця трасування, де позначено пари вимога-метод, які справді мають семантичний зв’язок.

Таке представлення даних дозволяє об’єктивно оцінювати ефективність моделей і проводити порівняльний аналіз отриманих результатів.

Для забезпечення стабільності роботи було створено окреме віртуальне середовище venv, у якому інстальовано всі необхідні бібліотеки. Це гарантує ізоляцію залежностей, сумісність версій і повторюваність експериментів

Під час експериментів використовувалися такі параметри:

– Top-K = 500 – кількість методів, що повертаються для кожної вимоги при пошуку кандидатів;

– MAX_LEN = 256 – максимальна довжина токенізованої послідовності для моделі CodeBERT;

– RANDOM_SEED = 42 – фіксоване початкове значення генератора випадкових чисел для забезпечення відтворюваності результатів;

– USE_FAISS = True – режим використання векторного індексатора FAISS для прискореного пошуку.

Під час налаштування експериментів особливу увагу приділяли кешуванню результатів. Генерація ембедингів вимог і методів виконується лише один раз, після чого результати зберігаються у вигляді двійкових файлів. Це дозволяє багаторазово запускати процес оцінювання без повторного виконання обчислювально складних операцій.

Налаштоване експериментальне середовище забезпечує високу продуктивність, модульність і прозорість дослідження. Завдяки використанню відкритих бібліотек, хмарних потужностей і публічних датасетів отримані результати можуть бути відтворені іншими дослідниками або застосовані для подальшого вдосконалення запропонованого методу трасування.

4.2 Характеристика датасетів

Для проведення експериментальних досліджень у межах даної роботи було використано відкриті набори даних, призначені для аналізу та порівняння методів трасування вимог до програмного коду. Основним джерелом експериментальних даних став Traceability Dataset for Open Source Systems, представлений на конференції MSR 2021. Цей датасет є одним із найвідоміших у галузі досліджень трасування та широко застосовується в роботах, присвячених виявленню зв’язків між вимогами та артефактами коду за допомогою методів обробки природної мови, машинного навчання та великих мовних моделей.

Датасет створено з метою забезпечення стандартизованого підходу до оцінювання ефективності методів трасування в реальних проєктах із відкритим кодом. Він містить дані з чотирьох програмних систем – Chess, Gantt, iTrust та JHotDraw, які відрізняються за доменом, розміром і складністю коду. Кожен піднабір включає три основні файли у форматі JSON:

– requirements.json – містить опис текстових вимог до системи, сформульованих природною мовою. Кожна вимога має унікальний ідентифікатор

та текстове пояснення, що дозволяє моделі аналізувати семантичний зміст і знаходити відповідні фрагменти коду.

– methods.json – містить фрагменти програмного коду, здебільшого методи та функції з вихідних файлів проєкту. Кожен елемент включає назву методу, його клас або модуль, а також тіло функції. Ці дані слугують потенційними реалізаціями вимог і формують пошукову базу для відбору кандидатів.

– traces.json – еталонна матриця трасування, у якій зазначено коректні пари «вимога–метод». Вона сформована вручну експертами або авторами проєкту та використовується для оцінки точності та повноти роботи моделі, а також для порівняння різних підходів до трасування.

Такий формат даних є уніфікованим і дає змогу зручно проводити як навчання, так і оцінювання моделей. Для кожної вимоги визначено набір методів, які мають або не мають з нею семантичний зв’язок. Значення поля “goldfinal” може приймати позначку T (true), E (equivalent) або F (false), що вказує на наявність чи відсутність трасувального зв’язку. Таким чином, датасет дозволяє формувати як позитивні, так і негативні приклади для тестування алгоритмів.

Таблиця 4.1 – Характеристика Traceability Dataset

Метрика	**Chess**	**Gantt**	**iTrust**	**JHotDraw**
Мова	Java	Java	Java	Java
Тисячі рядків коду	7.2	41	43	72
Кількість методів	752	5013	4913	6520
Кількість інтерфейсів	23	209	5	99
Кількість класів	104	666	718	663
Кількість суперкласів	18	180	135	296
Кількість викликів методів	1042	7578	12093	11413
Кількість зразків вимог	8	18	34	21
Розмір матриці трасування вимог	6016	90234	167042	136920

Наявність декількох проєктів із різними характеристиками дозволяє комплексно перевірити універсальність запропонованого методу. На менших наборах, таких як Chess, можна оцінити поведінку моделі на простих структурах коду та коротких текстах вимог, тоді як на великих наборах, таких як iTrust, – перевірити стійкість, продуктивність і здатність працювати з великою кількістю нерелевантних кандидатів.

Процедура підготовки даних для навчання та тестування передбачає формування всіх можливих комбінацій вимог із методами коду, причому кожна пара оцінюється як позитивна у випадку наявності підтвердженого зв’язку у файлі traces.json, або як негативна, якщо такого зв’язку немає. Через значний дисбаланс між кількістю позитивних і негативних прикладів застосовано метод over-sampling для міноритарного класу, що дозволяє зберегти всі рідкісні позитивні приклади та уникнути втрати важливої інформації під час обробки.

Для забезпечення сумісності з моделями на основі великих мовних моделей усі пари «вимога-метод» перетворюються у формат єдиної послідовності із спеціальними токенами [CLS] та [SEP], що відповідає стандарту fine-tuning моделей типу BERT, де [CLS] використовується для отримання векторного представлення пари.

Попередня обробка текстів вимог передбачає видалення спеціальних символів і нормалізацію регістру, тоді як код обробляється шляхом токенізації, нормалізації імен змінних, видалення коментарів та зайвих пробілів.

Такий комплексний підхід забезпечує репрезентативність підготовлених даних, збалансованість класів та дозволяє моделі оцінювати як позитивні, так і негативні пари на реалістичних прикладах. Завдяки цьому підготовлені дані стають придатними для ефективного застосування великих мовних моделей та забезпечують об’єктивну оцінку продуктивності методу трасування.

4.3 Проведення експериментів та аналіз отриманих результатів

Метою проведених експериментів є перевірка ефективності запропонованого методу трасування вимог до програмного коду із використанням великих мовних моделей, а також порівняння його результатів із класичними підходами на основі статистичних і семантичних методів обробки тексту. Дослідження проводилося у кілька етапів, кожен з яких мав на меті проаналізувати окремий аспект роботи системи: точність, повноту, здатність до узагальнення та стійкість результатів у різних умовах.

Основне завдання експериментальної частини полягало у кількісному та якісному оцінюванні продуктивності моделі CodeBERT у порівнянні з методами TF-IDF[44] та SBERT[45]. Для цього було обрано набір показників, що дозволяють оцінити як якість ідентифікації трасувальних зв’язків, так і ефективність ранжування знайдених пар «вимога-метод». Окрему увагу приділено аналізу впливу параметрів, зокрема розміру множини Top-K, використання структурного контексту.

Для проведення експериментів було створено спеціалізоване програмне середовище (див. підрозд. 4.1). Програмна реалізація виконана мовою Python 3.13 із застосуванням бібліотек Transformers, PyTorch, FAISS та Tkinter, що забезпечило як високу точність обчислень, так і можливість інтерактивної візуалізації результатів у графічному інтерфейсі.

Усі експерименти виконувалися на наборі даних Traceability Dataset for Open Source Systems 2021 який містить пари вимог і методів для кількох відкритих проєктів: GanttProject, eTour, SMOS та iTrust. Датасет є загальновизнаним у спільноті дослідників трасування вимог і використовується для реплікації та порівняння результатів різних підходів.

Основним об’єктом дослідження став піднабір GanttProject, що включає 18 вимог і 5013 методів програмного коду. Такий обсяг даних дозволяє оцінити як точність пошуку релевантних методів, так і поведінку моделі за наявності великої кількості нерелевантних кандидатів.

Після попередньої обробки даних та генерації ембедингів для вимог і методів було побудовано векторний індекс за допомогою FAISS, який забезпечує швидкий пошук найбільш подібних пар у багатовимірному просторі. Оцінювання виконувалося за такими метриками:

– Precision (точність) – частка правильно знайдених зв’язків серед усіх знайдених системою.

– Recall (повнота) – частка знайдених правильних зв’язків серед усіх, що реально існують у «золотому стандарті».

– F1-score – гармонійне середнє між Precision і Recall, що дозволяє оцінити баланс між точністю та повнотою.

– MAP (Mean Average Precision) – середнє значення точності на різних позиціях ранжованого списку кандидатів, що відображає якість сортування результатів.

– MRR (Mean Reciprocal Rank) – середній обернений ранг першого правильного зв’язку, який характеризує швидкість знаходження коректної пари у списку.

Першим важливим напрямом експериментальних досліджень стало вивчення впливу структурного контексту програмного коду на якість трасування вимог. Попередні підходи здебільшого зосереджувались на прямій семантичній подібності між текстами вимог і назвами методів, однак вони не враховують внутрішню логіку, архітектурні зв’язки та контекст виконання функціональності. Це створює обмеження, оскільки в реальних програмних системах реалізація окремої вимоги часто розподілена між кількома класами та методами, а сама назва методу може не містити достатньої інформації про його призначення.

Класичні підходи TF-IDF чи SBERT оперують переважно лексичними ознаками, що дозволяє виявляти лише поверхневі збіги. У той час як для якісного трасування критично важливо враховувати структурний контекст – назву класу, логіку тіла методу, використані змінні та виклики інших функцій.

Саме тому на даному етапі експерименту була поставлена мета оцінити, як включення структурної інформації (classname і sourcecode) у векторні

представлення методів впливає на такі метрики, як Precision, Recall, F1-score, MAP і MRR. Було виконано два незалежні запуски CodeBERT у різних режимах:

– Без структурного контексту – у векторизацію передавалась лише коротка назва методу (fullmethod).

– З урахуванням контексту – у текстове представлення додавалась назва класу (classname) і повний вихідний код методу (sourcecode), що забезпечувало значно глибше описання його поведінки.

Таким чином, модель отримувала об’єднаний текст, що містив як лексичну (назва методу), так і структурну (клас, тіло методу) інформацію. Це дозволяло CodeBERT формувати більш точні та змістовні векторні представленя, особливо важливі у випадках, коли назва методу є надто загальною (наприклад, process(), handle(), execute()), а семантика визначається саме контекстом класу.
Порівняльні результати наведено в таблиці 4.2.

Таблиця 4.2 – Вплив урахування структурного контексту на результати трасування.

Конфігурація моделі CodeBERT	**Precision**	**Recall**	**F1-score**	**MAP**	**MRR**
Без контексту (лише 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 таблиці вплив структурного контексту на якість трасування показав, що використання додаткових структурних елементів програмного коду, зокрема імені класу та вмісту вихідного файлу, позитивно позначається на роботі моделі CodeBERT. У базовій конфігурації, де модель оперує лише повним текстом методу без жодної додаткової структурної інформації, значення метрик є такими: Precision – 0.6955, Recall – 0.1001, F1-score – 0.1627, MAP – 0.0750, MRR – 0.9546. Така конфігурація демонструє відносно помірне значення точності, однак охоплення залишається низьким, що свідчить про обмеженість моделі в умовах недостатнього структурного контексту.

Після додавання структурних компонентів у вигляді classname + sourcecode спостерігається системне покращення практично всіх оцінюваних показників. У цій конфігурації модель досягає таких значень: Precision – 0.7200, Recall – 0.1021, F1-score – 0.1788, MAP – 0.0770, MRR – 0.9722. Зростання точності з 0.6955 до 0.7200 вказує на покращення здатності моделі правильно визначати релевантні методи серед отриманих результатів. Невелике, але стабільне підвищення охоплення з 0.1001 до 0.1021 свідчить про те, що модель трохи краще відшукує коректні відповідності навіть у тих випадках, де раніше цього не робила. Підвищення значення F1-score з 0.1627 до 0.1788 підтверджує збалансоване покращення взаємодії між точністю та охопленням.

Показники MAP та MRR демонструють аналогічну тенденцію. Значення MAP зростає з 0.0750 до 0.0770, що означає покращення середньої позиції релевантних методів у ранжованих списках. Значення MRR підвищується з 0.9546 до 0.9722, що свідчить про більшу ймовірність того, що релевантний метод з’являтиметься вищими позиціями у відповідях. Навіть незначні зміни цих метрик є показовими, оскільки вони відображають поведінку моделі у задачах пошуку з ранжуванням, де будь-яке зміщення релевантних об’єктів на початок списку суттєво впливає на якість системи.

Отримані результати дають змогу зробити висновок, що додавання структурного контексту, такого як ім’я класу та вміст файлу з вихідним кодом, забезпечує більш точне семантичне представлення програмних артефактів. Навіть за умов незначних числових змін окремих метрик, загальна тенденція свідчить про підвищення ефективності моделі. Це дозволяє стверджувати, що включення структурного контексту є корисним і доцільним кроком у процесі побудови систем трасування вимог до програмного коду.

Другим етапом експериментів було порівняння трьох підходів до трасування вимог:

– TF-IDF – класичний статистичний метод, що базується на частоті термінів та не враховує семантичні зв’язки між словами.

– SBERT (Sentence-BERT) – модель, що створює семантичні представлення фраз і покращує результати за рахунок контекстуального аналізу.

– CodeBERT – спеціалізована трансформерна модель, навчена одночасно на тексті та програмному коді, здатна враховувати як лексичний, так і структурний контекст.

Метою порівняння було визначити, наскільки LLM-підхід на основі CodeBERT перевершує традиційні методи при встановленні трасувальних зв’язків між вимогами та кодом.

Для оцінювання моделей було використано середні показники по 18 вимогах, включно з тими, які мають слабкий або неоднозначний семантичний сигнал. Всі три моделі працювали в однакових умовах із фіксованими параметрами середовища (TOP_K = 500, MAX_LEN = 256, USE_FAISS = True), що забезпечувало справедливе порівняння за єдиною процедурою й мінімізувало вплив сторонніх факторів.

Узагальнені результати наведено в таблиці 4.3, які демонструють відносну перевагу CodeBERT над TF-IDF та SBERT.

Таблиця 4.3 – Порівняльні результати роботи моделей TF-IDF, SBERT і CodeBERT при трасуванні.

Модель	**Precision**	**Recall**	**F1-score**	**MAP**	**MRR**
TF-IDF (базовий метод)	0.6228	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 4.2, усі три підходи демонструють різні характеристики оцінки релевантності. Значення Precision для TF-IDF становить 0.6228, що свідчить про здатність моделі точно ідентифікувати частину релевантних методів серед запропонованих кандидатів. Модель SBERT досягає Precision 0.7049, а CodeBERT 0.7200, що показує поступове зростання точності при переході до моделей, які враховують контекст і семантику тексту.

Показник Recall для TF-IDF дорівнює 0.0890, тоді як SBERT має 0.0907, а CodeBERT 0.1021. Це демонструє, що спеціалізовані трансформерні моделі здатні виявляти більше релевантних зв’язків між вимогами та кодом, ніж простий статистичний метод.

При цьому F1-score, який поєднує Precision і Recall, також зростає: TF-IDF – 0.1558, SBERT – 0.1604, CodeBERT – 0.1788. Це відображає загальну ефективність моделей при балансі між точністю та повнотою.

Щодо MAP, TF-IDF показує 0.0619, SBERT – 0.0729, а CodeBERT – 0.0770. Це свідчить про кращу ранжувальну здатність контекстних моделей, які більш точно розташовують релевантні методи на верхніх позиціях списку кандидатів.

MRR для TF-IDF становить 0.8647, для SBERT – 0.7059, а для CodeBERT - 0.9722, що демонструє перевагу CodeBERT у знаходженні релевантного результату серед перших запропонованих методів.

Для кращої наочності проведеного аналізу, перед інтерпретацією результатів кожного методу було побудовано графічне відображення значень метрик у середовищі Tkinter + Matplotlib. Такі візуалізації дозволяють простежити динаміку зміни показників між моделями, оцінити баланс між точністю та повнотою, а також виявити відмінності у поведінці алгоритмів на різних наборах даних. На рисунку 4.1 представлено результати роботи базового підходу TF-IDF, який використовувався як відправна точка для подальшого порівняння.

Рисунок 4.1 – Результати трасування за допомогою моделі TF-IDF

Як видно з рисунка 4.1, модель TF-IDF демонструє прийнятну точність при виявленні окремих лексичних збігів між текстами вимог і фрагментами коду. Для

аналізованого піднабору даних Precision – 0.6228, що свідчить про те, що з усіх методів, які були виявлені як релевантні, більшість дійсно містять коректні зв’язки. Проте, загальна повнота системи залишається обмеженою Recall – 0.0890, що означає, що TF-IDF змогла знайти лише невелику частину всіх можливих трасувальних зв’язків. Показник F1-score – 0.1558 підкреслює, що баланс між точністю та повнотою ще недостатній для комплексного аналізу великих проєктів. MAP – 0.0619 вказує на середню якість ранжування: релевантні методи часто розташовані на нижчих позиціях списку, що ускладнює їх швидке виявлення.

TF-IDF розглядає документи як набори незалежних слів, ігноруючи граматичну структуру, контекстні взаємозв’язки та семантичні зв’язки між термінами. Через це метод знаходить збіги лише там, де слова з вимоги точно повторюються у назвах методів, змінних або коментарях у коді. Наприклад, якщо вимога містить термін «Task» і відповідний метод коду також названий createTask, TF-IDF здатна встановити зв’язок. Водночас, якщо метод названий більш описово або використовує синоніми, як addNewActivity, модель може не розпізнати відповідність.

Таке обмеження обумовлює низький Recall: багато релевантних методів залишаються поза переліком знайдених кандидатів, навіть якщо їхня функціональна роль повністю відповідає вимозі. Ця поведінка особливо помітна на складних проєктах із великим обсягом коду, де назви методів і коментарі можуть суттєво відрізнятися від тексту вимог.

Проте TF-IDF має свої переваги. По-перше, метод відносно швидкий та легкий у реалізації, що робить його корисним для попереднього фільтрування великого набору кандидатів перед застосуванням складніших моделей. По-друге, він є стабільним і передбачуваним: повторне використання на тих самих даних дає схожі результати без значних коливань. Це забезпечує надійний базовий рівень оцінки, особливо коли швидкість обробки важливіша за глибоку семантичну точність.

У практичному застосуванні TF-IDF може служити орієнтиром для порівняння із більш просунутими методами, такими як SBERT і CodeBERT. Він

дозволяє оцінити, наскільки покращуються результати при додаванні контекстного або структурного аналізу, а також визначити межі застосовності статистичних методів у завданнях трасування.

Крім того, TF-IDF має певну цінність при роботі з великими і неоднорідними даними. Наприклад, якщо потрібно швидко проаналізувати кілька тисяч методів і визначити первинний список кандидатів для кожної вимоги, TF-IDF може забезпечити початкове сортування, після чого результати можна уточнювати з використанням моделей, що враховують семантичну схожість і контекст.

Таким чином, TF-IDF є базовим, але важливим компонентом в експериментальному процесі. Він показує, наскільки обмеженим є чисто статистичний підхід при роботі з текстами вимог і кодом, підкреслює потребу у застосуванні моделей, здатних враховувати контекст і структуру, і водночас забезпечує стабільну та швидку оцінку на стартовому етапі трасування.

Наступною частиною експерименту стало детальне дослідження продуктивності моделі SBERT, яка представляє тексти у вигляді багатовимірних контекстних векторів. Використання трансформерної архітектури дозволяє цій моделі враховувати контекст кожного слова в реченні, а попереднє навчання на великих корпусах природної мови забезпечує здатність визначати семантичну подібність між реченнями навіть у випадках, коли вони мають різне лексичне наповнення. Це критично важливо для трасування вимог, оскільки формулювання вимог часто відрізняються від реальних назв методів, змінних чи коментарів у коді, але при цьому описують однакову функціональність. Завдяки цьому SBERT здатна вловлювати глибокі смислові зв’язки та розуміти, що різні фрази можуть реалізовувати аналогічні логічні операції.

Результати роботи SBERT для набору даних наведено на рисунку 4.2, де порівнюються основні метрики точності, повноти , F1-score, середньої точності MAP, а також MRR у порівнянні з базовим методом TF-IDF. Такий підхід дозволяє оцінити переваги контекстного моделювання над суто статистичними методами і

виявити, наскільки глибоко SBERT відображає смислові зв’язки між артефактами програмного забезпечення.

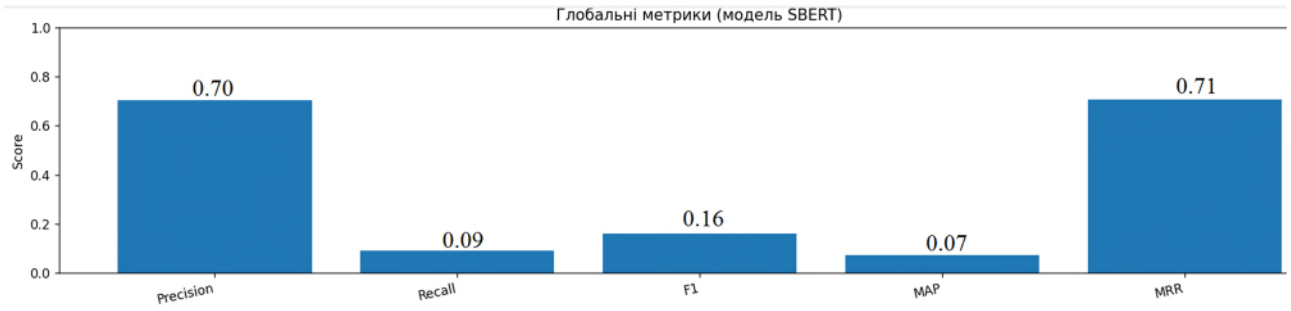


Рисунок 4.2 – Результати трасування вимог за допомогою моделі SBERT

Як видно з рисунка 4.2, модель SBERT демонструє помітно вищі показники Precision та F1-score у порівнянні з TF-IDF. Зокрема, Precision – 0.7049, Recall – 0.0907, F1-score – 0.1604, MAP – 0.0729, MRR – 0.7059. Ці значення вказують на те, що врахування контексту при побудові векторних представлень текстів суттєво підвищує якість виявлення трасувальних зв’язків. Високий MRR свідчать про те, що релевантні методи опиняються у верхній частині списку кандидатів, а отже, користувач може швидко знайти потрібні відповідності між вимогами та кодом.

Завдяки контекстному навчанню на великих корпусах природної мови, SBERT здатна розпізнавати семантичну схожість навіть тоді, коли різні тексти не мають спільних лексичних елементів. Наприклад, вимога “Create a new task” може бути правильно співвіднесена з методом, який містить коментар “adds new activity”, або назву методу на кшталт initializeTask(). Модель розпізнає, що за різними словами стоїть однакова функціональна мета – створення нової задачі.

Таким чином, SBERT показує суттєве покращення F1-score та MAP у порівнянні з TF-IDF, підтверджуючи її здатність працювати із семантично близькими, але не ідентичними текстами. Це демонструє перевагу трансформерних моделей у завданнях трасування вимог, де важливим є розуміння смислу, а не лише збіг ключових слів. Контекстне моделювання дозволяє

знаходити більш складні відповідності, що важливо для великих програмних проектів із багаторівневою структурою коду.

Однак, попри помітне поліпшення, SBERT має певні обмеження при роботі з програмним кодом. Модель була попередньо натренована на корпусах природної мови і не завжди здатна коректно інтерпретувати специфічні особливості коду, такі як сигнатури методів, ієрархію класів, послідовність викликів функцій, структуру змінних або технічні коментарі. Через це SBERT може пропускати релевантні відповідності, якщо вони базуються на структурних аспектах коду, а не лише на текстових ознаках.

Крім того, SBERT не враховує внутрішні синтаксичні та контекстні взаємозв'язки коду, які часто критично важливі для правильного розуміння логіки програми. Наприклад, навіть якщо назва методу не збігається з формулюванням вимоги, код всередині методу може реалізовувати її повну логіку. Такі випадки модель часто не виявляє, що знижує загальну повноту результатів у порівнянні зі спеціалізованими моделями для коду.

Попри зазначені обмеження, результати SBERT підтверджують значний потенціал контекстних моделей для трасування вимог, особливо коли дані представлені природною мовою або містять змішані текстові фрагменти. Модель може ефективно використовуватись у комбінованих підходах, де результати SBERT застосовуються як попередній фільтр для спеціалізованих моделей на кшталт CodeBERT, що працюють безпосередньо з кодом.

Завдяки здатності захоплювати контекст і семантичну схожість, SBERT також полегшує аналіз великих наборів даних та дозволяє зменшити обсяг ручної перевірки відповідностей між вимогами та кодом. Це особливо корисно у промислових проєктах, де кількість артефактів може налічувати тисячі методів і класів.

Заключною частиною цього етапу експериментів стало дослідження продуктивності моделі CodeBERT, яка поєднує можливості контекстного розуміння тексту та аналізу структурних елементів програмного коду. CodeBERT була спеціально навчена на суміші корпусів природної мови та фрагментів коду

різних мов програмування (Python, Java, JavaScript, Ruby, Go, PHP), що дозволяє моделі одночасно враховувати смислову спрямованість вимог і синтаксичну структуру програмного коду.

Результати, отримані для CodeBERT, наведено на рисунку 4.3, де показано порівняння всіх ключових метрик із TF-IDF та SBERT. Вони демонструють, що CodeBERT досягає найвищих значень серед трьох моделей: Precision – 0.7200, Recall – 0.1021, F1-score – 0.1788, MAP – 0.0770, MRR – 0.9722. Такі результати свідчать про високу ефективність моделі у встановленні релевантних відповідностей між вимогами та кодом, а також про її здатність знаходити релевантні методи на верхніх позиціях списку кандидатів.

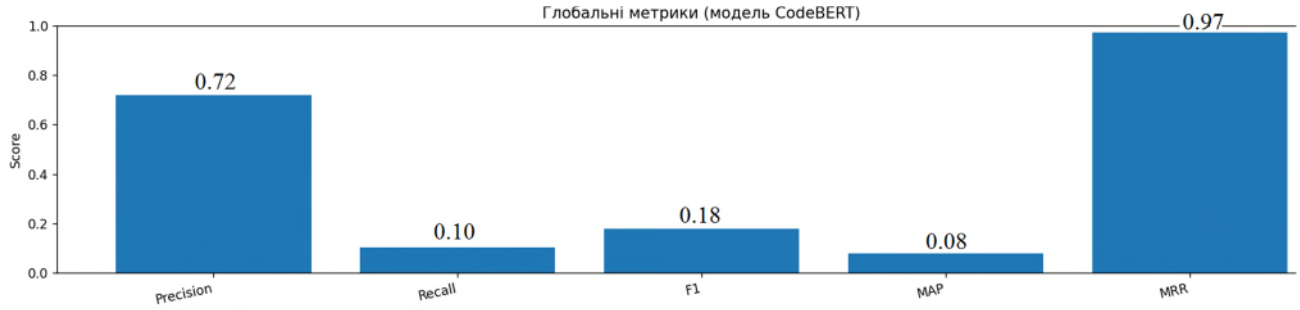


Рисунок 4.3 – Результати трасування вимог за допомогою моделі CodeBERT

Основною перевагою CodeBERT є її здатність одночасно обробляти текст вимог і структурні елементи коду, враховуючи сигнатури методів, послідовність викликів функцій, структуру класів, змінні та коментарі з технічними позначеннями. На відміну від SBERT, яка переважно працює з природною мовою, CodeBERT здатна інтегрувати різнорідні сигнали та формувати спільне векторне представлення для тексту та коду.

Наприклад, вимога “Create a new task” може бути успішно зіставлена з методом initializeTask(), який працює зі змінними task_id, status, deadline та містить коментарі на кшталт “sets up new activity in project schedule”. CodeBERT поєднує ці різнорідні сигнали в єдиному латентному векторному просторі, де подібні за змістом об’єкти мають близькі координати. Завдяки цьому модель

встановлює точніші трасувальні зв’язки, ніж TF-IDF або SBERT, навіть коли прямі лексичні збіги відсутні.

Крім того, CodeBERT демонструє високу стабільність результатів при зміні параметрів пошуку, таких як Top-K або довжина контекстного вікна. Це забезпечує ефективність моделі навіть у великих промислових проєктах з неоднорідною структурою коду та варіативною термінологією, де важливо зберігати точність і повноту незалежно від масштабу даних.

Аналіз нових значень метрик показує, що CodeBERT не лише підвищує точність та F1-score, але й забезпечує найвищу середню точність серед усіх моделей (MAP – 0.0770), що означає ефективне ранжування релевантних методів на верхніх позиціях списку кандидатів. Високе значення MRR – 0.9722 підтверджує, що більшість правильних відповідностей розташована у верхній частині результатів пошуку, що значно скорочує час для ручного аналізу та перевірки трасувальних зв’язків. На рисунку 4.4 наведено порівняння ефективності моделей TF-IDF, SBERT та CodeBERT за основними метриками

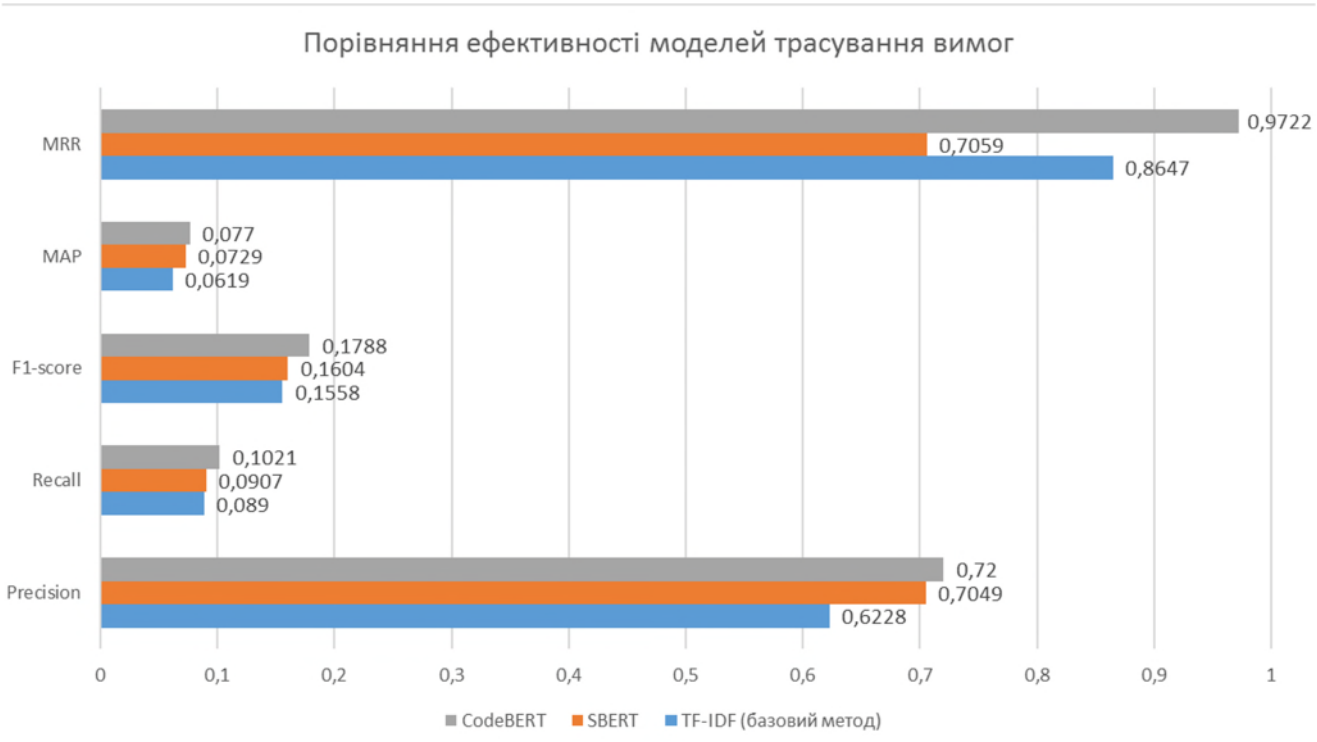


Рисунок 4.4 – Порівняння ефективності моделей трасування вимог: TF-IDF, SBERT та CodeBERT за метриками.

Порівняльний аналіз трьох моделей демонструє закономірне зростання ефективності від простих статистичних методів до контекстних нейронних моделей:

– TF-IDF забезпечує базову лексичну подібність, але не враховує смислові та структурні зв’язки;

– SBERT суттєво покращує розуміння семантики текстів, однак обмежена у роботі зі структурою коду;

– CodeBERT інтегрує обидва підходи, досягаючи найкращого балансу між точністю, повнотою та узагальненням результатів.

Таким чином, застосування CodeBERT дозволяє суттєво підвищити ефективність процесу трасування вимог, забезпечивши стабільні та точні результати навіть у складних сценаріях з багаторівневою семантикою. Високі значення всіх метрик, включно з MRR, підтверджують доцільність використання великих мовних моделей для інтегрованого аналізу текстових вимог та структурних елементів коду.

У ході проведених експериментів для цього етапу було здійснено порівняння трьох підходів до трасування вимог: класичного статистичного методу TF-IDF, контекстної моделі SBERT та спеціалізованої трансформерної моделі CodeBERT. Аналіз показав закономірне зростання ефективності від простих лексичних методів до складних нейронних підходів, які враховують семантичні та структурні зв’язки між вимогами та програмним кодом.

TF-IDF, попри свою простоту та низькі обчислювальні витрати, демонструє обмежену повноту та точність, що зумовлено відсутністю механізмів врахування контексту та синтаксичної структури коду. SBERT, застосовуючи контекстні векторні представлення, значно покращує точність і F1-score, однак її ефективність обмежена при аналізі програмного коду, оскільки модель навчалася переважно на текстах природної мови і не враховує структурні особливості методів та змінних.

CodeBERT поєднує переваги обох підходів, одночасно обробляючи текст вимог і структурні елементи коду. Вона забезпечує найвищі показники Precision, Recall, F1-score, MAP та MRR, що підтверджує її здатність ефективно встановлювати релевантні відповідності навіть у складних сценаріях із багаторівневою семантикою. Модель демонструє стабільність результатів при зміні параметрів пошуку та високий рівень покриття релевантних кандидатів, що робить її оптимальним інструментом для автоматизації трасування вимог у промислових проектах.

Отримані результати підтверджують гіпотезу про доцільність застосування великих мовних моделей для інтегрованого аналізу текстових вимог і програмних артефактів. Спільне моделювання тексту та коду у єдиному векторному просторі дозволяє виявляти семантичні відповідності, які складно або неможливо виявити класичними статистичними методами. Це відкриває перспективи для подальшого використання таких моделей у системах підтримки розробки програмного забезпечення, підвищуючи точність трасування, прискорюючи робочі процеси та знижуючи витрати часу на ручну перевірку.

Наступним етапом експериментального дослідження стало детальне вивчення впливу параметра Top-K на результати трасування вимог до програмного коду. Цей параметр є одним із ключових у задачах інформаційного пошуку та векторного порівняння, оскільки визначає, скільки найподібніших методів система повертає для кожної вимоги під час аналізу багатовимірних векторних представлень. Інакше кажучи, Top-K задає глибину ранжування – від обмеженого аналізу перших 10 – 50 найближчих методів до значного розширення пошуку на сотні або тисячі кандидатів (K = 500 – 2000).

Занадто мале значення K призводить до того, що частина релевантних методів може не потрапляти у список кандидатів, що суттєво обмежує повноту системи. Натомість надто велике K збільшує обсяг нерелевантних результатів, знижуючи точність та ускладнюючи подальший ручний аналіз. Тому вивчення впливу цього параметра є критично важливим для визначення оптимального балансу між повнотою та точністю в процесі трасування вимог.

Для експериментів використовувалася модель CodeBERT, при цьому всі інші параметри векторизації, такі як MAX_LEN, токенізація, обчислювальний пристрійта FAISS-індексація, залишалися сталими. Ембединги всіх методів зберігалися у кеші для забезпечення відтворюваності результатів. Це дозволило зосередитися саме на ефекті зміни Top-K, виключивши додаткові змінні, що могли вплинути на метрики.

Метрики Precision, Recall, F1-score, MAP та MRR обчислювалися за допомогою спеціальної функції eval_at_ks для кожного значення K, що дало змогу детально відстежувати динаміку зміни якості трасування при зростанні глибини пошуку та визначити точку насичення, після якої подальше збільшення K не дає значного покращення результатів.

Результати аналізу наведено у таблиці 4.4, яка відображає залежність середніх значень метрик для всіх вимог проєкту від параметра Top-K.

Таблиця 4.4 – Залежність метрик точності від параметра Top-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

Як видно з таблиці 4.4, зі збільшенням значення параметра K спостерігається монотонне зростання метрики Recall – від 0,0024 при K = 10 до 0,3937 при K = 2000. Це пояснюється тим, що розширення вибірки кандидатів дає змогу моделі CodeBERT охопити більшу кількість релевантних методів для кожної вимоги. Іншими словами, чим глибше аналізується ранжований список,

тим вищий шанс виявлення всіх потенційних відповідностей між вимогами та фрагментами коду.

Водночас спостерігається поступове зниження Precision, що обумовлено потраплянням у список кандидатів менш релевантних методів. Така поведінка є типовою для систем інформаційного пошуку і відображає класичний компроміс між точністю та повнотою: збільшення обсягу аналізованих кандидатів підвищує охоплення правильних результатів, але знижує відсоток релевантних методів серед перших позицій.

Метрика F1-score, яка є гармонійним середнім між Precision і Recall, демонструє значне зростання – з 0,0048 до 0,5018 при K = 2000. Це підтверджує, що при K ≥ 500 модель здатна одночасно підтримувати прийнятний рівень точності і значно збільшувати повноту виявлення релевантних методів. Подібна динаміка спостерігається і для MAP, яке зростає від 0,0022 до 0,2821, що свідчить про покращене ранжування релевантних методів у списку кандидатів та більш рівномірний розподіл релевантних елементів по топовій частині вибірки.

Важливо відзначити, що показник MRR залишається стабільним на рівні = 0,9722 для всіх значень K. Це означає, що принаймні один релевантний метод завжди потрапляє у верхні позиції списку, незалежно від глибини пошуку. Така стабільність свідчить про надійність і ефективність CodeBERT у практичних умовах трасування вимог, забезпечуючи користувачеві швидкий доступ до одного правильного кандидата без потреби переглядати весь список.

На рисунку 4.5 наведено графічне представлення середніх значень метрик по всіх вимогах залежно від параметра Top-K.

Графік демонструє характерний компроміс між точністю та повнотою: Precision поступово знижується зі збільшенням K, тоді як Recall, F1-score та MAP стабільно зростають, досягаючи плато при K = 500-1000. Така поведінка чітко ілюструє закономірності, властиві більшості систем інформаційного пошуку та рекомендаційних систем, де баланс між точністю та повнотою є критичним.

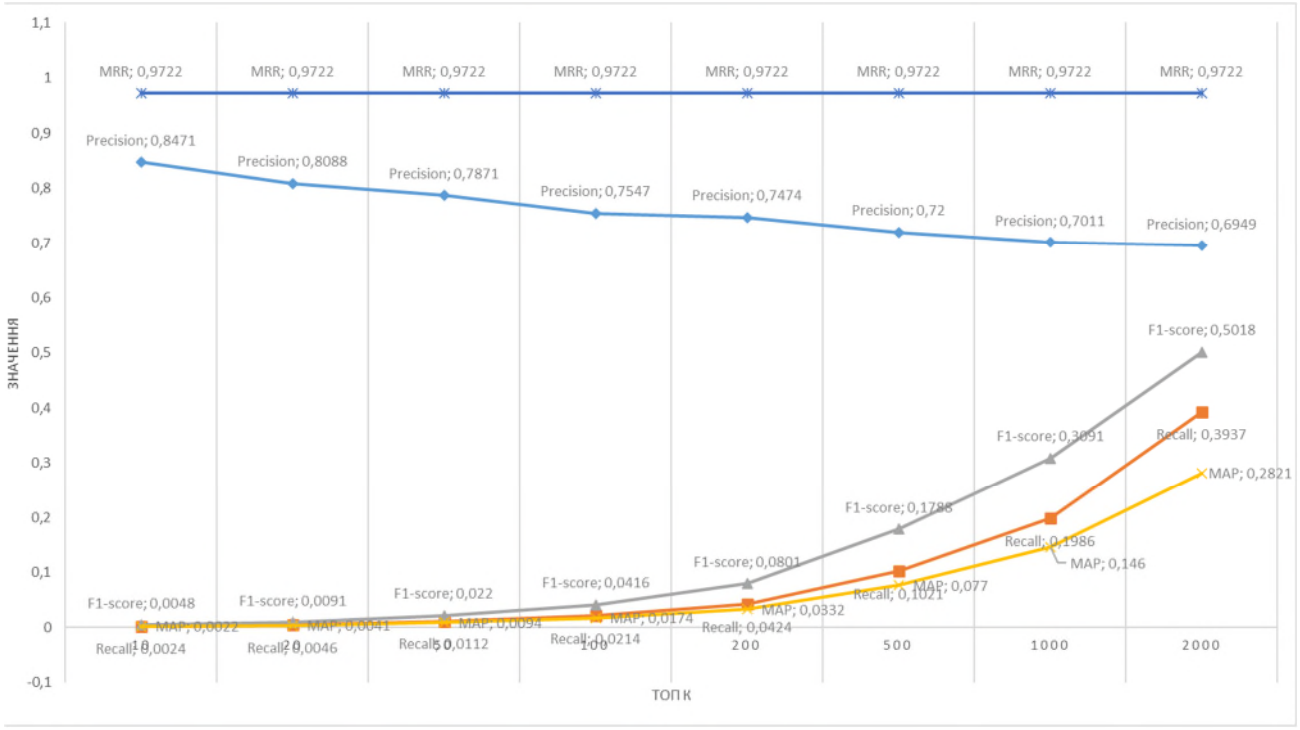


Рисунок 4.5 – Динаміка зміни основних метрик (Precision, Recall, F1-score, MAP, MRR) залежно від параметра Top-K

На рисунку 4.5 наведено середні значення основних метрик трасування для всіх вимог залежно від параметра Top-K. Графік демонструє характерний компроміс між точністю та повнотою, який є типовим для систем інформаційного пошуку та рекомендальних систем. Аналіз показує, що зміна K суттєво впливає на поведінку всіх метрик, і правильне його налаштування є ключовим для забезпечення збалансованого процесу трасування.

На початкових значеннях K = 10 – 50 система працює як високоточний, але вузько сфокусований фільтр: Precision знижується від 0,8471 до 0,7871, тоді як Recall зростає лише від 0,0024 до 0,0112. Це означає, що на малих глибинах вибірки у верхні позиції потрапляють лише найбільш релевантні методи, але більшість правильних трасувальних зв’язків залишаються поза списком кандидатів. У практичних умовах така конфігурація може бути корисною для швидкого виділення критично важливих методів, однак загальне покриття проекту буде недостатнім.

При подальшому збільшенні K до 100 – 200 спостерігається поступове зростання Recall (0,0214 – 0,0424) і відповідне збільшення F1-score (0,0416 –

0,0801), тоді як Precision зменшується помірно (0,7547 – 0,7474). Це свідчить про те, що модель починає охоплювати більшу частину релевантних методів без значного падіння точності. MAP при цьому зростає від 0,0174 до 0,0332, що показує поліпшення ранжування кандидатів: релевантні методи з'являються не лише на верхніх позиціях, а й на деяких середніх рівнях списку.

Найбільш помітні зміни відбуваються при K ≥ 500. Тут Recall зростає до 0,1021 (для K = 500), а F1-score до 0,1788, при цьому Precision ще залишається на прийнятному рівні 0,7200. MAP зростає до 0,0770, що вказує на значне покращення якості ранжування релевантних методів. MRR = 0,9722 залишається стабільним, демонструючи, що хоча б один релевантний метод постійно потрапляє у верхні позиції списку, забезпечуючи надійність пошуку навіть при великому розширенні вибірки.

При K = 1000 спостерігається подальше збільшення Recall до 0,1986 та F1-score до 0,3091. Precision трохи знижується до 0,7011, що є типовим компромісом для систем інформаційного пошуку: щоб знайти більше релевантних об'єктів, модель змушена включати у список кандидатів частину менш точних результатів. MAP у цей момент зростає до 0,1460, демонструючи, що релевантні методи займають більш рівномірні позиції у списку кандидатів, а ранжування стає стабільнішим і прогнозованішим.

Найбільше K = 2000 показує, що Recall досягає значення 0,3937, а F1-score 0,5018, тобто модель знаходить майже половину всіх можливих релевантних методів у списку. Precision знижується до 0,6949, MAP до 0,2821. Це явище насичення списку кандидатів свідчить про те, що подальше збільшення K не приносить пропорційного приросту повноти, тоді як точність продовжує знижуватися. MRR залишається стабільним на рівні 0,9722, підтверджуючи, що верхня частина списку завжди містить хоча б один релевантний метод.

Аналіз графіка також дозволяє відзначити важливі закономірності:

– При малих K система виявляє лише обмежену кількість релевантних методів, але робить це максимально точно. При великих K повнота значно зростає, але точність трохи падає.

– Середні значення метрик по всіх вимогах показують, що K = 500 – 1000 забезпечує найкращий баланс між точністю та повнотою, дозволяючи ефективно знаходити більшість релевантних методів без істотного падіння Precision.

– MRR залишається практично незмінним, що свідчить про здатність CodeBERT завжди виявляти принаймні один релевантний метод у верхній частині списку кандидатів.

– F1-score і MAP збільшуються зі зростанням K, демонструючи покращення збалансованості між повнотою та точністю та стабільність ранжування релевантних методів.

– При K > 1000 ефективність подальшого збільшення глибини вибірки зменшується: Recall росте повільніше, Precision знижується, а F1-score і MAP наближаються до плато.

Таким чином, отримані результати демонструють, що параметр Top-K є критично важливим для контролю якості трасування. Оптимальне налаштування K дозволяє значно покращити ефективність системи, зберігати високу якість ранжування та забезпечити стабільні результати незалежно від кількості вимог у проекті.

З практичної точки зору, такий аналіз показує, що для великих програмних проектів з великою кількістю вимог і методів коду вибір K у діапазоні 500 – 1000 дозволяє досягти компромісу між повнотою та точністю: система знаходить більшість релевантних методів, при цьому верхні позиції списку залишаються максимально точними для користувача. Крім того, стабільність MRR гарантує, що критично важливі методи завжди будуть виявлені першими, що підвищує надійність і ефективність процесу трасування.

Проведене експериментальне дослідження демонструє ефективність застосування сучасних методів трасування вимог до програмного коду із залученням великих мовних моделей. Зокрема, модель CodeBERT показала значно кращі результати порівняно з класичними підходами, такими як TF-IDF та SBERT. Це свідчить про високий потенціал нейронних моделей для аналізу зв’язку між текстовими вимогами та структурою програмного коду.

Одним із важливих факторів, що впливають на якість трасування, є використання структурного контексту. Включення додаткової інформації про назву класу та вихідний код методу дозволяє моделі точніше ідентифікувати відповідні елементи коду для кожної вимоги. Це підтверджується покращенням усіх ключових метрик: точності, повноти та їх комбінованого показника F1-score. Такий підхід знижує кількість помилкових співпадінь та пропущених відповідностей, що особливо важливо для великих та складних проектів.

Не менш значущим параметром у процесі трасування є Top-K, який визначає глибину аналізу та кількість найбільш релевантних елементів, що повертаються для кожної вимоги. Результати показали, що оптимальне налаштування цього параметра дозволяє досягти балансу між точністю та повнотою, тобто отримати максимально корисний набір кандидатів без перевантаження користувача зайвою інформацією. Високі значення Top-K забезпечують більшу повноту, тоді як нижчі значення дозволяють підвищити точність, що дає змогу гнучко налаштовувати процес під конкретні потреби проекту.

Загалом, результати дослідження підтверджують доцільність застосування LLM для інтегрованого аналізу тексту вимог та програмного коду. Використання таких моделей дозволяє підвищити ефективність процесу трасування, зменшити ймовірність помилок і забезпечити більш глибоке розуміння взаємозв’язків між вимогами та реалізацією. Це особливо актуально для великих проєктів із складною структурою, де ручне трасування є ресурсозатратним і не завжди точним.

Таким чином, експериментально підтверджено, що сучасні підходи на основі великих мовних моделей не лише підвищують якість трасування вимог, але й створюють передумови для подальшої автоматизації та оптимізації процесів забезпечення відповідності між специфікаціями та програмним кодом. Впровадження таких рішень може стати ключовим фактором підвищення якості розробки програмного забезпечення, скорочення часу на перевірку та зниження ризиків виникнення помилок у кінцевому продукті.

Висновки до розділу 4

У межах даного розділу було проведено детальні експериментальні дослідження запропонованого методу трасування вимог до програмного коду із застосуванням великих мовних моделей. Основною метою досліджень було оцінити ефективність моделей TF-IDF, SBERT та CodeBERT у поєднанні з різними конфігураціями векторного представлення коду, а також визначити вплив параметра Top-K на якість трасування. Результати експериментів дозволяють сформулювати ключові висновки, які підтверджують доцільність використання спеціалізованих LLM-підходів та структурно збагаченого представлення коду для підвищення ефективності трасування.

Дослідження показали, що включення додаткових елементів коду, таких як classname та sourcecode, у процес векторизації суттєво покращує якість трасування. При використанні лише fullmethod Precision становило 0.6955, Recall – 0.1001, F1-score – 0.1627, MAP – 0.0750, а MRR – 0.9546. Після включення структурного контексту всі метрики підвищилися: Precision = 0.7200, Recall = 0.1021, F1-score = 0.1788, MAP = 0.0770, а MRR збільшився до 0.9722. Це свідчить про те, що врахування контексту коду дозволяє моделі більш точно відображати семантичні зв’язки між текстом вимог і методами коду, забезпечує стабільне покращення всіх ключових метрик і робить результати трасування більш надійними.

Порівняльний аналіз показав, що CodeBERT у конфігурації з classname + sourcecode перевершує класичні методи TF-IDF та універсальний SBERT. Для TF-IDF Precision становило 0.6228, Recall – 0.0890, F1-score – 0.1558, MAP – 0.0619, а MRR – 0.8647. Модель SBERT показала дещо кращі результати: Precision = 0.7049, Recall = 0.0907, F1-score = 0.1604, MAP = 0.0729, MRR = 0.7059. Найвищі показники за всіма ключовими метриками забезпечила модель CodeBERT: Precision = 0.7200, Recall = 0.1021, F1-score = 0.1788, MAP = 0.0770, MRR = 0.9722.

Це підтверджує перевагу спеціалізованих LLM-підходів у завданнях трасування вимог, особливо при використанні структурно збагаченого представлення коду.

Дослідження впливу параметра Top-K продемонстрували типовий компроміс між точністю та повнотою. При невеликих значеннях K < 100, Precision залишався високим, проте Recall і F1-score були дуже низькими, що свідчить про те, що більшість релевантних методів залишалися поза верхніми позиціями результатів. Зі збільшенням K точність поступово знижується, тоді як Recall та F1-score значно зростають. Оптимальний баланс між точністю та повнотою досягається при K = 500 – 1000, де Precision коливається близько 0.72-0.70, а F1-score та Recall забезпечують прийнятний рівень ефективності для практичного використання. Це означає, що у вказаному діапазоні K система знаходить більшість релевантних методів без істотного зниження точності.

Для всіх розглянутих конфігурацій MRR залишався стабільно високим (0.9722 для CodeBERT із контекстом), що підтверджує надійність ранжування: релевантні методи майже завжди потрапляють у верхні позиції списку результатів незалежно від вибору Top-K. Це важлива характеристика для практичного використання, оскільки дозволяє користувачеві швидко знаходити потрібні методи без додаткового фільтрування.

Використання бібліотеки FAISS для кешування ембедингів забезпечило швидку і повторювану роботу системи навіть при великих значеннях K. Це дозволяє інтерактивно аналізувати результати трасування та оперативно перевіряти зміни в коді або вимогах, що робить систему придатною для використання в реальних проєктних умовах.

Подальше доналаштування CodeBERT на цільових проєктах, застосування підходів continual learning, інтеграція графових ознак (AST, графи викликів і потоків даних) та метаданих (імена пакетів, шляхи файлів) можуть значно підвищити ефективність трасування. Використання reranking-модулів поверх швидкого retrieval та включення human-in-the-loop фідбеку дозволить поступово підвищувати повноту без істотної втрати точності.

Підсумовуючи, запропонований підхід на базі CodeBERT, підсилений структурним контекстом коду та ефективною індексацією, забезпечує якісне початкове трасування «з коробки», перевершує класичні методи за ключовими метриками та має потенціал для подальшого вдосконалення.Найкращі результати очікуються за комбінації: CodeBERT + структурний контекст + K = 500 – 1000 + доменна адаптація / reranking, що дозволить суттєво підвищити повноту без істотної втрати точності та зробити метод придатним для промислового використання.

Цей підхід забезпечує надійне і практично застосовне рішення для інтерактивного аналізу та автоматизованого трасування вимог до програмного коду.

Загальні висновки

У магістерській роботі досягнуто поставленої мети – підвищення точності та повноти процесу трасування вимог до програмного коду шляхом розробки методу автоматичного виявлення зв’язків із використанням великих мовних моделей.

Запропонований підхід дозволяє поєднати сучасні трансформерні архітектури зі засобами семантичного пошуку, що забезпечує автоматизацію одного з найбільш трудомістких і критичних етапів інженерії вимог – встановлення відповідностей між специфікаціями та артефактами програмного забезпечення. Це дозволяє значно скоротити ручну роботу, знизити ймовірність помилок та підвищити точність і повноту процесу трасування.

Метод побудовано на двоетапній архітектурі retrieval → rerank, де перший етап забезпечує швидкий пошук найбільш релевантних фрагментів коду за допомогою векторних представлень вимог, а другий етап уточнює результати на основі семантичної подібності. Важливим аспектом є врахування структурного контексту коду, що дозволяє моделі більш повно сприймати логіку та призначення методів у проєкті, а не лише їх текстовий опис. Дослідження показали, що використання спеціалізованих моделей, навчених на змішаних корпусах тексту та програмного коду, дає значну перевагу над універсальними підходами загального призначення, оскільки такі моделі здатні захоплювати особливості синтаксису та семантики програмного коду.

Розроблено програмну реалізацію із модульною архітектурою та інтерактивним графічним інтерфейсом, який включає етапи підготовки даних, генерації векторних представлень вимог і коду, побудови індексу схожості та швидкого пошуку найбільш релевантних методів для кожної вимоги. Система також забезпечує оцінку якості трасування та інтерактивну візуалізацію результатів у реальному часі. Користувач може аналізувати окремі вимоги та методи, переглядати локальні метрики та спостерігати зміни показників у динаміці, що робить програмне забезпечення придатним не лише для

дослідницьких експериментів, а й для практичного використання під час реальної розробки програмного забезпечення.

Експериментальні дослідження охопили порівняння різних підходів, оцінку впливу структурного контексту та аналіз параметра Top-K, який визначає кількість найбільш релевантних методів, що повертаються для кожної вимоги. Результати підтвердили, що врахування структурного контексту дозволяє підвищити точність і стабільність ранжування, а використання параметра Top-K забезпечує можливість балансування між точністю та повнотою. Аналіз впливу Top-K показав класичний компроміс: зі збільшенням K зростає повнота, але знижується точність, що дозволяє адаптувати систему під конкретні потреби проєкту, наприклад, для більшого охоплення всіх потенційних відповідностей або для забезпечення максимальної точності.

Практична цінність запропонованого методу та реалізованого програмного прототипу полягає у значному скороченні ручної роботи при формуванні матриці трасованості, підтримці процесу верифікації реалізації вимог, прискоренні рев'ю змін у коді та виявленні потенційних проблем, таких як пропуски або дублювання реалізації вимог. Відкрита архітектура системи забезпечує інтеграцію з існуючими інструментами управління проєктами (Jira, GitLab, GitHub) і дозволяє адаптувати систему під специфіку будь-якого програмного середовища, що робить її універсальною і гнучкою для практичного використання.

Основний науковий результат роботи полягає в удосконаленні методу семантичного трасування, який завдяки поєднанню двоетапного пошуку та врахуванню структурного контексту на базі великих мовних моделей забезпечує суттєве підвищення точності та повноти виявлення зв'язків. Впровадження параметричної оцінки (Top-K) дозволило оптимізувати баланс метрик, забезпечуючи високу стабільність результатів навіть на великих масивах даних, що підтверджено експериментально.

Таким чином, поставлена в роботі мета повністю досягнута. Створено метод і програмний прототип, які забезпечують ефективне, автоматизоване та об'єктивне трасування вимог до програмного коду. Результати дослідження

мають високу наукову та практичну цінність, а реалізоване програмне рішення може бути впроваджене у реальні середовища розробки для підвищення прозорості, керованості та ефективності процесу розробки програмного забезпечення, зменшення ручної праці та підвищення якості продукту.

За темою кваліфікаційної роботи автором виконано дві наукові публікації.

Основні наукові та практичні результати пройшли апробацію на XVII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп'ютерних наук (АПКН – 2025)» (м. Хмельницький, 14–15 листопада 2025 р.) та опубліковані у фаховому виданні «Вісник Хмельницького національного університету. Технічні науки»:

1. Скрипнюк О.Ю., Манзюк Е.А., Багрій Р.О., Петровський С.С. Метод виявлення трасувальних зв'язків між вимогами та програмним кодом із використанням великих мовних моделей. Актуальні проблеми комп'ютерних наук АПКН-2025 : матеріали XVII Всеукр. науково-практ. конф., м. Хмельницький, 14–15 листопада 2025 р. Хмельницький, 2025. С. 380–384.

2. Скрипнюк О.Ю., Багрій Р.О., Манзюк Е.А., Скрипник Т.К. автоматизоване відновлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей. Вісник Хмельницького національного університету. Технічні науки. 2026. Т. 361, № 1.

Перелік посилань

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. Li X., Wang B., Wan H., Deng Y., Wang Z. *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

3. Charalampidou S. et al. *Empirical studies on software traceability: A mapping study*. Journal of Software: Evolution and Process. 2021. Vol. 33. No. 2. DOI: 10.1002/smr.2294

4. 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

5. 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

6. Dai P., Liu H., Zhang N. *Constructing traceability links between software requirements and source code based on self-attention neural networks.* Mathematics. 2023. Vol. 11. No. 2. P. 315. DOI: 10.3390/math11020315

7. Ahmadiyah A. S., Sungkono K. *Monthes: A compact software traceability dataset.* In: Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). 2023. P. 121-130. DOI: 10.5220/0012105700003464

8. Jiang J. et al. *A survey on large language models for code generation*. arXiv. 2024. arXiv: 2406.00515. URL: https://arxiv.org/abs/2406.00515

9. 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

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

11. Li T. et al. *Machine learning for requirements engineering (ML4RE): A systematic literature review complemented by practitioners' voices.* Information and Software Technology. 2024. Vol. 170. DOI: 10.1016/j.infsof.2024.107477

12. 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/2203.03850

13. Rosado da Cruz A. M. et al. *Machine learning techniques for requirements engineering.* Machine Learning and Knowledge Extraction. 2022. Vol. 4. No. 3. P. 610-627. DOI: 10.3390/make4030029

14. Legesse H., Bicknell G. *AI-enhanced requirements traceability using MBSE and large language models for complex systems.* In: AI4SE & SE4AI Workshop. 2025. URL: https://sercuarc.org/wp-content/uploads/2025/09/Legesse_AI_Enhanced_Requirements_Traceability_Using_MBSE_LLM_Complex_Systems.pdf

15. Ali S. J., Naganathan V., Bork D. *Establishing traceability between natural language requirements and source code using large language models and retrieval-augmented generation.* In: Conceptual Modeling. ER 2024. Lecture Notes in Computer Science. 2024. Vol. 15238. P. 295-314. DOI: 10.1007/978-3-031-76988-7_21

16. Wang Y. et al. *Unified pre-training for program understanding and generation.* arXiv. 2021. arXiv: 2103.06333. URL: https://arxiv.org/abs/2103.06333

17. Ge C., Wang T., Yang X., Treude C. *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

18. Wang F. et al. *Embedding traceability in large language model code generation: Towards trustworthy AI-augmented software engineering.* In: Proceedings

of the 1st Workshop on Software Genomics (SWGeno). 2025. DOI: 10.1145/3696630.3730569

19. Zhang J. et al. *Enhancing requirement traceability through data augmentation using large language models.* SSRN Electronic Journal. 2024. DOI: 10.2139/ssrn.5111271

20. Hassine J. *An LLM-based approach to recover traceability links between security requirements and goal models.* In: Proceedings of the 17th International Conference on Software Engineering and Formal Methods (SEFM). 2024. DOI: 10.1145/3661167.3661261

21. Guo J. L. C. *Natural language processing for requirements traceability.* arXiv. 2024. arXiv: 2405.10845. URL: https://arxiv.org/abs/2405.10845

22. Wang B. et al. *LLM-driven cost-effective requirements change impact analysis.* arXiv. 2025. arXiv: 2511.00262. URL: https://arxiv.org/abs/2511.00262

23. Wang S. et al. *Natural language to code: How far are we?* In: Proceedings of the ACM on Software Engineering. 2023. Vol. 1. No. FSE. P. 1364-1385. DOI: 10.1145/3597503

24. *Software traceability across SDLC: A comprehensive survey.* EasyChair Preprint. 2024. No. 12792. URL: https://easychair.org/publications/preprint/JJRn

25. Lin J. et al. *Enhancing automated software traceability by transfer learning from open-world domain data.* Empirical Software Engineering. 2022. Vol. 27. No. 5. Art. no. 110. DOI: 10.1007/s10664-022-10150-w

26. Cleland-Huang J. *Grand challenges of traceability: The next ten years* J. Cleland-Huang [et al.] IEEE Software. – 2023. – Vol. 40, № 1. – P. 4–11. – DOI: 10.1109/MS.2022.3221743.

27. Rodriguez A. D., Dearstyne K. R., Cleland-Huang J. *Prompts matter: Insights and strategies for prompt engineering in automated software traceability*. arXiv. 2023. arXiv: 2308.00229. URL: https://arxiv.org/abs/2308.00229

28. Hou X. et al. *Large language models (LLMs) for requirements engineering (RE): A systematic literature review*. arXiv. 2024. arXiv: 2509.11446. URL: https://arxiv.org/abs/2509.11446

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

30. Singh A. et al. *Evolution of meta's llama models and parameter-efficient fine-tuning methods: A survey*. arXiv. 2025. arXiv: 2510.12178. URL: https://arxiv.org/abs/2510.12178

31. Ouyang L. et al. *Training language models to follow instructions with human feedback.* In: Advances in Neural Information Processing Systems (NeurIPS). 2022. URL: https://arxiv.org/abs/2203.02155

32. Khan A. A. et al. *A systematic literature review of retrieval-augmented generation (RAG).* arXiv. 2024. arXiv: 2508.06401. URL: https://arxiv.org/abs/2508.06401

33. Chen M. et al. *Evaluating large language models trained on code*. arXiv. 2021. arXiv: 2107.03374. URL: https://arxiv.org/abs/2107.03374

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

35. Khalil F., Rebdawi G., Ghneim N. *Tracking the relationships between requirements and other development artifacts: A systematic mapping review*. ECTI Transactions on Computer and Information Technology. 2025. Vol. 19. No. 2. P. 321-335. DOI: 10.37936/ecticit.2025192.258621

36. Python. Офіційна документація мови програмування Python. URL: https://www.python.org

37. Visual Studio Code. Офіційний сайт середовища розробки. URL: https://code.visualstudio.com

38. Google Colab. Онлайн-середовище для виконання Python-ноутбуків. URL: https://colab.research.google.com

39. Hugging Face. Документація по моделям та бібліотекам Transformers. URL: https://huggingface.co/docs

40. PyTorch. Платформа глибокого навчання. URL: https://pytorch.org

41. Scikit-learn. Бібліотека машинного навчання для Python. URL: https://scikit-learn.org/stable

42. Pandas. Бібліотека аналізу даних для Python. URL: https://pandas.pydata.org

43. Facebook AI Similarity Search (FAISS). Бібліотека для пошуку за векторними поданнями. URL: https://github.com/facebookresearch/faiss

44. TF-IDF – Term Frequency–Inverse Document Frequency. URL: https://en.wikipedia.org/wiki/Tf-idf

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

ДОДАТКИ

Додаток А

Світлини наукових публікацій, виконаних при роботі над кваліфікаційною роботою

1. Скрипнюк О.Ю., Багрій Р.О., Манзюк Е.А., Скрипник Т.К. автоматизоване відновлення трасувальних зв’язків між вимогами та програмним кодом з використанням великих мовних моделей. Вісник Хмельницького національного університету. Технічні науки. 2026. Т. 361, № 1.

2. Скрипнюк О.Ю., Манзюк Е.А., Багрій Р.О., Петровський С.С. Метод виявлення трасувальних зв’язків між вимогами та програмним кодом із використанням великих мовних моделей. Актуальні проблеми комп’ютерних наук АПКН-2025 : матеріали XVII Всеукр. науково-практ. конф., м. Хмельницький, 14–15 листопада 2025 р. Хмельницький, 2025. С. 380–384.

УДК 004.8
DOI:

СКРИПНЮК ОЛЕКСАНДР
Хмельницький національний університет
ORCID ID: 0009-0006-1216-379X
e-mail: cawa.ckp@gmail.com
БАГРІЙ РУСЛАН
Хмельницький національний університет
ORCID ID: 0000-0001-5219-1185
e-mail: bahriiro@khmnu.edu.ua
МАНЗЮК ЕДУАРД
Хмельницький національний університет
ORCID ID: 0000-0002-7310-2126
e-mail: manziuk.e@khmnu.edu.ua
ПЕТРОВСЬКИЙ СЕРГІЙ
Хмельницький національний університет
ORCID ID: 0000-0002-0590-0484
e-mail: petrovskijss@khmnu.edu.ua

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.

Olexandr SKRYPNIUK, Ruslan BAHRII, Eduard Manziuk, Sergii PETROVSKYI
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

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.

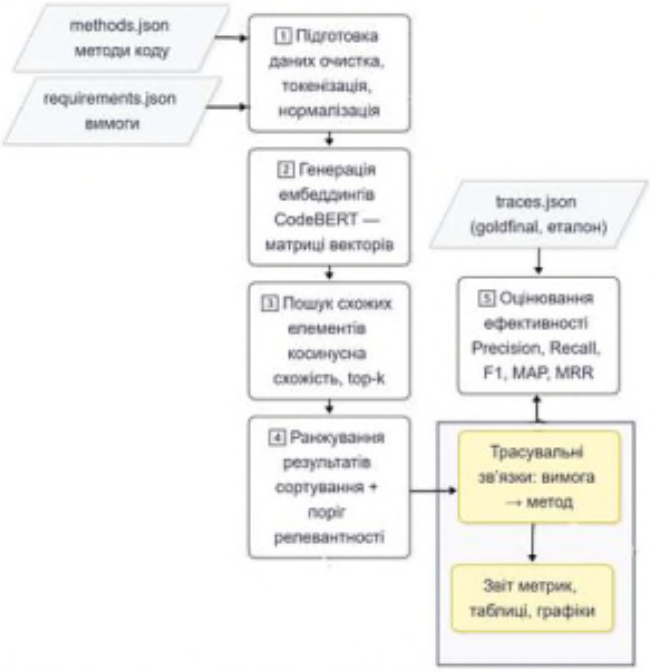


Рис. 1. Схема методу виявлення трасувальних зв'язків між вимогами та програмним кодом.

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. Ембедінги 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 ембедінгів.

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 ембедінгів 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.

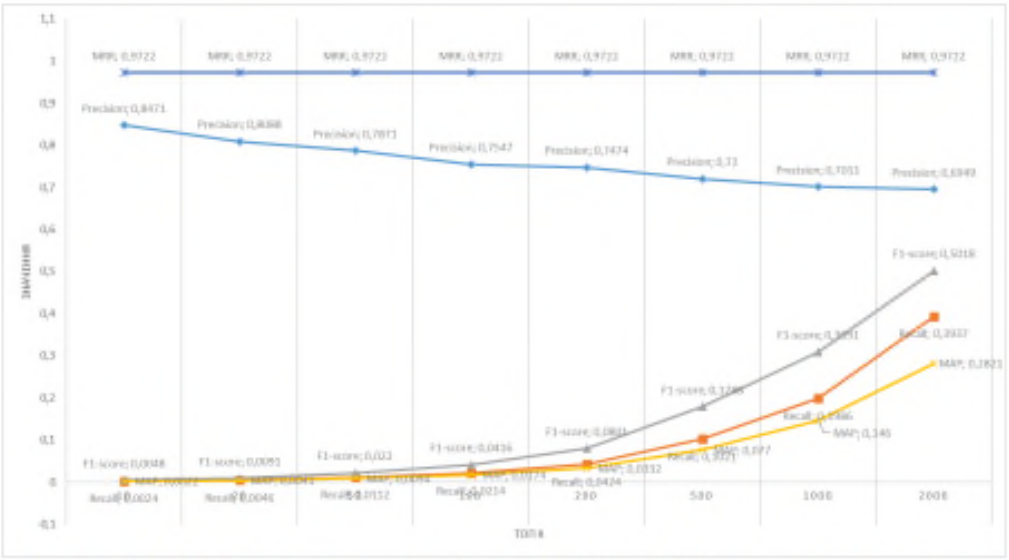


Рис. 2. – Динаміка зміни основних метрик (Precision, Recall, F1-score, MAP) залежно від параметра Top-K

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

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ

за матеріалами XVII Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп'ютерних наук АПКН-2025»

14-15 листопада 2025

Хмельницький 2025

Сиротенко Д.А., Троц В.В., Анікін В.А.
Ризики використання штучного інтелекту з перспективи кібербезпеки та захисту інформації 377

Скрипнюк О.Ю, Манзюк Е.А, Багрій Р.О, Петровський С.С.
Метод виявлення трасувальних зв'язків між вимогами та програмним кодом із використанням великих мовних моделей 380

Соколовський В.С., Манзюк Е.А.
Метод класифікації патологій листя рослин на основі згорткових нейронних мереж 385

Старостенко К.В.
Аналіз ефективності методів машинного навчання для виявлення мережевих атак типу DDoS 390

Стецюк П.П., Форкун Ю.В.
Метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання . 393

Тимофієв І.А., Мазурець О.В.
Нейромережевий підхід до виявлення депресивних патернів за аналізом текстового контенту цифрових сервісів у закладах освіти 395

Тростянецький Н.О., Кльоц Ю.П., Калій К.В. Откидач В.В.
Виявлення атак типу «блокування IP через NAT» в публічних мережах 405

Трохимчук О.В., Пасічник О.А., Поплавська О.А., Міхалевський В.Ц.
Підхід до оцінювання відповідності хештегів коротким текстам засобами NLP ... 409

Філюк Є.В., Джулій В.М.
Метод безпеки криптоактивів на основі технології багатосторонніх обчислень ... 413

Футорний Р.В., Медведчук Н.К.
Дослідження виявлення кібератак в Агро-IOT з використанням аналізу енергоспоживання 417

Ціцьвіра І.О., Радюк П.М., Скрипник Т. К.
Метод агентно-орієнтованого аналізу ринку криптовалют з використанням великих мовних моделей 421

Червончук І.С.
Аналіз методів та засобів виявлення логів у програмному забезпеченні 425

УДК 004.8

Скрипнюк О.Ю, Манзюк Е.А, Багрій Р.О, Петровський С.С.

Хмельницький національний університет

МЕТОД ВИЯВЛЕННЯ ТРАСУВАЛЬНИХ ЗВ'ЯЗКІВ МІЖ ВИМОГАМИ ТА ПРОГРАМНИМ КОДОМ ІЗ ВИКОРИСТАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

Запропоновано метод автоматичного встановлення трасувальних зв'язків між вимогами програмного забезпечення та програмним кодом на основі великих мовних моделей (LLM). Метод реалізує багаторівневу архітектуру, що складається з модулів підготовки даних, генерації ембедингів, пошуку, оцінювання та візуалізації результатів. Проведено експериментальне дослідження на наборі MSR-2021, у якому порівняно модель CodeBERT із TF-IDF та SBERT, а також оцінено вплив структурного контексту, параметра Top-K. Отримані результати підтверджують ефективність методу та можливість його масштабування для реальних проєктів.

A method for automatic detection of traceability links between software requirements and source code using Large Language Models (LLMs) is proposed. The method implements a multi-layer architecture with modules for data preparation, CodeBERT-based embedding generation, retrieval, evaluation, and visualization. Experiments on the MSR-2021 dataset compare CodeBERT with TF-IDF and SBERT, analyzing the effects of structural context, Top-K parameter. Results confirm the method's efficiency and scalability for real software projects.

Трасування вимог у програмному забезпеченні забезпечує контроль узгодженості між вимогами й реалізацією, даючи змогу виявляти невідповідності, оцінювати вплив змін і підтримувати прозорість процесу розробки [1]. Ручне виконання трасування є трудомістким і схильним до помилок, особливо у великих проєктах. Класичні методи, засновані на статистичних або лексичних підходах, таких як TF-IDF чи LSI, враховують лише частотні співвідношення слів, не розуміючи їхнього контексту [2]. Розвиток трансформерних архітектур і поява великих мовних моделей дозволяють застосовувати семантичне зіставлення між текстовими та програмними артефактами [3]. Що стало підґрунтям для подальших досліджень у цій сфері.

Проблема трасування вимог розглядається у численних працях як один із ключових аспектів управління вимогами. У подальших дослідженнях [4] було запропоновано референтні моделі та інструменти автоматизації. Впровадження глибинного навчання та моделей типу трансформерів дозволило формувати спільний embedding-простір для тексту й коду, де семантично близькі об'єкти розміщуються поруч [5]. Для оцінювання ефективності таких моделей створено

еталонний набір Traceability Dataset for Open Source Systems (MSR-2021), що містить чотири відкриті проєкти (GanttProject, eTour, SMOS, iTrust) [6].

А також застосування нейронних підходів до трасування показало можливість суттєвого підвищення якості ідентифікації зв'язків між вимогами та кодом [7], а дослідження у сфері Explainable AI підтвердили, що LLM можуть пояснювати знайдені відповідності природною мовою [8]. На основі цього було сформульовано мету роботи.

Метою дослідження є розроблення методу виявлення трасувальних зв'язків між вимогами та програмним кодом із використанням великих мовних моделей, що забезпечує підвищення точності трасування та скорочення часу аналізу.

Запропонований метод складається з п'яти послідовних етапів (рис. 1):

1. Підготовка даних. Вхідними даними є JSON-файли вимог, методів і трасувальних пар. Дані очищуються від шуму, токенізуються й нормалізуються.

2. Генерація ембеддингів. Для кожного тексту вимоги та методу коду обчислюється векторне представлення за допомогою моделі microsoft/CodeBERT-base. Отримані вектори зберігаються у матрицях для подальшого пошуку.

3. Пошук схожих елементів. Для кожної вимоги обчислюється косинусна схожість із усіма методами, формуються топ-k кандидатів за значенням схожості.

4.Ранжування результатів. Отримані кандидати упорядковуються за релевантністю, після чого застосовується поріг для формування трасувальних зв'язків.

5. Оцінювання ефективності. Порівняння виявлених зв'язків із еталонними проводиться за метриками Precision, Recall, F1-score, MAP та MRR.

На рисунку 1 показана схема методу трасування.

Рисунок 1 – Схему методу трасування вимог

Експериментальне дослідження проводилося на наборі MSR-2021 (з основним проєктом GanttProject) з використанням метрик Precision, Recall, F1-score, MAP та MRR.

Першочерговим завданням було визначити оптимальну конфігурацію методу на основі CodeBERT. Оскільки модель CodeBERT дозволяє враховувати структурні елементи коду, було досліджено вплив структурного контексту (додавання назви класу та тіла методу до аналізу) на якість трасування.

У таблиці 1 наведено порівняльні результати для конфігурації CodeBERT з урахуванням повного контексту (className + sourceCode) та без нього (fullmethod)

Таблиця 1 – Вплив урахування структурного контексту на якість трасування

Конфігурація CodeBERT	**Precision**	**Recall**	**F1-score**	**MAP**	**MRR**	**Top-K Recall**
Без контексту (fullmethod)	0.74	0.0185	0.0363	0.0159	1.00	1.00
З урахуванням className + sourceCode	0.85	0.0237	0.0460	0.0212	1.00	1.00

Як видно з таблиці 1, додавання структурної інформації забезпечує суттєвий приріст ефективності. Найбільш показовим є стрибок точності з 0.74 до 0.85. Показник F1-score, що балансує точність і повноту, також демонструє зростання (з 0.0363 до 0.0460). Це пояснюється тим, що модель отримує доступ до повного семантичного змісту коду, а не лише до його назв, що дозволяє їй генерувати надійніші трасувальні зв'язки.

Оскільки конфігурація з урахуванням контексту показала себе як значно ефективніша, саме вона була використана для подальшого порівняння з іншими базовими методами трасування: TF-IDF та SBERT. Для цього порівняння було зафіксовано параметр K=100. Порівняльні результати роботи моделей наведено в таблиці 2.

Таблиця 2 – Порівняльні результати моделей TF-IDF, SBERT і CodeBERT

Модель	**Precision**	**Recall**	**F1-score**	**MAP**	**MRR**	**Top-K Recall**
TF-IDF	0.78	0.0217	0.0422	0.0168	1.00	1.00
SBERT	0.79	0.0220	0.0428	0.0188	1.00	1.00
CodeBERT	0.85	0.0237	0.0460	0.0212	1.00	1.00

Як видно із таблиці 1, усі три моделі демонструють порівняно низький F1-score (в діапазоні 0.042–0.046). Це пояснюється вкрай низькими показниками

повноти (Recall ~0.022–0.024), що є характерним для налаштувань з високим пріоритетом точності.

Ключовим та найбільш показовим результатом з цього порівняння є суттєве зростання точності. Модель CodeBERT досягає Precision = 0.85, що значно перевершує TF-IDF (0.78) та SBERT (0.79).

Важливо, що це значне підвищення надійності знайдених зв'язків досягається без погіршення повноти, яка залишається на стабільному, хоча й низькому рівні для всіх моделей. Це свідчить про те, що CodeBERT здатний краще враховувати семантичний контекст і структурні зв'язки, генеруючи більш релевантні (точні) трасувальні зв'язки, ніж лексичні чи стандартні семантичні моделі.

Оскільки було доведено ефективність використання розширеного представлення коду, наступним важливим елементом аналізу став параметр Top-K. Він визначає кількість кандидатів у списку знайдених зв'язків і безпосередньо впливає на повноту трасування – тобто на те, наскільки велика частка правильних зв'язків буде знайдена серед перших K варіантів.

Залежність якості трасування від значення K наведено в таблиці 3.

Таблиця 3 – Залежність метрик від параметра Top-K

K	Precision	Recall	F1-score	MAP	MRR	Top-K Recall
10	1.0000	0.0028	0.0055	0.0028	1.0000	1.0000
20	0.9000	0.0050	0.0100	0.0050	1.0000	1.0000
50	0.8600	0.0120	0.0236	0.0110	1.0000	1.0000
100	0.8500	0.0237	0.0460	0.0212	1.0000	1.0000
200	0.7950	0.0442	0.0838	0.0381	1.0000	1.0000
500	0.7700	0.1071	0.1881	0.0871	1.0000	1.0000
1000	0.7180	0.1998	0.3126	0.1564	1.0000	1.0000
2000	0.7110	0.3957	0.5084	0.2957	1.0000	1.0000

Як видно з таблиці 3, параметр K має вирішальний вплив на компроміс між точністю та повнотою. Зі зростанням K від 10 до 2000 спостерігається чітка тенденція: показник Повноти зростає зі слідового значення 0.0028 до суттєвого 0.3957. Водночас Точність поступово знижується з 1.0 до 0.711, оскільки у ширшій вибірці неминуче з'являється більше хибних зв'язків.

Завдяки такому домінуючому зростанню повноти, збалансований F1-score також демонструє значне покращення, піднімаючись з 0.0055 до 0.5084. Це підтверджує, що зі збільшенням кількості кандидатів (тобто "глибини пошуку") модель знаходить значно більше релевантних зв'язків.

Узагальнюючи наведені результати, можна стверджувати, що запропонований метод трасування вимог до програмного коду на основі великих

мовних моделей суттєво перевершує традиційні підходи. Модель CodeBERT забезпечує найвищі показники точності (0.85) та збалансованості метрик (F1 = 0.046 при K = 100, F1 = 0.5084 при K = 2000). Врахування структурного контексту підвищує семантичну точність, а оптимальне значення Top-K = 500–1000 забезпечує раціональний баланс між точністю й повнотою.

Отримані результати підтверджують ефективність запропонованого підходу для автоматизованого трасування вимог у масштабних програмних системах.

Перелік посилань

1. Mucha J., Kaufmann A., Riehle D. A systematic literature review of pre-requirements specification traceability. Requirements Engineering. 2024. Vol. 29, № 1. P. 1–25. - DOI: 10.1007/s00766-023-00412-z.
2. Horber D., Schleich B., Wartzack S. Procedure Model for Structured Relational Modeling of Requirements to Support Requirements-Oriented Decision Making. Proceedings of the International Conference on Engineering Design (ICED21). 2021. P. 1–10. - DOI: 10.1017/pds.2021.505.
3. Rodriguez A. D., Dearstyne K. R., Cleland-Huang J. Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability. arXiv. 2023. arXiv:2308.00229. - URL: https://arxiv.org/abs/2308.00229.
4. Rodriguez A. D., Dearstyne K. R., Cleland-Huang J. Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability. arXiv. 2023. arXiv:2308.00229. - URL: https://arxiv.org/abs/2308.00229.
5. Gao H., Kuang H., Assunção W. K. G., Mayr-Dorn C., Rong G., Zhang H., Ma X., Egyed A. TRIAD: Automated Traceability Recovery based on Biterm-enhanced Deduction of Transitive Links among Artifacts. Proceedings of the 46th International Conference on Software Engineering (ICSE 2024). 2024. P. 1–13. - DOI: 10.1145/3597503.3639164.
6. Hammoudi M., Mayr-Dorn C., Mashkoor A., Egyed A. A Traceability Dataset for Open Source Systems. 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). 2021. P. 555–559. - DOI: 10.1109/MSR52588.2021.00073.
7. Ali S. J., Naganathan V., Bork D. Establishing Traceability between Natural Language Requirements and Software Artifacts by Combining RAG and LLMs. Proceedings of the 26th International Conference on Model-Driven Engineering Languages and Systems (MODELS '24). 2024. P. 1–16.
8. Borg M. Code Gradients: Towards Automated Traceability of LLM-Generated Code to Requirements Using Explainable AI. 2024 IEEE/ACM 46th International Conference on Software Engineering Companion (ICSE-Companion). 2024. P. 261–263. - DOI: 10.1109/ICSE-Companion58145.2024.00070.

Додаток Б

Програмний код

Програмний код створеної програмної реалізації методу доступний у репозиторії GitHub: git clone https://github.com/Skrypnyuk-Oleksandr/requirements-tracing-llm.git. На рисунку Б.1 подано хнімок екрану репозиторію на GitHub.

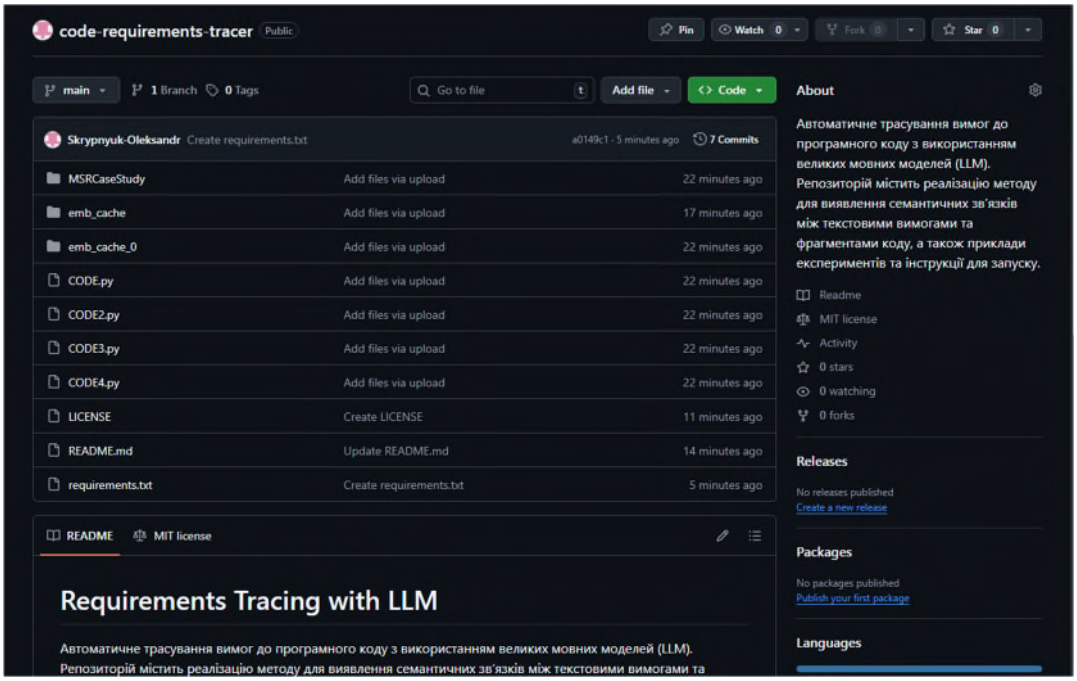


Рисунок Б.1 – Світлина головної сторінки репозиторію GitHub

Репозиторій містить наступні компоненти:

– CODE1.py - CODE4.py: модулі реалізації основних алгоритмів трасування вимог до коду.

– MSRCaseStudy/: папка з кейс-дослідженням та прикладами наборів даних для тестування запропонованого методу.

– emb_cache/ та emb_cache_0/: кешовані ембедінги тексту та коду, що забезпечують швидке повторне виконання без повторного обчислення векторів.

– README.md: файл із описом репозиторію, інструкціями щодо встановлення та запуску, а також прикладами використання модулів.

– requirements.txt: перелік бібліотек Python, необхідних для роботи програми.

– LICENSE: ліцензія MIT, яка дозволяє вільне використання та поширення коду з дотриманням авторства.

Додаток В

Презентаційний матеріал

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Метод виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей

Виконав:
студент 2 курсу, групи КНм-24-1
Скрипнюк Олександр Юрійович

Керівник:
к.т.н., доцент кафедри КН
Багрій Руслан Олександрович

Актуальність

- Трасування вимог є одним із ключових елементів життєвого циклу розробки програмного забезпечення, оскільки забезпечує контроль за відповідністю реалізованого функціоналу сформульованим вимогам, підтримку змін, а також виявлення впливу модифікацій на систему. У сучасних програмних проєктах, що характеризуються великою кількістю вимог та значним обсягом програмного коду, ручне виконання трасування стає надзвичайно трудомістким, затратним за часом і схильним до помилок.

- Традиційні методи інформаційного пошуку, такі як TF-IDF, LSI чи VSM, мають обмежену ефективність через неможливість повного урахування семантичного контексту між природномовним описом вимог і синтаксичною структурою програмного коду. Методи машинного навчання попереднього покоління дещо покращили якість порівняння артефактів, проте залишилися недостатньо глибокими у розумінні контекстних зв'язків.

- Суттєвий прогрес у розв'язанні цієї проблеми став можливим завдяки розвитку трансформерних архітектур та появі великих мовних моделей, таких як CodeBERT, GraphCodeBERT та UniXcoder, які здатні формувати узгоджене семантичне представлення тексту та коду. Їх застосування у процесі трасування вимог відкриває можливість значного підвищення точності й повноти встановлення зв'язків, що робить дослідження у цій сфері актуальним і науково значущим.

Мета і завдання роботи

Метою кваліфікаційної роботи магістра є підвищення точності та повноти автоматичного виявлення трасувальних зв'язків між текстовими вимогами та методами програмного коду шляхом розробки методу, що базується на використанні великих мовних моделей.

Для досягнення поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз існуючих підходів для автоматизації трасування зв'язків між вимогами та програмним кодом;
2. Розробити метод для автоматичного виявлення трасувальних зв'язків на основі векторного представлення вимог та коду з використанням великих мовних моделей;
3. Обґрунтувати вибір та виконати попередню обробку набору даних для навчання моделі, забезпечивши коректність структури, очищення та підготовку артефактів для подальших експериментів;
4. Провести експериментальну перевірку запропонованого методу за відомими статистичними показниками та порівняти отримані результати з відомими підходами.

Схема запропонованого методу виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей

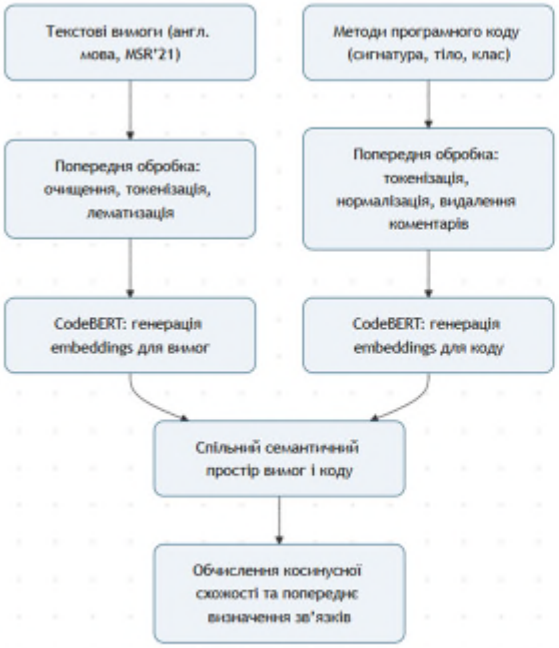


Схема етапу побудови векторних представлень вимог та методів коду

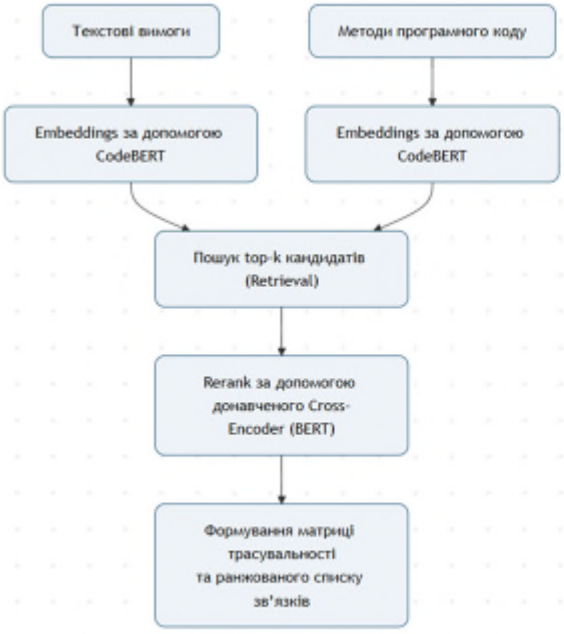
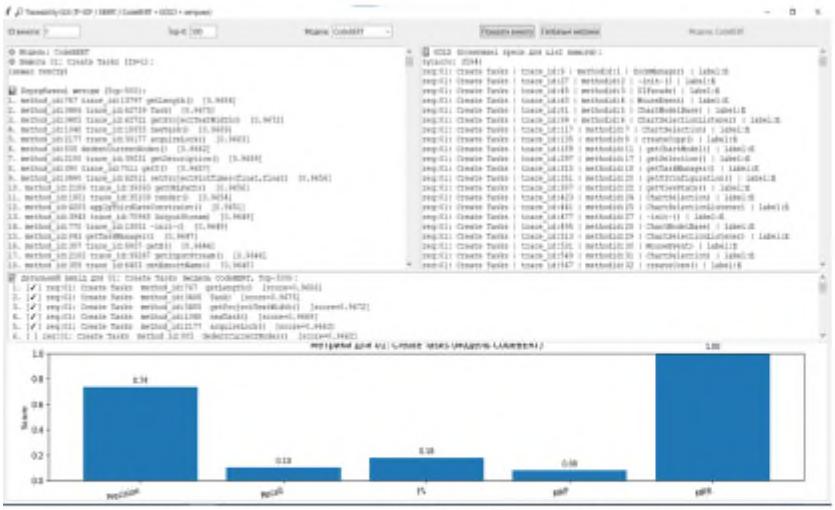


Схема архітектури методу трасування: retrieval → rerank за допомогою LLM

Метрики оцінювання роботи методу

Метрика	Формула	Пояснення
Precision	$\frac{TP}{TP+FP}$	Частка правильних позитивних прогнозів серед усіх передбачених як позитивні.
Recall	$\frac{TP}{TP+FN}$	Частка знайдених істинних позитивних прикладів серед усіх існуючих.
F1-score	$2 * \frac{Precision \cdot Recall}{Precision + Recall}$	Гармонійне середнє між Precision та Recall.
MAP	$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{m_1}\sum_{k=1}^{m_i}P_i(k)$	Середня точність на релевантних позиціях у списку кандидатів.
MRR	$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{rank_i}$	Середнє обернене значення рангу першого релевантного результату.



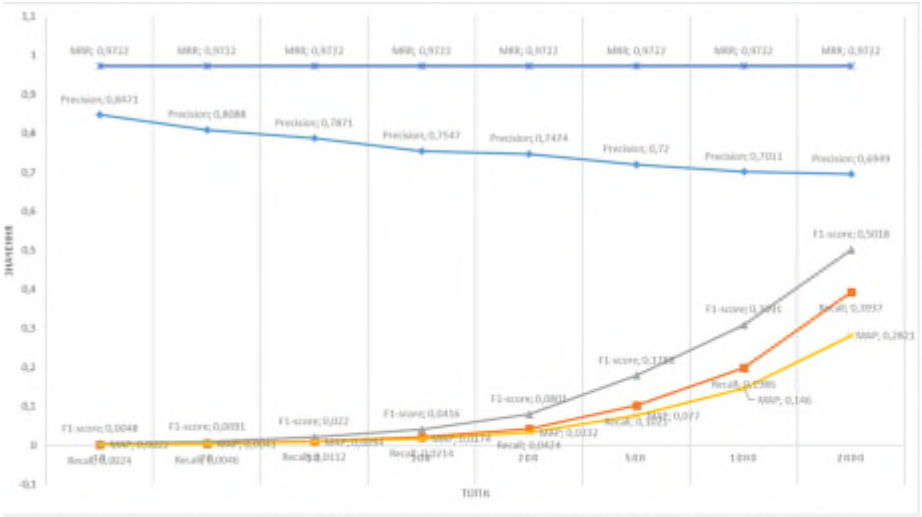
Інтерфейс забезпечує наочне представлення роботи запропонованого методу. Користувач може швидко оцінити якість виявлення трасувальних зв'язків для кожної вимоги, виявити найрезультативніші моделі та здійснити подальший аналіз у рамках дослідження.

Графічний інтерфейс користувача розробленої системи трасування вимог до програмного коду

Порівняння ефективності моделей трасування вимог: TF-IDF, SBERT та CodeBERT за метриками

Порівняльний аналіз трьох моделей демонструє закономірне зростання ефективності від простих статистичних методів до контекстних нейронних моделей:

- TF-IDF забезпечує базову лексичну подібність, але не враховує смислові та структурні зв'язки;
- SBERT суттєво покращує розуміння семантики текстів, однак обмежена у роботі зі структурою коду;
- CodeBERT інтегрує обидва підходи, досягаючи найкращого балансу між точністю, повнотою та узагальненням результатів.



Динаміка зміни основних метрик Precision, Recall, F1-score, MAP та MRR залежно від параметра Top-K

Графік демонструє характерний компроміс між точністю та повнотою: Precision поступово знижується зі збільшенням K, тоді як Recall, F1-score та MAP стабільно зростають, досягаючи плато при K = 500-1000. Така поведінка чітко ілюструє закономірності, властиві більшості систем інформаційного пошуку та рекомендаційних систем, де баланс між точністю та повнотою є критичним.

Висновки

- У магістерській роботі досягнуто поставленої мети — підвищено точність і повноту трасування вимог до програмного коду шляхом розробки методу автоматичного виявлення зв'язків із використанням великих мовних моделей. Запропонований підхід поєднує трансформерні архітектури з семантичним пошуком і забезпечує автоматизацію критичного етапу інженерії вимог, зменшуючи ручну працю та ймовірність помилок.
- Створено програмну реалізацію з модульною архітектурою та інтерактивним інтерфейсом: підготовка даних, генерація векторів, побудова індексу, швидкий пошук, візуалізація результатів у реальному часі та оцінка якості трасування. Система підтримує аналіз локальних метрик, відстеження змін і має відкриту архітектуру для інтеграції з Jira, GitLab і GitHub, що робить її придатною для досліджень і практики.
- Експериментально порівняно підходи та досліджено вплив структурного контексту і параметра Top-K: контекст підвищує точність і стабільність ранжування, а збільшення Top-K покращує повноту, дозволяючи гнучко налаштовувати баланс під завдання проекту. Практична цінність полягає у значному скороченні ручної праці під час формування матриці трасованості, підтримці верифікації вимог, прискоренні рев'ю та виявленні пропусків і дублювань. Основний науковий результат — удосконалення семантичного трасування через поєднання двоетапного пошуку та структурного контексту, що забезпечує істотне підвищення точності й повноти.
- За темою підготовлено дві наукові публікації та проведено апробацію на XVII Всеукраїнській наук.-практ. конференції «АПКН–2025» (Хмельницький, 14–15 листопада 2025 р.).Публікації:
  1. Скрипнюк О.Ю., Манзюк Е.А., Багрій Р.О., Петровський С.С. Метод виявлення трасувальних зв'язків між вимогами та програмним кодом із використанням великих мовних моделей. АПКН-2025, с. 380–384.
  2. Скрипнюк О.Ю., Багрій Р.О., Манзюк Е.А., Петровський С.С. Автоматизоване відновлення трасувальних зв'язків... Вісник ХНУ. Технічні науки. 2026. Т. 361, № 1.Поставлена мета повністю досягнута: створено ефективний метод і програмний прототип для автоматизованого, об'єктивного трасування вимог до коду з високою науковою та практичною цінністю.

Дякую за увагу!

Скрипнюк Олександр, КРМ, 2025

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 10%**

	Document		Sum coincidence on the DB	
ID: 252362 Title: КВАЛІФІКАЦІЙНА РОБОТА на тему Метод виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей Added in a DB: 2025-12-10 Authors: Олександр СКРИПНЮК Heads: Руслан БАГРІЙ Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	107097	1601	2173 (2%)	33 (2%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Олександр СКРИПНЮК

Співавтор:

Назва: КВАЛІФІКАЦІЙНА РОБОТА на тему Метод виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей

Науковий керівник: Руслан БАГРІЙ, к.т.н., доцент

Підрозділ: Кафедра комп'ютерних наук

Коефіцієнт подібності 1:2.3%

Коефіцієнт подібності 2:0.7%

Мікропробіли: 0

Заміна букв: 5

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2025-12-11 11:06:35.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-12-11

Дата

експерт Недзьвецький С. Р.

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНИХ НАУК

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей

Автор: Скрипнюк Олександр Юрійович

Спеціальність: 122 – Комп'ютерні науки

Освітня програма: освітньо-професійна

Науковий керівник: канд. техн. наук, доцент, Багрій Руслан Олександрович

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) за програмою Anti-Plagiarism виявлені 1,0 %, схожість виявлена зі звітом автора з науково-дослідної практики.

2) за програмою StrikePlagiarism КП1 2,3%

які містить матеріали огляду предметної області; інші схожості є фрагментарними – містять поширені конструкції, загальновідомі терміни, скорочення та визначення, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи. Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається

Керівник роботи Руслан БАГРІЙ

Гарант ОП Руслан БАГРІЙ

Завідувач кафедри КН Олександр БАРМАК

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МОН УКРАЇНИ
кафедра комп'ютерних наук

ВІДГУК НАУКОВОГО КЕРІВНИКА

на кваліфікаційну роботу магістра

гр. КНм-24-1 Олександра Скрипнюка за темою: «Метод виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей»

1. Актуальність теми

Актуальність теми кваліфікаційної роботи зумовлена складністю автоматизації управління вимогами за умов відсутності достатньої кількості розмічених даних для встановлення трасувальних зв'язків між текстовими вимогами та програмним кодом. Існуючі методи навчання з учителем потребують значних витрат часу експертів на ручне формування відповідностей, що є критичним бар'єром для нових ІТ-проєктів.

Запропонований метод виявлення трасувальних зв'язків, що базується на використанні векторних представлень і великих мовних моделей, забезпечує автоматичний аналіз семантичної подібності між вимогами та елементами програмного коду без необхідності створення великих розмічених вибірок, підвищуючи точність і повноту трасування та ефективність розробки на ранніх етапах.

2. Відповідність роботи предметній області спеціальності 122 Комп'ютерні науки та загальним вимогам до наукових робіт

Тема роботи повністю відповідає предметній області спеціальності 122 – Комп'ютерні науки та вимогам кваліфікаційної роботи. Об'єктом дослідження є процес автоматизованого виявлення трасувальних зв'язків між вимогами та програмним кодом у системах управління вимогами, а предметом – методи аналізу векторних представлень текстових вимог і коду з використанням великих мовних моделей.

3. Професійні та особистісні якості магістранта

Олександр СКРИПНЮК у процесі роботи продемонстрував високий рівень знань і навичок у галузі комп'ютерних наук, зокрема в сфері обробки природної мови та машинного навчання. Студент відзначився відповідальністю, цілеспрямованістю та здатністю самостійно вирішувати складні науково-технічні задачі.

4. Ступінь самостійності під час виконання кваліфікаційної роботи

Робота виконана самостійно, академічного плагіату не виявлено, стосовно всіх запозичень наведено відповідні посилання на джерела. Програмна реалізація та експериментальні дослідження виконані автором особисто.

5. Наукова новизна та оригінальність запропонованих підходів

5. Наукова новизна та оригінальність запропонованих підходів

Удосконалено метод виявлення трасувальних зв'язків між програмними вимогами та кодом шляхом використання доменно-адаптованих великих мовних моделей і аналізу векторних представлень текстових вимог та елементів коду, що відрізняється від існуючих підходів відсутністю навчального шару класифікатора та можливістю інтерпретації результатів на основі семантичної подібності. Це дозволило підвищити точність і повноту автоматичного трасування порівняно з традиційними підходами та зменшити помилки.

Отримані результати оприлюднені на науково-практичних конференціях.

6. Ступінь оволодіння методами дослідження

Студент продемонстрував впевнене володіння сучасними методами дослідження, зокрема аналізом векторних представлень текстових вимог і програмного коду та оцінюванням семантичної подібності з використанням великих мовних моделей. У роботі показано розуміння принципів доменної адаптації мовних моделей та їх застосування для автоматизованого виявлення трасувальних зв'язків.

7. Повнота та якість розкриття теми роботи

Мета роботи повністю розкрита. Отримані результати підтверджують наукову обґрунтованість положень, а також досягнення всіх поставлених завдань. Проведено обширний порівняльний аналіз ефективності різних алгоритмів.

8. Логічність, послідовність, аргументованість, літературна грамотність викладу матеріалу

Робота відзначається логічною структурою, послідовністю викладу та високим рівнем аргументованості. Використаний стиль відповідає сучасним стандартам наукового письма, що забезпечує легкість сприйняття матеріалу.

9. Можливість практичного застосування кваліфікаційної роботи, окремих її частин

Розроблений метод має значний потенціал для практичного застосування в інструментальних засобах управління вимогами та системах відстеження помилок. Його впровадження дозволяє автоматизувати встановлення трасувальних зв'язків між програмними вимогами та кодом, знизити навантаження на бізнес-аналітиків і розробників, і зменшити кількість помилок, пов'язаних із неповним або некоректним трасуванням вимог.

10. Висновок про можливість допуску кваліфікаційної роботи до захисту, на яку оцінку заслуговує робота

Кваліфікаційну роботу магістра Олександра СКРИПНЮКА рекомендовано до захисту. Вважаю, що робота заслуговує на оцінку «відмінно».

Науковий керівник _____ к.т.н., доц. КН Руслан БАГРІЙ

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МОН УКРАЇНИ

кафедра комп'ютерних наук

ВІДГУК ОПОНЕНТА

на кваліфікаційну роботу магістра

гр. КНм-24-1 Олександра СКРИПНЮКА за темою: *Метод виявлення трасувальних зв'язків між вимогами та програмним кодом з використанням великих мовних моделей*

1. Актуальність обраної теми

Розробка методу виявлення трасувальних зв'язків між програмними вимогами та кодом із використанням великих мовних моделей дозволяє значно покращити та прискорити процес розробки програмного забезпечення. Запропонований метод дає можливість автоматично встановлювати відповідності між вимогами та елементами коду без використання великих розмічених наборів даних. Тому робота, виконана автором, є актуальною та перспективною для підвищення ефективності процесів розробки програмного забезпечення та покращення точності автоматичного трасування.

2. Відповідність роботи предметній області спеціальності 122 Комп'ютерні науки та загальним вимогам до наукових робіт

Тема кваліфікаційної роботи повністю відповідає спеціальності 122 «Комп'ютерні науки». Виконані завдання з виявлення трасувальних зв'язків між програмними вимогами та кодом відповідають сучасним напрямам комп'ютерних наук. Робота відповідає вимогам до наукових досліджень і має практичне значення для підвищення ефективності процесів розробки програмного забезпечення.

3. Повнота розкриття мети та завдань дослідження

В роботі автор повністю розкриває мету дослідження та поставлені в межах теми завдання.

4. Наявність наукової новизни

У кваліфікаційній роботі представлено наукову новизну та інновації, що відповідають спеціальності 122 «Комп'ютерні науки». Запропоновано метод виявлення трасувальних зв'язків між програмними вимогами та кодом із використанням великих мовних моделей, який дозволяє підвищити точність і повноту автоматичного трасування. Отримані результати мають наукове та практичне значення й були оприлюднені на науково-практичній конференції «Актуальні проблеми комп'ютерних наук – 2025» та міжнародній конференції «ExplAI-2025: Advanced AI in Explainability and Ethics for the Sustainable Development Goals».

5. Зміст кожного розділу роботи

Робота складається з чотирьох розділів.

У першому розділі виконано аналіз предметної області та існуючих підходів до виявлення трасувальних зв'язків між вимогами та програмним кодом, зокрема із застосуванням великих мовних моделей, трансформерів та CNN, RNN.

Другий розділ присвячено методові виявлення трасувальних зв'язків. Описано архітектуру методу, етапи обробки даних, підходи до доменної адаптації та використані набори даних.

У третьому розділі наведено практичну реалізацію методу. Описано вибір середовища, технологій, бібліотек та компоненти системи.

У четвертому розділі проведено експериментальне дослідження розробленого методу. Проаналізовано результати роботи методу в умовах різних підходів до доменної адаптації великих мовних моделей..

6. Ступінь розкриття теми роботи

Тема кваліфікаційної роботи повною мірою розкрита та обґрунтована. Проведено детальний аналіз актуальності, сучасних підходів і відомих досліджень у сфері виявлення трасувальних зв'язків між програмними вимогами та кодом із використанням великих мовних моделей. Поставлені завдання виконані в повному обсязі, а результати дослідження проаналізовані та підтверджені тестуванням системи, що демонструє ефективність запропонованого методу.

7. Якість оформлення кваліфікаційної роботи

Оформлення роботи відповідає необхідним вимогам та стандартам, які ставляться до кваліфікаційних робіт. Текст написаний чіткою літературною мовою, а структура роботи відповідає встановленим нормам.

8. Недоліки кваліфікаційної роботи

Разом із тим, у роботі є певні недоліки. Зокрема, не розглянуто можливість застосування запропонованого методу на різноманітних наборах даних з іншими мовами та різними великими мовними моделями.

Проте ці недоліки не впливають на загальну позитивну оцінку роботи.

9. Загальний висновок (допускається чи не допускається до захисту), якої оцінки заслуговує кваліфікаційна робота.

Враховуючи високий рівень виконання роботи, відповідність поставленим завданням та вимогам, кваліфікаційна робота може бути допущена до захисту.

Рекомендована оцінка – «Відмінно».

Опонент (прізвище, ім'я, по батькові, посада, місце роботи)

Платюк Олександра Тетяніївна, д.т.н., професор, професор кафедри КІС

«19» 12 2025 р.

Підпис