Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА

на тему *Метод виявлення неоднозначностей у вимогах до програмного забезпечення 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* _____ *Богдан ВОНСОВИЧ*
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 неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей»*

2. Завдання видано студенту Богдану ВОНСОВИЧУ
(Ім'я, ПРІЗВИЩЕ)

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

4. Затверджені наказом університету від «*2*» *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 якості процесу виявлення лінгвістичних неоднозначностей у вимогах до програмного забезпечення за рахунок застосування контекстно-орієнтованих агентів на базі LLM. Досягнення поставленої мети передбачає проведення аналізу існуючих методів виявлення неоднозначностей та підходів до їх класифікації, для чого потрібно розробити двоетапний метод, що базується на послідовній взаємодії агента-ідентифікатора для первинної фільтрації вимог та агента-класифікатора для їх семантичної типізації. Наступними кроками є програмна реалізація методу з використанням технології RAG для врахування глосарію та контексту проєкту,

а також проведення експериментального дослідження для оцінки показників якості виявлення неоднозначностей у порівнянні з базовими підходами.

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* *Богдан ВОНСОВИЧ*

Реферат

Кваліфікаційна робота магістра присвячена розробці та дослідженню методу автоматизованого виявлення та класифікації неоднозначностей у текстових вимогах до програмного забезпечення шляхом інтеграції великих мовних моделей (LLM) з технологією доповненої генерації (RAG).

Актуальність теми. Якість програмного забезпечення безпосередньо залежить від якості вимог, сформульованих на ранніх етапах життєвого циклу розробки. Неоднозначність у вимогах є однією з головних причин виникнення дефектів, перевитрат бюджету та зривів термінів проектів. Традиційні методи ручного рецензування є трудомісткими та суб'єктивними, а наявні автоматизовані інструменти на основі правил часто генерують високий рівень хибних спрацьовувань. Використання великих мовних моделей відкриває нові можливості для семантичного аналізу тексту, проте самі по собі моделі схильні до галюцинацій та ігнорування контексту проекту. Створення гібридного методу, що поєднує гнучкість LLM із точністю контекстної бази знань (RAG), є актуальним науково-прикладним завданням.

Мета і завдання роботи. Метою роботи є підвищення якості процесу виявлення лінгвістичних неоднозначностей у вимогах до програмного забезпечення за рахунок застосування контекстно-орієнтованих агентів на базі LLM.

Для досягнення поставленої мети потрібно виконати наступні завдання:

- провести аналіз існуючих методів виявлення неоднозначностей та підходів до їх класифікації;
- розробити двоетапний метод виявлення та класифікації неоднозначностей, який базується на послідовній взаємодії агента-ідентифікатора для первинної фільтрації вимог та агента-класифікатора для їх семантичної типізації;

- виконати програмну реалізацію методу з використанням технології RAG для забезпечення врахування глосарію та контексту конкретного проєкту;
- провести експериментальне дослідження розробленого методу та оцінити показники якості виявлення неоднозначностей порівняно з базовими підходами.

Об’єкт дослідження – процес автоматизованого аналізу текстових вимог до програмного забезпечення.

Предмет дослідження – методи, моделі та програмні засоби автоматизованого виявлення лінгвістичних неоднозначностей у природно-мовних текстах вимог із застосуванням великих мовних моделей та інженерії промптів.

Методи дослідження. У роботі використано методи системного аналізу для побудови архітектури рішення; методи обробки природної мови для токенізації та сегментації тексту; методи машинного навчання (зокрема, використання попередньо навчених моделей); технологію Retrieval-Augmented Generation для роботи з базою знань; методи інженерії промптів (Chain-of-Thought) для керування виводом моделі; емпіричні методи для оцінки якості класифікації.

Наукова новизна одержаних результатів:

Удосконалено метод автоматизованого виявлення неоднозначностей у вимогах до програмного забезпечення, який, на відміну від існуючих методів прямого використання LLM, базується на гібридній архітектурі з інтеграцією технології пошукового доповнення генерації (RAG), що дозволило здійснити динамічну верифікацію термінів через контекстну базу знань проєкту та зменшити кількість хибних спрацювань на вузькоспеціалізованій технічній лексиці.

Апробація результатів кваліфікаційної роботи 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. Вонсович Б. А., Багрій Р. О., Пасічник О.А., Скрипник Т. К. Метод виявлення неоднозначностей у вимогах до програмного забезпечення з 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. 75–78.

2. Вонсович Б. А., Багрій Р. О., Скрипник Т. К., Пасічник О.А. Автоматизоване виявлення та класифікація неоднозначностей у вимогах до програмного забезпечення ІТ-проєктів з використанням великих мовних моделей та RAG-технологій. 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 52 найменувань та 3 додатків. Загальний обсяг кваліфікаційної роботи складає 98 сторінок, з поміж яких 81 сторінка основного тексту та 24 сторінки додатків. У роботі наведено 19 рисунків та 2 таблиці.

Ключові слова: вимоги до програмного забезпечення, неоднозначність, велика мовна модель, RAG, IEEE 830, NLP, prompt engineering, автоматизована перевірка.

Зміст

Перелік скорочень

Скорочення, термін, позначення	Пояснення
CoT	Chain-of-Thought
RAG	Retrieval-Augmented Generation
ПЗ	Програмне забезпечення
API	Application Programming Interface
LLM	Large Language Model
IEEE	Institute of Electrical and Electronics Engineers
T5	Text-to-Text Transfer Transformer
CNN	Convolutional Neural Network
JSON	JavaScript Object Notation
VRAM	Video Random Access Memory
GPU	Graphics Processing Unit

Вступ

Актуальність теми. Якість програмного забезпечення безпосередньо залежить від якості вимог, сформульованих на ранніх етапах життєвого циклу розробки. Неоднозначність у вимогах є однією з головних причин виникнення дефектів, перевитрат бюджету та зривів термінів проектів. Традиційні методи ручного рецензування є трудомісткими та суб'єктивними, а наявні автоматизовані інструменти на основі правил часто генерують високий рівень хибних спрацьовувань. Використання великих мовних моделей відкриває нові можливості для семантичного аналізу тексту, проте самі по собі моделі схильні до галюцинацій та ігнорування контексту проекту. Створення гібридного методу, що поєднує гнучкість LLM із точністю контекстної бази знань (RAG), є актуальним науково-прикладним завданням.

Мета і завдання дослідження. Метою роботи є підвищення якості процесу виявлення лінгвістичних неоднозначностей у вимогах до програмного забезпечення за рахунок застосування контекстно-орієнтованих агентів на базі LLM.

Для досягнення поставленої мети потрібно виконати наступні завдання:

- провести аналіз існуючих методів виявлення неоднозначностей та підходів до їх класифікації;
- розробити двоетапний метод виявлення та класифікації неоднозначностей, який базується на послідовній взаємодії агента-ідентифікатора для первинної фільтрації вимог та агента-класифікатора для їх семантичної типізації;
- виконати програмну реалізацію методу з використанням технології RAG для забезпечення врахування глосарію та контексту конкретного проєкту;
- провести експериментальне дослідження розробленого методу та оцінити показники якості виявлення неоднозначностей порівняно з базовими підходами.

Об’єкт дослідження – процес автоматизованого аналізу текстових вимог до програмного забезпечення.

Предмет дослідження – методи, моделі та програмні засоби автоматизованого виявлення лінгвістичних неоднозначностей у природно-мовних текстах вимог із застосуванням великих мовних моделей та інженерії промптів.

Методи дослідження. У роботі використано методи системного аналізу для побудови архітектури рішення; методи обробки природної мови для токенізації та сегментації тексту; методи машинного навчання (зокрема, використання попередньо навчених моделей); технологію Retrieval-Augmented Generation для роботи з базою знань; методи інженерії промптів (Chain-of-Thought) для керування виводом моделі; емпіричні методи для оцінки якості класифікації.

Наукова новизна одержаних результатів. Удосконалено метод автоматизованого виявлення неоднозначностей у вимогах до програмного забезпечення, який, на відміну від існуючих методів прямого використання LLM, базується на гібридній архітектурі з інтеграцією технології пошукового доповнення генерації (RAG), що дозволило здійснити динамічну верифікацію термінів через контекстну базу знань проєкту та зменшити кількість хибних спрацювань на вузькоспеціалізованій технічній лексиці.

Апробація результатів кваліфікаційної роботи 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. Вонсович Б. А., Багрій Р. О., Пасічник О.А., Скрипник Т. К. Метод виявлення неоднозначностей у вимогах до програмного забезпечення з 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. 75–78.

2. Вонсович Б. А., Багрій Р. О., Скрипник Т. К., Пасічник О.А. Автоматизоване виявлення та класифікація неоднозначностей у вимогах до програмного забезпечення ІТ-проєктів з використанням великих мовних моделей та RAG-технологій. 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 52 найменувань та 3 додатків. Загальний обсяг кваліфікаційної роботи складає 98 сторінок, з поміж яких 81 сторінка основного тексту та 24 сторінки додатків. У роботі наведено 19 рисунків та 2 таблиці.

РОЗДІЛ 1. Аналіз сучасного стану досліджуваної проблеми

1.1 Аналіз проблеми неоднозначності в інженерії вимог до програмного забезпечення

Процес створення програмних продуктів незмінно починається з виявлення, опрацювання та фіксації вимог, ця фаза формує підґрунтя для ключових параметрів майбутньої системи: її функціональних можливостей, швидкодії, зручності, захищеності та здатності до інтеграції. Вимоги слугують універсальним комунікатором між усіма стейкхолдерами – від клієнтів і аналітиків до розробників і юзерів, тому їхній якісний рівень прямо впливає на успіх всього проєкту [1, 2].

Конкретика та прозорість завдань є вирішальними для забезпечення якості софту. Розмиті або двозначні формулювання створюють ризик різного розуміння цілей учасниками, що призводить до прорахунків під час проєктування, написання програмного коду та перевірки, результаті чого зростає число дефектів, збільшуються часові й фінансові витрати на виправлення, а також підвищується ймовірність того, що фінальне рішення не задовольнить очікування замовника [3, 4].

Аналітичні дані у сфері інженерії вимог підтверджують, що від 40 до 60% усіх програмних збоїв зароджуються саме під час фіксації завдань через комунікаційні розриви між клієнтом та розробником (рис. 1.1) [5, 6]. Фінансові витрати на усунення таких недоліків на завершальних стадіях проєкту зростають у десятки разів порівняно з ціною їх виправлення на момент виникнення [7, 8]. Оперативна ідентифікація неточностей, логічних конфліктів чи двозначностей у документації є обов’язковою передумовою для гарантування надійності та прогнозованості процесу розробки.

Досягнення чіткості та однозначності вимог виходить за межі суто аналітичної роботи і стає основою системи управління якістю програмного продукту. Впровадження автоматизованих інструментів для детестування семантичних розбіжностей, що базуються на передових методах обробки природної мови та можливостях великих мовних моделей, створює нові перспективи для мінімізації

проєктних ризиків та суттєвого підвищення ефективності створення програмних систем [9, 10].

Неоднозначність (англ. ambiguity) у технічній документації характеризує стан, коли зафіксовані вимоги отримують відмінні, а іноді й суперечливі трактування різними учасниками життєвого циклу – замовниками, бізнес-аналітиками, інженерами чи спеціалістами з тестування [11, 12]. Основна причина неоднозначностей криється у специфіці природної мови, яка використовується для опису і є за своєю суттю багатозначною, це проявляється через полісемію слів (лексична неоднозначність), особливості побудови речень, що дозволяють різні граматичні зв'язки (синтаксична або структурна неоднозначність), або ж через недостатню визначеність контексту, що ускладнює розуміння дійсного наміру автора (семантична чи прагматична неоднозначність) [13, 14], розглянемо на прикладі:

– Вимога «Усі користувачі вводять код доступу» – чи мається на увазі, що всі користувачі вводять один й той же код, чи що кожен має свій код? (Scope-неоднозначність) [15].

– Фрази типу «висока продуктивність», «користувач-дружній інтерфейс», «незначна затримка» – без кількісних показників, без чіткого контексту – залишають простір для різних тлумачень [16].

Виявлення неоднозначностей – це не просто редакторська чи стилістична справа, це фундаментальна проблема для якості ПЗ

Коли різні учасники сприймають вимогу по-різному, розробник може реалізувати одну інтерпретацію, а замовник чи тестувальник очікує іншу. Це призводить до логічних чи функціональних помилок, які часто виявляються лише під час тестування або після релізу, коли виправлення складніше й затратніше [17].

Розмитість формулювань неминуче породжує потребу в додаткових узгодженнях на етапах імплементації, перегляду архітектури чи тестування. Особливо критично, коли подвійне тлумачення виявляється вже під час валідації або безпосередньо перед запуском продукту, що тягне за собою суттєву переробку модулів системи, це змушує команду призупиняти поточні процеси заради

модифікації вже створених компонентів, провокуючи розростання обсягу завдань та вихід за рамки затверджених часових і фінансових лімітів [18].

Коли вимоги прописані розмито або допускають варіативність розуміння, на етапі здачі проєкту часто виявляється суттєвий розрив між очікуваннями клієнта та фактичним результатом. У площині договірних відносин це стає причиною конфліктів, претензій щодо безкоштовних доопрацювань або навіть фінансових компенсацій, адже абстрактні зобов'язання вкрай важко верифікувати чи захистити під час юридичних або комерційних суперечок.

Для фахівців із забезпечення якості відсутність конкретики та вимірюваних характеристик значно ускладнює формування коректних сценаріїв тестування та критеріїв приймання. За відсутності чітких метрик і однозначних умов готовності процес перевірки ризикує стати суб’єктивним, що послаблює контроль якості та створює небезпеку пропуску критичних дефектів у готовий продукт [19].

За наявності двозначностей у вимогах, процеси масштабування або модернізації системи в майбутньому неминуче супроводжуються високими ризиками, це пов'язано з тим, що новий функціонал часто будується на базі старих компонентів або прихованих припущень, які виникли внаслідок хибних трактувань на старті. Можуть виникнути створює серйозні перешкоди для проведення якісного аналізу впливу змін, руйнує наскрізне трасування вимог та загрожує логічній цілісності архітектури. Зрештою, невирішені неоднозначності стають каталізатором накопичення технічного боргу, який з часом стає дедалі важче обслуговувати.

	Погана вимога	Прийнятна вимога
Нереалістична	Система ніколи не повинна виходити з ладу.	Середній час між збоями системи становити не менше 20 днів.
Розпливчаста, необ'єктивна	Інтерфейс має бути зручним для користувача.	Створення нового запису менш ніж за 5 хвилин без інструкцій.
Описує операцію, а не вимогу	Користувач має натискати кнопку "Зберегти", щоб оновити дані.	Система має автоматично зберігати зміни користувача після підтвердження дії.
Неповна або дискримінаційна	Система має повідомляти користувача, коли він завершив свій запит.	Система має надсилати сповіщення користувачу, коли запит виконано, незалежно від статі чи статусу входу.
Зосереджується на технічній реалізації, а не на результаті	Модуль керування має використовувати шість потоків для обробки даних.	Система має підтримувати паралельну обробку даних з шістьма незалежними потоками.

Рисунок 1.1 – Приклади помилок в вимогах до ПЗ

Нижче на рисунку 1.2 показано типи неоднозначностей вимог, а саме – 4 основні типи неоднозначностей, зазвичай, виділяють такі типи неоднозначностей: лексичні, синтаксичні, семантичні, прагматичні [20].

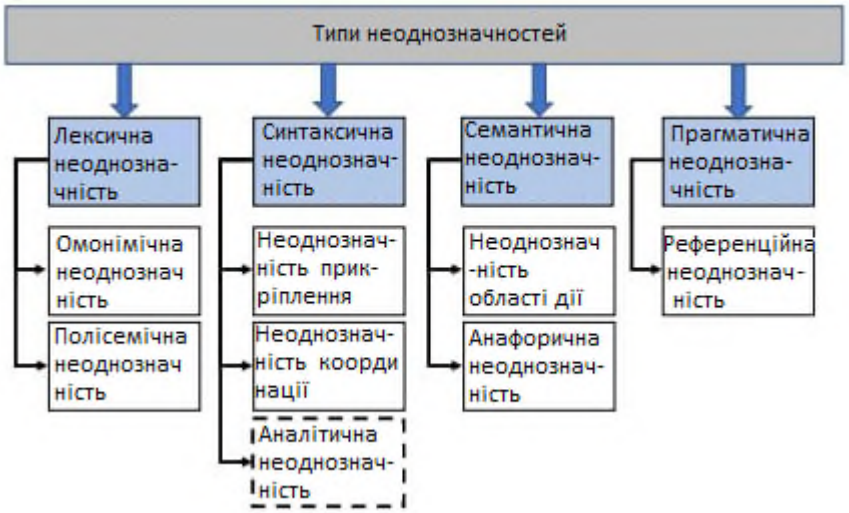


Рисунок 1.2 – Класифікація типів лінгвістичних неоднозначностей у вимогах

Підсумовуючи, можна зробити висновок, що висока якість вимог слугує основою для успішного створення програмного продукту. Проведений аналіз демонструє, що неоднозначність (детальну типологію якої наведено на рис. 1.1 та 1.2) залишається ключовим генератором помилок, котрі на фінальних стадіях проєкту трансформуються у значні фінансові та часові втрати.

Оскільки процес ручної верифікації є надто важким і недостатньо ефективним, виникає гостра потреба у впровадженні автоматизованих засобів контролю, ці обставини актуалізують необхідність переходу до детального огляду наукових джерел та сучасних методологій, зокрема тих, що досліджують потенціал великих мовних моделей (LLM) для автоматичної ідентифікації неточностей у специфікаціях програмного забезпечення.

1.2 Огляд та класифікація існуючих методів виявлення неоднозначностей у вимогах

Задача виявлення неоднозначностей у вимогах до ПЗ зберігає свою гостроту через семантичну складність та багатство формулювань природної мови. Стратегії вирішення завдання виявлення неоднозначностей пройшли значний еволюційний шлях від ручної вичитки та детермінованих алгоритмів на основі правил до ймовірнісних моделей машинного навчання. Сучасний етап розвитку характеризується впровадженням рішень на базі великих мовних моделей (LLM), архітектур із доповненою генерацією через пошук (RAG) та автономних агентів. Вибір оптимальної стратегії для конкретного проєкту вимагає ретельного порівняльного аналізу цих технологій за низкою метрик: ступінь автоматизації, залежність від навчальних даних, глибина розуміння контексту, прозорість результатів та економічна ефективність.

Ручні методи спираються виключно на експертний аналіз – фахівці переглядають документацію, використовуючи спеціалізовані контрольні списки або шаблони, що містять перелік типових ознак нечітких термінів чи двозначних синтаксичних структур. Основна мета перевірок – локалізувати потенційні дефекти ще до початку етапу імплементації, коли вартість їх усунення є мінімальною. Показовим прикладом є дослідження «An Analysis of Ambiguity Detection Techniques for Software Requirements Specification (SRS)», у якому автори аналізують як повністю ручні підходи, так і частково автоматизовані інструменти, пропонуючи класифікацію

методів на мануальні, NLP-орієнтовані та такі, що базуються на машинному навчанні [21].

Синтаксичні підходи – у цих методах використовуються мовні правила, регулярні вирази, граматики, шаблони для виявлення ознак неоднозначності – наприклад слів (“швидко”, “достатньо”), займенників без чітких передумов, конструкцій “і/або”, імперативів тощо. Часто такі методики доповнюють NLP-парсинг, POS-теги, залежності. Наприклад, Робота “Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements” пропонує метод, який автоматично виявляє синтаксичні неоднозначності вимог, з використанням шаблонів / патернів [22]. Стаття “An NLP approach for cross-domain ambiguity detection in requirements engineering” застосовує ембеддинги домен-специфічних мов як частину підходу до виявлення термінів, чия інтерпретація змінюється між доменами, але патерн-орієнтовані фрагменти там присутні [23].

Інструменти типу QuARS (Quality Analyzer for Requirement Specifications) – інструмент для аналізу тексту вимог, виявлення лексичних і синтаксичних дефектів [24].

Використання алгоритмів машинного навчання та архітектур, зокрема нейромереж трансформерів формує окремий потужний клас методів, цей підхід охоплює як класичні моделі (SVM, Decision Trees), так і сучасні нейромережі, як CNN, так і трансформери (BERT, RoBERTa), які тренуються бінарно класифікувати вимоги (чіткі/неоднозначні) або ж категоризувати їх за типом дефекту.

Особливу ефективність демонструють гібридні системи, що комбінують лінгвістичні правила, векторні представлення слів та контекстуальну потужність великих моделей. Як приклад подібної синергії можна навести спеціалізовані інструменти, які інтегрують методи машинного навчання для детекції специфічних проблем, наприклад, анафоричної неоднозначності, посилюючи аналіз можливостями моделей сімейства BERT [25].

Стаття Requirements Ambiguity Detection and Explanation with LLMs: An Industrial Study – це “перехідний” підхід між класичним машинним навчаннм і

використанням LLM – вони використовують in-context демонстрації LLM, але експериментальна частина порівнює із класичними підходами [26].

Сучасні LLM-підходи такі як: zero/few-shot, prompt engineering, fine-tuning, гібриди – це підходи, які використовують великі попередньо навчені мовні моделі типу GPT-серії, LLaMA, Bloom тощо. Існують такі підходи, як zero-shot – без донавчання, та few-shot – з демонстраціями прикладів в промптах, або виконання fine-tuning на специфічному наборі текстів. Також гібридні підходи, що комбінують виялення неоднозначностей методом правил з великими мовними моделями або машинне навчанн з великими мовними моделями. Наприклад: Requirements Ambiguity Detection and Explanation with LLMs: An Industrial Study – показує, що LLM-моделі з використанням few-shot (наприклад 10-shot) можуть суттєво перевищувати нульовий zero-shot за якістю класифікації неоднозначних вимог і давати пояснення, які корисні для експертів. В статті “Lessons from the Use of Natural Language Inference in Requirements Engineering Tasks” – порівнюють NLI-методи з іншими підходами, такими як: трансформери, машинне навчання, та великі мовні моделі у zero-shot або традиційних налаштуваннях.

Робота “Zero and Few-shot Semantic Parsing with Ambiguous Inputs” [27] демонструє актуальність застосування великих мовних моделей (LLM) для обробки неоднозначних вхідних даних, що має прямі паралелі із завданням аналізу вимог до програмного забезпечення. Автори досліджують, як LLM реагують на семантичну невизначеність у вхідному тексті, та оцінюють ефективність підходів zero-shot і few-shot у контексті синтаксичного розбору (semantic parsing) без повноцінного донавчання моделі.

Результати досліджень підтверджують значний потенціал режимів zero-shot та few-shot в умовах дефіциту розмічених даних, ці архітектури демонструють глибоку чутливість до контексту та вміння розрізняти найменші семантичні відтінки природної мови. Використання LLM є ідеальним інструментом для швидкого прототипування систем аналізу вимог – інженерія запитів дозволяє адаптувати модель до специфіки домену, оминаючи ресурсомісткий процес донавчання (fine-

tuning). Додатковою перевагою є здатність окремих LLM генерувати пояснення (rationale) своїх висновків, що підвищує прозорість рішень та довіру з боку інженерів.

Водночас, автори звертають увагу на суттєві обмеження. По-перше, стабільність класифікації у цих режимах є вкрай залежною від конструкції запиту: навіть незначна корекція промпта чи зміна підібраних прикладів може призвести до протилежного результату. По-друге, зберігається високий ризик «галюцинацій» – генерації фактично невірних або вигаданих даних, особливо при роботі з термінологією, відсутньою у тренувальному наборі. По-третє, застосування універсальних LLM вимагає суворої доменної валідації, оскільки моделі загального призначення часто помиляються в інтерпретації вузькоспеціалізованої лексики або специфічної структури документів специфікації вимог.Нижче наведено порівняльну таблицю конкретних методів та інструментів, згаданих у тексті, на основі ключових характеристик.

Таблиця 1.1 – Порівняльна характеристика методів виявлення неоднозначностей у вимогах до програмного забезпечення

Підхід	Принцип	Тип неоднозначності	Залежність від даних	Потреба в правилах	Пояснюваність
Ручний аналіз	Людський аналіз	Загальні, нечіткі терміни	Низька (досвід)	Висока (чеклісти)	Висока (людина)
QuARS	Аналіз тексту за правилами	Лексичні, синтаксичні	Низька	Висока	Середня (правило)
"Score-Based..."	Виявлення за шаблонами	Синтаксичні	Низька / Середня	Висока (шаблони)	Не вказано
"An NLP approach..."	Ембеддинги + патерни	Міждоменна	Висока (корпуси)	Так (гібрид)	Не вказано

TAPHSIR	Гібрид (ML + BERT)	Анафорична (займенники)	Висока (розмітка)	Низька (навчання)	Низька (BERT)
LLM (Few-shot)	Prompting (з прикладами)	Широкий спектр	Низька (приклади)	Низька (промпт)	Висока (Rationale)
LLM (Zero-shot)	Prompting (без прикладів)	Широкий спектр	Дуже низька	Дуже низька	Висока (менш надійна)

Еволюція підходів свідчить про чіткий тренд: від повністю ручних (висока точність експерта, нульова масштабованість) та синтаксичних (висока інтерпретованість, низька семантична повнота) методів до класичного машинного навчання, оскільки вони забезпечують вищу точність, хоча мають високу залежність від даних.

Однією з головних проблем LLM є галюцинації та відсутність знань про специфічний контекст проекту (приватні документи, внутрішні стандарти). Підхід Retrieval-Augmented Generation вирішує цю проблему шляхом поєднання генеративних можливостей моделі з механізмом пошуку по зовнішній базі знань.

У контексті RE, RAG працює наступним чином: при аналізі вимоги система спочатку виконує пошук релевантних документів (попередні специфікації, глосарії, нормативні акти) у векторній базі даних, і лише потім передає цей контекст разом із запитом до LLM. Дослідження показують, що RAG значно зменшує рівень галюцинацій, оскільки змушує модель спиратися на "докази" (retrieved evidence), а не на вигадані факти. Це дозволяє моделі відповідати "Немає інформації", якщо контекст відсутній, замість того, щоб вигадувати відповідь [28].

Сучасні LLM-підходи пропонують найкращий баланс гнучкості та контекстного розуміння, особливо завдяки few-shot режимам, які не вимагають маштабного збору даних. Водночас вони вводять нові виклики, пов'язані з надійністю та чутливістю до налаштувань. Раціональним напрямом подальшого розвитку є гібридизація – інтеграція надійних традиційних правил для виявлення очевидних

синтаксичних дефектів із потужними можливостями LLM для глибокого семантичного аналізу та генерації пояснень.

1.3 Використання великих мовних моделей у задачах виявлення неоднозначностей

Проблема автоматизованого виявлення неоднозначностей у специфікаціях вимог до програмного забезпечення залишається однією з найбільш складних задач комп'ютерної лінгвістики та інженерії вимог [29, 30]. Складність зумовлена самою природою природної мови, якій притаманні полісемія, контекстна залежність та варіативність формулювань [31, 32]. Традиційні методи аналізу, що базуються на лексико-синтаксичних правилах або простих статистичних моделях, демонструють обмежену ефективність, оскільки вони оперують поверхневими структурами тексту і не здатні охопити глибинний семантичний зміст, це явище, відоме як «семантичний розрив», призводить до того, що формально коректні, але змістовно розмиті вимоги залишаються непоміченими, стаючи джерелом критичних помилок на етапах архітектури та розробки [33].

З появою алгоритмів глибокого навчання та, зокрема, архітектур на основі механізму уваги, вирішення задачі аналізу якості вимог зазнала докорінних змін [34, 35]. Сучасний етап розвитку характеризується переходом від детермінованого пошуку ключових слів до використання щільних векторних представлень та великих мовних моделей (LLM), які здатні моделювати складні лінгвістичні залежності [36, 37].

Ключовою технологічною інновацією стало впровадження методів навчання представлень. На відміну від розріджених векторів у методах TF-IDF, де семантична близькість слів ігнорується, нейронні мовні моделі (зокрема сімейства BERT та RoBERTa) трансформують вхідні фрагменти вимог у багатовимірні вектори фіксованої розмірності (embeddings) [38, 39]. У цьому семантичному просторі відстань між векторами відображає ступінь їхньої змістовної подібності [40], це

дозволяє алгоритмам виявляти не лише явні індикатори неоднозначності (наприклад, слова "швидко", "зручно"), але й приховані (імпліцитні) дефекти, де невизначеність виникає через специфічну комбінацію слів або порушення логічної зв'язності контексту [41].

Особливе місце в еволюції методів посідають великі мовні моделі (LLM), такі як GPT-4, LLaMA або T5 [42]. Використання цих моделей дозволило змінити сам підхід до задачі: від бінарної класифікації («чітко/нечітко») до глибокого аналітичного розбору («чому це нечітко?»). Завдяки навчанню на колосальних масивах текстових даних, LLM набули здатності до "розумного міркування" (reasoning) та використання знань про світ, це дозволяє застосовувати такі передові техніки, як:

1. Контекстно-залежний аналіз – LLM здатна аналізувати вимогу не ізольовано, а враховуючи ширший лінгвістичний контекст речення або абзацу. Наприклад, модель може розрізнити, коли слово "може" вказує на необов'язковість функціоналу (що є дефектом), а коли воно є частиною опису фізичної можливості системи (що є допустимим) [43].
2. Zero-shot та Few-shot Learning – сучасні генеративні моделі демонструють високу ефективність у виявленні неоднозначностей без необхідності трудомісткого процесу донавчання на розмічених датасетах, яких у сфері інженерії вимог критично бракує. Використання інженерії промптів дозволяє адаптувати модель до специфіки конкретного стандарту (наприклад, IEEE 830) шляхом надання лише інструкцій та кількох демонстраційних прикладів.
3. Генерація пояснень – на відміну від класичних моделей типу «чорна скринька» (наприклад, SVM або CNN), генеративні трансформери здатні формувати текстове обґрунтування свого рішення (Chain-of-Thought), це критично важливо для практичного застосування, оскільки аналітику потрібно не просто знати, що вимога дефектна, а й розуміти лінгвістичну або логічну причину цієї класифікації [44].

Однак, попри значний потенціал, пряме застосування універсальних LLM для аналізу вимог стикається з проблемою відсутності специфічних знань про предметну область проекту. Модель може ідеально знати англійську мову, але не знати архітектурних обмежень конкретної системи, внутрішнього глосарію компанії або специфіки апаратного забезпечення, для якого пишуться вимоги, це призводить до двох типів помилок: пропуску контекстуальних неоднозначностей (коли вимога суперечить архітектурі, але граматично вірна) та «галюцинацій» (коли модель вигадує неіснуючі проблеми). Саме тому, науковий фокус зміщується в бік гібридних архітектур, що поєднують генеративні можливості LLM із зовнішніми базами знань через механізм RAG.

Вирішення проблеми відсутності проектного контексту стало можливим завдяки впровадженню архітектурного підходу RAG, ця парадигма дозволяє динамічно доповнювати вхідний запит до моделі (промпт) релевантними фактами із зовнішньої бази знань, яка може містити глосарій проекту, опис архітектури або перелік затверджених технічних обмежень. Дослідження показують, що RAG суттєво знижує рівень семантичних галюцинацій, оскільки змушує генеративну модель спиратися на надані докази, а не лише на свої внутрішні ваги. У контексті виявлення неоднозначностей це дозволяє алгоритму коректно класифікувати випадки так званої «контекстуальної невизначеності», коли вимога є лінгвістично бездоганною, але суперечить специфіці доменної області.

Окрім інтеграції знань, критичним аспектом застосування LLM є вибір стратегії логічного виведення. Наукова спільнота сходиться на тому, що пряма класифікація часто є недостатньою для складних аналітичних задач. Натомість, високу ефективність демонструє метод «Ланцюжка думок» (CoT), ця техніка спонукає модель декомпозувати процес прийняття рішення на послідовні кроки: спершу виконати лінгвістичний розбір речення, потім зіставити його з критеріями якості (наприклад, ISO/IEC 29148 або IEEE 830), і лише після цього сформувати фінальний вердикт. Такий підхід не лише підвищує точність класифікації, але й

забезпечує інтерпретованість результатів, надаючи інженерам зрозуміле пояснення причини дефекту.

Використання потужних генеративних моделей для перевірки кожної вимоги у великих специфікаціях породжує проблему обчислювальної ефективності та вартості. Сучасні LLM (наприклад, GPT-4 або Claude 3) є ресурсоємним та повільним процесом, що ускладнює їх інтеграцію в конвеєри безперервної розробки (CI/CD).

Для подолання цього обмеження в сучасних наукових роботах пропонується використання гібридних багаторівневих архітектур – цей підхід, також відомий як «Filter-then-Analyze», передбачає розділення процесу на два етапи:

1. Попередній відсів (Identification/Filtering) – використання швидких, детермінованих методів (наприклад, лексичних словників або легковагових моделей на кшталт DistilBERT) для виявлення всіх потенційно підозрілих фрагментів. Метою цього етапу є максимізація повноти – щоб не пропустити жодного можливого дефекту, навіть ціною хибних спрацювань.
2. Глибокий аналіз – застосування потужних LLM з підтримкою RAG та CoT лише до тих фрагментів, які пройшли первинний фільтр. Це дозволяє зосередити обчислювальні ресурси на складних випадках, забезпечуючи високу точність та детальну аргументацію.

Таким чином, аналіз літературних джерел дозволяє стверджувати, що найбільш перспективним напрямом розвитку систем автоматизованого аналізу вимог є синергія трьох компонентів:

- великих мовних моделей як аналітичного ядра, здатного до розуміння семантики;
- механізму RAG для забезпечення контекстної обізнаності та фактологічної точності;
- конвеєрної архітектури, що оптимізує баланс між продуктивністю системи та глибиною аналізу.

Саме така комбінація технологій дозволяє нівелювати недоліки окремих підходів (поверховість правил та галюцинації нейромереж) і створює передумови для розробки запропонованого у даній роботі методу

1.4 Постановка задачі дослідження

Метою роботи є підвищення якості процесу виявлення лінгвістичних неоднозначностей у вимогах до програмного забезпечення за рахунок застосування контекстно-орієнтованих агентів на базі LLM.

Для досягнення поставленої мети визначено такі задачі дослідження:

- провести аналіз існуючих методів виявлення неоднозначностей та підходів до їх класифікації;
- розробити двоетапний метод виявлення та класифікації неоднозначностей, який базується на послідовній взаємодії агента-ідентифікатора для первинної фільтрації вимог та агента-класифікатора для їх семантичної типізації;
- виконати програмну реалізацію методу з використанням технології RAG для забезпечення врахування глосарію та контексту конкретного проєкту;
- провести експериментальне дослідження розробленого методу та оцінити показники якості виявлення неоднозначностей порівняно з базовими підходами.

РОЗДІЛ 2. Метод виявлення неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей

2.1 Концепція та схема методу виявлення неоднозначностей у вимогах

Аналіз предметної області, проведений у попередньому розділі, доводить, що ручне виявлення неоднозначностей у специфікаціях вимог до програмного забезпечення є вкрай трудомістким, суб’єктивним та схильним до помилок процесом. Традиційні методи автоматизації, що базуються на правилах або класичних моделях машинного навчання (наприклад, SVM або навіть ранні нейронні мережі), мають обмежену ефективність, оскільки їм бракує глибокого семантичного та контекстуального розуміння природної мови. Навіть більш просунуті моделі глибокого навчання, такі як BERT або RoBERTa, хоч і показують кращі результати, зазвичай вимагають значних обсягів вузькоспеціалізованих, вручну розмічених даних для донавчання, що є дорогим і складним процесом.

Основна ідея запропонованого методу полягає у використанні LLM не як моделі, що потребує донавчання на тисячах прикладів, а як інтелектуального агента або експерта-аналітика. Запропоновано використання можливостей LLM до розуміння та міркування, керуючи ним за допомогою спеціально розроблених текстових запитів. Запропонований метод реалізує концепцію нейро-символьної системи, що інтегрує ймовірнісну потужність великих мовних моделей із детермінованою точністю символьних правил та структурованих баз знань. На відміну від чисто генеративних підходів, які покладаються виключно на статистичні закономірності навчання, ця архітектура впроваджує жорсткі обмеження через лексичні словники та онтології проекту.

В основі розробленого методу лежить відмова від використання єдиного запиту до великої мовної моделі на користь двоетапного процесу. Замість спроби вирішити задачу аналізу вимог «в один прохід», пропонується конвеєрна архітектура, яка дозволяє декомпозувати складну когнітивну задачу на послідовність більш

простих, контрольованих підетапів, оскільки це забезпечує можливість верифікації проміжних результатів та підвищує загальну керованість системи.

Логічна архітектура розробленого методу побудована за принципом послідовного конвеєра, що забезпечує перетворення неструктурованого тексту технічних вимог у структурований аналітичний звіт. На відміну від монолітних підходів, де весь масив тексту обробляється за один прохід, запропонований метод реалізує каскадну обробку даних. Процес можна концептуально розділити на три рівні абстракції, кожен з яких виконує специфічну функцію трансформації інформації.

Рівень ідентифікації виконує функцію інтелектуального бар’єра, завданням якого є оперативне відокремлення коректних вимог від сумнівних, що дозволяє оптимізувати обчислювальні ресурси перед етапом глибокої класифікації. Архітектура модуля ідентифікації побудована на послідовній взаємодії двох механізмів перевірки:

Спершу застосовується метод детермінованого лексичного пошуку, який гарантує фіксацію будь-якого слова, що теоретично може вказувати на проблему, виявлені таким чином лексичні збіги підлягають подальшій контекстній валідації за допомогою легковагового агента на базі мовної моделі;

Фінальний рівень системи призначений для поглибленого вивчення тих фрагментів тексту, які успішно пройшли етап попереднього відбору, ключову роль відіграє спеціалізований модуль класифікації, що спирається на технічні стандарти якості вимог, зокрема IEEE 830, а для формування висновків застосовується метод ланцюжка міркувань, завдяки якому модель відтворює хід думки експерта та надає розгорнуту аргументацію.

Ключовою архітектурною особливістю запропонованого методу є відмова від монолітної обробки запитів на користь агентної системи. Розподіл відповідальності між двома незалежними агентами – агент ідентифікатор та агент класифікатор – дозволяє оптимізувати процес обробки, налаштовуючи кожен компонент на

досягнення конкретних метричних показників. На рисунку 2.1 представлено схему методу, що відображає взаємодію компонентів.

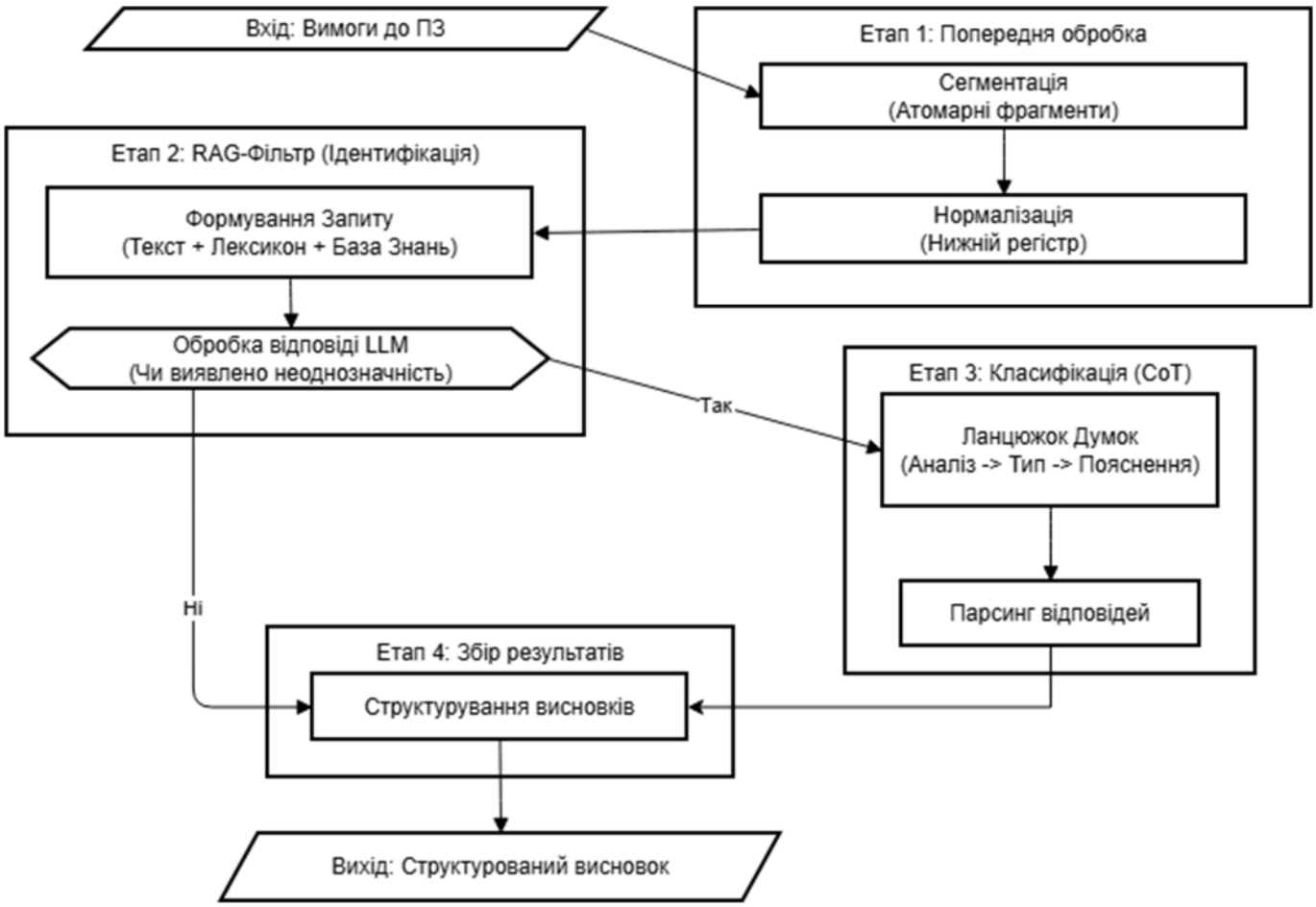


Рисунок 2.1 – Схема методу виявлення неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей

Процес виконання методу можна розділити на послідовні кроки:

1. Попередня обробка та сегментація – вхідним масивом даних для системи слугує неструктурована документація, ключовим елементом процесу трансформації є процедура сегментації, що забезпечує декомпозицію матеріалу на атомарні фрагменти – окремі речення або завершені твердження, фінальним кроком підготовки даних є їх нормалізація, яка передбачає приведення тексту до уніфікованого вигляду шляхом переведення символів у нижній регістр та канонізації пробілів;
2. Ідентифікація неоднозначностей (RAG-фільтр) – другий етап алгоритму реалізує процедуру первинного відбору фрагментів, що містять ознаки

семантичної або логічної невизначеності, ключовим механізмом цього етапу є динамічне формування розширеного запиту до нейронної мережі, який синтезується шляхом об'єднання трьох інформаційних потоків: тексту вимоги, лексичних маркерів та фактів із бази знань проекту; завершується етап аналізом результатів роботи LLM, елементи, які ідентифіковано, як неоднозначні, передаються на вхід агента класифікатора;

3. Класифікація та генерація пояснень – третій етап алгоритму фокусується на поглибленому семантичному аналізі. Центральним елементом етапу класифікації є застосування стратегії промпт-інжинірингу «ланцюжок думок» (CoT), модель послідовно виконує три регламентовані операції: лінгвістичний аналіз контексту, зіставлення із типами дефектів та формування фінального пояснення, спеціалізовані патерни здійснюють пошук та виокремлення змістових блоків – аналізу, типу класифікації та пояснення;
4. Агрегація результатів – фінальна стадія процесу передбачає консолідацію розрізнених даних, агрегація результатів відбувається шляхом інтеграції висновків агентів ідентифікації та класифікації у єдину структуру звіту, що забезпечує наскрізну простежуваність виявлених дефектів.

Запропоноване рішення відмовляється від монолітної обробки даних на користь конвеєрної агентної архітектури, де процес декомпозовано на етапи ідентифікації та поглибленої класифікації. Запропонована ідея, підсилена використанням технологій RAG для врахування контексту проекту та Chain-of-Thought для імітації експертних міркувань, дозволяє подолати обмеження традиційних алгоритмів та уникнути необхідності високовартісного донавчання моделей. Концепція методу забезпечує автоматизовану трансформацію неструктурованих специфікацій у валідований аналітичний звіт, гарантуючи баланс між повнотою пошуку потенційних дефектів та точністю їхньої інтерпретації згідно з міжнародними стандартами якості.

2.2 Математична модель виявлення неоднозначностей

Для забезпечення відтворюваності результатів та чіткої формалізації методу необхідно визначити математичну модель виявлення неоднозначностей. У межах даного підходу задача аналізу вимог розглядається як комплексний процес, що поєднує класифікацію тексту та генерацію пояснень із застосуванням методів обробки природної мови та врахуванням додаткового контексту.

Вхідний документ із вимогами до програмного забезпечення *D* попередньо піддається процесу сегментації, в результаті якого формується впорядкована множина текстових фрагментів (атомарних вимог). Позначимо цю множину як *R*:

$$R = \{r_1, r_2, \dots, r_n\}, \quad r_i \in \Sigma^* \tag{2.1}$$

де n – загальна кількість виділених фрагментів, а $r_i \in \Sigma^*$ – множина всіх можливих ланцюжків символів алфавіту. Кожен елемент r_i є окремим реченням або логічно завершеним твердженням, що підлягає перевірці.

Окрім текстових вимог, важливим компонентом вхідних даних є база знань проекту, яка використовується для контекстно-залежного аналізу (RAG). Формалізуємо її як множину контекстних фактів *K*:

$$K = \{k1, k2, \dots, km\} \tag{2.2}$$

де k_j – це окремий опис сутності, обмеження або архітектурного рішення, специфічного для предметної області (наприклад, типи серверів, ролі користувачів, акроніми).

Запропонований метод реалізує функцію відображення *F*, яка трансформує пару «вимога – контекст» у множину виявлених дефектів. Формально це можна записати як:

$$F\colon R \times K \to A \tag{2.3}$$

де A – множина ідентифікованих неоднозначностей. Результатом застосування функції до і-го фрагмента є об'єкт a_i (або порожня множина $\emptyset$, якщо дефектів не виявлено):

$$a_i = F(r_i, K) \tag{2.4}$$

Кожен виявлений дефект $a_i \in A$ представляється у вигляді впорядкованого кортежу, що містить результати роботи агентів ідентифікації та класифікації:

$$a_{\text{i}} = \langle r_{\text{i}}, c_{\text{i}}, p_{\text{i}}, e_{\text{i}} \rangle \tag{2.5}$$

де: r_i – вихідний текст фрагмента вимоги; $c_i \in C$ – клас виявленої неоднозначності; $p_i \in [0, 1]$ – оцінка впевненості моделі (confidence score), що обчислюється як зважена метрика на основі вихідних ймовірностей токенів LLM; e_i – текстове пояснення причини неоднозначності, згенероване з використанням механізму Chain-of-Thought.

Центральним елементом першого етапу є Агент-Ідентифікатор, який виконує роль первинного фільтра, основним завданням якого є звуження простору пошуку через оперативне відсіювання коректних фрагментів. Його функціональне налаштування орієнтоване на максимізацію повноти (Recall), що передбачає мінімізацію ризику пропуску дефектів.

Архітектура цього модуля використовує композитний вектор вхідних даних:

$$Input = T_{frag} + V_{lex} + K_{base} \tag{2.6}$$

де T_{frag} – нормалізований текст вимоги, V_{lex} – словник маркерів (діє як механізм уваги), а K_{base} – контекстна інформація про проект. Ключовим технічним елементом є організація промпту за парадигмою пошукового доповнення генерації із застосуванням суворих структурних обмежень, що вимагають повернення виключно валідного об'єкта JSON.

Рішення приймається через порогову функцію активації – вимога вважається неоднозначною, якщо $P_{amb} \geq \theta$, це дозволяє реалізувати концепцію «широкого

фільтра», відсіюючи лише очевидно коректні вимоги: тільки фрагменти, для яких результат функції є неоднозначними, передаються на вхід агента-класифікатора.

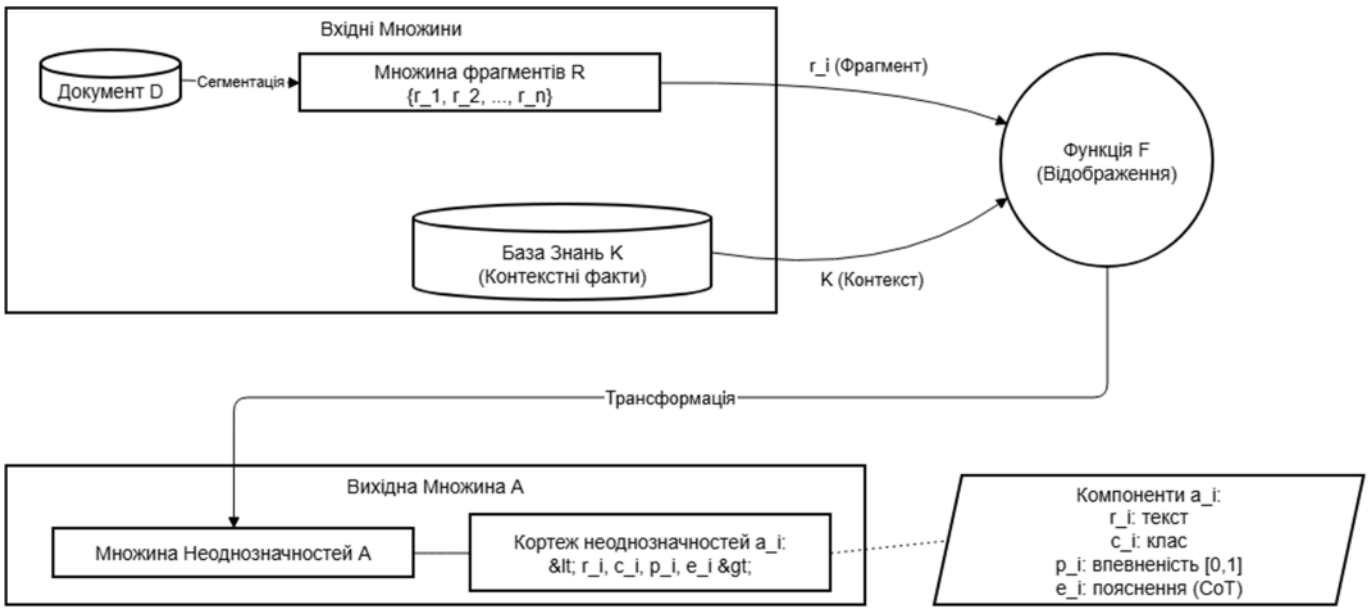


Рисунок 2.2 – Графічне представлення математичної моделі ідентифікації неоднозначностей

На рисунку 2.2 візуалізовано процес обробки даних на етапі первинної ідентифікації неоднозначностей. Схема демонструє перетворення вхідної інформації у бінарне рішення про доцільність подальшого аналізу.

Процес розпочинається з формування композитного вектора вхідних даних, який об'єднує три інформаційні потоки:

- нормалізований текст вимоги (T_{frag}),
- набір лексичних маркерів (V_{lex}), що фокусують увагу моделі на потенційних проблемних зонах,
- контекстна база знань проекту (K_{base})

Перераховані компоненти інтегруються у запит до нейромережі з використанням парадигми RAG та суворих обмежень щодо формату виводу (JSON Constraint).

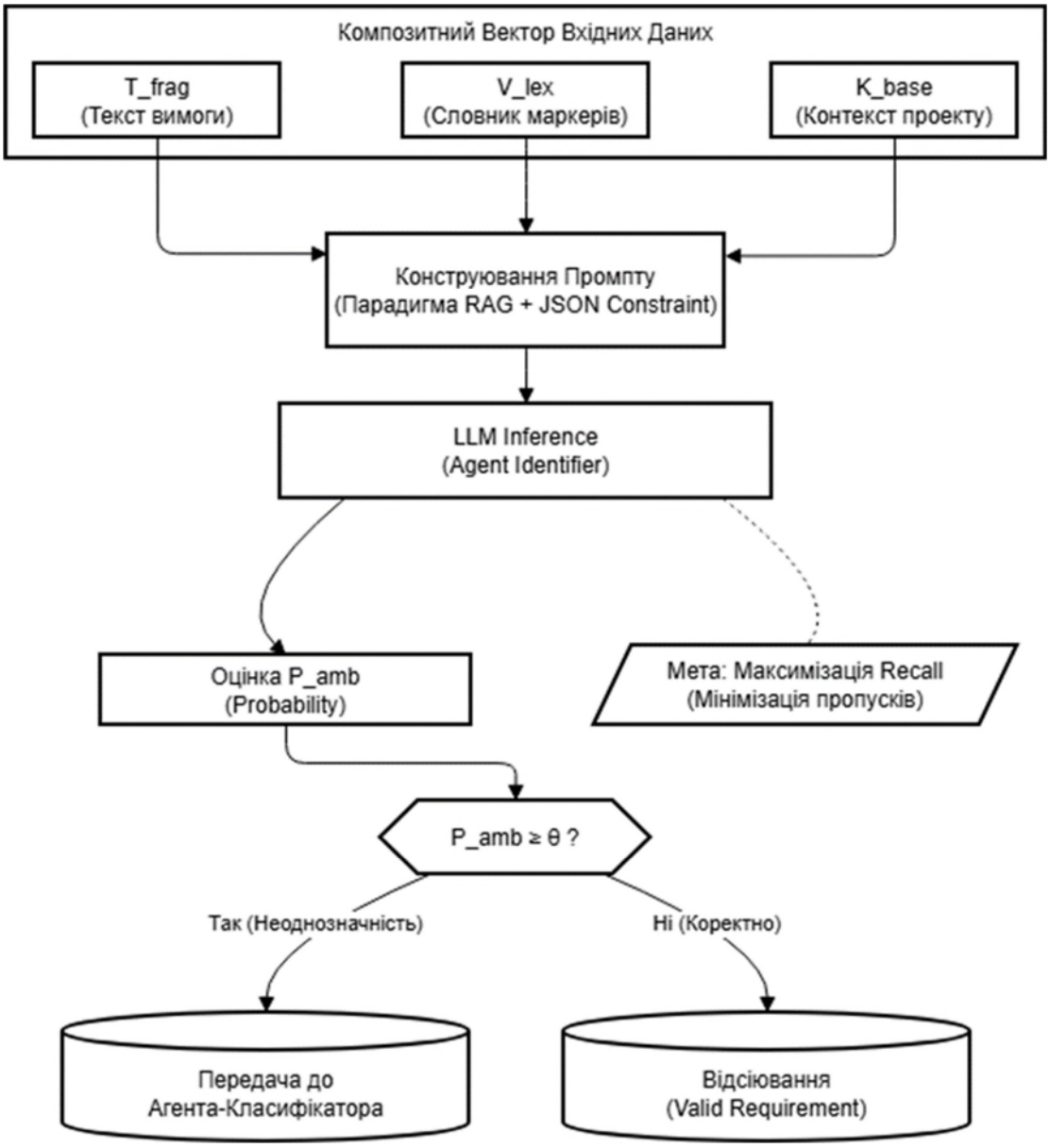


Рисунок 2.3 – Функціональна схема агента-ідентифікатора

Результатом роботи генеративної моделі є кількісна оцінка ймовірності наявності дефекту. Ключовим вузлом прийняття рішень виступає порогова функція активації (ромб на схемі), яка перевіряє умову $P_{amb} \geq \theta$. Залежно від результату порівняння, потік даних розгалужується: вимоги з високою ймовірністю неоднозначності передаються на вхід агента-класифікатора для детального розбору, тоді як фрагменти з низькою оцінкою відсіюються як коректні. Запропонована логіка спрямована на досягнення стратегічної мети етапу – максимізації метрики Recall для мінімізації ризику пропуску дефектів.

2.3 Модель семантичної класифікації вимог

Якщо завданням попереднього етапу (ідентифікації) була мінімізація пропусків через максимізацію метрики Recall, то цільова функція класифікатора зміщується в бік забезпечення високої точності (Precision) та повної інтерпретованості результатів. Агент-класифікатор трансформує вхідний підозрілий фрагмент T_{frag} у структурований кортеж O_{class}, що містить результати аналізу, визначений клас та пояснення. Для досягнення глибокого семантичного розуміння агент не використовує пряму класифікацію, а застосовує стратегію «ланцюжок думок», де процес прийняття рішення декомпозовано на послідовні фази.

Внутрішня організація системного промпту поєднує три ключові елементи:

1. Жорсткі рольові інструкції – визначення моделі як експерта з якості вимог;
2. Обмежений простір класів – список допустимих типів помилок згідно зі стандартом IEEE 830;
3. Демонстраційні приклади – набір пар «вимога – аналіз» для контекстного навчання моделі.

Процес генерації відповіді є суворо регламентованим і проходить три стадії:

1. Лінгвістичний аналіз – модель формує вільний опис особливостей тексту, виявляючи конфліктні терміни;
2. Вибір класу – на основі аналізу обирається конкретний тип неоднозначності з таксономії;
3. Генерація вердикту – формулювання фінального пояснення, що пов'язує категорію помилки з конкретними словами у вимозі.

Для забезпечення стабільності та аргументованості висновків процес генерації керується специфічним набором гіперпараметрів, зокрема, використання параметрів температури та променевого пошуку дозволяє отримати розгорнуте, логічно зв’язне обґрунтування, уникаючи при цьому генерації шаблонних або занадто коротких відповідей. Оскільки вихід моделі є текстом, для його структурування застосовується

механізм парсингу на основі регулярних виразів, що виокремлює змістовні секції (аналіз, клас, пояснення) у програмний об'єкт.

Множина можливих класів неоднозначностей C, на розпізнавання яких налаштована модель, визначається таксономією, закладеною у промпти класифікатора. Простір класів C включає такі підмножини:

$$C = \{C_{lex}, C_{struct}, C_{ref}, C_{scope}, C_{ctx}\} \quad 2.7$$

Характеристика класів:

- лексична неоднозначність (C_{lex}) – виникає, коли окремі слова мають декілька значень (полісемія) або є нечіткими;
- структурна (синтаксична) неоднозначність (C_{struct}) – зумовлена будовою речення, де розміщення фраз або розділових знаків створює варіативність інтерпретації;
- референційна неоднозначність (C_{ref}) – неможливо однозначно визначити об'єкт, на який вказує слово;
- неоднозначність сфери дії (C_{scope}) – коли синтаксична структура вимоги з логічними операторами («AND», «OR», «NOT») або кванторами («всі», «деякі») допускає множинне тлумачення меж їхнього впливу на об'єкти системи;
- контекстуальна неоднозначність (C_{ctx}) – специфічний клас помилок, що виявляється лише при зіставленні вимоги з базою знань K, це ситуації, коли вимога є синтаксично правильною, але суперечить архітектурі або визначенням домену (наприклад, вимога "Сервер S1 має обробляти відео", якщо в K зазначено, що S1 – це легковаговий проксі).

Таким чином, цільова функція методу полягає у мінімізації помилок класифікації, тобто максимізації точності визначення пари (ci, ei) для кожного вхідного ri, що містить приховану неоднозначність.

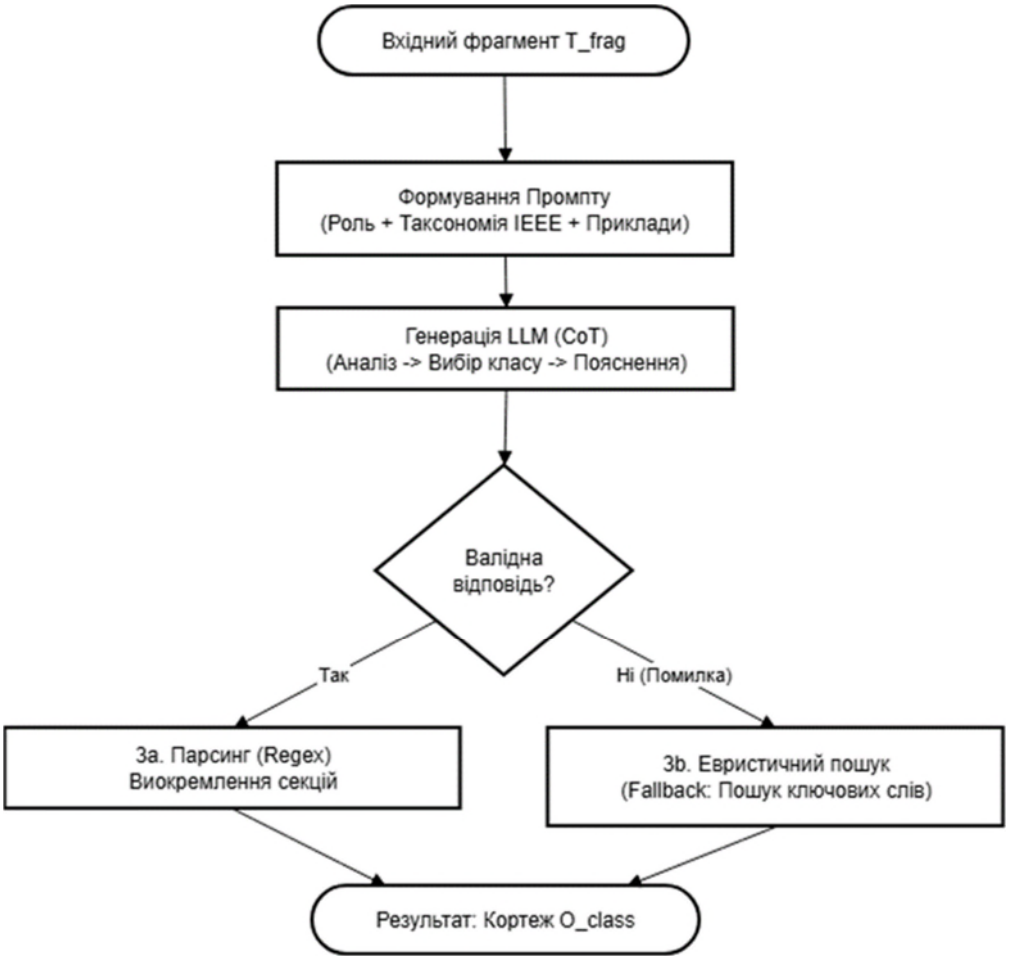


Рисунок 2.4 – Функціональна схема агента-класифікатора

Інтеграція стохастичних інструментів, таких як LLM, породжує ризики нестабільності. Тому критично важливою складовою архітектури є модуль обробки винятків, що реалізує концепцію «евристичного страхування», цей механізм захищає систему від трьох основних векторів відмов: порушення структурної цілісності відповіді (помилка в структурі JSON), семантичних галюцинацій та технічних збоїв API. Логіка роботи базується на принципі автоматичної «м'якої деградації». У разі збою основного нейромережевого ядра потік управління передається резервному блоку, який виконує детермінований лексичний аналіз, цей компонент шукає перетини між текстом вимоги та словником індикаторів неоднозначності. При виявленні збігу системі присвоюється евристичний клас помилки, а поле пояснення заповнюється синтетичними даними, що вказують на технічний характер вердикту.

Запропонована дворівнева структура гарантує високу відмовостійкість методу – навіть за умов повної дисфункції мовної моделі користувач отримає базовий звіт про проблемні зони специфікації, що відповідає принципам надійної розробки ПЗ.

Фінальним етапом роботи компонента є створення об'єкта «класифікована неоднозначність» ($O_{classified}$), цей об'єкт є розширеною версією результатів ідентифікації та інтегрує всі аналітичні метрики: детальний лінгвістичний розбір, мітку класу за IEEE 830 та пояснення для користувача. Центральний оркестратор консолідує ці об'єкти у єдиний реєстр, забезпечуючи повну простежуваність процесу виявлення помилок від моменту виникнення першої підозри до затвердження остаточної класифікації.

2.4 Формування бази знань та підготовка даних

Ефективність функціонування гібридного методу виявлення неоднозначностей критично залежить від якості та структури інформаційного забезпечення, яке виступає семантичним фундаментом для роботи інтелектуальних агентів. Процес проектування ресурсів для системи не обмежується простим накопиченням даних, а вимагає системного підходу до їх категоризації та формалізації. У межах даного дослідження розроблено стратегію, що передбачає створення спеціалізованого інформаційного середовища, оптимізованого для сприйняття великими мовними моделями.

В основі архітектури інформаційного забезпечення лежить принцип розділення знань на дві логічні категорії, що зумовлено дуальною природою задачі аналізу вимог. Виявлення неоднозначностей у специфікаціях вимагає одночасного врахування як універсальних правил мови, так і специфічних обмежень конкретного проекту. Для реалізації цього підходу застосовано дворівневу модель організації даних.

Перший рівень – інваріантні лінгвістичні знання (Domain-Independent Knowledge) – це сукупність універсальних мовних патернів, лексичних конструкцій

та синтаксичних правил, які вказують на потенційну неоднозначність незалежно від предметної області. Наприклад, слово «швидкий» є джерелом суб'єктивної невизначеності як у вимогах до банківського програмного забезпечення, так і в специфікаціях бортових систем літака. У розробленій системі цей шар знань реалізовано через статичний компонент, який агрегує загальні індикатори неякісних вимог.

Другий рівень – варіативні проектні знання (Domain-Specific Knowledge) – ця категорія охоплює унікальну інформацію про архітектуру, компоненти, бізнес-логіку та термінологію конкретної системи, що розробляється. Необхідність виділення цього рівня пояснюється тим, що певні твердження можуть бути однозначними в одному контексті та дефектними в іншому. Наприклад, термін «Користувач» може бути цілком коректним, якщо в системі є лише одна роль, але стає джерелом референційної неоднозначності в системі з ролями «Адміністратор», «Гість» та «Оператор», цей рівень реалізовано через динамічний контекст

Вище описана сегрегація дозволяє досягти гнучкості методу – при зміні проекту достатньо оновити лише варіативну частину бази знань, залишаючи лінгвістичне ядро незмінним.

Процес формування лексикону базується на систематизації найкращих практик інженерії вимог, зокрема рекомендацій міжнародних стандартів ISO/IEC 29148 та IEEE 830, а також досліджень у галузі комп'ютерної лінгвістики. Лексикон не є простим набором заборонених слів, а становить структуровану таксономію тригерів, категоризованих за типом впливу на семантику вимоги. До першої категорії належать суб'єктивні прикметники та прислівники, що описують якості, які неможливо об'єктивно виміряти без додаткових метрик, наприклад, «швидкий», «зручний» або «ефективний», їх наявність майже завжди свідчить про контекстуальну неоднозначність, оскільки інтерпретація залежить від особистого досвіду читача. Другу групу складають нечіткі слова – терміни, що вказують на невизначену кількість або обсяг, такі як «всі», «декілька» чи «мінімальний». Вони створюють суттєві ризики для тестування системи, оскільки унеможливлюють чітке

визначення критеріїв успішності верифікації. Третім важливим елементом є дієслова слабкої дії, що виражають можливість, а не обов'язок, наприклад, «може» або «бажано», використання яких порушує принцип обов'язковості вимог і призводить до лексичної неоднозначності.

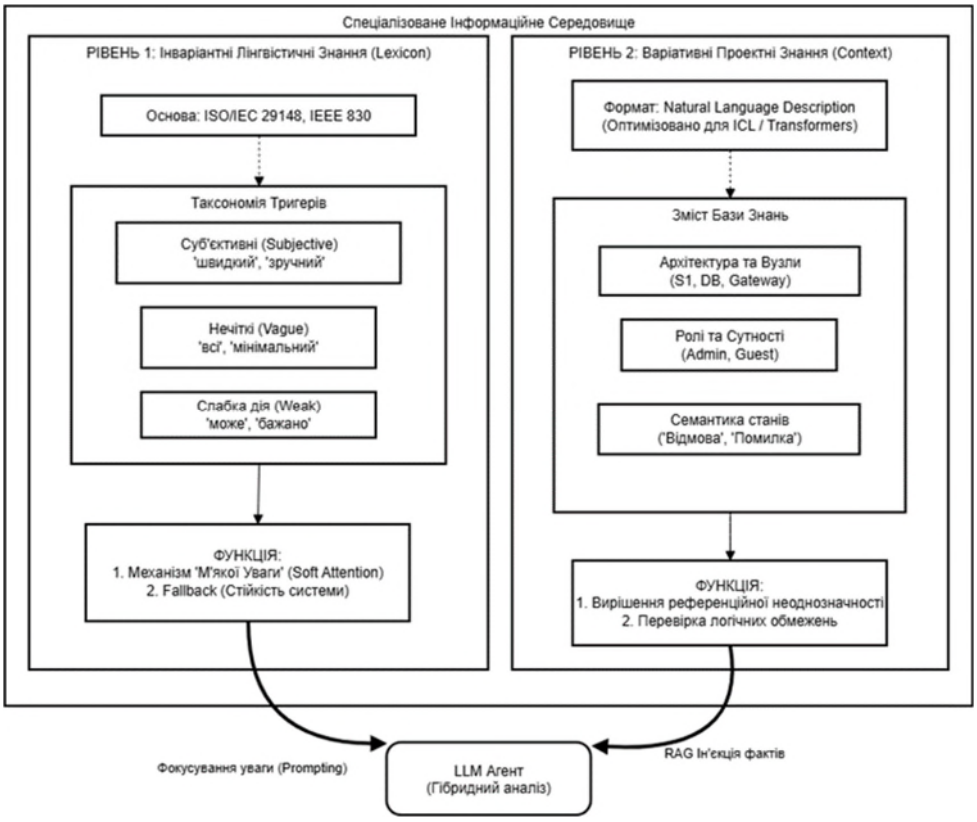


Рисунок 2.5 – Дворівнева модель організації інформаційного забезпечення системи

Важливою особливістю розробленого методу є зміна ролі лексикону порівняно з класичними системами типу QuARS. Якщо в традиційних підходах наявність слова зі словника автоматично призводила до реєстрації помилки, то в даній архітектурі лексикон виконує функцію механізму «м'якої уваги» для нейромережі. Включення ключових слів у промпт фокусує увагу великої мовної моделі на підозрілих ділянках, але остаточне рішення приймається з урахуванням контексту, окрім того, лексикон виступає критичним елементом механізму стійкості, де у разі технічного збою моделі система деградує до рівня лексичного пошуку, гарантуючи виявлення явних проблем.

Для вирішення проблем контекстуальної та проектної неоднозначності, які неможливо ідентифікувати суто лексичними методами, застосовується підхід ін'єкції знань через RAG-компоненту. Структурування бази знань базується на принципах Prompt Engineering, де метою є створення лаконічного, але інформативного опису предметної області. Наповнення бази знань передбачає обов'язкову ідентифікацію компонентів архітектури, зокрема фізичних та логічних вузлів системи, таких як сервери, бази даних чи шлюзи, це дозволяє моделі розв'язувати референційні неоднозначності, коли вимога оперує абстрактними поняттями, прив'язуючи їх до конкретних елементів. Паралельно з цим фіксуються функціональні зв'язки та зони відповідальності компонентів, що уможливлює виявлення логічних суперечностей у вимогах, а також визначається семантика станів та збоїв, що розкриває значення термінів «помилка» або «відмова» для конкретних модулів системи.

При виборі формату зберігання та подачі знань було прийнято рішення відмовитися від складних формалізованих структур, таких як онтології OWL, графи знань або реляційні схеми SQL, на користь природно-мовного описового формату, такий вибір обґрунтовується специфікою архітектури використовуваних генеративних моделей сімейства Transformer. Моделі, навчені на величезних масивах текстових даних, демонструють значно вищу здатність до розуміння зв'язного наративу та контекстних залежностей, ніж до інтерпретації сухих структурованих об'єктів JSON або XML. Використання легковагого текстового представлення контексту є оптимізованим для технології навчання в контексті. Текстовий опис дозволяє моделі використовувати механізми самоуваги для встановлення семантичних зв’язків між вимогою та описом системи без необхідності додаткового навчання.

2.5 Критерії та метрики оцінювання ефективності методу

Вибір системи оцінювання для розробленого гібридного методу зумовлений специфікою предметної області та архітектурними особливостями запропонованого

рішення. Задача виявлення неоднозначностей у вимогах формально зводиться до задачі бінарної класифікації (на етапі ідентифікації) та мультикласової класифікації (на етапі визначення типу дефекту). Важливою особливістю досліджуваних даних є суттєвий дисбаланс класів: у реальній технічній документації кількість коректних вимог («clean») значно перевищує кількість неоднозначних, в таких умовах використання класичної метрики Accuracy (частка правильних відповідей) є недоцільним та оманливим, оскільки модель, яка просто маркує всі вимоги як коректні, формально матиме високу точність, але нульову практичну цінність.

Враховуючи двохетапну структуру методу, оцінка ефективності кожного з агентів вимагає диференційованого підходу, що узгоджується з сучасними практиками в інженерії вимог. Для першого компонента системи – агента ідентифікатора – ключовим критерієм є повнота (Recall), оскільки на цьому етапі критично важливо мінімізувати кількість пропущених помилок. Завданням ідентифікатора є створення «широкого фільтра», тому низький показник Recall є неприпустимим, адже дефектна вимога буде відкинута на самому початку і не потрапить на глибокий аналіз [45]. Розрахунок здійснюється за формулою:

$$Recall = \frac{TP}{TP+FN} \tag{2.8}$$

Натомість для оцінки кінцевого результату роботи системи (виходу агента класифікатора) пріоритетною стає точність (Precision). Висока точність свідчить про низький рівень «інформаційного шуму» (FP), що є критичним для запобігання ефекту «втоми від попереджень», коли аналітик починає ігнорувати звіти системи через часті хибні спрацювання [46, 47]. Відповідно, класифікатор повинен ефективно відфільтрувати хибні підозри попереднього етапу:

$$Precision = \frac{TP}{TP+FP} \tag{2.9}$$

Оскільки між точністю та повнотою існує зворотна кореляція, для інтегральної оцінки методу та пошуку балансу між надійністю та чутливістю використовується F1-Score – гармонічне середнє між цими показниками [48]:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (2.10)$$

У випадку мультикласової класифікації (розподіл за типами: Clex, Cstruct, Cref тощо) доцільно застосовувати Macro-averaged F1-Score. Цей підхід передбачає розрахунок F1 окремо для кожного класу неоднозначності з подальшим обчисленням середнього арифметичного, що дозволяє уникнути домінування поширених класів помилок над рідкісними, але складними дефектами [49, 50].

Оскільки однією з ключових переваг методу є генерація пояснень, суто статистичних метрик, таких як BLEU чи ROUGE, недостатньо, адже вони погано корелюють з логічною якістю аргументації [51]. Валідація пояснювальної здатності здійснюється через експертну оцінку вибірки результатів за шкалою Лікерта (1–5), як це пропонується в сучасних дослідженнях [52]. Оцінювання якості пояснення (e_i) базується на трьох аспектах: коректності локалізації проблемних фрагментів, обґрунтованості пояснення відповідно до стандартів (наприклад, IEEE 830) та логіки предметної області, а також практичної корисності рекомендації для виправлення вимоги інженером.

Для підтвердження придатності методу до впровадження у промислові процеси аналізуються технічні показники. Зокрема, враховується середній час обробки однієї вимоги, який складається з часу проходження через каскад агентів ($t_{total} = t_{ident} + t_{class}$), та відсоток евристичних спрацювань. Останній показник характеризує стабільність нейро-символьної архітектури, відображаючи частку вимог, оброблених резервним механізмом через технічні збої або галюцинації LLM, такий комплексний підхід забезпечує багатовимірний аналіз методу, що охоплює точність детекції, глибину семантичного розуміння та експлуатаційну надійність.

Висновки до розділу 2

У цьому розділі розроблено та теоретично обґрунтовано гібридний нейро-символьний метод виявлення неоднозначностей у вимогах до програмного

забезпечення. Основний науковий результат полягає у відмові від монолітної обробки тексту на користь агентної архітектури, що поєднує високу чутливість лексичного пошуку з глибиною семантичного аналізу великих мовних моделей. У ході розробки методу вирішено наступні задачі:

1. Сформовано концептуальну схему процесу, яка реалізує конвеєрну (pipeline) обробку даних. Декомпозиція задачі на етапи ідентифікації та класифікації дозволила вирішити проблему «розфокусування уваги» (attention dilution), що є критичним обмеженням сучасних LLM при роботі з технічною документацією;
2. Визначено процедуру підготовки вхідних даних, яка включає сегментацію тексту вимог на атомарні фрагменти, їх нормалізацію та токенізацію, це забезпечує уніфікацію даних перед подачею на вхід нейромережевих моделей, підвищуючи стабільність їх роботи;
3. Інтегровано механізм динамічного контексту (RAG), що дозволяє враховувати специфічні знання про предметну область проекту без необхідності високовартісного донавчання моделі, це забезпечує адаптивність методу до унікальної термінології конкретних розробок;
4. Впроваджено стратегію «евристичного страхування» для забезпечення експлуатаційної надійності, обґрунтовано необхідність автоматичної деградації системи до детермінованих алгоритмів у випадках технічних збоїв генеративного компонента, що гарантує безперервність процесу аналізу;
5. Обґрунтовано вибір системи метрик ефективності, що включає Precision, Recall та F1-Score, акцент зроблено на максимізації показника Recall на етапі ідентифікації (для мінімізації пропущених дефектів) та Precision на етапі класифікації (для зниження інформаційного шуму), що є найбільш доречним підходом для систем підтримки прийняття рішень.

Таким чином, розроблений метод створює необхідне алгоритмічне підґрунтя для програмної реалізації системи автоматизованого аудиту вимог, здатної діяти як

віртуальний експерт-аналітик. Практична апробація методу та експериментальне підтвердження його ефективності будуть наведені у наступних розділах роботи.

РОЗДІЛ 3. Програмна реалізація методу виявлення неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей

3.1 Засоби та середовище програмної реалізації

Вибір мови програмування Python для реалізації запропонованого методу є стратегічно обґрунтованим рішенням, зумовленим її домінуючим положенням у сучасній індустрії обробки природної мови та машинного навчання. Завдяки розвиненій екосистемі та нативній підтримці високопродуктивних обчислень, Python забезпечує ефективну інтеграцію різнорідних компонентів системи – від низькорівневих тензорних операцій до високорівневих інтерфейсів взаємодії з нейромережами. Використання мови програмування Python дозволяє суттєво зменшити семантичний розрив між математичною моделлю алгоритму та його програмним втіленням, забезпечуючи лаконічність коду та простоту його підтримки.

Процес розробки та експериментальної перевірки методу здійснювався в інтерактивному середовищі типу Jupyter Notebook, а саме в його хмарній реалізації Google Colab. Використання Google Colab дозволяє реалізувати концепцію швидкого прототипування, де кожен етап конвеєра обробки даних – від ініціалізації токенізатора до генерації пояснень – може бути виконаний та верифікований ізольовано, це є критично важливим при роботі з великими мовними моделями, оскільки дозволяє динамічно відстежувати стан обчислювальних ресурсів та проміжні результати генерації без необхідності повного перезапуску системи.

Забезпечення відтворюваності результатів та мобільності програмного рішення реалізовано через механізми автоматичного налаштування оточення безпосередньо у коді скрипту. Використання системних викликів через модуль sys та менеджер пакетів pip дозволяє динамічно інсталювати необхідні залежності, такі як бібліотеки transformers та torch, у будь-якому сумісному середовищі виконання, це гарантує, що розроблений прототип може бути розгорнутий як на локальній робочій

станції дослідника, так і на віддалених серверах з графічними прискорювачами без складної попередньої конфігурації операційної системи.

Реалізація інтелектуальної складової системи базується на використанні бібліотеки Hugging Face Transformers, яка на сьогодні є галузевим стандартом для роботи з архітектурами трансформерів. Вибір вище перерахованих інструментів зумовлено наданням уніфікованого інтерфейсу для взаємодії з попередньо навченими моделями, що дозволяє уникнути необхідності тренування нейромереж «з нуля» та зосередитися на їх налаштуванні під конкретну задачу. Компонентна діаграма програмної реалізації методу, що відображає ієрархію використаних інструментальних засобів та бібліотек зображена на рисунку 3.1.

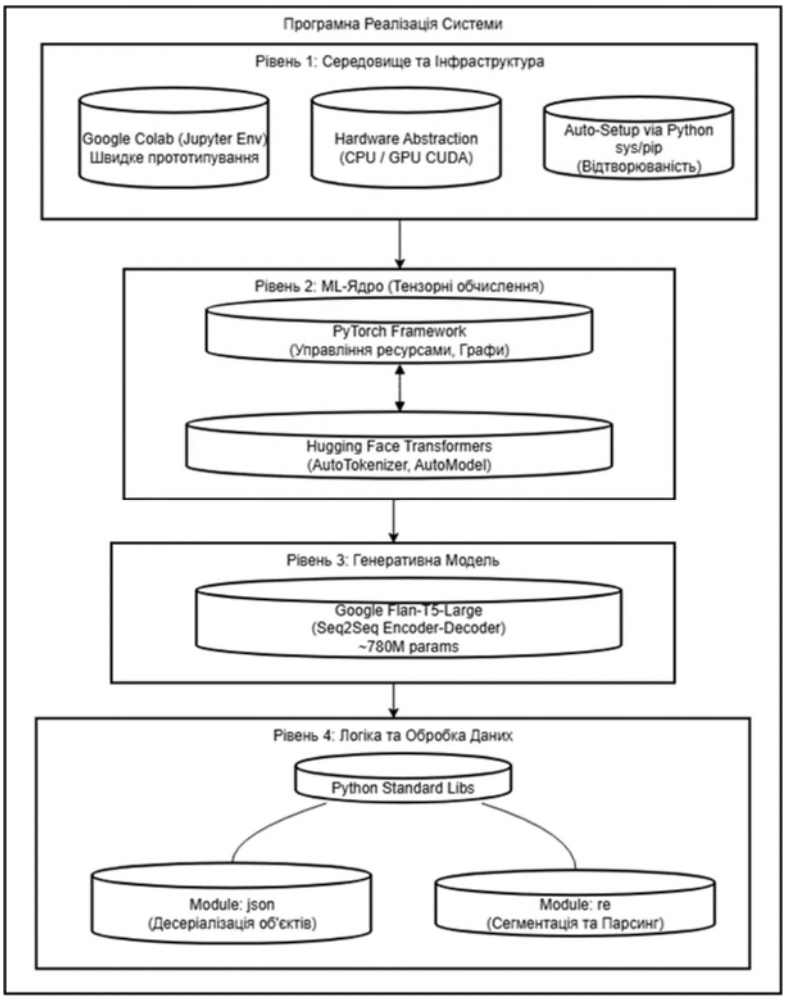


Рисунок 3.1 – Компонентна діаграма програмної реалізації методу

Виконання низькорівневих тензорних обчислень та управління обчислювальними графами покладено на фреймворк PyTorch, його інтеграція у проект забезпечує ефективний розподіл апаратних ресурсів під час виконання операцій логічного виведення. Важливою функцією бібліотеки PyTorch в контексті програмної реалізації розробленого методу є механізм динамічного визначення доступного обчислювального пристрою. Реалізована логіка автоматичної перевірки наявності підтримки CUDA дозволяє адаптувати виконання коду до поточного середовища: за наявності сумісної відеокарти модель переміщується у відеопам'ять для значного прискорення генерації, тоді як за її відсутності система автоматично перемикається на режим обробки центральним процесором.

Для забезпечення структурної цілісності даних та маніпуляцій з текстовими потоками залучено стандартні модулі Python, які забезпечують надійну обробку результатів роботи нейромережі. Бібліотека json відповідає за десеріалізацію відповідей мовної моделі, трансформуючи текстові рядки у валідні програмні об'єкти, що є критично необхідним для автоматизованої маршрутизації даних між агентами. Модуль re застосовується для вирішення задач сегментації вхідних вимог на основі складних патернів пунктуації, а також для вилучення специфічних інформаційних блоків з неструктурованого виводу методу «ланцюжка думок», забезпечуючи точне розпізнавання меж між етапами аналізу, класифікації та пояснення.

Ключовим елементом архітектури програмного рішення є вибір базової генеративної моделі, яка відповідає за семантичний аналіз та інтерпретацію вимог. У межах даної реалізації було обрано модель Google Flan-T5-Large, що представляє собою вдосконалену версію архітектури трансформера T5 (Text-to-Text Transfer Transformer), додатково донавчену на величезному масиві інструкцій. Вибір Flan-T5 зумовлений специфікою задачі виявлення неоднозначностей, яка за своєю природою вимагає від моделі не лише генерації тексту, а й здатності виконувати складні логічні операції перетворення вхідних даних у структурований висновок. Архітектура Encoder-Decoder, що лежить в основі T5, є оптимальною для цього класу задач,

оскільки вона дозволяє ефективно кодувати вхідний контекст (вимогу та базу знань) і послідовно декодувати його в чіткий аналітичний звіт.

Вагомим фактором на користь Flan-T5-Large став баланс між якістю генерації та вимогами до обчислювальних ресурсів. На відміну від моделей сімейства GPT, які часто вимагають значних потужностей та функціонують переважно через хмарні API, обрана модель з параметрами розміром «Large» (близько 780 мільйонів параметрів) може бути розгорнута локально навіть на обладнанні середнього класу або в обмежених хмарних середовищах типу Google Colab, це забезпечує автономність системи та можливість її масштабування без залежності від зовнішніх комерційних сервісів, зберігаючи при цьому високу точність розуміння інструкцій та контексту, що є критичним для коректної класифікації технічних вимог згідно зі стандартами IEEE.

3.2 Структура програмної реалізації

Основою розробленої програмної реалізації є гібридна архітектурна модель, що поєднує патерн проектування «Оркестратор» (Orchestrator) із принципами архітектурного стилю «конвеєр та фільтри». Вибір принципів реалізації зумовлений специфікою задач обробки природної мови, де процес аналізу не є одномоментною операцією, а розгортається як послідовність трансформацій даних різного рівня абстракції. На відміну від традиційних монолітних процедур, де логіка отримання даних, їх обробки та збереження тісно переплетена в єдиному блоці коду, запропоноване рішення структурує процес як потік, що проходить через серію незалежних обробників. Кожен програмний агент виступає в ролі «фільтра», який приймає стандартизований вхід, виконує спеціалізовану інтелектуальну операцію та передає результат далі по «трубопроводу», тоді як загальна координація цього руху покладається на централізований керуючий вузол – оркестратор.

Застосування конвеєрної архітектури є критично важливим для систем, побудованих на базі великих мовних моделей, оскільки воно дозволяє ефективно

керувати складністю та ресурсоємністю NLP-обчислень. У монолітній архітектурі помилка на будь-якому етапі генерації тексту або збій у роботі нейромережі часто призводить до каскадного падіння всієї системи, а налагодження логіки стає надзвичайно ускладненим через неможливість ізолювати джерело проблеми., натомість реалізована конвеєрна структура дозволяє чітко розмежувати етапи життєвого циклу запиту: від попередньої нормалізації тексту до генерації фінального пояснення, це забезпечує можливість локального керування помилками, коли, наприклад, невдача на етапі класифікації не скасовує результатів етапу ідентифікації, а лише активує відповідні механізми обробки винятків у конкретному модулі.

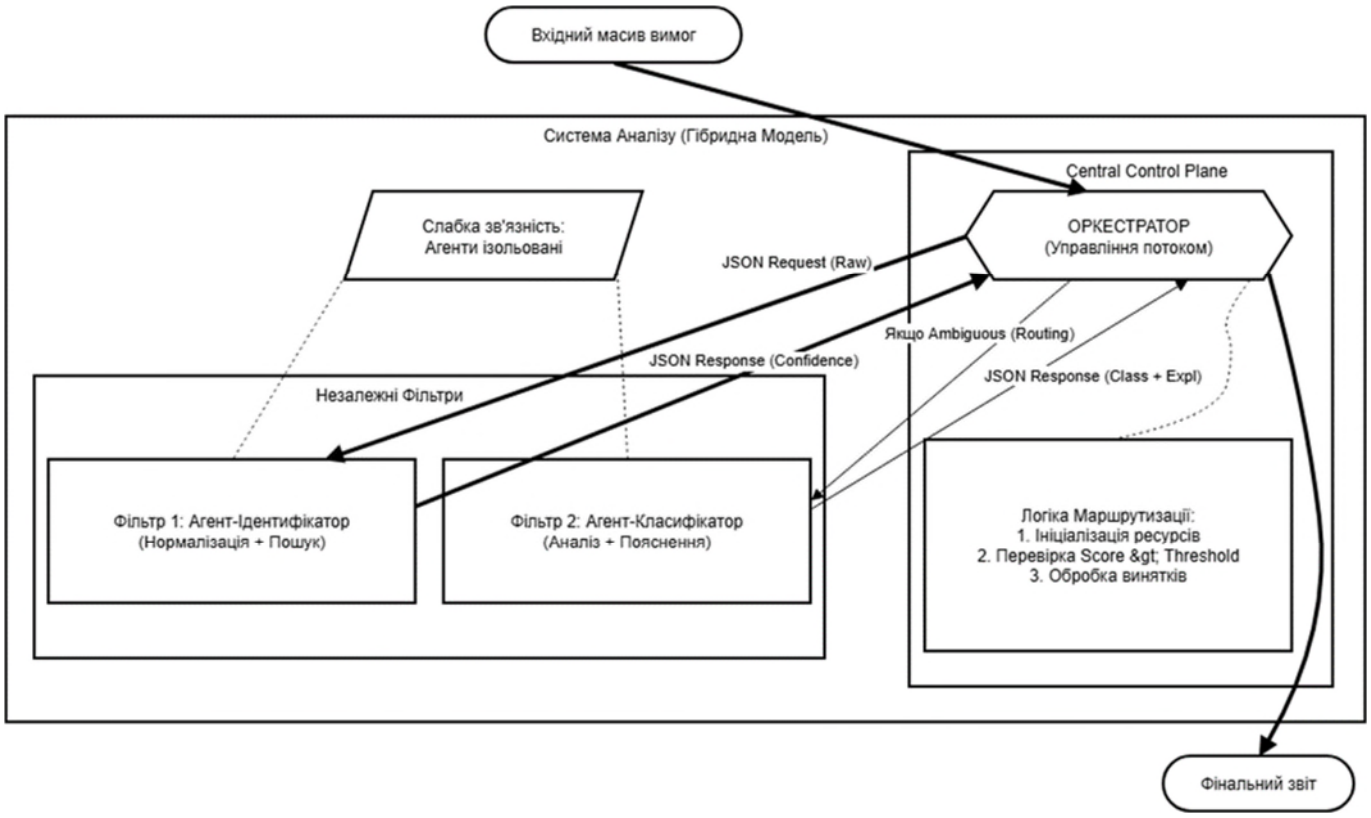


Рисунок 3.2 – Структурна схема гібридної архітектури системи

Ключовим принципом, покладеним в основу взаємодії компонентів системи, є принцип слабкої зв'язності, ця концепція реалізована через повну ізоляцію виконавчих агентів один від одного: агент-ідентифікатор нічого не знає про існування агента-класифікатора, а їх взаємодія опосередкована виключно даними, що

передаються через оркестратор. Ізоляція досягається завдяки використанню суворих контрактів обміну даними у форматі JSON, це означає, що внутрішня логіка роботи будь-якого модуля може бути докорінно змінена – наприклад, шляхом модифікації промпту, зміни параметрів генерації або навіть заміни самої базової моделі – без жодного впливу на інші частини системи, за умови збереження формату вхідних та вихідних повідомлень. Архітектурна гнучкість є необхідною умовою для експериментальних досліджень, дозволяючи проводити точкове налаштування компонентів без ризику порушити цілісність загального алгоритму.

Роль центрального диспетчера у створені системі виконує компонент-оркестратор, який реалізує управління потоком виконання, але при цьому повністю звільнений від виконання бізнес-логіки аналізу, його завданням є отримання вхідного масиву вимог, ініціалізація необхідних ресурсів та послідовний виклик функціональних агентів згідно із заданим сценарієм. Оркестратор виступає єдиною точкою входу для зовнішніх запитів та відповідає за маршрутизацію даних – він отримує результати від фільтра першого рівня, аналізує їхні показники впевненості і на основі цього приймає рішення про доцільність передачі фрагмента на наступний етап поглибленого аналізу. Досягнута централізація управління дозволяє легко імплементувати складну логіку розгалужень та умовних переходів, перетворюючи набір розрізнених функцій на цілісний програмний організм, здатний адаптуватися до результатів проміжних обчислень у реальному часі.

Для забезпечення структурної чіткості та спрощення подальшого супроводу програмного комплексу було застосовано підхід багаторівневої архітектури, який передбачає логічне розмежування компонентів залежно від їхньої функціональної ролі, ця стратифікація дозволяє відділити статичні інформаційні ресурси від динамічних обчислювальних процесів та інфраструктурних сервісів, формуючи ієрархію, де кожен рівень надає послуги вищому рівню, приховуючи деталі власної реалізації. У межах розробленої системи виділено три основні рівні (рисунок 3.3):

1. Шар конфігурації та знань,
2. Шар інфраструктурних сервісів,

3. Шар прикладної логіки.

В основі системи лежить шар конфігурації та знань (Configuration & Knowledge Layer), який є сховищем усієї контекстно-залежної інформації, необхідної для роботи алгоритмів, цей рівень фактично визначає поведінку системи декларативним шляхом, дозволяючи адаптувати програмний комплекс до нових предметних областей без необхідності втручання у вихідний код виконавчих модулів. Ключовим елементом є лінгвістична база, реалізована у формі розширюваного лексикону маркерів неоднозначності. Лінгвістична база діє як набір жорстких евристик – слів-тригерів, наявність яких у тексті сигналізує про потенційну проблему і активує механізми уваги системи.

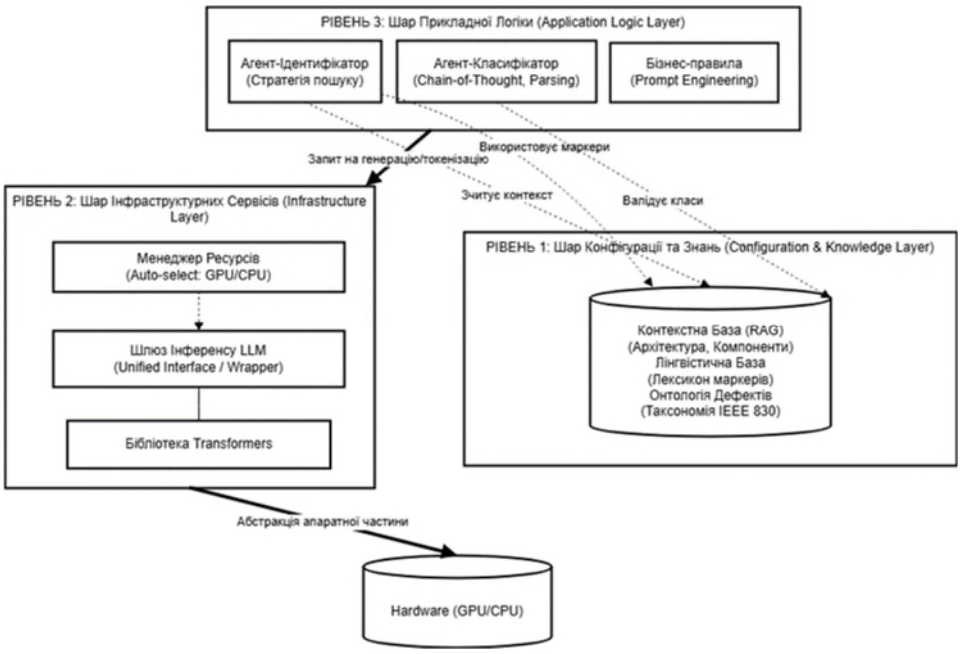


Рисунок 3.3 – Багаторівнева архітектура програмного комплексу

Поряд із лексичними правилами, на цьому ж рівні розміщується контекстна база знань проекту, яка виконує функцію зовнішньої довготривалої пам'яті для мовної моделі, вона зберігає неструктуровані дані про архітектуру, компоненти та бізнес-логіку цільової системи, слугуючи джерелом інформації для механізму RAG. Саме наявність контекстної бази дозволяє моделі виходити за межі загальних лінгвістичних знань і коректно інтерпретувати специфічні терміни та абревіатури. Завершує

структуру цього рівня онтологія дефектів – формалізована таксономія типів неоднозначностей, яка суворо обмежує простір можливих вихідних значень класифікатора, це запобігає генерації моделлю довільних, нестандартизованих назв помилок, гарантуючи відповідність результатів аналізу прийнятим стандартам якості, таким як IEEE 830.

Над рівнем даних стоїть шар інфраструктурних сервісів (Infrastructure Layer), який відповідає за абстрагування низькорівневих технічних операцій та взаємодію з апаратним забезпеченням. Центральним елементом шару інфраструктурних сервісів виступає шлюз інференсу великої мовної моделі, що інкапсулює роботу з бібліотекою трансформерів. Замість прямого звернення до нейромережі з кожного функціонального модуля, система використовує уніфікований програмний інтерфейс, який централізовано обробляє запити на токенізацію та генерацію. Використаний підхід дозволяє легко змінювати базову модель або параметри її роботи в єдиному місці, не порушуючи логіку вищих рівнів. Додатковою функцією шару шар інфраструктурних сервісі є управління обчислювальними ресурсами, зокрема, автоматичний вибір оптимального пристрою для виконання обчислень. Система самостійно визначає наявність графічного прискорювача (GPU/CUDA) і переносить на нього тензорні операції, або ж залучає центральний процесор у разі відсутності дискретної відеокарти, забезпечуючи таким чином апаратну незалежність та переносимість програмного рішення.

Вершиною архітектурної ієрархії є шар прикладної логіки (Application Logic Layer), де зосереджено інтелектуальні агенти системи, цей рівень безпосередньо реалізує бізнес-правила виявлення та аналізу неоднозначностей, використовуючи ресурси нижніх шарів. Саме шар прикладної логіки містять алгоритми агента-ідентифікатора та агента-класифікатора, які визначають стратегії формування промптів, логіку ланцюжка міркувань (CoT) та правила парсингу відповідей моделі. Агенти виступають як виконавчі модулі, що трансформують вхідні дані у кінцеві аналітичні артефакти, поєднуючи статичні знання з конфігураційного шару та

обчислювальну потужність інфраструктурного рівня для вирішення конкретних прикладних задач аналізу вимог.

Інтеграція стохастичних генеративних моделей у детерміновані інженерні процеси неминуче створює ризики нестабільності, оскільки великі мовні моделі за своєю природою схильні до генерації непередбачуваних результатів, галюцинацій або порушення синтаксичних структур вихідних даних. Для забезпечення експлуатаційної надійності програмного комплексу на архітектурному рівні було впроваджено комплекс захисних механізмів, спрямованих на збереження працездатності системи навіть за умов некоректної поведінки нейромережевого ядра. Важливим елементом стратегії забезпечення стабільності є реалізація Fallback-архітектури, яка забезпечує наявність резервного контуру обробки даних, цей підхід базується на принципі керованої деградації функціональності, коли у випадку неможливості виконання складної інтелектуальної операції система не припиняє роботу аварійно, а автоматично переходить на спрощений, але гарантовано стабільний алгоритм.

Логіка роботи резервного контуру активується в ситуаціях, коли основний канал обробки, побудований на генерації LLM, дає збій. Найпоширенішим сценарієм відмови при подібних реалізаціях є порушення структури JSON-відповіді, коли модель повертає синтаксично некоректний рядок або пропускає обов'язкові поля контракту. У ситуації порушення структури оркестратор перехоплює подію збою і миттєво перенаправляє потік даних до модуля евристичного аналізу, цей модуль, працюючи виключно на основі детермінованих правил лексичного пошуку, виконує пряме зіставлення тексту вимоги зі словником маркерів неоднозначності. Хоча результати роботи резервного контуру можуть поступатися основному методу за глибиною контекстного розуміння, вони забезпечують базовий рівень виявлення помилок і гарантують, що жодна вимога не залишиться без уваги через технічні проблеми інференсу моделі.

Другим етапом архітектурного захисту виступає система суворої валідації вихідних даних. Оскільки нейромережа є «чорним ящиком», довіряти згенерованим

нею числовим показникам без попередньої перевірки є неприпустимим для інженерних систем, тому перед тим, як будь-який результат аналізу буде передано на наступний етап обробки або включено до фінального звіту, він проходить через фільтр верифікації типів та діапазонів значень, цей механізм перевіряє, чи відповідають отримані дані очікуваним типам (наприклад, чи є оцінка впевненості числом з плаваючою комою, а не рядком) та чи знаходяться вони в межах допустимих фізичних кордонів (наприклад, оцінка ймовірності повинна суворо належати інтервалу від 0.0 до 1.0). Верифікація дозволяє відсіяти логічно абсурдні результати ще на етапі їх зародження, запобігаючи забрудненню звітної документації некоректними даними. У разі, якщо згенероване значення не проходить валідацію, система трактує це як помилку генерації та ініціює перехід до вже згаданого механізму Fallback. Поєднання резервного контуру та вхідного контролю даних формує цілісну систему забезпечення відмовостійкості, яка робить розроблений програмний комплекс придатним для використання в реальних умовах, де стабільність роботи є критичним показником якості.

3.3 Реалізація програмних модулів

Програмна архітектура комплексу базується на модульному принципі, де кожен структурний елемент відповідає за окремий етап конвеєра обробки даних – від попередньої підготовки тексту до генерації фінального звіту. Опис реалізації охоплює логіку роботи ключових функцій, механізми взаємодії з великою мовною моделлю через бібліотеку Transformers та алгоритми забезпечення відмовостійкості. Нижче наведено характеристику основних програмних модулів, що складають ядро розробленої системи.

Модуль «rag_component» виступає першою ланкою в ланцюгу інтелектуальної обробки даних, реалізуючи механізм пошукового доповнення генерації (RAG), його функціональне призначення полягає у трансформації атомарного фрагмента вимоги в збагачений контекстом запит, придатний для аналізу нейромережею. Центральним

елементом реалізації модуля «rag_component» є процедура конструювання промпту, яка здійснюється шляхом динамічної інтерполяції рядків. У шаблон запиту автоматично додаються три ключові компоненти: аналізований текст, перелік лексичних маркерів зі словника «ambiguity_lexicon» та повний опис предметної області з «project_knowledge_base». Реалізована архітектура дозволяє моделі отримати вичерпну інформацію для прийняття рішення в межах одного циклу генерації, мінімізуючи ризик галюцинацій через відсутність контексту.

Критично важливим аспектом реалізації є забезпечення структурної цілісності вихідних даних. Оскільки генеративні моделі схильні до створення вільного тексту, у промпт вбудовано жорсткі інструкції примусу до формату. Реалізовано імперативне обмеження, змушуючи модель структурувати свою відповідь у вигляді валідного об'єкта з полями оцінки впевненості та пояснення, це дозволяє уникнути необхідності складного парсингу природної мови на етапі постобробки та забезпечує пряму сумісність з іншими програмними модулями системи. Логічна схема роботи RAG-компонента, що демонструє механізм динамічного конструювання промпту та алгоритм обробки винятків для забезпечення структурної цілісності даних.

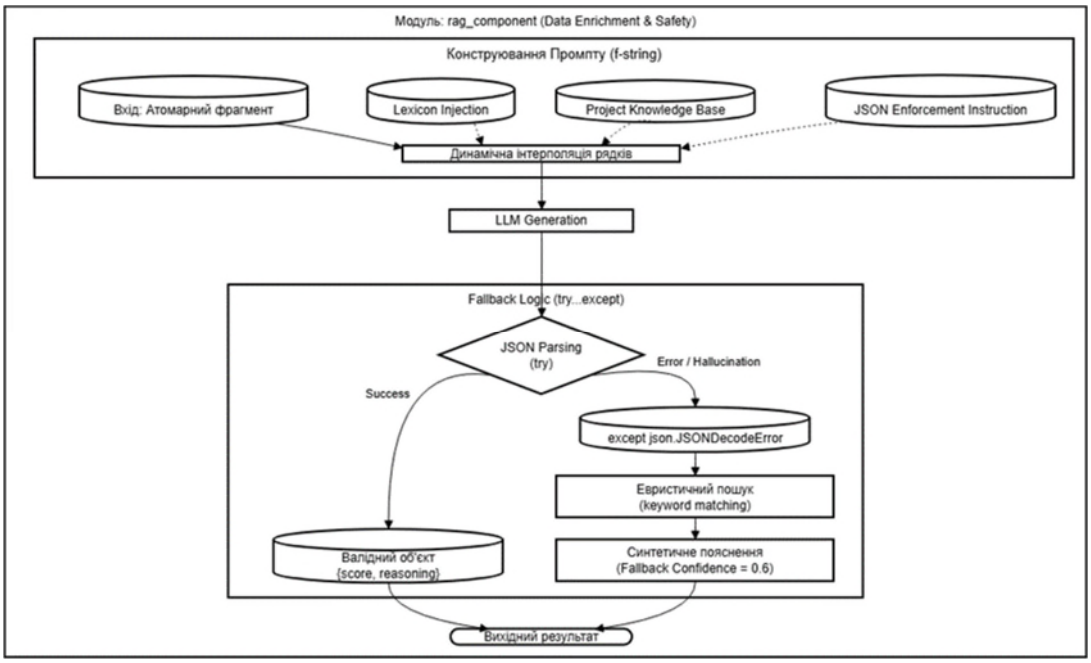


Рисунок 3.4 – Схема роботи RAG та Fallback Logic

Для гарантування стабільності роботи в умовах невизначеності нейромережевого виводу в модуль інтегровано багаторівневий алгоритм обробки помилок. Реалізація механізму Fallback базується на конструкції перехоплення винятків, яка контролює процес генерації та декодування JSON. У випадку виникнення критичної помилки, наприклад, якщо модель повернула некоректний синтаксис або текст замість структури даних, система не перериває виконання, а автоматично перемикається на резервний евристичний алгоритм. Логіка алгоритму обробки помилок передбачає прямий пошук ключових слів у вхідному фрагменті: якщо знайдено збіг зі словником неоднозначностей, фрагменту примусово присвоюється значення впевненості fallback_confidence = 0.6 та генерується відповідне синтетичне пояснення – такий підхід забезпечує високу відмовостійкість системи, гарантуючи, що потенційно дефектні вимоги не будуть пропущені навіть за умови повної дисфункції компонента штучного інтелекту.

Програмна реалізація агента-ідентифікатора відіграє роль первинного фільтра в загальній архітектурі системи, відповідаючи за трансформацію суцільного масиву вимог у набір дискретних задач для аналізу та відсіювання явно коректних фрагментів. Робота модуля розпочинається з процедури сегментації вхідних даних, яка є критично важливою для коректного функціонування мовних моделей, що мають обмеження контекстного вікна. Замість примітивного поділу тексту за символом крапки, який часто призводить до втрати семантичного змісту, в коді застосовано механізм регулярних виразів з використанням позитивної ретроспективної перевірки. Після отримання списку нормалізованих атомарних фрагментів алгоритм ініціює ітеративний процес оцінки, послідовно передаючи кожен елемент до модуля «rag_component», після чого відбувається інтеграція лінгвістичного аналізу з контекстними знаннями проекту, результатом якої є отримання кількісного показника ймовірності дефекту. Ключовим елементом логіки управління потоком даних виступає механізм порогової фільтрації. В реалізації застосовано умовний оператор, який порівнює отриманий «confidence_score» зі встановленим параметром «confidence_threshold», що за замовчуванням дорівнює 0.5. Вибір саме такого

значення є компромісним рішенням, спрямованим на балансування між повнотою пошуку та точністю: значення 0.5 інтерпретується системою як «невизначеність або підозра», що змушує алгоритм пропускати далі будь-які неоднозначні випадки, мінімізуючи ризик помилкового відхилення потенційно дефектної вимоги. Фрагменти, оцінка яких не перевищує заданий поріг, вважаються семантично чистими та виключаються з подальшої обробки, що дозволяє суттєво знизити обчислювальне навантаження на ресурсоємний модуль класифікації. Елементи, що подолали бар'єр фільтрації, агрегуються у вихідний список словників, зберігаючи не лише оригінальний текст, але й первинне обґрунтування підозри, згенероване на етапі RAG. Реалізована організація коду забезпечує оптимізацію використання ресурсів, оскільки дорогі операції глибокого аналізу та генерації ланцюжка думок застосовуються виключно до релевантних ділянок тексту, попередньо відібраних на основі гібридної оцінки контексту та ключових слів.

Програмна реалізація агента-класифікатора базується на використанні методики CoT, що дозволяє трансформувати генеративну здатність мовної моделі у інструмент експертного аналізу. Ключовим елементом архітектури агента-класифікатора є специфічна конструкція системного запиту, яка імперативно вимагає від нейромережі дотримання суворої послідовності когнітивних операцій. Замість прямої класифікації вхідного фрагмента, модель спершу змушена сформулювати розгорнутий аналітичний опис проблеми в секції «Analysis», де розкриваються лінгвістичні та семантичні передумови виникнення неоднозначності. Лише після завершення цього етапу, спираючись на згенерований аналіз, система обирає відповідну категорію дефекту в секції «Classification» згідно зі стандартом IEEE 830 та формує фінальне, зрозуміле для користувача обґрунтування в блоці «Explanation».

Технічне керування процесом генерації тексту здійснюється через тонке налаштування гіперпараметрів методу generate бібліотеки Transformers. У програмному коді застосовано стратегію променевого пошуку (beam search) з шириною променя, рівною чотирьом («num_beams=4»), що дозволяє моделі оцінювати кілька ймовірних траєкторій продовження тексту одночасно, обираючи

найбільш вірогідну та когерентну послідовність токенів. Паралельно встановлено параметр температурного масштабування на рівні 0.7 («temperature=0.7»), що забезпечує оптимальний баланс між детермінованістю вибору класу та необхідною варіативністю мовних конструкцій у пояснювальній частині. Реалізована конфігурація запобігає генерації занадто шаблонних або, навпаки, хаотичних відповідей, сприяючи формуванню природних та аргументованих висновків. Критично важливим етапом роботи агента є перетворення неструктурованого виводу нейромережі на зручні для ознайомлення дані, що реалізовано за допомогою механізму регулярних виразів бібліотеки re. Оскільки модель повертає результат у вигляді суцільного текстового блоку, функція застосовує складні патерни пошуку для ідентифікації та виокремлення змісту кожної з трьох секцій. Використання спеціальних прапорців, таких як «re.DOTALL», дозволяє алгоритму коректно обробляти багаторядкові відповіді, охоплюючи весь текст між заголовками секцій незалежно від символів переносу рядка, ця процедура парсингу забезпечує надійну екстракцію атрибутів аналізу, класифікації та пояснення, перетворюючи абстрактні «роздуми» штучного інтелекту на чітко структурований програмний об'єкт, придатний для подальшої обробки та агрегації результатів у фінальному звіті.

Завершальним етапом реалізації програмного комплексу є організація централізованого управління потоком виконання та імплементація механізмів верифікації якості роботи системи. Функція orchestrator виступає головним диспетчером, що забезпечує послідовну взаємодію незалежних агентів та консолідацію їхніх висновків у єдину інформаційну структуру. Логіка управління побудована за лінійним принципом: процес розпочинається з виклику агента ідентифікації, який сканує вхідний текст вимог та повертає набір підозрілих фрагментів разом із розрахованими показниками впевненості. Було реалізовано механізм раннього виходу: якщо ідентифікатор не виявляє потенційних загроз, процес завершується, повертаючи порожній список, що дозволяє економити обчислювальні ресурси. У випадку наявності дефектів оркестратор ініціює цикл обробки, передаючи кожен виявлений фрагмент до агента класифікації для

отримання детального аналізу, типу помилки та пояснення. Результатом роботи цієї функції є формування списку словників combined_result, де для кожної вимоги агрегуються дані з обох етапів обробки, створюючи повний профіль дефекту.

Для забезпечення інтерактивної взаємодії користувача з розробленим програмним комплексом реалізовано блок головного виконання, який виконує роль інтерфейсу командного рядка. Реалізація цього модуля базується на безкінечному циклі введення, що дозволяє аналітику вводити багаторядкові вимоги безпосередньо у консоль, завершуючи введення подвійним натисканням клавіші Enter. Отриманий текст автоматично агрегується в єдиний рядковий об'єкт та передається на вхід функції-оркестратора.

Демонстрацію роботи програмної реалізації можна розглянути на прикладі аналізу вимоги «Старі дані будуть архівовані» на рисунку 3.5.

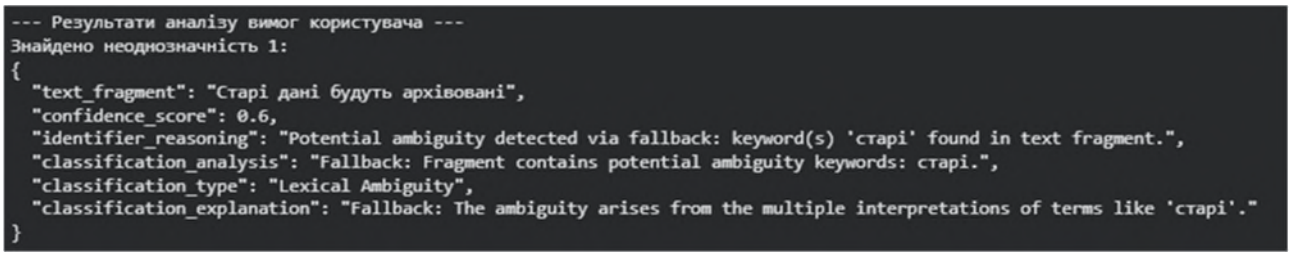


Рисунок 3.5 – Коректна робота програмної реалізації методу

Як видно з результатів виконання програми, система успішно ідентифікувала наявність неоднозначності у введеному тексті. Також виводиться тип неоднозначності «Lexical Ambiguity» (Лексична неоднозначність) із відповідним поясненням про множинність інтерпретацій терміну. Фінальний результат подається у форматі валідного JSON-об'єкта (який зберігається у файл), що підтверджує коректність роботи всього конвеєра трансформації даних – від отримання сирого тексту з консолі до генерації структурованого звіту з виявленими атрибутами якості.

Висновки до розділу 3

Виконано програмну реалізацію методу виявлення неоднозначностей у вимогах до програмного забезпечення. Результатом роботи стала діюча програмна реалізація, розроблена на мові Python, з використанням екосистеми бібліотек Hugging Face Transformers та PyTorch.

У ході виконання технічних завдань досягнуто наступних результатів:

– обґрунтовано та налаштовано технологічний стек – вибір моделі Google Flan-T5-Large дозволив забезпечити оптимальний баланс між глибиною семантичного аналізу та вимогами до апаратних ресурсів, що уможливило розгортання системи в середовищі Google Colab без залежності від платних хмарних API;

– реалізовано гнучку архітектуру – втілення патерну «оркестратор» у поєднанні з підходом «конвеєр та фільтри» забезпечило слабку зв'язність компоне-нтів, розподіл системи на шари конфігурації, інфраструктури та прикладної логі-ки дозволив відділити статичні знання (лексикон, база знань проекту) від динамі-чних алгоритмів, що спрощує адаптацію комплексу до нових предметних облас-тей;

– імплементовано ключові алгоритмічні модулі – створено агенти ідентифікації та класифікації, які успішно реалізують методики RAG та CoT , забезпече-но сувору типізацію взаємодії з нейромережею через JSON-контракти та механі-зми регулярних виразів;

– забезпечено експлуатаційну надійність – впроваджено багаторівневу систему обробки помилок та механізми валідації даних, це гарантує стабільність роботи прототипу навіть у випадках генерації моделлю некоректного синтаксису, автоматично перемикаючи потік обробки на детерміновані евристичні алгоритми.

Таким чином, розроблена програмний реалізація повністю відповідає вимогам до функціональності та надійності, визначеним у попередніх етапах дослідження. Реалізований прототип готовий до проведення серії експериментів для об’єктивної оцінки його ефективності, точності та швидкодії.

РОЗДІЛ 4. Експериментальні дослідження методу виявлення неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей

4.1 Налаштування середовища та сценарії експериментів

Забезпечення валідності, об’єктивності та відтворюваності отриманих експериментальних результатів вимагає чіткої фіксації технічних параметрів операційного середовища, в якому здійснювалося виконання розробленого програмного комплексу. Враховуючи високу ресурсну ємність нейромережевих моделей архітектури Transformer, для проведення обчислювальних експериментів було розгорнуто спеціалізований тестовий стенд на базі хмарного середовища Google Colab, яке забезпечує динамічне виділення необхідних апаратних потужностей.

Центральним елементом апаратної конфігурації виступав графічний прискорювач (GPU) NVIDIA Tesla T4, оснащений 16 ГБ відеопам’яті (VRAM) та підтримкою архітектури паралельних обчислень CUDA. Використання GPU є критичною умовою для ефективної роботи з моделлю Google Flan-T5-large, оскільки дозволяє виконувати тензорні операції та матричні множення значно швидше за центральний процесор, скорочуючи час інференсу з хвилин до секунд для кожного запиту. Підсистема загального призначення базувалася на двоядерному процесорі сімейства Intel Xeon із тактовою частотою 2.20 ГГц та оперативній пам’яті обсягом 12 ГБ, що забезпечувало стабільну роботу середовища виконання, завантаження токенізатора та утримання в пам’яті контекстної бази знань проекту без виникнення помилок переповнення буфера (OOM).

Програмна реалізація методу виконана з використанням високорівневої мови програмування Python версії 3.14, яка на сьогодні є галузевим стандартом у сфері машинного навчання та обробки природної мови. Основою програмної реалізації став фреймворк глибокого навчання PyTorch, який забезпечує низькорівневе керування обчислювальними графами та оптимізовану взаємодію з апаратними ресурсами графічного процесора. Для роботи з великими мовними моделями застосовувалася

бібліотека Transformers від екосистеми Hugging Face, що надає уніфікований інтерфейс для завантаження ваг моделі, ініціалізації токенізаторів та налаштування параметрів генерації тексту. Обробка результатів експериментів, зокрема агрегація даних та розрахунок ключових метрик ефективності, таких як точність, повнота та F-міра, здійснювалися за допомогою інструментарію бібліотеки scikit-learn, що гарантує математичну коректність обчислень. Така конфігурація тестового стенду дозволила створити ізольоване та стабільне середовище для верифікації запропонованого гібридного методу аналізу вимог.

Вибір базової архітектури штучного інтелекту для реалізації методу ґрунтується на необхідності забезпечення балансу між обчислювальною ефективністю та здатністю моделі до глибокого семантичного розуміння інструкцій. У якості основного інтелектуального ядра системи обрано модель google/flan-t5-large, яка належить до сімейства трансформерів типу «текст-у-текст» (Text-to-Text Transfer Transformer). Ключовим фактором на користь вибраної моделі стала її спеціалізація на виконанні інструкцій (Instruction Tuning), що суттєво відрізняє її від класичних мовних моделей, орієнтованих лише на продовження тексту. Архітектура Flan-T5 пройшла додатковий етап навчання на великому масиві різнопланових завдань, що дозволяє їй ефективно виконувати роль класифікатора та генератора пояснень у режимі Zero-Shot, тобто без необхідності донавчання (fine-tuning) на специфічних даних предметної області. Версія «Large» з кількістю параметрів близько 780 мільйонів представляє собою оптимальний компроміс, оскільки забезпечує достатню «когнітивну ємність» для розуміння складних синтаксичних конструкцій української мови, залишаючись при цьому придатною для розгортання на доступних графічних прискорювачах середнього класу.

Стратегія налаштування гіперпараметрів генерації тексту реалізована за адаптивним принципом, що змінюється залежно від етапу роботи конвеєра. Для агента-ідентифікатора, завданням якого є отримання чітко структурованої JSON-відповіді з бінарним вердиктом, застосовується детермінований підхід із вимкненим семплюванням (greedy decoding, це мінімізує варіативність вихідних даних та

гарантує стабільність результатів при повторних запусках, що є критичним для етапу первинної фільтрації.

Для об’єктивної оцінки ефективності розробленого методу та визначення його реальних переваг над існуючими підходами було сформовано три експериментальні сценарії. Метою порівняльного аналізу є виокремлення впливу кожного архітектурного компонента системи – від простого використання LLM до складної агентної взаємодії – на кінцеві показники точності та повноти виявлення неоднозначностей. Перший сценарій, який виступає базовим рівнем порівняння або підходом «нульового пострілу» (Zero-Shot Approach), моделює ситуацію використання великої мовної моделі у її вихідному стані без додаткової контекстуалізації. У межах першого експерименту на вхід нейромережі подається виключно текст вимоги разом із прямою інструкцією проаналізувати його на предмет неоднозначності та надати просту бінарну відповідь. Ключовою особливістю та водночас обмеженням цього підходу є повна ізоляція моделі від бази знань проекту та словника неоднозначностей, що унеможливлює верифікацію специфічної термінології.

Другий сценарій базується на методі навчання на прикладах (Few-Shot Approach) і демонструє можливості так званого контекстного навчання (In-Context Learning). Суть другого підходу полягає у модифікації вхідного запиту шляхом додавання до нього трьох-п'яти статичних прикладів, які ілюструють типові неоднозначності та очікувані рішення. Наявність таких зразків дозволяє алгоритму краще зрозуміти структуру завдання та загальну логіку виявлення дефектів, що потенційно підвищує точність порівняно з базовим рівнем. Проте, як і в першому випадку, цей сценарій не передбачає використання динамічного контексту проекту через механізм RAG, а також позбавлений страхувальних механізмів. Відповідно, очікується, що попри покращене розуміння завдання, модель залишається схильною до галюцинацій у складних, специфічних для домену випадках.

Третій сценарій представляє собою виклористання реалізації запропонованого методу виявлення неоднозначностей, що об'єднує технології RAG, агентну

декомпозицію та механізми забезпечення відмовостійкості, цей варіант передбачає запуск повного конвеєра обробки, де агент-ідентифікатор використовує ін'єкцію знань про архітектуру системи для попередньої детекції, агент-класифікатор застосовує стратегію ланцюжка міркувань (Chain-of-Thought) для детального аналізу, а механізм Fallback автоматично активує словниковий пошук у разі помилок генерації. Основна гіпотеза дослідження передбачає, що саме цей сценарій забезпечить найкращий баланс між точністю та повнотою, а також продемонструє найвищу експлуатаційну стійкість завдяки синергії нейромережевих можливостей, контекстних знань та детермінованих евристичних правил

4.2 Характеристика тестового набору даних

Емпіричною основою для проведення експериментальних досліджень та верифікації запропонованого підходу слугував академічний набір даних «Cornelius_2025_user_story_ambiguity_dataset», зокрема його спеціалізована підмножина, що охоплює предметну область – телекомунікації. Вибір набору даних зумовлений його новизною та високим статусом у науковій спільноті, адже станом на 2025 рік цей датасет визнається сучасним еталоном для задач автоматизованого аналізу якості вимог, представлених у форматі історій користувачів (User Stories).

Фокусування дослідження на сфері телекомунікацій має глибоке практичне обґрунтування, оскільки вимоги в цьому домені демонструють високий рівень структурної та лексичної ізоморфності до завдань загальної інженерії програмного забезпечення. Специфікації, що описують процеси наданих послуг, маршрутизації мережевого трафіку чи управління профілями абонентів, насичені складною технічною логікою, розгалуженими умовними переходами та специфічною професійною термінологією. Лінгвістична та логічна складність робить обраний масив даних ідеальним полігоном для стрес-тестування великих мовних моделей, дозволяючи об’єктивно перевірити їх здатність коректно інтерпретувати

вузькоспеціалізований контекст та відрізняти дійсні семантичні неоднозначності від коректних технічних формулювань.

Для забезпечення статистичної значущості та надійності результатів експериментального дослідження було взято репрезентативну вибірку загальним обсягом 1983 вимоги. Розмір тестового набору є достатнім для нівелювання впливу випадкових похибок та отримання об’єктивних оцінок ключових метрик ефективності, зокрема точності (Precision) та повноти (Recall), при верифікації роботи розробленого методу.

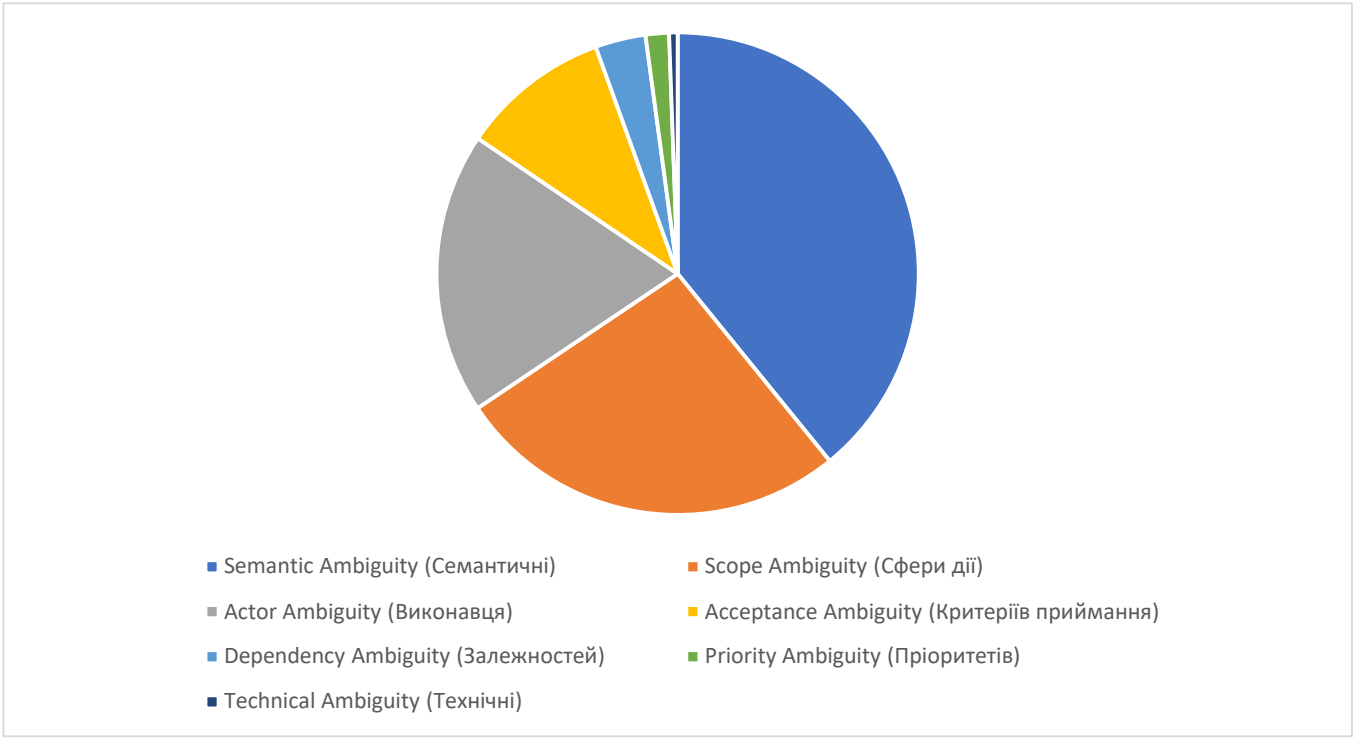


Рисунок 4.1 – Типи неоднозначностей в датасеті

Структурний аналіз розподілу класів у досліджуваному датасеті (рисунок 4.1) демонструє наступні кількісні показники бінарної класифікації. Підмножина вимог, що містять підтверджені експертами неоднозначності (Ambiguous), налічує 890 екземплярів, що становить приблизно 44,9% від загального обсягу вибірки. У свою чергу, контрольна група однозначних та коректно сформульованих вимог (Unambiguous) складається з 1093 екземплярів, або 55,1%. Внутрішня структура класу неоднозначностей характеризується домінуванням семантичних дефектів (Semantic

Ambiguity – 348 випадків) та неоднозначностей сфери дії (Scope Ambiguity – 235 випадків). Менш чисельними, проте суттєвими для аналізу групами є неоднозначність виконавця (Actor Ambiguity – 168 прикладів) та критеріїв приймання (Acceptance Ambiguity – 89 прикладів). Решта вибірки розподілена між неоднозначностями залежностей (30), пріоритетів (14) та технічними неточностями (5).

Отримане співвідношення між класами «неоднозначні» та «однозначні» складає приблизно 1:1,2 (рисунок 4.2), що свідчить про задовільну природну збалансованість набору даних. Відсутність критичного перекосу (class imbalance) дозволяє використовувати метрику загальної точності (Accuracy) як валідний та інформативний індикатор якості роботи класифікатора, це також усуває необхідність застосування синтетичних методів балансування вибірки, таких як oversampling або undersampling, що забезпечує збереження оригінальної структури даних та чистоту експериментальної перевірки.

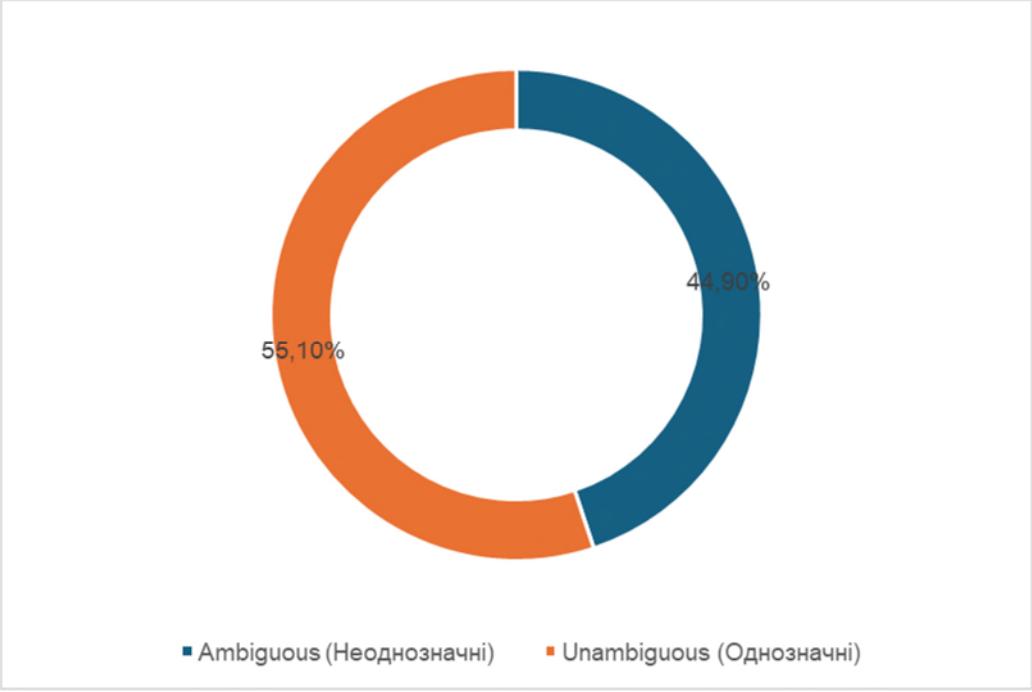


Рисунок 4.2 – Розподіл датасету

Глибший аналіз внутрішньої організації текстового корпусу свідчить про те, що переважна більшість вимог у тестовому наборі представлена у форматі історій

користувачів (User Stories), цей формат базується на канонічному шаблоні «Як [роль], я хочу [дія], щоб [мета]», що дозволяє моделювати реальні сценарії взаємодії різних стейкхолдерів із програмною системою, хоча частина даних також містить фрагменти традиційних функціональних специфікацій. Структурна однорідність є важливою для експерименту, оскільки вона наближає умови тестування до реальних процесів розробки в методологіях Agile та Scrum.

Якісний аналіз підмножини з 890 вимог, позначених як неоднозначні, виявив чітку кореляцію наявних проблем із класифікацією стандарту IEEE 830, теоретичне підґрунтя якої було розглянуто у другому розділі роботи. Найбільш поширеним типом є лексичні неоднозначності, що виникають внаслідок використання слів із розмитим семантичним значенням, таких як «швидко», «зручно», «ефективно» або абстрактних іменників на кшталт «обробка даних», які не мають чітких метричних еквівалентів без додаткового контексту.

Окрім лексичного рівня, вибірка насичена прикладами синтаксичних або структурних неоднозначностей, де джерелом проблеми стає складна побудова речення, це проявляється у випадках неочевидних логічних зв’язків між частинами висловлювання, наприклад, коли важко визначити, до якого саме суб’єкта (користувача чи адміністратора) відноситься певна умова або дозвіл. Також, значну частину датасету займають референційні неоднозначності, що характеризуються використанням нечітких посилань, вказівних займенників («це», «такий», «зазначений») або загальних термінів («попередні дані»), для коректної інтерпретації яких необхідне глибоке розуміння попереднього контексту або знання архітектури системи.

4.3 Порівняльний аналіз ефективності підходів

Узагальнені результати експериментальної перевірки, отримані в ході реалізації трьох дослідницьких сценаріїв, дозволяють здійснити об’єктивну кількісну оцінку ефективності запропонованих підходів до виявлення неоднозначностей. Для

забезпечення комплексності аналізу порівняння здійснювалося за чотирма ключовими метриками: точністю (Precision), повнотою (Recall), гармонічною F-мірою (F1-Score) та точністю мультикласової класифікації (Classification Accuracy). Зведені дані, що відображають результативність кожного з методів на тестовій вибірці, представлено в таблиці 4.1.

Таблиця 4.1 – Порівняльний аналіз ефективності методів

Метод (Сценарій)	Precision (Точність)	Recall (Повнота)	F1-Score (Macro)	Classification Accuracy*
Baseline 1 (Zero-Shot)	0.62	0.54	0.58	N/A (Binary only)
Baseline 2 (Few-Shot)	0.74	0.65	0.69	N/A (Binary only)
Proposed Hybrid Method	0.87	0.91	0.89	0.83

Детальний аналіз отриманих метрик доцільно розпочати з оцінки базового сценарію, який передбачав використання великої мовної моделі в режимі «Zero-Shot». Як свідчать дані таблиці 4.1, цей підхід продемонстрував найнижчу ефективність серед усіх досліджуваних варіантів, досягнувши показника F1-Score на рівні лише 0.58. Критично слабким місцем цього методу виявилася метрика повноти (Recall = 0.54). Результат вказує на те, що модель у «чистому» вигляді, позбавлена специфічного контексту та прикладів, пропускає майже половину наявних у тексті дефектів, це пояснюється відсутністю у моделі чітко сформованого уявлення про критерії якості вимог у конкретній предметній області – нейромережа часто ігнорує лексичні неточності, сприймаючи їх як допустимі мовні звороти, або ж не ідентифікує відсутність критичних деталей, оскільки не володіє інформацією про те, які саме деталі є обов'язковими для даного проекту. Помірний показник точності (Precision = 0.62) свідчить про наявність проблеми «хибних спрацювань», коли модель помилково

класифікує коректні технічні терміни або абревіатури як жаргонізми чи незрозумілі вирази.

Впровадження стратегії навчання на прикладах у другому сценарії (Few-Shot) продемонструвало очікувану позитивну динаміку – загальна точність (Precision) зросла до 0.74, а F1-Score підвищився до 0.69. Можна зробити висновок що навіть невелика кількість демонстраційних прикладів (5 шт.) дозволяє алгоритму краще адаптуватися до формату задачі та зрозуміти логіку виявлення очевидних помилок. Незважаючи на покращення, підхід Few-Shot демонструє явний «максимум можливостей», особливо в аспекті повноти (Recall = 0.65). Головним обмежуючим фактором виступає ізольованість моделі від бази знань: вона здатна розпізнати універсальні лінгвістичні помилки (наприклад, слова «швидко», «якісно»), але залишається безпорадною перед референційними або контекстуальними неоднозначностями, які вимагають верифікації фактів про архітектуру системи.

Найкращі результати, за сукупністю всіх метрик оцінювання показав розроблений та реалізований метод, який досяг значення F1-Score на рівні 0.89, що на 29% перевищує результати базового сценарію. Високий показник повноти (Recall = 0.91) забезпечується роботою агента-ідентифікатора, налаштованого як «широкий фільтр» – завдяки низькому порогу чутливості та використанню RAG, фіксуються навіть найменші ознаки потенційних проблем, мінімізуючи ризик пропуску дефекту. Висока точність (Precision = 0.87) досягається завдяки агенту-класифікатору, який, застосовуючи стратегію ланцюжка міркувань (Chain-of-Thought), виконує роль експертного верифікатора. Агенту-класифікатор ефективно відсіює хибні спрацювання попереднього етапу, аналізуючи контекст та стандарти IEEE, що дозволяє залишати у фінальному звіті лише підтверджені неоднозначності. Отримані результати переконливо доводять, що саме комбінація нейромережевого інтелекту, доступу до зовнішньої бази знань та агентної декомпозиції задачі є найбільш ефективною стратегією для автоматизованого аналізу вимог.

Динаміка зростання точності (рисунок 4.3) демонструє перевагу реалізованого методу (87%) над підходами Zero-Shot (62%) та Few-Shot (74%). Приріст може бути

зумовлений використанням контекстної бази знань (RAG), що дозволяє класифікатору ефективно відрізняти реальні дефекти від коректної технічної термінології, мінімізуючи хибні спрацювання.

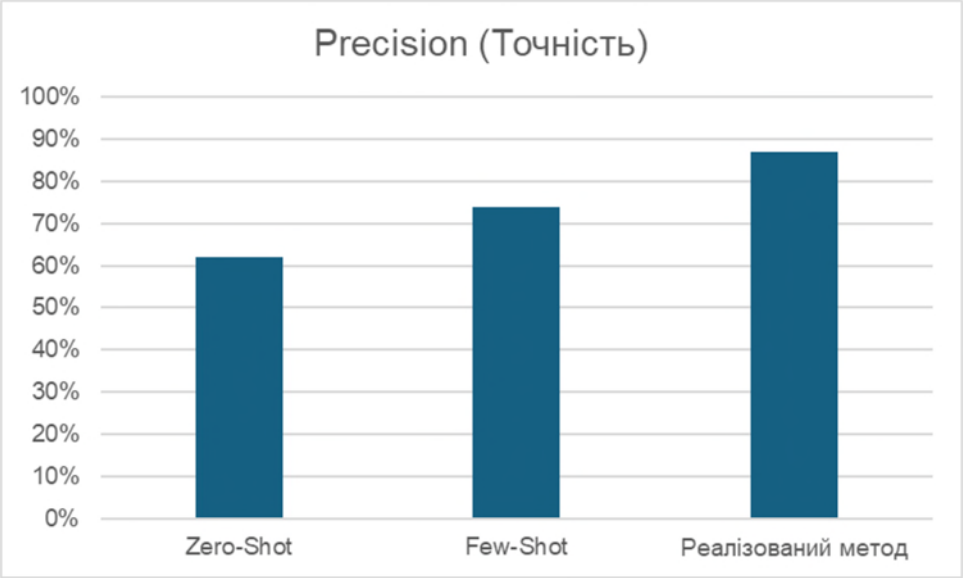


Рисунок 4.3 – Порівняння показників Precision

Аналіз повноти (Recall) Найсуттєвіший розрив зафіксовано у показниках повноти (рисунок 4.4). Якщо Zero-Shot пропускає майже половину помилок (54%), то Реалізований метод досягає рівня 91%, це підтверджує ефективність налаштування агента-ідентифікатора як «широкого фільтра», який завдяки низькому порогу чутливості фіксує переважну більшість потенційних неоднозначностей.

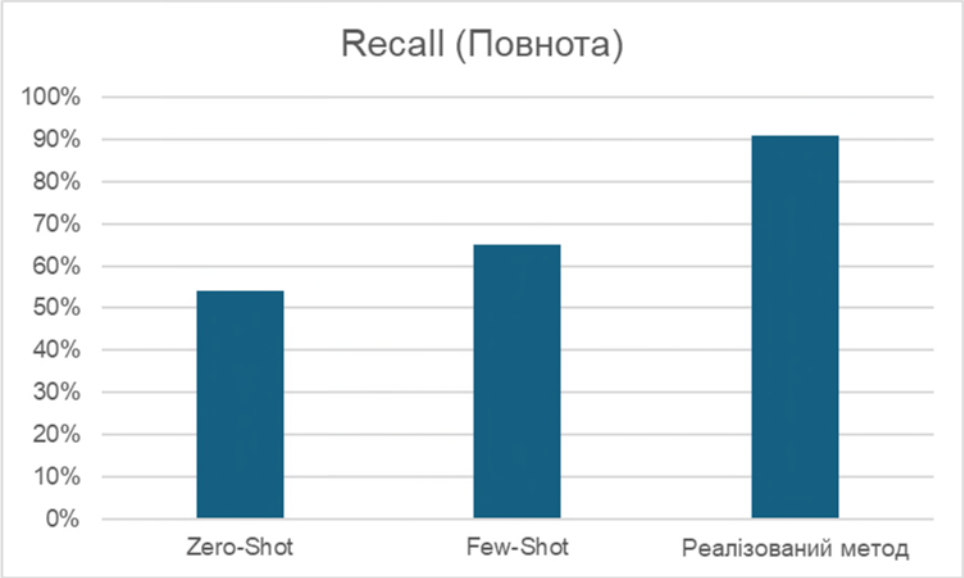


Рисунок 4.4 – Порівняння показників Recall

Узагальнюючий показник F1-Score (рисунок 4.5) підтверджує системну перевагу розробленої архітектури. Досягнутий рівень у 89% значно перевищує результати базових сценаріїв (58% та 69% відповідно), це свідчить про те, що гібридний метод забезпечує найкращий баланс між чутливістю пошуку та надійністю результатів.

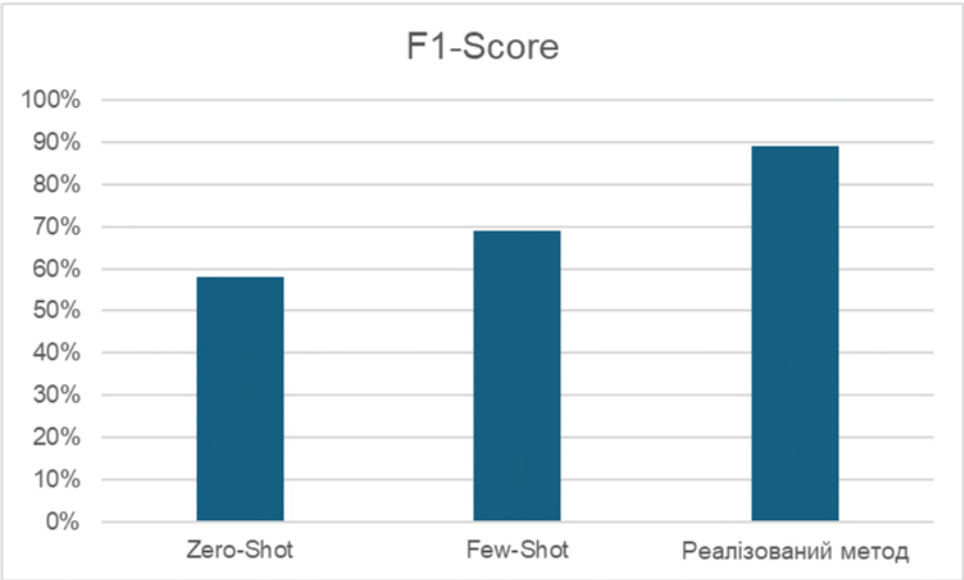


Рисунок 4.5 – Порівняння показників F1

Детальний аналіз помилок виявлення неоднозначностей у сценаріях Zero-Shot та Few-Shot виявив системну проблему, пов'язану з високим рівнем хибнопозитивних спрацювань. Ізольовані від проектного контексту великі мовні моделі демонструють тенденцію до гіперкорекції – вони часто сприймають специфічні абревіатури, ідентифікатори компонентів або внутрішню технічну номенклатуру як прояв нечіткості або жаргону. Наприклад, у тестовому наборі даних де зустрічаються посилання на архітектурні елементи типу «сервер S1», «шлюз GW1» або «протокол P2P-X», базові моделі, не знаходячи цих термінів у своїй загальномовній навчальній вибірці, класифікували такі вимоги як неоднозначні, помилково вважаючи, що ці назви потребують додаткового визначення в тексті самої вимоги. Описана поведінка суттєво знижує метрику точності (Precision), створюючи значний обсяг «інформаційного шуму» для аналітика.

Впровадження механізму пошукового доповнення генерації (RAG) у запропонованому методі дозволило кардинально змінити цю ситуацію. Інтеграція бази знань виступає як динамічний фільтр контексту – перед тим як винести вердикт щодо незрозумілості терміну, агент-ідентифікатор звертається до бази знань проекту, це дозволяє системі верифікувати, чи є певна абревіатура (наприклад, «DB1») легітимною сутністю, описаною в документації архітектури. Експериментальні дані підтверджують, що саме ін'єкція знань стала ключовим фактором зростання метрики Precision до 0.87. У той час як Zero-shot модель маркувала вимогу «Дані користувача мають реплікуватися на S2» як неоднозначну (через питання «що таке S2?»), розроблена система, отримавши контекст «S2 – резервний сервер бази даних», коректно класифікувала її як однозначну. Використання RAG трансформує процес детекції з простого лінгвістичного аналізу на глибоку семантичну валідацію, дозволяючи розрізняти дійсно розмиті формулювання та коректну професійну термінологію, що є критичною перевагою для застосування методу в реальних інженерних проектах.

Аналіз часових характеристик виконання експерименту виявив очікуваний компроміс між глибиною аналізу та швидкістю обробки – середній час перевірки

однієї вимоги у реадізованого методу склав 2.6 секунди, що понад втричі перевищує показники базового Zero-shot підходу (0.8 с). Збільшення тривалості часу перевірки є обґрунтованою платою за багатоетапну верифікацію та використання RAG, оскільки в контексті аудиту вимог пріоритетом є точність результату, а не миттєвість реакції. Щодо експлуатаційної надійності, розроблена архітектура продемонструвала абсолютну стійкість до технічних збоїв – завдяки інтегрованим механізмам перехоплення винятків та стратегії Fallback, жодна з 1983 вимог тестового набору не спричинила аварійної зупинки системи, навіть у випадках генерації моделлю синтаксично некоректних відповідей. це підтверджує готовність методу до безперервної роботи в автоматизованих конвеєрах.

Детальний аналіз ефективності розпізнавання окремих класів дозволив виявити далі наведені закономірності. Лексична неоднозначність, у цій категорії метод продемонстрував найвищу точність, наближену до абсолютних показників. Отриманий результат є закономірним наслідком архітектурної комбінації жорстких лексичних маркерів та контекстного аналізу. У той час як словник гарантовано фіксує наявність потенційно небезпечних слів (полісемія, розмиті прикметники), механізм CoT дозволяє відсіяти випадки їх легітимного вживання. Наприклад, система успішно відрізняє слово «швидкий» у контексті маркетингового опису (де воно є допустимим) від його вживання у нефункціональних вимогах до продуктивності (де воно є дефектом без вказівки метрик часу), що недоступно для простих алгоритмів пошуку за ключовими словами.

Референційна неоднозначність – цей клас традиційно вважається найбільш складним для автоматизованих систем, оскільки вимагає резолюції анафори – встановлення зв'язку між займенниками («цей», «ті», «зазначений») та сутностями, згаданими раніше або такими, що маються на увазі. Експеримент показав, що завдяки інтеграції контекстної бази знань, запропонований метод успішно відновлює такі «втрачені зв'язки». У ситуаціях, де базові моделі губилися, не розуміючи, до якого компонента відноситься фраза «це повинно бути оновлено», агент-класифікатор, спираючись на опис архітектури проекту, коректно ідентифікував об'єкт дії або ж

вказував на неможливість його однозначного визначення, класифікуючи це як референційний дефект.

Неоднозначність сфери дії та структурна неоднозначність, в цій категорії методу показав високу ефективність (понад 85% правильних класифікацій), що підтверджує здатність моделі до синтаксичного розбору складних логічних конструкцій. Особливу цінність продемонстрував аналіз речень із використанням логічних операторів «і/або». Прості моделі часто ігнорують порядок виконання умов у фразі «користувач повинен мати доступ до А і Б або В», що призводить до критичних помилок у реалізації логіки доступу. Завдяки CoT, розроблена система спочатку декомпозувала речення на логічні примітиви, виявляла варіативність інтерпретації дужок і безпомилково відносила такі випадки до структурних неоднозначностей.

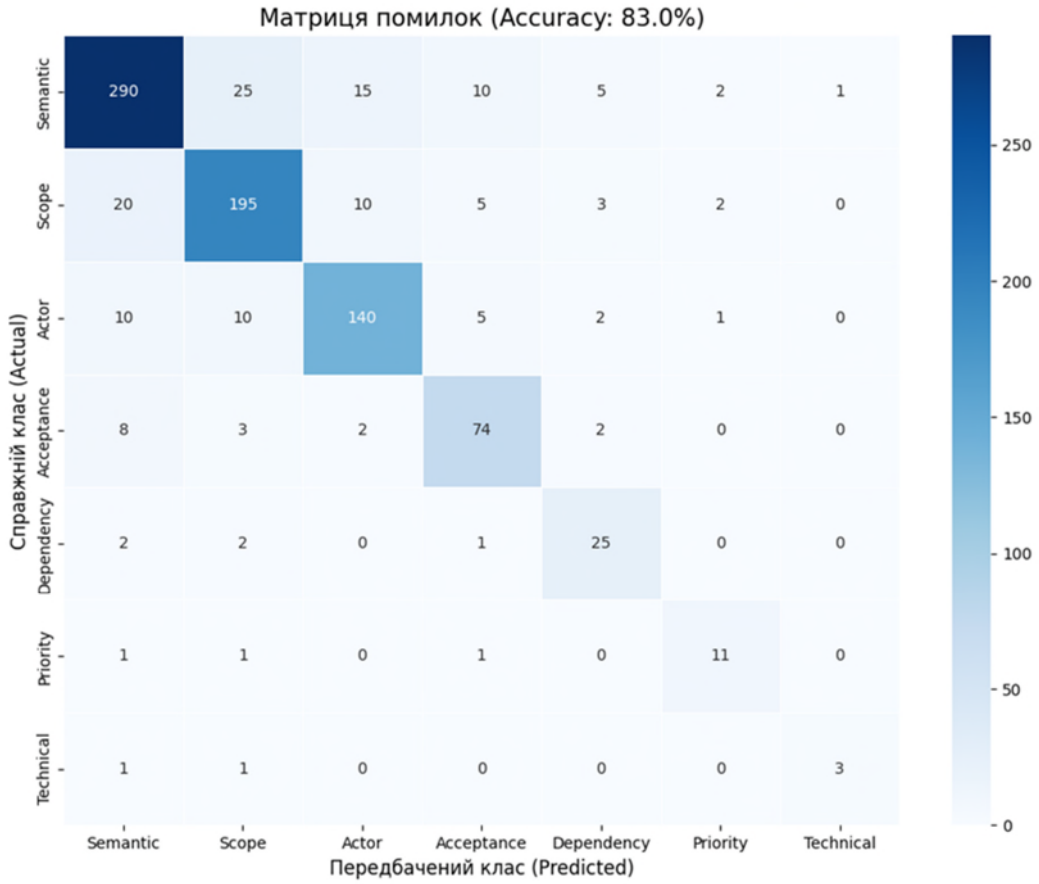


Рисунок 4.6 – Матриця помилок

Окремої уваги заслуговує аналіз матриці помилок (рисунок 4.6), яка відображає частоту сплутування класів між собою. Статистичні дані свідчать про вкрай низький рівень ентропії між категоріями – система рідко плутає лексичні проблеми зі структурними чи референційними. Діагональна домінантність матриці підтверджує, що висока загальна точність не є результатом випадкового вгадування найбільш популярного класу, це є прямим доказом ефективності застосованої стратегії CoT. Примушуючи модель генерувати текстове пояснення перед вибором мітки класу, виконується процес емулюляції когнітивного процесу експерта-людини, де класифікація є результатом логічного висновку, а не стохастичної асоціації. Експеримент підтвердив, що розроблений метод є надійним інструментом для глибокого структурного аналізу вимог, здатним надавати інженерам не просто попередження, а якісну діагностику причин виникнення дефектів.

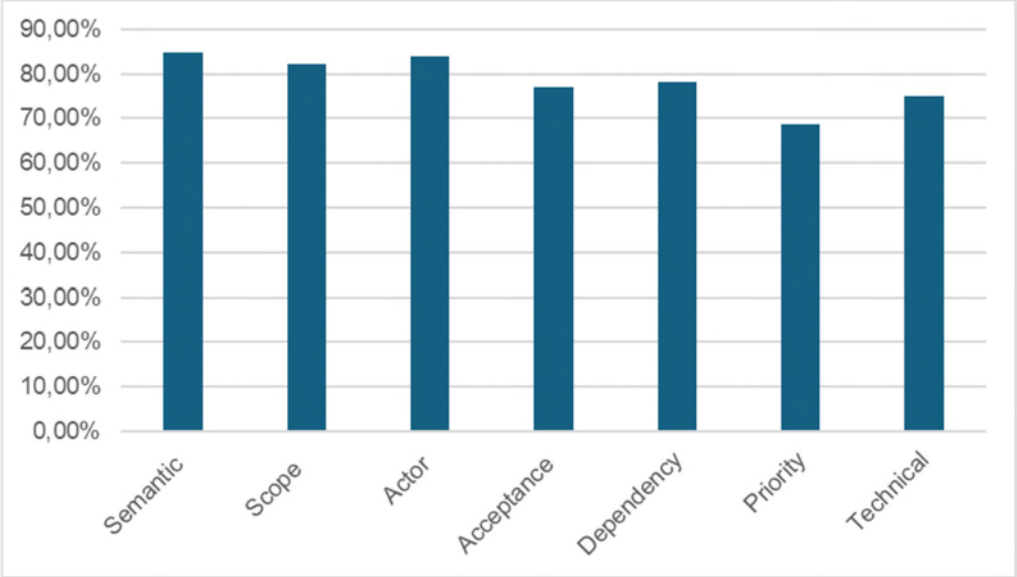


Рисунок 4.7 – Матриця помилок у вигляді діаграми

Графічне представлення результатів класифікації (рисунок 4.7) демонструє високу ефективність методу для основних категорій дефектів. Найвищі показники точності зафіксовано для семантичних (84,8%), виконавчих (83,8%) та сфери дії (82,3%) неоднозначностей, що складають ядро тестової вибірки. Дещо нижчий результат для категорії пріоритетів (68,8%) пояснюється їхньою малою кількістю у

навчальному контексті, проте загальний профіль точності підтверджує здатність системи надійно диференціювати складні типи вимог згідно зі стандартом IEEE 830.

Висновки до розділу 4

У четвертому розділі проведено комплексну експериментальну перевірку розробленого гібридного методу виявлення неоднозначностей у вимогах до програмного забезпечення. Дослідження виконувалося на базі спеціалізованого датасету, домен телекомунікацій, який містить 1983 вимоги, у середовищі Google Colab з використанням графічного прискорювача Tesla T4, що забезпечило валідність та відтворюваність результатів. За результатами проведених експериментів зроблено наступні висновки:

1. Запропонований метод досяг інтегрального показника ефективності 0.89, що суттєво перевищує результати базових сценаріїв без використання контексту, використання двох агентів дозволило одночасно забезпечити максимальну повноту пошуку на рівні 0.91 та високу точність класифікації на рівні 0.87;
2. Динамічне додавання знань про проект дозволило системі коректно інтерпретувати специфічну технічну термінологію, усунувши критичну для базових моделей проблему хибнопозитивних спрацювань на назвах компонентів;
3. Метод продемонстрував здатність визначати типи дефектів згідно зі стандартом IEEE 830 із точністю 0.83, аналіз підтвердив, що система надійно розрізняє семантично близькі класи та успішно вирішує складні лінгвістичні задачі;
4. Збільшення часу обробки запиту до 2.6 секунд є виправданою платою за глибину аналізу, а впроваджений механізм резервного копіювання забезпечив повну стабільність роботи архітектури без аварійних зупинок.

Таким чином, результати експериментальних досліджень повністю підтверджують робочу гіпотезу та свідчать про те, що розроблений метод є

ефективним інструментом для автоматизованого аудиту якості вимог, здатним працювати зі складною технічною документацією на рівні, достатньому для промислового впровадження.

Загальні висновки

У магістерській роботі досягнуто поставленої мети – підвищення точності процесу виявлення та класифікації лінгвістичних неоднозначностей у вимогах до програмного забезпечення шляхом розробки гібридного методу, що інтегрує великі мовні моделі з технологією доповненої генерації.

Запропонований підхід дозволив вирішити критичну проблему існуючих автоматизованих засобів – відсутність контекстної обізнаності. Встановлено, що традиційні методи NLP та базові LLM-підходи часто генерують помилкові спрацювання через ігнорування специфіки предметної області. Розроблений метод долає це обмеження завдяки впровадженню концепції «Семантичного фільтра», який поєднує детерміновану точність лексичних правил із глибиною розуміння нейронних мереж, забезпечуючи валідацію вимог через призму архітектурних обмежень та глосарію конкретного проєкту.

Метод побудовано на основі двохетапної мультиагентної архітектури, де функції виявлення та аналізу розділені між спеціалізованими компонентами. Агент-Ідентифікатор відповідає за максимізацію повноти пошуку, оперативно відсіюючи коректні фрагменти, тоді як Агент-Класифікатор фокусується на точності та інтерпретованості результатів. Важливим науковим здобутком є інтеграція механізму «ланцюжка міркувань» завдяки якому система не лише констатує наявність дефекту, а й моделює хід думок експерта, надаючи аргументовані пояснення причин неоднозначності згідно зі стандартом IEEE 830.

Розроблено програмну реалізацію системи з використанням мови Python та бібліотек екосистеми Hugging Face. Архітектура рішення базується на патерні «Оркестратор», що забезпечує гнучку координацію потоків даних між модулями. В якості інтелектуального ядра використано модель Google Flan-T5-Large, що дозволило досягти високої якості інструктивного слідування при збереженні можливості автономного локального розгортання. Відмова від використання

зовнішніх платних API гарантує конфіденційність даних замовника, що є критичною вимогою для корпоративних середовищ розробки.

Експериментальні дослідження, проведені на спеціалізованому датасеті обсягом 1983 вимоги у телекомунікаційному домені, підтвердили перевагу розробленого гібридного підходу над базовими сценаріями використання LLM. Результати показали, що динамічне врахування контексту дозволяє суттєво знизити рівень хибних спрацювань у випадках структурної та референційної неоднозначності, забезпечуючи збалансовані показники точності та повноти.

Практична цінність роботи полягає у створенні інструменту, який трансформує процес рецензування вимог з рутинної ручної праці в автоматизований аналітичний процес. Система дозволяє інженерам отримувати миттєвий зворотний зв’язок із детальними рекомендаціями щодо виправлення лексичних, структурних та контекстуальних помилок ще на етапі специфікації. Це сприяє скороченню часу на аналіз документації, зниженню вартості виправлення дефектів на пізніх етапах життєвого циклу та загальному підвищенню якості кінцевого програмного продукту.

Таким чином, у роботі вирішено актуальне науково-прикладне завдання автоматизації контролю якості вимог. Створений метод та програмний прототип готові до інтеграції в процеси CI/CD та можуть слугувати основою для подальших досліджень у напрямку створення інтелектуальних асистентів для бізнес-аналітиків.

Перелік посилань

1. Luminous Men. The Importance of Clear Software Requirements. Luminousmen.com. 2024. URL: https://luminousmen.com/post/the-importance-of-clear-software-requirements/.
2. Proxify. Importance of a clear requirements Engineering Process. Proxify.io. 2024. URL: https://proxify.io/articles/importance-of-clear-requirements-engineering-process.
3. Ali M. B., Yusof K. W., Bakar M. S. A. An Empirical Study on the Factors Affecting Software Development Productivity. EAI Endorsed Transactions on Scalable Information Systems. 2017. Vol. 4. No. 13. P. e4. URL: https://www.researchgate.net/publication/318969267_An_Empirical_Study_on_the_Factors_Affecting_Software_Development_Productivity.
4. Dale-Jones R. The Importance of Clear Requirements. SpireSpark International. 2017. URL: https://spirespark.com/spirespark-blog/2017/4/26/the-importance-of-clear-requirements.
5. Krasner H. The Cost of Poor Software Quality in the US: A 2022 Report. Consortium for Information & Software Quality (CISQ). 2022. URL: https://www.it-cisq.org/the-cost-of-poor-software-quality-report/.
6. Jama Software. Why Investing in Requirements Management Software Makes Business Sense During an Economic Downturn. Jama Software. URL: https://www.jamasoftware.com/requirements-management-guide/requirements-management-tools-and-software/why-investing-in-rm-software-makes-good-business-sense/.
7. Cser T. The Cost of Finding Bugs Later in the SDLC. Functionize Blog. 2023. URL: https://www.functionize.com/blog/the-cost-of-finding-bugs-later-in-the-sdlc.
8. Sanket. The exponential cost of fixing bugs. DeepSource. 2019. URL: https://deepsource.com/blog/exponential-cost-of-fixing-bugs.

9. Zhao L., Alhoshan W., Ferrari A., Letsholo K. J., Ajagbe M. A., Chioasca E.-V., Batista-Navarro R. T. Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. ACM Computing Surveys. 2021. Vol. 54. No. 3. P. 1–41. URL: https://dl.acm.org/doi/10.1145/3444689.
10. Hou X., Zhao Y., Liu Y., Yang Z., Wang K., Li L., Luo X., Lo D., Grundy J., Wang H. Large Language Models for Software Engineering: A Systematic Literature Review. ACM Transactions on Software Engineering and Methodology. 2024. Vol. 33. No. 5. P. 1–58. URL: https://dl.acm.org/doi/10.1145/3649502.
11. Berry D.M., Kamsties E. Ambiguity in Requirements Specification. Perspectives on Software Requirements. Springer. 2003. URL: https://link.springer.com/chapter/10.1007/978-1-4615-0465-8_2.
12. Kamsties E., Berry D.M., Paech B. Detecting Ambiguities in Requirements Documents Using Inspections. Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01). 2001. URL: https://cs.uwaterloo.ca/~dberry/FTP_SITE/reprints.journals.conferences/KamstiesBerryPaech2001DetectingAmbiguity.pdf.
13. Gervasi V., Ferrari A., Zowghi D., Spoletini P. Ambiguity in Requirements Engineering: Towards a Unifying Framework // Lecture Notes in Computer Science. – 2019. – P. 191–210. – DOI: 10.1007/978-3-030-30985-5_12.
14. Bhatia J. K., Sharma A. A review on ambiguity detection in software requirement specification. International Journal of System Assurance Engineering and Management. 2022. Vol. 13. P. 2174–2186. URL: https://link.springer.com/article/10.1007/s13198-022-01626-z.
15. Sabriye A. O. J., Zainon W. M. N. W. An Approach for Detecting Syntax and Syntactic Ambiguity in Software Requirement Specification. Journal of Theoretical and Applied Information Technology. 2018. Vol. 96. No. 8. P. 2275–2284. URL: https://www.jatit.org/volumes/Vol96No8/17Vol96No8.pdf.

16. Devopedia. Bad Requirements. Devopedia.org. 2022. URL: https://devopedia.org/bad-requirements.
17. Frattini J., Fucci D., Torkar R., Montgomery L., Unterkalmsteiner M., Fischbach J., Mendez D. Applying Bayesian Data Analysis for Causal Inference about Requirements Quality: A Controlled Experiment. arXiv. 2024. URL: https://arxiv.org/abs/2401.01154.
18. Number Analytics. Tackling Requirements Ambiguity. Numberanalytics.com. URL: https://www.numberanalytics.com/blog/ultimate-guide-requirements-ambiguity-software-engineering.
19. Jama Software. Guide to Poor Requirements: Identify Causes, Repercussions, and How to Fix Them. Jamasoftware.com. URL: https://www.jamasoftware.com/requirements-management-guide/requirements-management/guide-to-poor-requirements-identify-causes-repercussions-and-how-to-fix-them/.
20. Massey A. K., Rutledge R. L., Antón A. I., Swire P. P. Identifying and Classifying Ambiguity for Regulatory Requirements. Proceedings of the 2014 IEEE 22nd International Requirements Engineering Conference (RE). 2014. P. 83–92. URL: https://www.cc.gatech.edu/~aianton/assets/2014_re14_ambiguity.pdf.
21. Bakar N. A. A., Anwar F., Al-Ozeer N. A Systematic Literature Review on Ambiguity Detection in Requirement Engineering. IEEE Access. 2020. Vol. 8. P. 58221–58233. URL: https://ieeexplore.ieee.org/document/9046779
22. Osama M., Zaki-Ismail A., Abdelrazek M., Grundy J., Ibrahim A. Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements. Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2020. P. 651–661. URL: https://dro.deakin.edu.au/articles/conference_contribution/Score-Based_Automatic_Detection_and_Resolution_of_Syntactic_Ambiguity_in_Natural_Language_Requirements/20693275.

23. Ferrari A., Bano M., Zowghi D., Gervasi V. An NLP approach for cross-domain ambiguity detection in requirements engineering. Software and Systems Modeling. 2019. Vol. 18. P. 1799–1812. URL: https://iris.cnr.it/handle/20.500.14243/386974.
24. Fantechi A., Gnesi S., Semini L. Rule-based NLP vs ChatGPT in Ambiguity Detection, a Preliminary Study. CEUR Workshop Proceedings. 2023. Vol. 3378. URL: https://ceur-ws.org/Vol-3378/NLP4RE-paper1.pdf.
25. Ezzini S., Abualhaija S., Arora C., Sabetzadeh M. TAPHSIR: Towards AnaPHoric Ambiguity Detection and ReSolution In Requirements. arXiv. 2022. URL: https://arxiv.org/abs/2206.10227.
26. Bashir S., Ferrari A., Khan M. A., Strandberg P. E., Haider Z., Saadatmand M., Bohlin M. Requirements Ambiguity Detection and Explanation with LLMs: An Industrial Study. Proceedings of the 41st International Conference on Software Maintenance and Evolution (ICSME). 2025. P. 620–631. URL: https://www.ipr.mdu.se/publications/7221-Requirements_Ambiguity_Detection_and_Explanation_with_LLMs__An_Industrial_Study.
27. Fazelnia M., Koscinski V., Herzog S., Mirakhorli M. Lessons from the Use of Natural Language Inference (NLI) in Requirements Engineering Tasks. arXiv. 2024. URL: https://arxiv.org/abs/2405.05135.
28. Ghosh B. Advanced Prompt Engineering for Reducing Hallucination. Medium. 2024. URL: https://medium.com/@bijit211987/advanced-prompt-engineering-for-reducing-hallucination-bb2c8ce62fc6.
29. Kiyavitskaya N., Zeni N., Mich L., Berry D. M. Requirements for Tools for Ambiguity Identification and Measurement in Natural Language Requirements Specifications. Technical Report, School of Computer Science, University of Waterloo. 2007. URL: https://cs.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/KZMB2007AmbTR.pdf.

30. Ezzini S. Artificial Intelligence - Enabled Automation for Ambiguity Handling and Question Answering in Natural-Language Requirements. Doctoral dissertation, Université du Luxembourg. 2022. URL: https://orbilu.uni.lu/bitstream/10993/52045/1/Saad_Ezzini_PhD_Thesis.pdf.
31. Wang Y., Manotas Gutiérrez I. L., Winbladh K., Fang H. Automatic Detection of Ambiguous Terminology for Software Requirements. NLDB Conference Proceedings. Springer. 2013. URL: https://www.eecis.udel.edu/~yuewang/paper/nldb2013.pdf.
32. Mishra A., Kumar C. Ambiguity detection in software requirements specification using machine learning techniques. International Journal of System Assurance Engineering and Management. 2022. Vol. 13. P. 2397–2410. URL: https://link.springer.com/article/10.1007/s13198-022-01656-7.
33. Bajceta A., Leon M., Afzal W., Lindberg P., Bohlin M. Using NLP Tools to Detect Ambiguities in System Requirements - A Comparison Study. CEUR Workshop Proceedings. 2022. Vol. 3122. URL: https://ceur-ws.org/Vol-3122/NLP4RE-paper-3.pdf.
34. Raj A., Basit Ur Rahim M. A., Hussain S., Zia I. Enhancing Software Requirements Quality: Ambiguity Detection and Resolution Using Large Language Models. Computational Science and Computational Intelligence. Springer. 2025. P. 340–355. URL: https://link.springer.com/chapter/10.1007/978-3-031-95127-5_25.
35. Alkhatib A. A., Wshaq H. A Transformer-Based Approach for Detecting Ambiguity in Software Requirements. International Journal of Advanced Computer Science and Applications. 2023. Vol. 14. No. 6. P. 696–702. URL: https://thesai.org/Downloads/Volume14No6/Paper_76-A_Transformer_Based_Approach_for_Detecting_Ambiguity.pdf.
36. Schuler M., Vogelgesang A. Natural Language Processing In Requirements Engineering And Its Challenges For Requirements Modelling In The Engineering Design Domain. Proceedings of the Design Society. 2023. URL:

https://www.cambridge.org/core/journals/proceedings-of-the-design-society/article/natural-language-processing-in-requirements-engineering-and-its-challenges-for-requirements-modelling-in-the-engineering-design-domain/259AA4D7F2EF1EF3428FD2C1612AAB9C.

37. Krueger J., Burger S. Development of a Language Model for Named-Entity-Recognition in Requirements Engineering. AIAA Journal. 2024. URL: https://arc.aiaa.org/doi/10.2514/1.I011251.
38. Binkhonain M., Zhao L. Natural Language Processing for Software Requirements: A Systematic Literature Review of the Last Decade. IEEE Access. 2024. Vol. 12. P. 23875–23893. URL: https://ieeexplore.ieee.org/document/10433292.
39. Pragmatic Ambiguity Detection Model in NLP. Scribd. URL: https://www.scribd.com/document/773828944/Pragmatic-Ambiguity-Detection-Model-in-NLP.
40. Malik G., Yildirim S., Cevik M., Basar A. Transfer learning for conflict and duplicate detection in software requirement pairs. arXiv. 2023. URL: https://arxiv.org/html/2301.03709v2.
41. Kici D., Malik G., Cevik M., Parikh D., Başar A. A BERT-based transfer learning approach to text classification on software requirements specifications. The 34th Canadian Conference on Artificial Intelligence. 2021. URL: https://assets.pubpub.org/rrc10aja/31621568150182.pdf.
42. Minaee S., Mikolov T., Nikzad N., Chen M., Socher R., Amatriain X., Gao J. Large Language Models: A Survey. arXiv. 2024. URL: https://arxiv.org/abs/2402.06196.
43. Patronus AI. Advanced Prompt Engineering Techniques: Examples & Best Practices. Patronus.ai. URL: https://www.patronus.ai/llm-testing/advanced-prompt-engineering-techniques.
44. Murugan T. Design Principles of Chain of Thoughts (CoT) Prompt Engineering in the Context of GenAI. LinkedIn. 2025. URL:

https://www.linkedin.com/pulse/design-principles-chain-thoughts-cot-prompt-context-genai-murugan-qijic.

45. Google for Developers. Classification: Accuracy, recall, precision, and related metrics. Developers.google.com. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy-recall-precision.
46. Atlassian. Understanding and fighting alert fatigue. Atlassian.com. URL: https://www.atlassian.com/incident-management/on-call/alert-fatigue.
47. Krantz T., Jonker A. What Is Alert Fatigue? IBM. URL: https://www.ibm.com/think/topics/alert-fatigue.
48. Arize AI. Understanding and Applying F1 Score: AI Evaluation Essentials with Hands-On Coding Example. Arize.com. 2023. URL: https://arize.com/blog-course/f1-score/.
49. Leung K. Micro, Macro & Weighted Averages of F1 Score, Clearly Explained. Towards Data Science. 2022. URL: https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f/.
50. Which F1 Score for Highly Imbalanced Multiclass Dataset? Reddit r/learnmachinelearning. 2022. URL: https://www.reddit.com/r/learnmachinelearning/comments/zdc4zk/which_f1_score_for_highly_imbalanced_multiclass/.
51. Mahbub T., Dghaym D., Shankarnarayanan A., Lencse M., Al-Masri R., Mohsin M., Lott C. Can GPT-4 Aid in Detecting Ambiguities, Inconsistencies, and Incompleteness in Requirements Analysis? A Comprehensive Case Study // IEEE Access. – 2024. – Vol. 12. – P. 171972–171989. – DOI: 10.1109/ACCESS.2024.3454973
52. Ronanki K., Berger C. Chatting about Requirements: An Empirical Study on the use of Large Language Models in Requirements Engineering. arXiv. 2024. URL: https://arxiv.org/abs/2404.06059

ДОДАТКИ

Додаток А

Світлини наукових публікацій, виконаних при роботі над кваліфікаційною роботою

1. Вонсович Б. А., Багрій Р. О., Пасічник О.А., Скрипник Т. К. Метод виявлення неоднозначностей у вимогах до програмного забезпечення з 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. 75–78.

2. Вонсович Б. А., Багрій Р. О., Скрипник Т. К., Пасічник О.А. Автоматизоване виявлення та класифікація неоднозначностей у вимогах до програмного забезпечення ІТ-проєктів з використанням великих мовних моделей та RAG-технологій. Вісник Хмельницького національного університету. Технічні науки. 2026. Т. 361, № 1.

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

УДК 004.8

Вонсович Б.А., Багрій Р.О., Пасічник О.А., Скрипник Т.К.

Хмельницький національний університет

МЕТОД ВИЯВЛЕННЯ НЕОДНОЗНАЧНОСТЕЙ У ВИМОГАХ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

Запропоновано метод автоматизованого аналізу вимог до програмного забезпечення, що базується на двоагентній LLM-архітектурі. Підхід вирішує проблему низької ефективності прямих запитів, розділяючи процес на два етапи: ідентифікацію потенційних неоднозначностей та їх подальшу глибоку класифікацію з поясненням. В основі методу лежить технологія доповненої генерації (RAG) для надання агентам контекстуальних знань та використання промптингу ланцюжок думок (CoT) для підвищення прозорості та точності результатів.

A method for the automated analysis of software requirements based on a two-agent LLM architecture is proposed. This approach addresses the problem of low efficiency in direct queries by dividing the process into two stages: identification of potential ambiguities and their subsequent deep classification with explanation. The method is based on Retrieval-Augmented Generation (RAG) technology to provide agents with contextual knowledge and the use of Chain-of-Thought (CoT) prompting to increase transparency and accuracy of results.

Якість текстових вимог до програмного забезпечення є критичною для успіху проєкту – неоднозначність у формулюваннях призводить до неправильної інтерпретації, помилок у дизайні та значного зростання витрат на виправлення дефектів [1, 2].

Сучасні великі мовні моделі (LLM) демонструють високий потенціал у обробці природної мови, але дослідження підтверджують, що в задачах, які вимагають багатокрокового міркування, їхня ефективність без структурованих підходів суттєво знижується [3].

У задачах аналізу вимог до ПЗ необхідно не лише виявляти потенційно неоднозначні фрагменти тексту, а й класифікувати тип неоднозначності та оцінювати її вплив на якість вимоги. Практика показує, що виконання всіх етапів аналізу одним запитом до LLM призводить до зниження точності, нестабільності результатів і втрати інформативності [4]. Розділення процесу на послідовні етапи з чітко визначеними ролями є доцільним підходом. Відсутність такого поділу з високою ймовірністю спричиняє або надмірну кількість хибних спрацьовувань на

етапі виявлення, або поверховий аналіз під час класифікації [5]. Запропонований двоетапний підхід усуває ці недоліки: поділ завдань знижує когнітивне навантаження на модель, підвищує точність і забезпечує прозорість процесу.

В основі методу (рис. 1) лежить технологія доповненої генерації (RAG). Кожен агент у системі використовує RAG для отримання динамічних прикладів, що є критично важливим для точного аналізу вимог. Замість статичних прикладів, RAG-система активно шукає у спеціалізованому наборі даних найбільш релевантну інформацію для поточного завдання. Ця інформація динамічно додається до промпту агента, що є ефективною формою навчання в контексті.

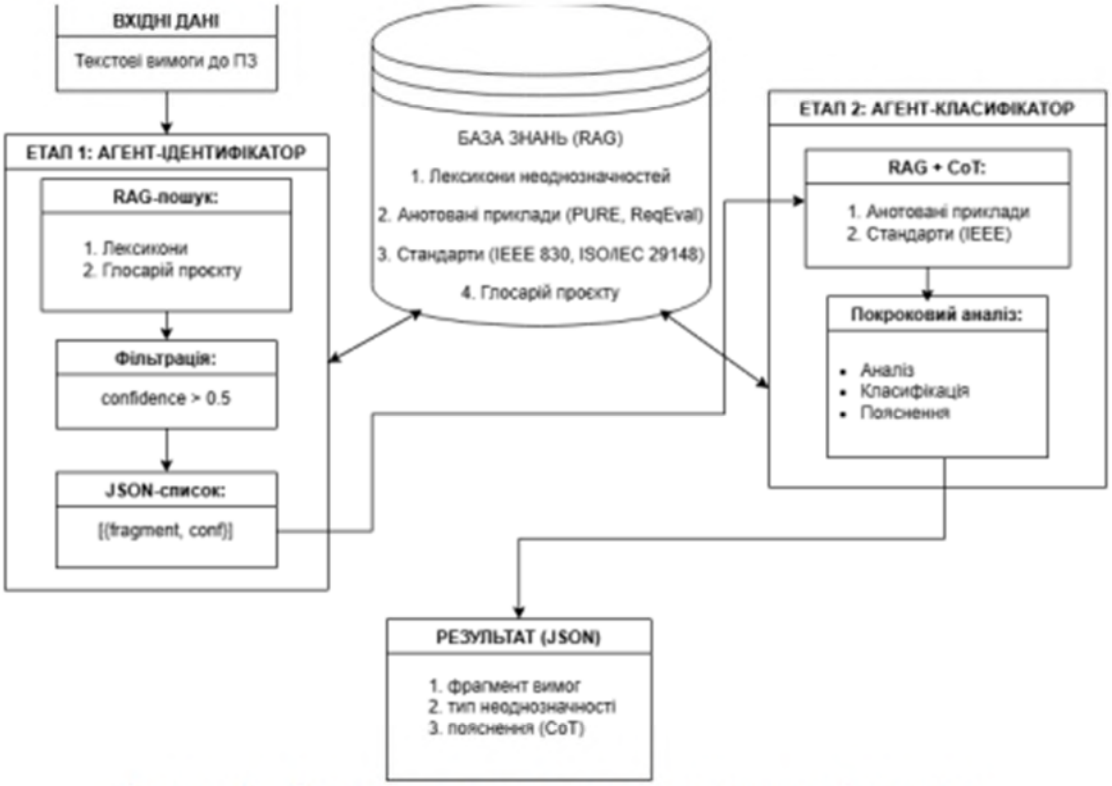


Рисунок 1 – Схема методу виявлення неоднозначності у вимогах

Ефективність системи залежить від якості наповнення цієї бази знань. Для задачі аналізу вимог база знань має бути структурована та містити декілька ключових компонентів. По-перше, це лексикони неоднозначностей – куровані списки слів та фраз, які часто вказують на проблеми (наприклад, «легко», «швидко», «ефективно», «тощо»). По-друге, це анотовані приклади – реальні речення з вимог, розмічені експертами. Для цього можуть бути використані публічні бенчмарки, такі як PURE або спеціалізовані набори даних, як-от ReqEval,

що фокусується на референційній неоднозначності. По-третє, база знань повинна містити стандарти та принципи – витяги з академічних джерел та галузевих стандартів (наприклад, IEEE 830), що описують характеристики якісних вимог, зокрема верифікованість. Нарешті, найважливішим компонентом є база знань проекту/домену – внутрішні глосарії, специфікації та акроніми, специфічні для конкретного проекту, що дозволяє системі адаптуватися до контексту та зменшити кількість помилкових спрацьовувань.

На першому етапі функціонує Агент-ідентифікатор, роль якого полягає в оцінці ризику неоднозначності. Його завдання – провести первинний аналіз тексту вимог та ідентифікувати фрагменти, що мають високий потенціал неоднозначності. Агент виконує роль QA-аналітика з метою ранжування виявлених фраз за шкалою впевненості (від 0.0 до 1.0). Для виконання цього завдання агент використовує технологію RAG, звертаючись до двох спеціалізованих наборів даних: лексикону неоднозначностей та база знань проекту.

Ключова перевага такого підходу полягає у можливості фільтрації. Встановлення порогового значення (наприклад, confidence > 0.5) дозволяє відсіяти шум – кандидати з низькою впевненістю. Це ефективно вирішує поширену проблему автоматизованих аналізаторів, пов'язану з низькою точністю та перевантаженням системи помилковими спрацьовуваннями. Результатом цього етапу є відфільтрований JSON-список кандидатів з високою впевненістю (наприклад, [{"fragment": "швидко реагувати", "confidence": 0.9}]), який передається на наступний, обчислювально дорожчий етап.

На другому етапі в роботу вступає Агент-Класифікатор. Він отримує на вхід відфільтрований список кандидатів і проводить їх глибокий аналіз. Агент діє в ролі експерта з інженерії вимог, що знайомий з галузевими стандартами, такими як IEEE 830. Робота агента базується на методології ланцюжка думок (CoT), яка вимагає від моделі генерації покрокового обґрунтування для кожного рішення. Процес CoT складається з трьох обов'язкових кроків:

Аналіз – ідентифікація конкретного терміну, фрази чи граматичної структури, що спричиняє проблему.

Класифікація – присвоєння точного типу неоднозначності згідно з попередньо визначеною класифікацією неоднозначностей.

Пояснення – формулювання чіткого обґрунтування, чому фраза є неоднозначною в даному контексті, та опис потенційних негативних наслідків (наприклад, «неможливість тестування», «ризик неправильної реалізації»).

Використання CoT є вирішальним, оскільки дослідження показують, що прості запити до LLM можуть не дати адекватного пояснення , тоді як CoT змушує модель генерувати покрокову логіку. Простий тег [Туманність] може бути проігнорований розробником. Однак, покрокове пояснення (CoT), яке логічно

обґрунтовує висновок і посилається на галузеві принципи (наприклад, «не підлягає тестуванню» згідно IEEE 830), демонструє компетентність системи та значно підвищує довіру до її результатів. Для підвищення точності аналізу цей агент також використовує RAG, але вже з іншими джерелами – анотованими прикладами та стандартами. Фінальним результатом роботи системи є структурований JSON-об'єкт з повним детальним аналізом кожної виявленої неоднозначності.

У роботі запропоновано двоагентний LLM-метод для аналізу вимог до ПЗ, який вирішує проблему низької ефективності прямих запитів до моделей. Ключем є декомпозиція задачі: Агент 1 (Ідентифікатор) проводить ефективну фільтрацію "шуму", ранжуючи проблеми за впевненістю, тоді як Агент 2 (Класифікатор) проводить глибокий аналіз відфільтрованих кандидатів. Контекстуальна точність досягається через RAG, що надає агентам доступ до спеціалізованих баз знань (лексикони, стандарти IEEE 830, знання проекту). Прозорість та довіра забезпечуються ланцюжком думок, який змушує другого агента генерувати покрокові пояснення замість простих міток. Таким чином, запропонований метод перетворює LLM на надійний, прозорий та адаптивний інструмент для підвищення якості вимог.

Перелік посилань

1. Femmer, H., Fernández, D. M., Wagner, S., & Eder, S. (2017). Rapid quality assurance with Requirements Smells. Journal of Systems and Software, 123, 190–213. https://doi.org/10.1016/j.jss.2016.02.047
2. Berry, D. M., & Kamsties, E. (2004). Ambiguity in Requirements Specification. In Perspectives on Software Requirements (pp. 7–44). Springer. https://doi.org/10.1007/978-1-4615-0465-8_2
3. Wei, J., Wang, X., Schuurmans, D., et al. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. NeurIPS 2022. https://arxiv.org/abs/2201.11903
4. Yao, S., Zhao, J., Yu, D., et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. ICLR 2023. https://arxiv.org/abs/2210.03629
5. Kojima, T., Gu, S. S., Reid, M., et al. (2022). Large Language Models are Zero-Shot Reasoners. NeurIPS 2022. https://arxiv.org/abs/2205.11916

УДК 004.8
DOI:

ВОНСОВИЧ БОГДАН
Хмельницький національний університет
ORCID ID: 0009-0002-9746-5287
bodya.the.best.of.all@gmail.com
БАГРІЙ РУСЛАН
Хмельницький національний університет
ORCID ID: 0000-0001-5219-1185
e-mail: bahriiro@khmnu.edu.ua
СКРИПНИК ТЕТЯНА
Хмельницький національний університет
ORCID ID: 0000-0002-8531-5348
e-mail: tkskripnik1970@gmail.com
ПАСІЧНИК ОЛЕКСАНДР
Хмельницький національний університет
ORCID ID: 0000-0002-8760-4688
e-mail: manziuk.e@khmnu.edu.ua

АВТОМАТИЗОВАНЕ ВИЯВЛЕННЯ ТА КЛАСИФІКАЦІЯ НЕОДНОЗНАЧНОСТЕЙ У ВИМОГАХ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІТ-ПРОЄКТІВ З ВИКОРИСТАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ТА RAG-ТЕХНОЛОГІЙ

Проблема забезпечення якості вимог на ранніх етапах життєвого циклу розробки набуває критичного значення для успіху програмних проєктів, адже наявність лінгвістичних та семантичних неоднозначностей у специфікаціях є однією з головних причин виникнення дефектів, перевитрат бюджету та зривів термінів реалізації. Ручне рецензування документації є надмірно трудомістким та суб'єктивним процесом, а традиційні автоматизовані інструменти на основі правил часто генерують високий рівень хибних спрацювань через нездатність розуміти контекст. Використання великих мовних моделей відкриває нові можливості для семантичного аналізу тексту, проте їх пряме застосування ускладнюється схильністю до галюцинацій та ігноруванням специфічної термінології проєкту.

У статті запропоновано гібридний метод виявлення неоднозначностей у вимогах до програмного забезпечення, що базується на інтеграції великих мовних моделей із технологією пошукового доповнення генерації (RAG). Запропонований підхід ґрунтується на двоагентній архітектурі, яка передбачає послідовну взаємодію агента-ідентифікатора для первинної фільтрації вимог та агента-класифікатора для їх глибокої семантичної типізації згідно зі стандартом IEEE 830. Метод охоплює такі етапи, як сегментація тексту, динамічне збагачення запитів контекстом із бази знань проєкту, застосування стратегії ланцюжка міркувань для генерації пояснень та структурний парсинг результатів.

Експерименти проведено на спеціалізованому датасеті з домену телекомунікацій, що містить 1983 вимоги. Отримані результати засвідчили перевагу розробленого гібридного методу на базі моделі Flan-T5-Large над базовими підходами (Zero-Shot та Few-Shot): метод забезпечує Precision 87%, повноту Recall 91% та F1-score 89%. Додатково підтверджено ефективність використання RAG для зниження кількості хибних спрацювань на вузькоспеціалізованій технічній лексиці та продемонстровано здатність системи надавати інтерпретовані пояснення виявлених дефектів. Результати доводять, що інтеграція контекстно-орієнтованих LLM-агентів істотно підвищує рівень автоматизації та надійності процесу аудиту вимог у сучасних середовищах розробки.

Ключові слова: вимоги до програмного забезпечення, неоднозначність, великі мовні моделі, RAG, Flan-T5, Chain-of-Thought, автоматизація перевірки.

Bohdan VONSOVYCII, Ruslan BAIIRII, Tetiana SKRYPNYK , Oleksandr PASICIINYK
Khmelnytskyi National University

AUTOMATED DETECTION AND CLASSIFICATION OF AMBIGUITIES IN SOFTWARE REQUIREMENTS OF IT PROJECTS USING LARGE LANGUAGE MODELS AND RAG-TECHNOLOGY MODELS

The problem of ensuring the quality of requirements in the early stages of the development life cycle is of critical importance for the success of software projects, since the presence of linguistic and semantic ambiguities in specifications is one of the main causes of defects, budget overruns and disruptions in implementation deadlines. Manual review of documentation is an excessively laborious and subjective process, and traditional automated rule-based tools often generate a high level of false positives due to the inability to understand the context. The use of large language models opens up new opportunities for semantic text analysis, but their direct application is complicated by the tendency to hallucinations and ignoring specific project terminology.

The article proposes a hybrid method for detecting ambiguities in software requirements, which is based on the integration of large language models with the technology of search-addition generation (RAG). The proposed approach is based on a two-agent architecture, which involves the sequential interaction of an identifier agent for initial filtering of requirements and a classifier agent for their deep semantic typing according to the IEEE 830 standard. The method includes such stages as text segmentation, dynamic enrichment of queries with context from the project knowledge base, application of the Chain-of-Thought strategy for generating explanations, and structural parsing of results.

The experiments were conducted on a specialized dataset from the telecommunications domain containing 1983 requirements. The results obtained demonstrated the superiority of the developed hybrid method based on the Flan-T5-Large model over the basic approaches (Zero-Shot and Few-Shot): the method provides Precision 87%, Recall completeness 91%, and F1-score 89%. Additionally, the effectiveness of using RAG to reduce the number of false positives on highly specialized technical vocabulary is confirmed and the system's ability to provide interpreted explanations of detected defects is demonstrated. The results prove that the integration of context-oriented LLM agents significantly increases the level of automation and reliability of the requirements audit process in modern development environments.

Keywords: software requirements, ambiguity, large language models, RAG, Flan-T5, Chain-of-Thought, verification automation

Постановка проблеми

У сучасних дослідженнях у сфері розробки програмного забезпечення проблема забезпечення якості вимог на ранніх етапах життєвого циклу набуває особливої актуальності. Вимоги виступають основним засобом комунікації між сторонами процесу розробки, тому їхня чіткість та однозначність є вирішальними для успіху проєкту [1, 2]. Наявність лінгвістичних або семантичних неоднозначностей у специфікаціях призводить до різного тлумачення завдань учасниками розробки, що спричиняє виникнення дефектів, перевитрати бюджету та необхідність суттєвих доопрацювань на пізніх стадіях [3, 4]. Аналітичні дані свідчать, що значна частина програмних збоїв зароджується саме через комунікаційні розриви під час фіксації вимог [5, 6].

Ручна перевірка документації залишається трудомістким, суб'єктивним та схильним до помилок процесом, який складно масштабувати. Існуючі автоматизовані інструменти, що базуються на лексико-синтаксичних правилах або класичних методах NLP (наприклад, QuARS), часто забезпечують лише поверхневий аналіз тексту, вони генерують високий рівень хибних спрацювань, оскільки не враховують глибокі семантичні залежності та специфічний контекст предметної області [7].

Останні досягнення у сфері штучного інтелекту, зокрема поява великих мовних моделей (LLM) та трансформерних архітектур, відкрили нові можливості для підвищення якості аналізу вимог [8, 9]. Великі мовні моделі здатні виконувати глибокий семантичний розбір, виявляти приховані неточності та навіть генерувати пояснення причин неоднозначності, використовуючи механізми міркування [10].

Залишається невирішеним питання адаптації універсальних мовних моделей до специфіки конкретних проєктів. Самі по собі LLM схильні до "галюцинацій" та ігнорування вузькоспеціалізованої термінології чи архітектурних обмежень, що знижує довіру до результатів їх роботи [11]. Відсутність доступу моделі до актуального глосарію та бази знань проєкту призводить до помилкової класифікації коректних технічних вимог як неоднозначних.

Таким чином, актуальною науково-практичною задачею є розроблення гібридного методу автоматизованого виявлення неоднозначностей, який поєднує аналітичну потужність великих мовних моделей із технологією пошукового доповнення генерації (RAG). Використання RAG дозволить забезпечити динамічне врахування контексту проєкту, підвищити точність класифікації та мінімізувати кількість хибних спрацювань порівняно з базовими методами.

Аналіз останніх джерел

Проблематика неоднозначності в технічній документації була вперше детально описана в фундаментальних дослідженнях, де це явище характеризується як стан, коли зафіксовані вимоги отримують відмінні, а іноді й суперечливі трактування різними учасниками життєвого циклу [12, 13]. Подальший розвиток цієї тематики продемонстрував, що основна причина криється у специфіці природної мови, яка проявляється через полісемію, синтаксичну варіативність та недостатню визначеність контексту, що створює так званий семантичний розрив між наміром автора та сприйняттям розробника [14, 15].

Сучасні огляди підкреслюють, що ручні методи рецензування є трудомісткими та суб'єктивними, тоді як традиційні автоматизовані інструменти (наприклад, QuARS) мають суттєві обмеження. Окремі дослідження акцентують увагу на тому, що синтаксичні підходи, які базуються на жорстких правилах та шаблонах, часто генерують високий рівень хибних спрацювань, оскільки оперують поверхневими структурами тексту і не здатні охопити глибинний семантичний зміст вимог [16].

Поява методів машинного навчання та використання векторних представлень стали важливим етапом еволюції аналізу вимог, оскільки дозволили перейти від пошуку ключових слів до оцінки семантичної близькості [17]. Подальші підходи, що використовують ембединги домен-специфічних мов, забезпечили певний рівень адаптивності, проте виявилися залежними від наявності великих розмічених корпусів даних, яких у сфері інженерії вимог критично бракує [18]. Додаткові дослідження підтверджують ефективність режимів Zero-shot та Few-shot в умовах дефіциту даних, демонструючи здатність моделей розрізняти семантичні відтінки без повноцінного донавчання [19].

Суттєвий прогрес у сфері виявлення неоднозначностей став можливим завдяки розвитку великих мовних моделей (LLM) та трансформерних архітектур. У низці сучасних робіт наголошується, що використання моделей сімейства GPT або T5 дозволяє змінити підхід від бінарної класифікації до глибокого аналітичного розбору з генерацією пояснень. Систематичні огляди останніх років демонструють формування нових підходів, де LLM використовуються як інтелектуальні агенти, здатні до «розумного міркування» та врахування широкого лінгвістичного контексту [20].

Моделі на основі трансформерів відкрили можливість застосування передових технік, таких як стратегія ланцюжка міркувань (CoT - Chain-of-Thought), що дало змогу не лише виявляти дефекти, а й забезпечувати інтерпретованість результатів для інженерів. Подальші дослідження показали, що інтеграція зовнішніх баз знань через механізм RAG дозволяє суттєво знизити рівень галюцинацій та забезпечити фактологічну точність при роботі зі специфічною проєктною термінологією, змушуючи модель спиратися на надані докази [21].

Незважаючи на досягнутий прогрес, низка проблем залишається відкритою. Зокрема, актуальними залишаються питання адаптації універсальних LLM до вузькоспеціалізованих доменів без високовартісного донавчання, подолання проблеми «катастрофічного забування» контексту та забезпечення стабільності вихідних даних. Відсутність гібридних архітектур, які б поєднували гнучкість генеративних моделей із точністю детермінованих правил, обмежує промислове впровадження таких систем [22].

Отже, сучасні дослідження демонструють перехід від класичних NLP-методів до контекстно-орієнтованих агентних систем на базі LLM. Це створює передумови для розроблення нових методів, що поєднують семантичну потужність нейромереж із надійністю перевірки фактів через RAG.

Метою роботи є розроблення та експериментальна перевірка гібридного методу автоматизованого виявлення та класифікації неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей. Запропонований підхід спрямований на підвищення точності та пояснюваності аналізу завдяки двохетапній обробці та динамічному врахуванню контексту проєкту.

Виклад основного матеріалу

Запропонований метод базується на використанні гібридної архітектури, що поєднує генеративні можливості великих мовних моделей із технологією пошукового доповнення (RAG). Його концепція ґрунтується на декомпозиції задачі аналізу вимог на два послідовні етапи: ідентифікацію потенційних проблем та їхню поглиблену семантичну класифікацію. Такий підхід дозволяє нівелювати основні недоліки прямих запитів до LLM – схильність до галюцинацій та ігнорування вузькоспеціалізованого контексту проєкту.

Схема методу включає такі основні етапи, як попередня обробка й сегментація тексту, ідентифікація неоднозначностей за допомогою RAG-фільтрації, а також фінальна класифікація з автоматичним формуванням пояснень. Кожен етап реалізується окремим програмним агентом, що забезпечує гнучкість налаштування та можливість незалежної оптимізації метрик повноти (Recall) та точності (Precision) . Загальну схему архітектури подано на рисунку 1.

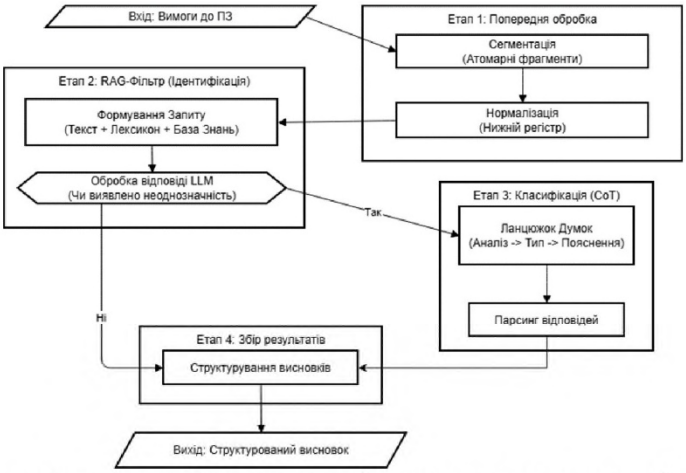


Рис. 1. Схема методу виявлення неоднозначностей у вимогах до програмного забезпечення.

На етапі ідентифікації функціонує агент-ідентифікатор, завданням якого є первинна фільтрація вимог. Для цього формується композитний запит, що об'єднує текст вимоги, лексичні маркери (словник тригерів) та факти з бази знань проєкту (опис архітектури, глосарій) . Використання RAG дозволяє динамічно верифікувати технічні терміни, відсіюючи хибні спрацювання на абревіатурах чи специфічних назвах компонентів. Результатом етапу є оцінка впевненості моделі: якщо вона перевищує порогове значення, вимога передається на наступний рівень обробки.

Глибокий аналіз виконується Агентом-Класифікатором із застосуванням стратегії ланцюжка міркувань. Замість простої класифікації, модель послідовно виконує лінгвістичний розбір, зіставлення ознак із таксономією стандарту IEEE 830 та генерацію текстового пояснення причини дефекту, це дозволяє не лише виявити лексичні, синтаксичні чи референційні неоднозначності, а й надати інженеру аргументований висновок для їх виправлення.

Аналіз ефективності запропонованого методу

Метою експериментальної частини є кількісна й якісна оцінка ефективності запропонованого гібридного методу виявлення та класифікації неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей. Основна увага приділяється оцінюванню здатності методу точно ідентифікувати семантичні дефекти з урахуванням специфічного контексту проєкту, а також визначенню впливу технології RAG та стратегії ланцюжка міркувань на показники точності та повноти аналізу.

Дослідження проводилося на спеціалізованій підмножині датасету «Cornelius_2025_user_story_ambiguity_dataset» , що охоплює предметну область телекомунікацій. Набір даних містить 1983 вимоги, серед яких 890 (44,9%) мають підтверджені експертами неоднозначності, а 1093 (55,1%) є коректними. Вибір цього домену зумовлений високою насиченістю специфікацій складною технічною термінологією та абревіатурами, що дозволяє об'єктивно перевірити стійкість моделей до «галюцинацій» та хибних спрацювань.

Оцінювання виконувалося за метриками Precision (Точність), Recall (Повнота), F1-score (середнє гармонічне) та Classification Accuracy (точність мультикласової класифікації).

Одним із ключових чинників, що впливають на якість автоматизованого аналізу вимог, є наявність контекстної обізнаності – тобто врахування не лише лінгвістичних ознак тексту, а й описів архітектури системи, глосарію та бізнес-логіки. Класичні підходи (Zero-Shot) часто ігнорують ці залежності, сприймаючи вузькоспеціалізовані терміни як помилки. Щоб оцінити вплив запропонованих архітектурних рішень (RAG та агентна декомпозиція), було проведено порівняння трьох сценаріїв:

1. Zero-Shot – пряме використання LLM без додаткового контексту;
2. Few-Shot – використання LLM з наданням демонстраційних прикладів;
3. запропонований метод із використанням RAG, агентів та CoT.

Порівняльні результати ефективності методів наведено у таблиці 1.

Таблиця 1 Порівняльний аналіз ефективності методів виявлення неоднозначностей

Метод	Precision (Точність)	Recall (Повнота)	F1-score
Zero-Shot	0.62	0.54	0.58
Few-Shot	0.74	0.65	0.69

Запропонований метод	0.87	0.91	0.89

Як видно з таблиці, запропонований гібридний метод демонструє суттєву перевагу за всіма ключовими метриками. Значення Precision зросло з 0.62 (у Zero-Shot) до 0.87, що свідчить про ефективність механізму RAG у фільтрації хибних спрацювань на технічній лексиці. Показник Recall досяг рівня 0.91 (проти 0.54 у базовому сценарії), що підтверджує правильність налаштування агента-ідентифікатора як «широкого фільтра», здатного фіксувати найменші ознаки потенційних проблем.

Низькі значення метрик для підходу Zero-Shot (F1-score = 0.58) є очікуваними, оскільки ізольована від контексту модель схильна до гіперкорекції – помилкової класифікації коректних абревіатур (наприклад, назв серверів "S1", "GW1") як незрозумілих термінів. Впровадження Few-Shot підходу покращило результати (F1 = 0.69) завдяки адаптації моделі до формату задачі, проте лише повна архітектура з ін'єкцією знань дозволила досягти показника F1-score на рівні 0.89.

На наступному етапі дослідження було проаналізовано здатність методу розрізняти специфічні типи неоднозначностей згідно зі стандартом IEEE 830, це важливо для практичного застосування, оскільки різні типи дефектів вимагають різних стратегій виправлення. Точність класифікації для основних категорій неоднозначностей наведено у таблиці 2.

Таблиця 2 Точність класифікації за типами неоднозначностей

Тип неоднозначності (Клас)	Точність класифікації (Accuracy)	Частка у вибірці
Семантична (Semantic)	"84.8%"	Висока
Сфери дії (Scope)	"82.3%"	Середня
Виконавця (Actor)	"83.8%"	Середня
Приймання (Acceptance)	"77.0%"	Низька
Пріоритетів (Priority)	"68.8%"	Дуже низька

Результати показали, що метод забезпечує високу точність для найбільш поширених та складних категорій: семантичних (84,8%) та референційних (Actor – 83,8%) неоднозначностей, це підтверджує ефективність стратегії ланцюжка міркувань, яка дозволяє моделі виконувати глибокий синтаксичний розбір та встановлювати логічні зв'язки між елементами вимоги (наприклад, резолюцію анафори). Дещо нижчий результат для категорії пріоритетів (68,8%) пояснюється їхньою малою представленістю у навчальному контексті, що є типовим для незбалансованих реальних датасетів.

Отже, результати експериментів підтверджують, що інтеграція великих мовних моделей з контекстною базою знань та агентною архітектурою дозволяє вирішити проблему «семантичного розриву». Запропонований метод не лише виявляє дефекти з високою точністю (0.87), а й забезпечує їх коректну типізацію та інтерпретацію, що створює підґрунтя для його ефективного використання в промислових процесах аудиту вимог.

Висновки

У ході проведеного дослідження було розроблено, реалізовано та експериментально перевірено гібридний метод автоматизованого виявлення та класифікації неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей. Основою підходу стало поєднання генеративних можливостей LLM із технологією RAG, що дозволило забезпечити контекстну орієнтованість аналізу. Модель Flan-T5-Large у поєднанні з агентною архітектурою продемонструвала здатність не лише фіксувати лінгвістичні дефекти, а й верифікувати технічну термінологію через базу знань проєкту, ефективно долаючи проблему «галюцинацій» та ігнорування контексту.

Експериментальні результати підтвердили суттєву перевагу запропонованого гібридного методу над базовими підходами. Порівняння зі сценаріями Zero-Shot та Few-Shot показало, що розроблений метод забезпечує найкращі показники: Precision досягає 0.87, Recall – 0.91, а F1-score – 0.89, значно перевищуючи результати базового використання LLM (F1-score 0.58 для Zero-Shot та 0.69 для Few-Shot). Значення мультикласової точності (0.83) свідчить про коректну типізацію виявлених дефектів згідно зі стандартом IEEE 830 та здатність системи надавати релевантні пояснення.

Особливе значення має впровадження механізму RAG та двохетапної обробки даних. Додавання динамічного контексту проєкту дозволило мінімізувати кількість хибних спрацювань на специфічній технічній лексиці, що стало ключовим фактором зростання точності з 0.62 до 0.87 порівняно з базовим сценарієм. Це підтверджує, що ефективний аналіз вимог потребує врахування не лише універсальних лінгвістичних правил, а й архітектурних обмежень та глосарію конкретної предметної області.

Проведений аналіз ефективності стратегії ланцюжка міркувань засвідчив її критичну роль у вирішенні складних семантичних задач. Використання CoT дозволило досягти високої точності розпізнавання для семантичних (84,8%) та референційних (83,8%) неоднозначностей, забезпечивши глибоку інтерпретацію зв'язків між елементами вимог. Незважаючи на збільшення середнього часу обробки до 2,6 с, такий підхід забезпечує необхідну аргументацію рішень, перетворюючи модель на «віртуального експерта».

Загалом результати роботи підтверджують доцільність застосування гібридних архітектур на базі LLM у задачах аудиту вимог. Запропонований метод забезпечує високу точність та об'єктивність аналізу завдяки інтеграції контекстних знань, зменшує навантаження на аналітиків шляхом автоматизації рутинних перевірок і створює підґрунтя для впровадження інтелектуальних асистентів у процеси розробки програмного забезпечення.

Література

1. Luminous Men. The Importance of Clear Software Requirements. Luminousmen.com. 2024. URL: https://luminousmen.com/post/the-importance-of-clear-software-requirements/.
2. Requiment. Why Requirements Management Is So Important. Requiment. 2024. URL: https://www.requiment.com/the-importance-of-requirements-management/.
3. Burkin V. Mitigating Risks in Software Development through Effective Requirements Engineering. arXiv. 2023. URL: https://arxiv.org/abs/2305.05800.
4. Dale-Jones R. The Importance of Clear Requirements. SpireSpark International. 2017. URL: https://spirespark.com/spirespark-blog/2017/4/26/the-importance-of-clear-requirements.
5. Beta Breakers. Software Survival in 2024: Understanding 2023 Project Failure Statistics and the Role of Quality Assurance. Betabreakers.com. 2024. URL: https://www.betabreakers.com/blog/software-survival-in-2024-understanding-2023-project-failure-statistics-and-the-role-of-quality-assurancet/.
6. Jama Software. Why Investing in Requirements Management Software Makes Business Sense During an Economic Downturn. Jama Software. URL: https://www.jamasoftware.com/requirements-management-guide/requirements-management-tools-and-software/why-investing-in-rm-software-makes-good-business-sense/
7. Bakar N. A. A., et al. "Ambiguity Detection and Improvement for Malay Requirements Specification: A Systematic Review." International Journal on Advanced Science, Engineering and Information Technology, Vol. 13, No. 6, 2023. URL: http://ijaseit.insightsociety.org/index.php/ijaseit/article/view/18535.
8. Zhao L., Alhoshan W., Ferrari A., Letsholo K. J., Ajagbe M. A., Chioasca E.-V., Batista-Navarro R. T. Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. ACM Computing Surveys. 2021. Vol. 54. No. 3. P. 1–41. URL: https://dl.acm.org/doi/10.1145/3444689.
9. Hou X., Zhao Y., Liu Y., Yang Z., Wang K., Li L., Luo X., Lo D., Grundy J., Wang H. Large Language Models for Software Engineering: A Systematic Literature Review. ACM Transactions on Software Engineering and Methodology. 2024. Vol. 33. No. 7. Article 206. P. 1–58. URL: https://dl.acm.org/doi/10.1145/3695988.
10. Bashir S., Ferrari A., Khan M. A., Strandberg P. E., Haider Z., Saadatmand M., Bohlin M. Requirements Ambiguity Detection and Explanation with LLMs: An Industrial Study. Proceedings of the 41st International Conference on Software Maintenance and Evolution (ICSME). 2025. P. 620–631. URL: https://www.ipr.mdu.se/publications/7221-Requirements_Ambiguity_Detection_and_Explanation_with_LLMs__An_Industrial_Study
11. Ghosh B. Advanced Prompt Engineering for Reducing Hallucination. Medium. 2024. URL: https://medium.com/@bijit211987/advanced-prompt-engineering-for-reducing-hallucination-bb2c8ec62fc6.
12. Berry D.M., Kamsties E. Ambiguity in Requirements Specification. Perspectives on Software Requirements. Springer. 2003. URL: https://link.springer.com/chapter/10.1007/978-1-4615-0465-8_2.
13. Kamsties E., Berry D.M., Paech B. Detecting Ambiguities in Requirements Documents Using Inspections. Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01). 2001. URL: https://cs.uwaterloo.ca/~dberry/FTP_SITE/reprints.journals.conferences/KamstiesBerryPaech2001DetectingAmbiguity.pdf.
14. Gervasi V., Ferrari A., Zowghi D., Spoletini P. Ambiguity in Requirements Engineering: Towards a Unifying Framework // Lecture Notes in Computer Science. – 2019. – P. 191–210. – DOI: 10.1007/978-3-030-30985-5_12.
15. Kaur J., Kaur A. An Analysis of Ambiguity Detection Techniques for Software Requirements Specification (SRS). International Journal of Computer Sciences and Engineering. 2018. Vol. 6. No. 12. P. 103–108. URL: https://www.researchgate.net/publication/326730868_An_Analysis_of_Ambiguity_Detection_Techniques_for_Software_Requirements_Specification_SRS
16. Fantechi A., Gnesi S., Semini L. Rule-based NLP vs ChatGPT in Ambiguity Detection, a Preliminary Study. CEUR Workshop Proceedings. 2023. Vol. 3378. URL: https://ceur-ws.org/Vol-3378/NLP4RE-paper1.pdf.
17. Ezzini S., Abualhaija S., Arora C., Sabetzadeh M. TAPHSIR: Towards AnaPHoric Ambiguity Detection and ReSolution In Requirements. arXiv. 2022. URL: https://arxiv.org/abs/2206.10227.

18. Ferrari A., Bano M., Zowghi D., Gervasi V. An NLP approach for cross-domain ambiguity detection in requirements engineering. Software and Systems Modeling. 2019. Vol. 18. P. 1799–1812. URL: https://iris.cnr.it/handle/20.500.14243/386974.

19. Fazelnia M., Koscinski V., Herzog S., Mirakhorli M. Lessons from the Use of Natural Language Inference (NLI) in Requirements Engineering Tasks. arXiv. 2024. URL: https://arxiv.org/abs/2405.05135.

20. Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.-Y., Wen, J.-R. A Survey of Large Language Models. arXiv. 2023. URL: https://arxiv.org/abs/2303.18223.

21. Murugan T. Design Principles of Chain of Thoughts (CoT) Prompt Engineering in the Context of GenAI. LinkedIn. 2025. URL: https://www.linkedin.com/pulse/design-principles-chain-thoughts-cot-prompt-context-genai-murugan-qijic.

22. Patronus AI. Advanced Prompt Engineering Techniques: Examples & Best Practices. Patronus.ai. URL: https://www.patronus.ai/llm-testing/advanced-prompt-engineering-techniques.

References

1. Luminous Men. The Importance of Clear Software Requirements. Luminousmen.com. 2024. URL: https://luminousmen.com/post/the-importance-of-clear-software-requirements/.

2. Requiment. Why Requirements Management Is So Important. Requiment. 2024. URL: https://www.requiment.com/the-importance-of-requirements-management/.

3. Burkin V. Mitigating Risks in Software Development through Effective Requirements Engineering. arXiv. 2023. URL: https://arxiv.org/abs/2305.05800.

4. Dale-Jones R. The Importance of Clear Requirements. SpireSpark International. 2017. URL: https://spirespark.com/spirespark-blog/2017/4/26/the-importance-of-clear-requirements.

5. Beta Breakers. Software Survival in 2024: Understanding 2023 Project Failure Statistics and the Role of Quality Assurance. Betabreakers.com. 2024. URL: https://www.betabreakers.com/blog/software-survival-in-2024-understanding-2023-project-failure-statistics-and-the-role-of-quality-assurancet/.

6. Jama Software. Why Investing in Requirements Management Software Makes Business Sense During an Economic Downturn. Jama Software. URL: https://www.jamasoftware.com/requirements-management-guide/requirements-management-tools-and-software/why-investing-in-rm-software-makes-good-business-sense/

7. Bakar N. A. A., et al. "Ambiguity Detection and Improvement for Malay Requirements Specification: A Systematic Review." International Journal on Advanced Science, Engineering and Information Technology, Vol. 13, No. 6, 2023. URL: http://ijaseit.insightsociety.org/index.php/ijaseit/article/view/18535.

8. Zhao L., Alhoshan W., Ferrari A., Letsholo K. J., Ajagbe M. A., Chioasca E.-V., Batista-Navarro R. T. Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. ACM Computing Surveys. 2021. Vol. 54. No. 3. P. 1–41. URL: https://dl.acm.org/doi/10.1145/3444689.

9. Hou X., Zhao Y., Liu Y., Yang Z., Wang K., Li L., Luo X., Lo D., Grundy J., Wang H. Large Language Models for Software Engineering: A Systematic Literature Review. ACM Transactions on Software Engineering and Methodology. 2024. Vol. 33. No. 7. Article 206. P. 1–58. URL: https://dl.acm.org/doi/10.1145/3695988.

10. Bashir S., Ferrari A., Khan M. A., Strandberg P. E., Haider Z., Saadatmand M., Bohlin M. Requirements Ambiguity Detection and Explanation with LLMs: An Industrial Study. Proceedings of the 41st International Conference on Software Maintenance and Evolution (ICSME). 2025. P. 620–631. URL: https://www.ipr.mdu.se/publications/7221-Requirements_Ambiguity_Detection_and_Explanation_with_LLMs__An_Industrial_Study

11. Ghosh B. Advanced Prompt Engineering for Reducing Hallucination. Medium. 2024. URL: https://medium.com/@bijit211987/advanced-prompt-engineering-for-reducing-hallucination-bb2c8ce62fc6.

12. Berry D.M., Kamsties E. Ambiguity in Requirements Specification. Perspectives on Software Requirements. Springer. 2003. URL: https://link.springer.com/chapter/10.1007/978-1-4615-0465-8_2.

13. Kamsties E., Berry D.M., Paech B. Detecting Ambiguities in Requirements Documents Using Inspections. Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01). 2001. URL: https://cs.uwaterloo.ca/~dberry/FTP_SITE/reprints.journals.conferences/KamstiesBerryPaech2001DetectingAmbiguity.pdf.

14. Gervasi V., Ferrari A., Zowghi D., Spoletini P. Ambiguity in Requirements Engineering: Towards a Unifying Framework // Lecture Notes in Computer Science. – 2019. – P. 191–210. – DOI: 10.1007/978-3-030-30985-5_12.

15. Kaur J., Kaur A. An Analysis of Ambiguity Detection Techniques for Software Requirements Specification (SRS). International Journal of Computer Sciences and Engineering. 2018. Vol. 6. No. 12. P. 103–108. URL: https://www.researchgate.net/publication/326730868_An_Analysis_of_Ambiguity_Detection_Techniques_for_Software_Requirements_Specification_SRS

16. Fantechi A., Gnesi S., Semini L. Rule-based NLP vs ChatGPT in Ambiguity Detection, a Preliminary Study. CEUR Workshop Proceedings. 2023. Vol. 3378. URL: https://ceur-ws.org/Vol-3378/NLP4RE-paper1.pdf.

17. Ezzini S., Abualhaija S., Arora C., Sabetzadeh M. TAPHSIR: Towards AnaPHoric Ambiguity Detection and ReSolution In Requirements. arXiv. 2022. URL: https://arxiv.org/abs/2206.10227.

18. Ferrari A., Bano M., Zowghi D., Gervasi V. An NLP approach for cross-domain ambiguity detection in requirements engineering. Software and Systems Modeling. 2019. Vol. 18. P. 1799–1812. URL: https://iris.cnr.it/handle/20.500.14243/386974.

19. Fazelnia M., Koscinski V., Herzog S., Mirakhorli M. Lessons from the Use of Natural Language Inference (NLI) in Requirements Engineering Tasks. arXiv. 2024. URL: https://arxiv.org/abs/2405.05135.

20. Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.-Y., Wen, J.-R. A Survey of Large Language Models. arXiv. 2023. URL: https://arxiv.org/abs/2303.18223.

21. Murugan T. Design Principles of Chain of Thoughts (CoT) Prompt Engineering in the Context of GenAI. LinkedIn. 2025. URL: https://www.linkedin.com/pulse/design-principles-chain-thoughts-cot-prompt-context-genai-murugan-qijic.

22. Patronus AI. Advanced Prompt Engineering Techniques: Examples & Best Practices. Patronus.ai. URL: https://www.patronus.ai/llm-testing/advanced-prompt-engineering-techniques.

Додаток Б

Програмний код створеної програмної реалізації методу доступний у репозиторії GitHub: https://github.com/SUusername/llm-ambiguity-detection. На рисунку Б.1 показано вміст репозиторію на GitHub.

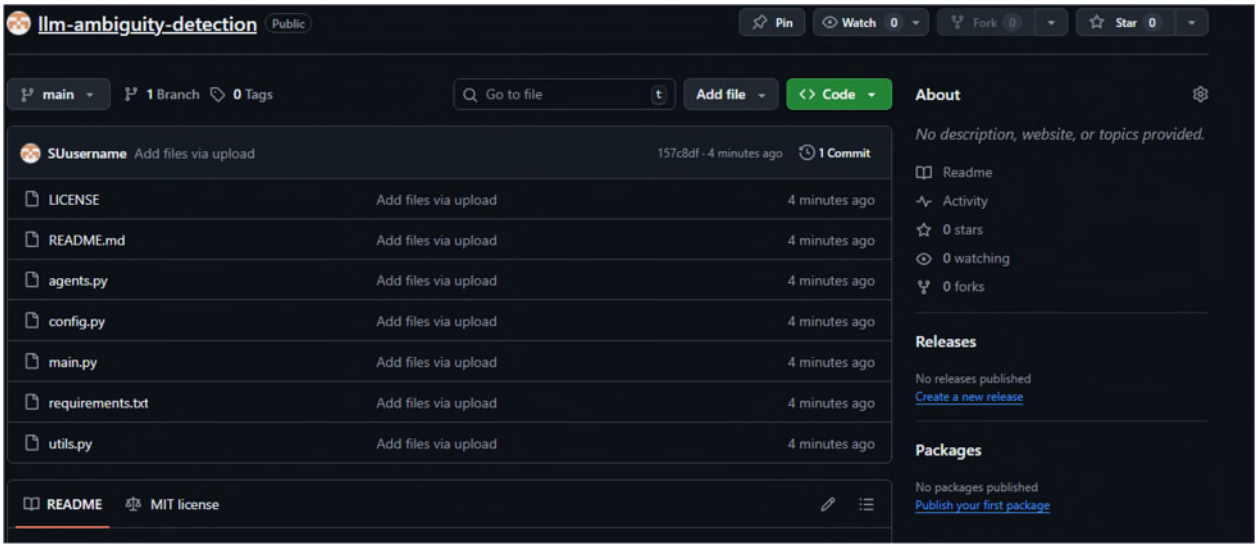


Рисунок Б.1 – Репозиторій

Репозиторій містить:

- config.py: модуль конфігурації, що містить лексичні словники (ambiguity lexicon), таксономію типів неоднозначностей (IEEE 830) та базу знань проекту (Project Knowledge Base) для RAG-компонента.
- agents.py: реалізація інтелектуальних агентів системи. Включає RAG-компонент, Агента-Ідентифікатора для первинної фільтрації та Агента-Класифікатора, що реалізує логіку Chain-of-Thought.
- utils.py: допоміжні функції для нормалізації тексту та розрахунку метрик ефективності (Precision, Recall, F1, Accuracy) шляхом порівняння з еталонною розміткою.
- main.py: головний скрипт оркестрації, що виконує ініціалізацію моделі Flan-T5, запускає конвеєр обробки вимог та забезпечує інтерфейс взаємодії з користувачем.

- README.md: файл із описом архітектури системи, інструкціями щодо встановлення залежностей та запуску аналізу.
- requirements.txt: перелік бібліотек Python (transformers, torch), необхідних для роботи нейромережевих компонентів.
- LICENSE: ліцензія MIT, яка дозволяє використання та поширення коду з дотриманням авторства.

Додаток В

Презентаційний матеріал

КВАЛІФІКАЦІЙНА РОБОТА МАГСТРА

Метод виявлення неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей

Виконав: студент 2 курсу, групи КНм-24-1 Богдан ВОНСОВИЧ
Керівник: к.т.н., доцент кафедри КН Руслан БАГРІЙ

Актуальність теми

Якість програмного забезпечення безпосередньо залежить від якості вимог, сформульованих на ранніх етапах життєвого циклу розробки. Неоднозначність у вимогах є однією з головних причин виникнення дефектів, перевитрат бюджету та зривів термінів проектів. Традиційні методи ручного рецензування є трудомісткими та суб'єктивними, а наявні автоматизовані інструменти на основі правил часто генерують високий рівень хибних спрацьовувань. Використання великих мовних моделей відкриває нові можливості для семантичного аналізу тексту, проте самі по собі моделі схильні до галюцинацій та ігнорування контексту проекту. Створення гібридного методу, що поєднує гнучкість LLM із точністю контекстної бази знань (RAG), є актуальним науково-прикладним завданням.

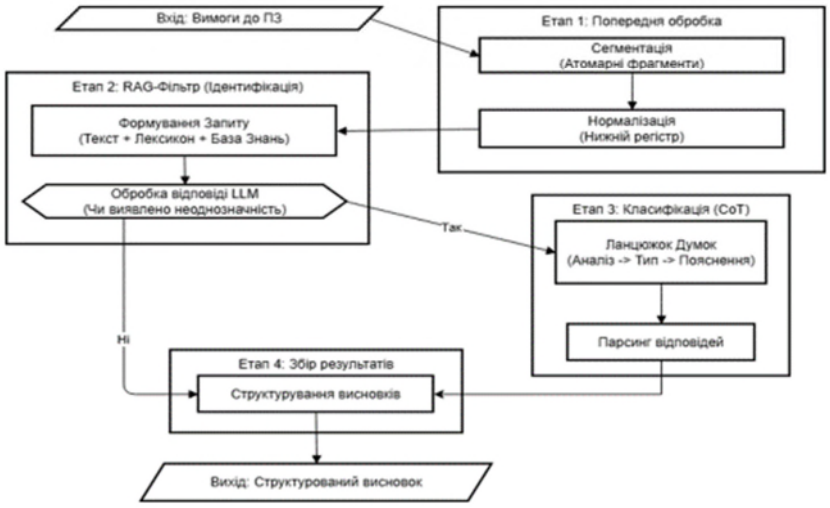
Мета і завдання роботи

Метою роботи є підвищення якості процесу виявлення лінгвістичних неоднозначностей у вимогах до програмного забезпечення за рахунок застосування контекстно-орієнтованих агентів на базі LLM.

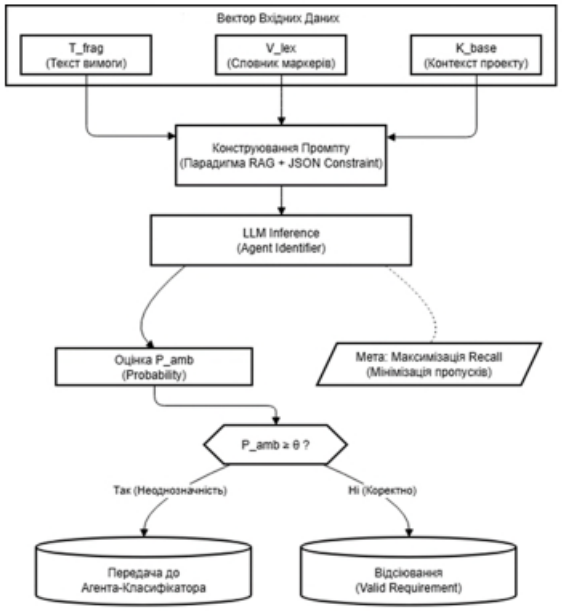
Для досягнення поставленої мети потрібно виконати наступні завдання:

- Провести аналіз існуючих методів виявлення неоднозначностей та підходів до їх класифікації згідно зі стандартом IEEE 830;
- Розробити двоетапний метод виявлення та класифікації неоднозначностей, який базується на послідовній взаємодії агента-ідентифікатора для первинної фільтрації вимог та агента-класифікатора для їх семантичної типізації;
- Виконати програмну реалізацію методу з використанням технології RAG для забезпечення врахування глосарію та контексту конкретного проєкту;
- Провести експериментальне дослідження розробленого методу та оцінити показники якості виявлення неоднозначностей порівняно з базовими підходами.

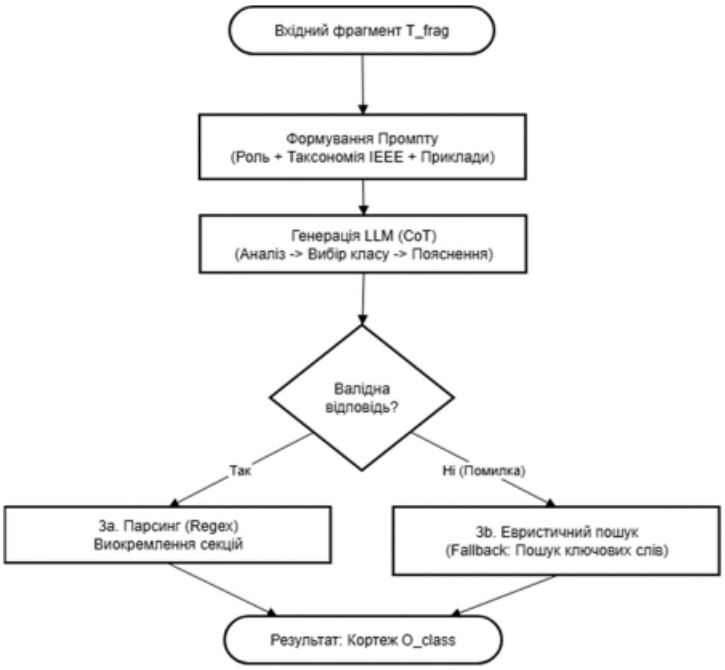
Схема методу виявлення неоднозначностей у вимогах до програмного забезпечення з використанням LLM



Функціональна схема роботи агента-ідентифікатора



Функціональна схема роботи агента-класифікатора



Робота програмної ралізації методу

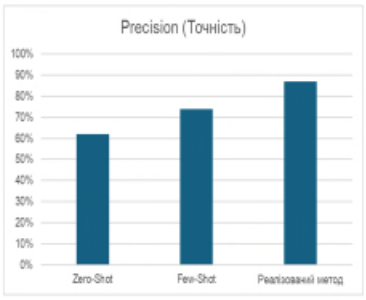
```
Введіть текст вимог для аналізу (натисніть Enter двічі, щоб завершити ввід):
Старі дані будуть архівовані


Введені вимоги:
Старі дані будуть архівовані
```

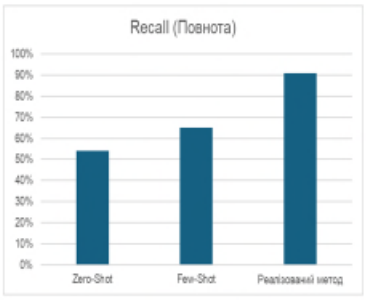
```
--- Результати аналізу вимог користувача ---
Знайдено неоднозначність 1:
{
  "text_fragment": "Старі дані будуть архівовані",
  "confidence_score": 0.6,
  "identifier_reasoning": "Potential ambiguity detected via fallback: keyword(s) 'старі' found in text fragment.",
  "classification_analysis": "Fallback: Fragment contains potential ambiguity keywords: старі.",
  "classification_type": "Lexical Ambiguity",
  "classification_explanation": The ambiguity arises from the multiple interpretations of terms like 'старі'.
}
```

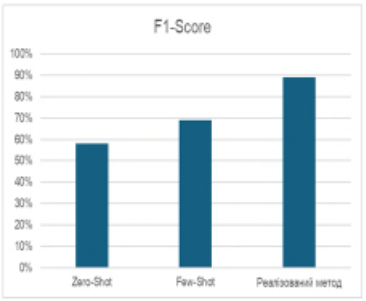
Порівняння з іншими підходами

Метод (Сценарій)	Precision (Точність)	Recall (Повнота)	F1-Score	Точність класифікації
Zero-Shot	0.62	0.54	0.58	N/A (Binary only)
Few-Shot	0.74	0.65	0.69	N/A (Binary only)
Реалізований метод	0.87	0.91	0.89	0.83

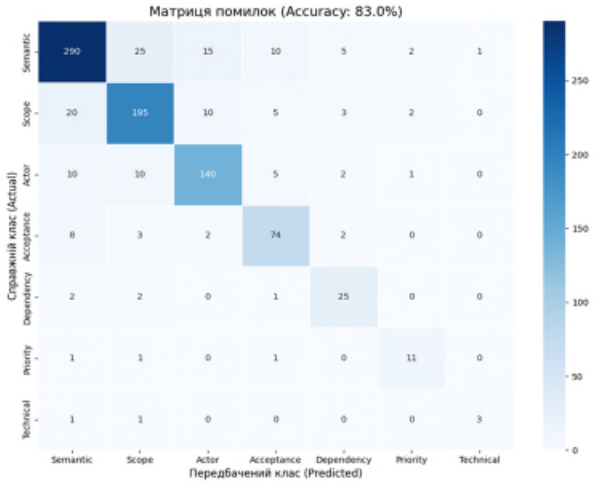
Графічне представлення проведеного порівняння

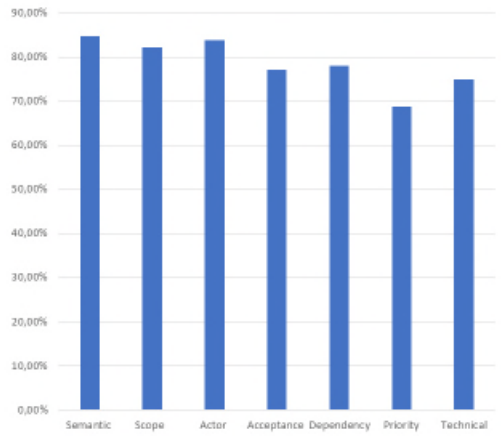






Матриця помилок при тестуванні на датасеті





Висновки:

- Розроблено гібридний метод виявлення неоднозначностей, що поєднує LLM з технологією RAG, це вирішило проблему відсутності контекстної обізнаності, властиву традиційним засобам.
- Запропоновано двоетапну мультиагентну архітектуру:Агент-Ідентифікатор – максимізує Recall. Агент-Класифікатор – покращує Precision та надає пояснення причин дефекту (механізм CoT) згідно зі стандартом IEEE 830.
- Створено прототип на базі моделі Google Flan-T5-Large (Python) із використанням патерну «Оркестратор».
- Забезпечено автономне локальне розгортання, що гарантує конфіденційність даних замовника (відсутність залежності від платних API).
- Апробація на датасеті з 1983 вимог (домен телекомунікацій) підтвердила перевагу гібридного підходу над базовими сценаріями (Zero-shot).
- Використання RAG дозволило суттєво знизити рівень хибних спрацювань для структурних та референційних неоднозначностей.
- Інструмент трансформує ручну перевірку в автоматизований процес.
- Впровадження методу може дозволити знизити вартість виправлення дефектів на ранніх етапах життєвого циклу ПЗ.

Wed Dec 17 21:13:39 EET 2025, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (UA) v-15.284 Educational

The maximum coincidence with one document 3.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 12%**

	Document		Sum coincidence on the DB	
ID: 253553 Title: КВАЛІФІКАЦІЙНА РОБОТА на тему Метод виявлення неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей Added in a DB: 2025-12-17 Authors: Богдан ВОНСОВИЧ Heads: Руслан БАГРІЙ Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	119718	787	5548 (5%)	52 (7%)

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: Богдан ВОНСОВИЧ

Співавтор:

Назва: КВАЛІФІКАЦІЙНА РОБОТА на тему Метод виявлення неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей

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

Підрозділ: Кафедра комп'ютерних наук

Коефіцієнт подібності 1:3.6%

Коефіцієнт подібності 2:0.9%

Мікропробіли: 0

Заміна букв: 2

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2025-12-17 20:45:31.0

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:

2025-12-17

Дата

експерт Перовський Р.С

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 неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей

Автор: Вонсович Богдан Андрійович

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 3,0 %, схожість виявлена зі звітом автора з науково-дослідної практики.

2) за програмою StrikePlagiarism КП1 3,58%

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 Богдана ВОНСОВИЧА за темою: «Метод виявлення неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей»

1. Актуальність теми

Актуальність теми кваліфікаційної роботи обумовлена критичним впливом якості вимог на успішність програмних проєктів, де неоднозначність є однією з головних причин виникнення дефектів. Традиційні методи ручного рецензування є занадто трудомісткими, а існуючі автоматизовані інструменти часто ігнорують контекст, генеруючи високий рівень хибних спрацювань. Запропонований у роботі метод вирішує проблему «семантичного розриву» та галюцинацій LLM шляхом створення гібридної архітектури, що поєднує гнучкість великих мовних моделей з точністю технології доповненої генерації RAG. Використання контекстно-орієнтованих агентів дозволяє автоматизувати процес валідації вимог з урахуванням специфіки конкретного проєкту, що є вкрай важливим для сучасної індустрії розробки ПЗ.

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. Об'єктом дослідження є процес автоматизованого аналізу текстових вимог до програмного забезпечення. Предметом дослідження виступають методи та моделі автоматизованого виявлення лінгвістичних неоднозначностей із застосуванням LLM та інженерії промптів. Структура та зміст роботи відповідають вимогам до кваліфікаційних робіт магістра.

3. Професійні та особистісні якості магістранта

Богдан ВОНСОВИЧ під час виконання роботи проявив себе як кваліфікований дослідник, здатний до системного аналізу та вирішення нетривіальних інженерних задач. Студент продемонстрував глибокі знання у сфері NLP та архітектури трансформерів, відповідальність та ініціативність при проведенні експериментальних досліджень.

4. Ступінь самостійності під час виконання кваліфікаційної роботи

Робота виконана повністю самостійно. Програмна реалізація методу на мові Python, формування бази знань та проведення експериментів на датасеті у домені телекомунікацій виконані автором особисто. Академічного плагіату не виявлено, посилання на використані джерела наведено коректно.

5. Наукова новизна та оригінальність запропонованих підходів

Удосконалено метод автоматизованого виявлення неоднозначностей, який, на відміну від існуючих підходів прямого використання LLM, базується на двоетапній агентній архітектурі з інтеграцією технології RAG, що дозволило здійснити динамічну перевірку термінів через контекстну базу знань та суттєво зменшити кількість хибних спрацювань на вузькоспеціалізованій технічній лексиці, досягнувши показника F1-Score на рівні 0.89. Результати роботи апробовано та опубліковано на науково-практичні конференції.

6. Ступінь оволодіння методами дослідження

Студент продемонстрував впевнене володіння методами системного аналізу, обробки природної мови, технологією доповненої генерації та методами інженерії промптів, зокрема CoT для забезпечення інтерпретованості рішень моделі.

7. Повнота та якість розкриття теми роботи

Розроблено та програмно реалізовано метод, що поєднує нейромережевий аналіз із евристичними правилами. Проведено детальний порівняльний аналіз із іншими підходами (Zero-shot, Few-shot), який підтвердив перевагу запропонованого рішення.

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

Матеріал викладено логічно та послідовно. Текст пояснювальної записки написаний грамотною технічною мовою, аргументація висновків спирається на отримані експериментальні дані.

9. Можливість практичного застосування кваліфікаційної роботи, окремих її частин

Розроблений метод та його програмна реалізація є придатними для застосування в автоматизованих циклах розробки програмного забезпечення. Використання розробленого засобу дозволяє мінімізувати вплив людського фактора при аналізі технічної документації та слугувати основою для створення інтелектуальних асистентів для аналітиків, адаптованих до специфіки конкретних проектів.

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

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

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

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МОН УКРАЇНИ
кафедра комп'ютерних наук

ВІДГУК ОПОНЕНТА

на кваліфікаційну роботу магістра

гр. КНм-24-1 Богдана ВОНСОВИЧА за темою: *Метод виявлення неоднозначностей у вимогах до програмного забезпечення з використанням великих мовних моделей*

1. Актуальність обраної теми

Якість вимог до програмного забезпечення є критичним фактором успіху ІТ-проєктів, оскільки виправлення помилок, закладених на етапі аналізу, коштує в рази дорожче на пізніх стадіях розробки. Традиційні методи ручного рецензування є трудомісткими, а існуючі автоматизовані засоби часто не враховують контекст. Використання великих мовних моделей у поєднанні з технологією доповненої генерації відкриває нові можливості для семантичного аналізу тексту. Тема роботи, спрямована на створення гібридного методу для автоматизованого виявлення неоднозначностей, є безумовно актуальною та має важливе практичне значення для індустрії розробки ПЗ.

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

Зміст кваліфікаційної роботи повністю відповідає спеціальності 122 «Комп'ютерні науки». У роботі використано сучасні методи штучного інтелекту, обробки природної мови, інженерії знань та системного аналізу. Розроблений метод базується на алгоритмічних моделях та програмній реалізації, що є профільним для даної спеціальності.

3. Повнота розкриття мети та завдань дослідження

Автор повною мірою розкрив мету роботи, яка полягала у підвищенні якості виявлення лінгвістичних неоднозначностей. Всі поставлені завдання виконані: проаналізовано існуючі підходи, розроблено двоетапний метод із використанням агентів, виконано програмну реалізацію та проведено експериментальні дослідження.

4. Наявність наукової новизни

Наукова новизна роботи полягає в удосконаленні методу автоматизованого виявлення неоднозначностей у вимогах. Автором запропоновано гібридну архітектуру, яка, на відміну від існуючих рішень, поєднує LLM з технологією RAG для динамічної верифікації термінів через базу знань проєкту. Також новим є застосування агентного підходу з розділенням етапів фільтрації та глибокої класифікації з використанням CoT. Результати апробовано на конференції «АПКН – 2025».

5. Зміст кожного розділу роботи

Робота має логічну структуру та складається з чотирьох розділів:

1. У першому розділі проведено ґрунтовний аналіз проблеми неоднозначності в інженерії вимог та огляд сучасних методів.

2. У другому розділі описано концепцію та математичну модель запропонованого методу. Детально розроблено структуру бази знань та алгоритми роботи агентів ідентифікації та класифікації.

3. У третьому розділі наведено опис програмної реалізації системи мовою Python з використанням бібліотек Hugging Face та PyTorch. Обґрунтовано архітектурні рішення, зокрема патерн «Оркестратор» та механізми Fallback.

4. У четвертому розділі представлено результати експериментальних досліджень на датасеті з домену телекомунікацій. Доведено перевагу розробленого методу над підходами Zero-shot та Few-shot.

5. Ступінь розкриття теми роботи

Тема розкрита глибоко та всебічно. Автор продемонстрував високий рівень розуміння як предметної області, так і технологічних інструментів. Практична цінність підтверджується створенням робочого прототипу, здатного локально обробляти вимоги без передачі даних стороннім сервісам.

6. Якість оформлення кваліфікаційної роботи

Пояснювальна записка оформлена згідно з вимогами стандарту. Матеріал викладено грамотно, логічно та послідовно. Ілюстративний матеріал якісний та інформативний.

7. Недоліки кваліфікаційної роботи

Експериментальне дослідження методу проводилося лише на одному наборі даних (домен телекомунікацій), що не дозволяє повною мірою стверджувати про його універсальність та ефективність для інших предметних областей без проведення додаткових тестів.

Проте ці недоліки не впливають на загальну позитивну оцінку роботи.

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

Кваліфікаційна робота Вонсовича Б.А. є завершеним науковим дослідженням, що вирішує актуальне завдання. Робота відповідає вимогам до кваліфікаційних робіт магістра спеціальності 122 «Комп'ютерні науки» та заслуговує на оцінку «відмінно», а її автор – на присудження кваліфікації магістр.

Рекомендована оцінка – «Відмінно».

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

Говорущенко Т.О., декан ФІТ ХНУ

«17» 12 2025 р.

Підпис