Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА

на тему *Метод класифікації програмних вимог 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* — Підпис — *Богдан РОМАНОВ*
Курс, група виконавця — Ім'я, ПРІЗВИЩЕ

Керівник: *д.т.н., професор кафедри КН* — Підпис — *Олександр БАРМАК*
Науковий ступінь, посада — Ім'я, ПРІЗВИЩЕ

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

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

12 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
« *25* » *08* *2025* року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

1. Тема кваліфікаційної роботи магістра: *«Метод класифікації програмних вимог з використанням великих мовних моделей»*

2. Завдання видано студенту *Богдану РОМАНОВУ*
(Ім'я, прізвище)

3. Керівник роботи *зав. кафедри КН, д.т.н., професор Олександр БАРМАК*
(посада, ім'я, прізвище)

4. Затверджені наказом університету від « *25* » *08* *2025* р. № *65*

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

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

Мета роботи – підвищення якості автоматичної класифікації програмних вимог шляхом розробки методу некерованої кластеризації векторних представлень доменно-адаптованої великої мовної моделі. Досягнення мети роботи передбачає аналіз існуючих підходів, які реалізують методи класифікації програмних вимог, розробку методу класифікації на основі некерованої кластеризації з використанням великих мовних моделей та програмну реалізацію запропонованого рішення. Результативність методу підтверджується результатами експериментальних досліджень на відомих наборах даних з оцінкою якості кластеризації та інтерпретації результатів.

7. Календарний план виконання кваліфікаційної роботи:

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

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

Керівник: *зав. каф. КН д.т.н.,проф.* (Науковий ступінь, посада) _____ (Підпис) *Олександр БАРМАК* (Ім'я, ПРІЗВИЩЕ)

Реферат

Кваліфікаційна робота магістра спрямована на підвищення якості класифікації програмних вимог при використанні некерованих методів кластеризації за доменною адаптацією моделі.

Актуальність теми. Розв’язання задачі класифікації програмних вимог є актуальною проблемою у галузі розробки програмного забезпечення. При великій кількості поставлених вимог, людина може мати труднощі під час складання специфікацій та проєктування архітектури продукту, витрачаючи значну кількість часу на фільтрацію функціональних та не функціональних вимог. Автоматична система класифікації програмних вимог дозволить скоротити час формування технічного завдання, що в свою чергу дає можливість значно пришвидшити цикл розробки програмного забезпечення. Це набуває особливої актуальності при використанні гнучких та ітеративних підходів до розробки, де вимоги постійно змінюються.

Об’єкт дослідження – процес автоматизованої класифікації програмних вимог у системах управління вимогами.

Предмет дослідження – методи некерованої кластеризації векторних представлень текстів з використанням доменно-адаптованих великих мовних моделей.

Мета роботи – підвищення якості автоматичної класифікації програмних вимог шляхом розробки методу некерованої кластеризації векторних представлень доменно-адаптованої великої мовної моделі.

Для досягнення мети роботи, необхідно виконати наступні завдання:

– провести аналіз сучасних підходів до автоматизованої класифікації програмних вимог, що базуються на методах обробки природної мови та глибокого навчання.

– розробити метод класифікації програмних вимог, що базується на формуванні векторного простору ознак за допомогою LLM та його подальшій некерованій кластеризації.

– виконати доменну адаптацію мовної моделі для покращення семантичної якості векторних представлень.

– провести експериментальне дослідження розробленого методу із застосуванням ансамблевих підходів, оцінити отримані метрики якості та порівняти їх з базовими підходами та існуючими аналогами.

Методи дослідження. У роботі використано методи доменної адаптації мовних моделей (classification, contrastive, POS loss), алгоритми кластеризації (HDBSCAN, KMeans, GaussianMixture, AgglomerativeClustering, SpectralClustering, OPTICS, Birch) та ансамблеві архітектури кластеризації (ConsensusClustering, AggregatedClustering) для класифікації текстів; методи зниження розмірності (t-SNE, UMAP) для візуалізації даних; об’єктно-орієнтований підхід для проєктування реалізації.

Наукова новизна одержаних результатів. Удосконалено метод класифікації програмних вимог шляхом поєднання доменної адаптації великих мовних моделей та ансамблевої некерованої кластеризації векторних представлень, що, відрізняється від існуючих підходів відсутністю навчального шару класифікатора та наявністю візуалізації векторного простору ознак, що дало змогу підвищити точність класифікації на 1-2% в порівнянні з керованими методами та забезпечити можливість інтерпретації розподілу ознак.

Апробація результатів кваліфікаційної роботи магістра та публікації. Основні наукові та практичні результати пройшли апробацію на XVII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп’ютерних наук (АПКН – 2025)» (м. Хмельницький, 14–15 листопада 2025 р.) та опубліковано статтю в рамках міжнародної конференції ExplAI 2025:

1. Романов Б.А., Бармак О.В., Багрій Р.О., Скрипник Т.К. Метод класифікації програмних вимог з використанням великих мовних моделей (LLM) Актуальні проблеми комп’ютерних наук АПКН-2025 : матеріали XVII Всеукр. науково-практ. конф., м. Хмельницький, 14–15 листопада 2025 р. Хмельницький, 2025. С. 368–372. URL: https://kn.khmnu.edu.ua/wp-content/uploads/sites/18/apkn-2025_corpuspaper.pdf.

2. Bahrii R., Skrypnyk T., Romanov B., Zaitseva E., El Bouhissi H., Lytvynenko V. An approach to identifying functional and non-functional requirements in IT-project management using deep learning models. ExplAI-2025: Advanced AI in Explainability and Ethics for the Sustainable Development Goals. Khmelnytskyi, 2025

Структура та обсяг роботи. Кваліфікаційна робота магістра складається із завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань з 47 найменувань та 3 додатки. Загальний обсяг кваліфікаційної роботи складає 115 сторінок, з поміж яких 84 сторінки основного тексту та 31 сторінки додатків. У роботі наведено 42 рисунки та 7 таблиць.

Ключові слова: класифікація програмних вимог, некерований класифікатор, кластеризація, RoBERTa, доменна адаптація, контрастивне навчання.

Зміст

Перелік скорочень

Скорочення, термін, позначення	Пояснення
CF	Clustering Features. Ознаки кластеризації
FR	Функціональні вимоги
NFR	Нефункціональні вимоги
VS Code	Visual studio code
ПЗ	Програмне забезпечення
NLP	Natural Language Processing Обробка природної мови
CNN	Convolutional Neural Network Згорткова нейронна мережа
RNN	Recurrent Neural Network Рекурентна нейронна мережа
LSTM	Long Short-Term Memory Довга короткочасна пам’ять
BERT	Bidirectional Encoder Representations from Transformers Двонаправлені представлення кодувальника з трансформерів
RoBERTa	Robustly optimized BERT approach Надійно оптимізований підхід BERT
LLM	Large Language Model Надійно оптимізований підхід BERT

Вступ

Актуальність теми. Розв’язання задачі класифікації програмних вимог є актуальною проблемою у галузі розробки програмного забезпечення. При великій кількості поставлених вимог, людина може мати труднощі під час складання специфікацій та проєктування архітектури продукту, витрачаючи значну кількість часу на фільтрацію функціональних та не функціональних вимог. Автоматична система класифікації програмних вимог дозволить скоротити час формування технічного завдання, що в свою чергу дає можливість значно пришвидшити цикл розробки програмного забезпечення. Це набуває особливої актуальності при використанні гнучких та ітеративних підходів до розробки, де вимоги постійно змінюються.

Об’єкт дослідження – процес автоматизованої класифікації програмних вимог у системах управління вимогами.

Предмет дослідження – методи некерованої кластеризації векторних представлень текстів з використанням доменно-адаптованих великих мовних моделей.

Мета роботи – підвищення якості автоматичної класифікації програмних вимог шляхом розробки методу некерованої кластеризації векторних представлень доменно-адаптованої великої мовної моделі.

Для досягнення мети роботи, необхідно виконати наступні завдання:

– провести аналіз сучасних підходів до автоматизованої класифікації програмних вимог, що базуються на методах обробки природної мови та глибокого навчання.

– розробити метод класифікації програмних вимог, що базується на формуванні векторного простору ознак за допомогою LLM та його подальшій некерованій кластеризації.

– виконати доменну адаптацію мовної моделі для покращення семантичної якості векторних представлень.

– провести експериментальне дослідження розробленого методу із застосуванням ансамблевих підходів, оцінити отримані метрики якості та порівняти їх з базовими підходами та існуючими аналогами.

Методи дослідження. У роботі використано методи доменної адаптації мовних моделей (classification, contrastive, POS loss), алгоритми кластеризації (HDBSCAN, KMeans, GaussianMixture, AgglomerativeClustering, SpectralClustering, OPTICS, Birch) та ансамблеві архітектури кластеризації (ConsensusClustering, AggregatedClustering) для класифікації текстів; методи зниження розмірності (t-SNE, UMAP) для візуалізації даних; об’єктно-орієнтований підхід для проєктування реалізації.

Наукова новизна одержаних результатів. Удосконалено метод класифікації програмних вимог шляхом поєднання доменної адаптації великих мовних моделей та ансамблевої некерованої кластеризації векторних представлень, що, відрізняється від існуючих підходів відсутністю навчального шару класифікатора та наявністю візуалізації векторного простору ознак, що дало змогу підвищити точність класифікації на 1-2% в порівнянні з керованими методами та забезпечити можливість інтерпретації розподілу ознак.

Апробація результатів кваліфікаційної роботи магістра та публікації. Основні наукові та практичні результати пройшли апробацію на XVII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп’ютерних наук (АПКН – 2025)» (м. Хмельницький, 14–15 листопада 2025 р.) та опубліковано статтю в рамках міжнародної конференції ExplAI 2025:

1. Романов Б.А., Бармак О.В., Багрій Р.О., Скрипник Т.К. Метод класифікації програмних вимог з використанням великих мовних моделей (LLM) Актуальні проблеми комп’ютерних наук АПКН-2025 : матеріали XVII Всеукр. науково-практ. конф., м. Хмельницький, 14–15 листопада 2025 р. Хмельницький, 2025. С. 368–372. URL: https://kn.khmnu.edu.ua/wp-content/uploads/sites/18/apkn-2025_corpuspaper.pdf.

2. Bahrii R., Skrypnyk T., Romanov B., Zaitseva E., El Bouhissi H., Lytvynenko V. An approach to identifying functional and non-functional requirements

in IT-project management using deep learning models. ExplAI-2025: Advanced AI in Explainability and Ethics for the Sustainable Development Goals. Khmelnytskyi, 2025

Структура та обсяг роботи. Кваліфікаційна робота магістра складається із завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань з 47 найменувань та 3 додатки. Загальний обсяг кваліфікаційної роботи складає 115 сторінок, з поміж яких 84 сторінки основного тексту та 31 сторінки додатків. У роботі наведено 42 рисунки та 7 таблиць.

Розділ 1. Аналіз існуючих методів класифікації програмних вимог

1.1 Характеристика задачі класифікації програмних вимог

У сучасному світі розробки ПЗ точне та ефективне виявлення функціональних та нефункціональних вимог є критично важливим для успіху будь-якого проєкту. Неправильна або неповна класифікація цих вимог на ранніх етапах може призвести до значних фінансових втрат, затримок у випуску продукту та, врешті-решт, до створення ПЗ, яке не відповідає очікуванням користувачів [1]. Саме тому актуальність задачі автоматизації цього процесу за допомогою методів глибокого навчання стрімко зростає [2].

Функціональні вимоги визначають, що саме повинна робити система, описуючи її конкретні функції та поведінку. На противагу їм, нефункціональні вимоги встановлюють, як система має виконувати ці функції, визначаючи такі її атрибути, як продуктивність, безпека, надійність та зручність використання. Розуміння цієї різниці є фундаментальним для побудови якісного програмного продукту [3, 4].

Традиційно, аналіз та класифікація вимог до ПЗ виконується бізнес-аналітиками та системними інженерами вручну. Цей підхід, хоч і перевірений часом, має низку суттєвих недоліків: трудомісткість та висока вартість, людський фактор, неповнота виявлення, конфліктні вимоги, динамічність вимог тощо. Для вирішення означених проблем застосовуються технології глибокого навчання та обробки природної мови [5]. Ці методи дозволяють створювати моделі, здатні автоматично аналізувати та класифікувати текстову інформацію з високою точністю. Основними перевагами застосування глибокого навчання для виявлення функціональних та нефункціональних вимог є: автоматизація та швидкість, підвищення точності, об’єктивність, раннє виявлення проблем [6].

Однією з основних проблем для класифікації програмних вимог є неоднозначність та суб’єктивність, оскільки вимоги часто можуть формуватися повсякденною мовою, це може призвести до їх неоднозначного сприйняття при спілкуванні з клієнтом.

Як приклад неоднозначності, слово «productivity» у загальному контексті може стосуватись ефективності праці, тоді як у вимогах до ПЗ воно часто набуває технічного відтінку, наприклад, швидкість обробки даних, тощо. Такі терміни, як «масштабованість», «зручність супроводу» чи «безпека», мають складні, багатогранні значення, повне розуміння яких вимагає спеціальних знань. Модель, яка не «розрізняє» тонкощі предметної області, змушена орієнтуватися лише на поверхневі синтаксичні шаблони, що знижує її надійність [7].

Водночас із семантичними проблемами, значним викликом для впровадження методів глибокого навчання є складність створення якісних розмічених датасетів. Кероване навчання класичних моделей, які часто є основою для класифікації, вимагає тисячі правильно розмічених прикладів для досягнення необхідної точності класифікації [8]. Отримання такого обсягу даних є серйозною проблемою для нових проєктів розробки ПЗ, оскільки процес ручної розмітки текстових вимог експертами є дуже трудомістким, повільним і дорогим [9]. Це створює замкнуте коло, для автоматизації класифікації в новому проєкті потрібна велика кількість розмічених даних, яких немає, а їх створення вручну суттєво уповільнює початок роботи та збільшує витрати.

Зазначене обґрунтовує доцільність використання методів некерованого навчання (кластеризації), які не потребують розмічених датасетів.

1.2 Аналіз існуючих публікацій та наукових підходів

Для класифікації програмних вимог використовуються різні методи обробки текстів, від моделей з фіксованими ознаками до нейромережевих підходів. Сучасні дослідження спрямовані на створення спеціалізованих моделей, адаптованих до мови розробки програмного забезпечення [10]. Порівняння різних підходів для класифікації текстів наведено в таблиці 1.

На початковому етапі для класифікації вимог активно використовувалися архітектури CNN та RNN. Моделі на основі CNN ефективно виділяли локальні ознаки, зокрема ключові слова та фрази, тоді як RNN/LSTM добре вловлювали

послідовні залежності в тексті вимоги [11]. Часто ці архітектури комбінували для використання переваг обох підходів [12]. Перевагами зазначених підходів є значне покращення точності порівняно з традиційними методами, наприклад, SVM або Naive Bayes та здатність автоматично вивчати ознаки, що усуває потребу в ручній інженерії ознак. Їх основним недоліком є обмежене розуміння контексту – ці моделі мали труднощі з аналізом довгих та складних речень через проблеми із «затухаючим градієнтом» та обмеженою здатністю враховувати контекст усієї вимоги.

Поява трансформерних архітектур, зокрема BERT (Bidirectional Encoder Representations from Transformers), стала революційною для обробки природної мови і, зокрема, для класифікації вимог. Активно застосовується механізм тонкого налаштування fine-tuning попередньо навчених моделей, таких як BERT, RoBERTa, DeBERTa, для специфічної задачі класифікації ФВ та НФВ [13, 14, 15]. Такі підходи, як NoRBERT та PRCBERT, демонструють найвищу на сьогодні точність на стандартних наборах даних, таких як PROMISE [14]. Перевагами трансформерних архітектур є глибоке контекстуальне розуміння, оскільки завдяки механізму уваги *attention* та двонаправленому навчанню, BERT враховує контекст кожного слова з обох боків, що дозволяє краще розуміти семантику вимоги [13], а також навчання з передачею знань *Transfer Learning,* оскільки BERT попередньо навчений на величезних масивах загальномовних текстів, що дозволяє йому досягати високої точності навіть при тонкому налаштуванні на відносно невеликих наборах даних вимог [14]. До недоліків можна віднести високі обчислювальні потреби, адже тонке налаштування та використання великих трансформерних моделей вимагає значних обчислювальних ресурсів. Іншою проблемою є «чорна скринька», оскільки складність архітектури ускладнює інтерпретацію того, чому модель ухвалила те чи інше рішення, що є критичним для відповідальних систем [16]. Також варто відзначити чутливість до якості даних, оскільки ефективність моделей сильно залежить від якості та розміру даних, на яких проводиться тонке налаштування, при цьому існуючі датасети, такі як PROMISE, є відносно невеликими та можуть містити застарілі вимоги [14].

Останнім кроком в еволюції підходів стало застосування великих мовних моделей, таких як моделі останнього покоління зокрема, серії GPT-4, Google Gemini, Anthropic Claude, тощо. На відміну від BERT-подібних архітектур, які переважно використовуються для класифікації, LLM здатні виконувати завдання в режимі «нульового» або «кількох прикладів», а також генерувати вимоги або відповідати на питання про них. [17]. Однак їх використання пов’язане з такими викликами, як схильність до «галюцинацій» (генерування неправдивої інформації) та висока вартість використання через API [18].

Незважаючи на вражаючі результати трансформерних моделей, головна прогалина сучасних досліджень полягає у так званій «семантичній прірві». Більшість робіт спираються на стандартні моделі, навчені на загальномовних текстах (напр., Вікіпедія, новини), які не вловлюють специфічну семантику та нюанси термінології, притаманної галузі інженерії програмного забезпечення [19].

З усіх перелічених напрямків, найбільш прямим та поширеним на практиці способом створення доменно-специфічної моделі є донавчання існуючої потужної загальномовної моделі на релевантних даних. Цей підхід, відомий як доменна адаптація, дозволяє «спеціалізувати» модель на нюансах конкретної галузі, не вимагаючи навчання з нуля. Однак, в той час як більшість досліджень застосовують доменну адаптацію для створення керованих класифікаторів, які потребують наявності міток для кожного нового набору даних, у реальних умовах дані часто є нерозміченими.

Таблиця 1.1 – Переваги та недоліки існуючих підходів.

Підхід	**Переваги**	**Недоліки**	**Ключові публікації**
CNN/RNN/LSTM	Автоматичне вивчення ознак, краще за традиційні моделі	Обмежений контекст, проблеми з довгими залежностями	Khayashi et al [11]
Гібридні моделі (CNN+RNN)	Комбінують переваги обох архітектур	Підвищена складність, все ще	Rahman et al. [12]

		поступаються трансформерам	
Трансформери (BERT, RoBERTa)	Глибоке контекстуальне розуміння, SOTA точність, ефективне навчання	Високі вимоги до ресурсів, проблема «чорної скриньки»	Luo et al. [14], Xu [13]
Великі мовні моделі (LLMs)	Генерація та класифікація вимог, нульове/кілька-прикладове навчання.	Схильність до «галюцинацій», висока вартість використання	Almonte et al. [17] Ebrahim et al. [18]

Також, варто зазначити, що більшість існуючих робіт зосереджені на використанні керованого підходу навчання класифікатора [5]. Великі мовні моделі формують вектори ознак, де подібні вимоги можуть утворювати природні кластери, але при цьому некеровані методи класифікації, наприклад кластеризація, є малодослідженими.

Некеровані методи, зокрема кластеризація, пропонують альтернативу для ситуацій, коли розмічені дані відсутні. На відміну від класичних класифікаторів, їм не потрібні заздалегідь визначені категорії. Натомість вони виявляють природні групи в даних на основі схожості, що дозволяє досліджувати їхню структуру та потім інтерпретувати знайдені кластери [20].

Щільнісні алгоритми кластеризації, такі як HDBSCAN [21] та OPTICS не вимагають заздалегідь заданої кількості кластерів, а знаходять області високої щільності даних, автоматично відокремлюючи їх від шуму. Це робить їх придатними для виявлення рідкісних категорій вимог. HDBSCAN відомий стійкістю до викидів, а OPTICS краще справляється з кластерами різної щільності, що часто зустрічається в текстових даних. Однак їхня ефективність суттєво залежить від правильного вибору гіперпараметрів, таких як мінімальний розмір кластера [22].

Центроїдний алгоритм K-Means вимагає явного завдання кількості кластерів (K), що є його основним недоліком для нових недосліджених наборів вимог. Він також припускає, що кластери мають сферичну форму та приблизно однаковий розмір [23], що рідко виконується для семантичних векторів тексту. Його переваги – швидкість та масштабованість.

Імовірнісний підхід на основі Gaussian Mixture Models (GMM) пропонує більшу гнучкість, ніж K-Means. Він моделює дані як суміш кількох гаусівських розподілів і підтримує м’яке кластерування [24]. Це означає, що кожна вимога може з певною ймовірністю належати до кількох кластерів одночасно, що може краще відображати реальну семантичну неоднозначність.

Графовий метод Spectral Clustering працює не безпосередньо з векторами, а з матрицею подібності між ними [25]. Це дозволяє йому ефективно виявляти кластери складної форми, які важко відокремити центроїдними методами. Недоліком є висока обчислювальна складність для великих наборів даних та чутливість до вибору міри подібності.

Ієрархічні алгоритми, такі як Agglomerative Clustering та BIRCH, пропонують інший підхід. Agglomerative Clustering будує дендрограму, даючи аналітику змогу вибрати рівень деталізації кластеризації після обчислення [25]. Однак він стає обчислювально витратним для великих датасетів. BIRCH, натомість, спеціалізований для роботи з дуже великими обсягами даних. Він будує CF-дерево для інкрементального стиснення даних, що забезпечує високу швидкість, але може втрачати точність при роботі з високо-вимірними ембеддингами [25].

1.3 Огляд мовних моделей для класифікації текстів

На сьогоднішній день найбільш актуальними є два основних підходи: спеціалізовані трансформерні моделі (Рис. 1.1), наприклад BERT та його похідні, які тонко налаштовуються для конкретних завдань, та універсальні великі мовні моделі, зокрема GPT-4 або Gemini, здатні виконувати задачі через природні інструкції.

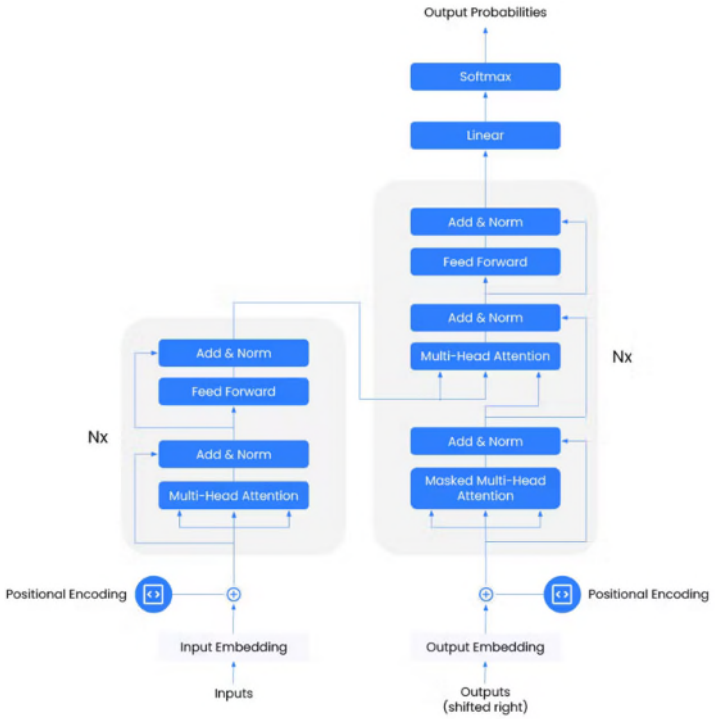


Рисунок 1.1 – Приклад трансформерної архітектури

Тонко налаштовані трансформери, зокрема BERT-base, можуть досягати високих показників точності при класифікації текстів. Наприклад, в задачах класифікації довгих академічних документів BERT-base досягнув значення середнього F1 метрики на рівні 82%, що є відносно високим результатом [26]. Навчання BERT може займати десятки хвилин і вимагати більше 2 ГБ пам’яті GPU. Також в умовах обмежених даних або для надто довгих документів продуктивність окремих трансформерів, наприклад RoBERTa-base, може різко падати, іноді поступаючись простішим методам [26].

Ключовою перевагою трансформерів є можливість доменної адаптації через попереднє навчання та тонке налаштування. Для подолання «семантичної прірви» між загальномовними та спеціалізованими текстами були розроблені доменно-специфічні варіанти, такі як SciBERT для наукових текстів або BioBERT для біомедицини [27]. Систематичний огляд свідчить, що такі спеціалізовані моделі, попередньо навчені на текстах цільової галузі, значно перевершують загальні трансформери у відповідних доменах, краще розуміючи термінологію та контекст [26]. Таким чином, напрямком із найбільшим

практичним потенціалом є саме доменна адаптація базових архітектур на релевантних даних.

Головним обмеженням трансформерів залишаються високі вимоги до ресурсів. Процес тонкого налаштування вимагає потужних GPU та значних обсягів пам’яті. Крім того, стандартне обмеження довжини вхідних даних у 512 токенів для BERT моделей ускладнює роботу з довгими документами, вимагаючи складних стратегій, як-от розбиття тексту на частини та агрегації результатів (Рис. 1.2), що додатково збільшує обчислювальне навантаження.

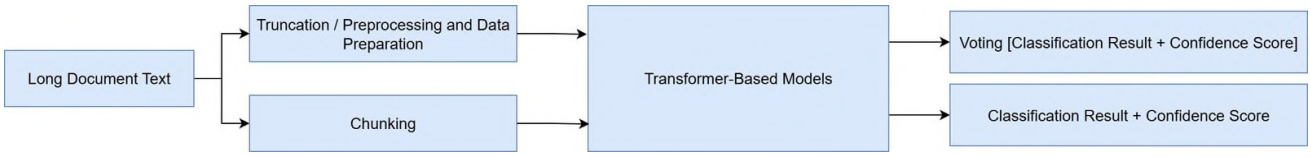


Рисунок 1.2 – Приклад класифікації довгих текстів з використанням трансформерних моделей [26]

Великі мовні моделі (LLM) здатні виконувати широкий спектр завдань, від генерації коду до аналізу текстів, без явного тонкого налаштування для кожної окремої задачі [28]

Як і трансформери, сучасні LLM також базуються на архітектурі трансформера, але зазвичай мають значно більшу кількість параметрів і навчаються великій кількості даних [29]. Це надає їм широкі загальні знання, але ускладнює контроль над виводом у вузьких галузях.

Головною перевагою LLM є здатність до навчання за кількома прикладами або навіть у режимі нульового навчання. Замість ресурсомісткого процесу тонкого налаштування, користувач може сформулювати завдання класифікації у вигляді текстового запиту-інструкції. Це робить LLM неймовірно гнучкими та швидкими у прототипуванні рішень, особливо для нових або складних схем класифікації, де немає великих розмічених наборів даних [28]. Однак точність у такому режимі може бути нижчою, ніж у спеціально налаштованого трансформера, і сильно залежить від конкретного запиту. Крім

того, LLM схильні до так званих «галюцинацій», генерації впевнених, але фактично неправильних відповідей [29].

Поряд із здатністю великих мовних моделей (LLM) працювати у режимі кількох прикладів, існує інший потужний підхід до вирішення складних завдань без великих розмічених даних, а саме ансамблеві методи кластеризації. Вони вирішують фундаментальну проблему неузгодженості результатів окремих алгоритмів, які можуть давати різні розбиття одних і тих самих даних через різні внутрішні припущення або чутливість до параметрів. Замість пошуку єдиного алгоритму, ансамблевий підхід генерує кількість різних кластеризацій, а потім обєднує їх у єдине, більш надійне та стійке рішення. Це підвищує якість кластеризації та зменшує залежність від вибору конкретного методу [30].

Одним з основних підходів у цій галузі є консенсусна кластеризація, або consensus clustering. Її основна ідея полягає в об’єднанні множини окремих розбиттів отриманих, різними алгоритмами з різними параметрами або на різних підвибірках даних у єдине консенсусне розбиття (Рис. 1.3). Цей процес часто реалізується через побудову матриці спільної появи об’єктів в одних кластерах. Як зазначається в огляді щодо зважених ансамблів кластеризації, такий підхід значно підвищує як стійкість, так і надійність фінального результату порівняно з будь-яким окремим методом. Консенсусна кластеризація особливо корисна в областях, де стабільність результатів критично важлива, наприклад, у біоінформатиці для виявлення молекулярних підтипів захворювань [31].

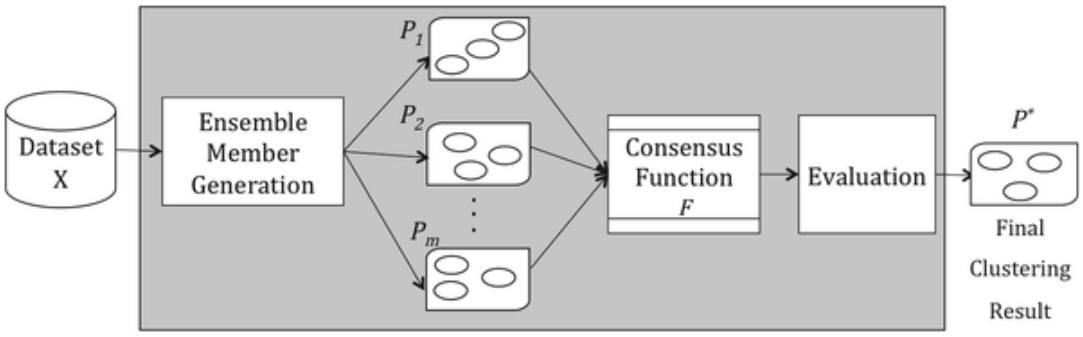


Рисунок 1.3 – Типова схема реалізації консенсусної кластеризації [31]

Тісно пов’язаним підходом є агресована кластеризація, яку іноді розглядають як конкретну задачу в рамках консенсусної кластеризації (Рис. 1.4). Її мета, це знайти єдине розбиття, яке максимально узгоджується з набором вхідних кластеризацій, мінімізуючи сумарну кількість розбіжностей між ними. Якщо консенсусні методи часто працюють через аналіз подібностей, агрегація може формулюватися як задача оптимізації з явною мірою розбіжностей. Таке формулювання дозволяє ефективно інтегрувати навіть суперечливі результати, отримуючи компактне і обґрунтоване рішення. У літературі цей процес описується як пошук консолідованого розбиття шляхом комбінування ансамблевих членів без доступу до оригінальних ознак [30].

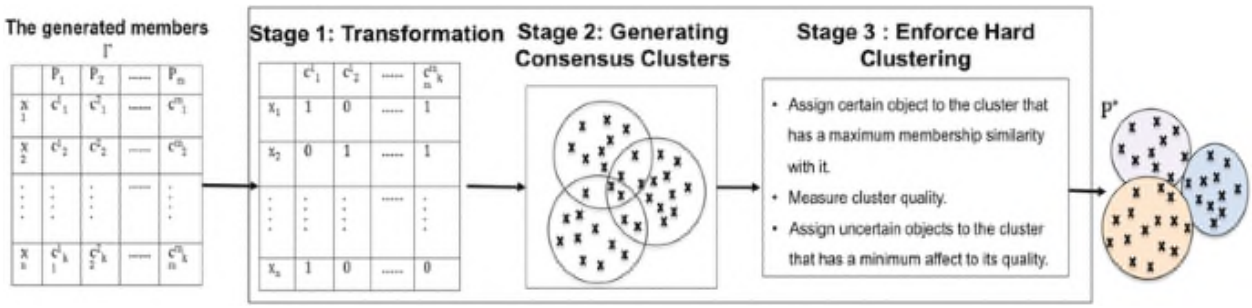


Рисунок 1.4 – Алгоритм роботи ансамблевого методу кластеризації [31]

Впровадження найпотужніших моделей, таких як GPT-4, через API може призвести до значних операційних витрат, особливо при інтенсивному використанні. Альтернативою є використання менших відкритих моделей, але їх розгортання власними силами також вимагає потужної інфраструктури [32]. Для інтеграції LLM в спеціалізовані рішення існують фреймворки, як-от LangChain або LlamaIndex, які спрощують створення складних конвеєрів [32].

Отже, ансамблеві методи, такі як консенсусна та агрегована кластеризація, пропонують потужну альтернативу для ситуацій, коли дані не мають чіткої внутрішньої структури або коли результати стандартних алгоритмів нестійкі. Вони доповнюють підхід LLM, оскільки також не вимагають великих розмічених наборів даних, але зосереджуються на підвищенні надійності та об’єктивності результатів кластеризації шляхом інтеграції множини гіпотез, а не їхньої генерації з нуля.

1.4 Мета та завдання кваліфікаційної роботи

Мета роботи – підвищення якості автоматичної класифікації програмних вимог шляхом розробки методу некерованої кластеризації векторних представлень доменно-адаптованої великої мовної моделі.

Для досягнення мети роботи, необхідно виконати наступні завдання:

– провести аналіз сучасних підходів до автоматизованої класифікації програмних вимог, що базуються на методах обробки природної мови та глибокого навчання.

– розробити метод класифікації програмних вимог, що базується на формуванні векторного простору ознак за допомогою LLM та його подальшій некерованій кластеризації.

– виконати доменну адаптацію мовної моделі для покращення семантичної якості векторних представлень.

– провести експериментальне дослідження розробленого методу із застосуванням ансамблевих підходів, оцінити отримані метрики якості та порівняти їх з базовими підходами та існуючими аналогами.

Розділ 2 Метод класифікації програмних вимог з використанням некерованих методів кластеризації

2.1 Концепція та схема методу класифікації програмних вимог

Основні кроки запропонованого методу класифікації мають наступний вигляд (Рис. 2.1):

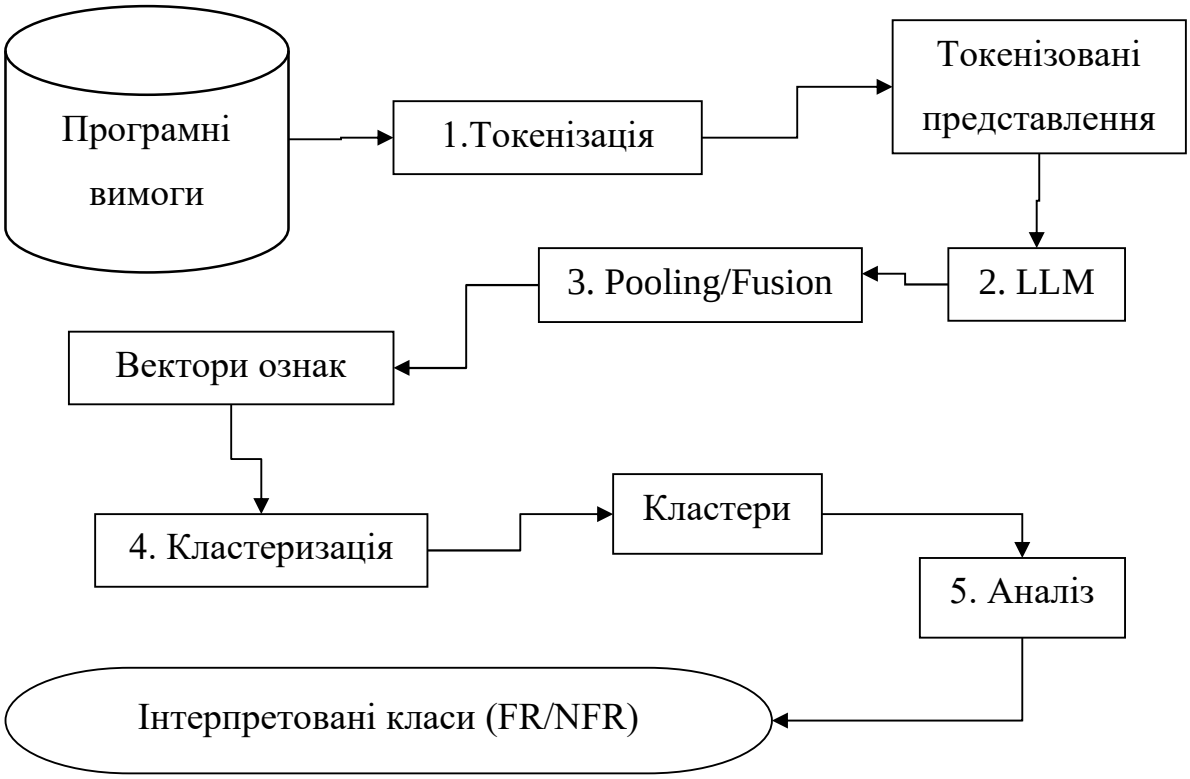


Рисунок 2.1 – Загальна схема методу класифікації програмних вимог, з використанням методів кластеризації

Запропонований метод класифікації починається з опрацювання вихідних програмних вимог, які зберігаються у вигляді текстового корпусу. На першому етапі текст проходить процес токенізації, де відбувається його підготовка до обробки мовною моделлю. Токенізація розкладає вимоги на послідовність лексичних одиниць, що дозволяє моделі сприймати текст не як рядок символів, а

як структуровані елементи, оптимізовані для подальшої семантичної інтерпретації (Рис. 2.2).

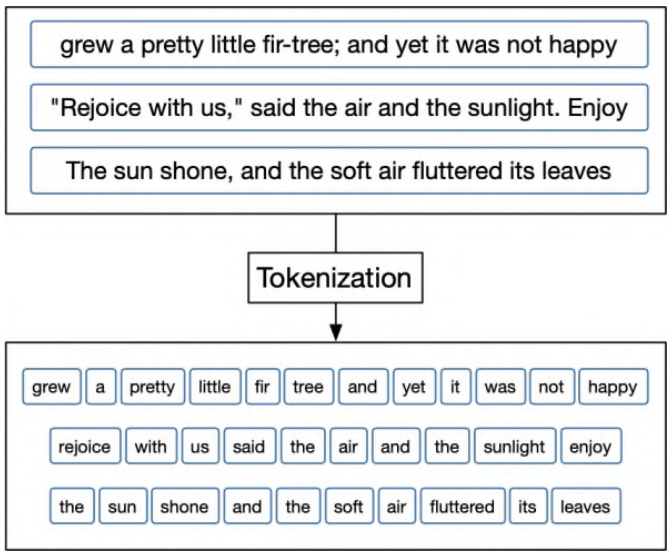


Рисунок 2.2 – Приклад роботи токенізатора [33]

Після цього токенізований текст передається до великої мовної моделі (LLM), яка формує високорівневі репрезентації змісту. На цьому етапі модель ідентифікує приховані смислові зв'язки між частинами тексту, виявляє конструкції, що несуть ключову інформацію, та трансформує вимоги у форму, яка відображає їх семантику значно точніше, ніж традиційні ембеддинги (Рис. 2.3). Варто зазначити, що вивід LLM, зазвичай, має багатовимірний або послідовний характер, що потребує агрегування.

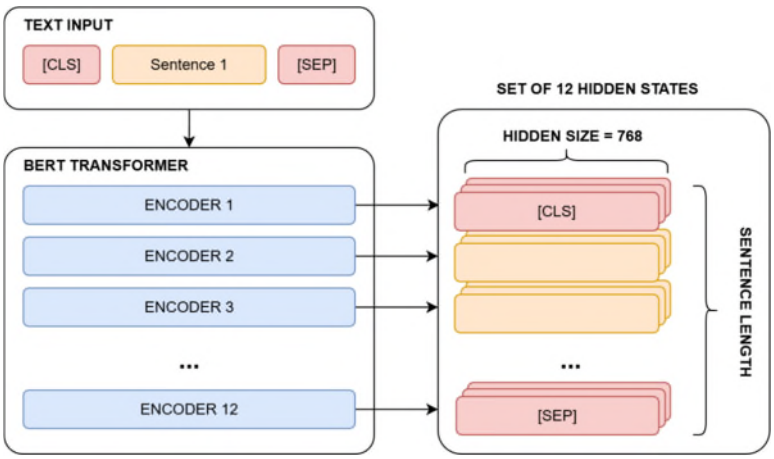


Рисунок 2.3 – Процес створення багатошарових векторних представлень використовуючи модель BERT

Далі формується компактне векторне представлення кожної вимоги. Для цього використовується метод pooling або fusion, який узагальнює вихідні ознаки, повернуті мовною моделлю. Цей компонент забезпечує перехід від послідовності токенів до одного узгодженого вектора ознак, що зберігає найважливіші характеристики. У результаті кожна вимога отримує математичне подання у багатовимірному просторі, придатне для подальшої обробки некерованими методами кластеризації.

Утворений векторний простір використовується в якості вхідних даних для кластеризації. На цьому етапі відбувається аналіз розташування векторів відносно один одного, виявляючи природні групи, які виникають внаслідок семантичної подібності. Кластеризація дає змогу згрупувати вимоги, що мають близький зміст, структуру або функціональну спрямованість, навіть у випадку відсутності ручної розмітки. Результатом є набір кластерів, кожен з яких представляє семантично зв’язані типи вимог.

Після розподілу даних на кластери, на етапі аналізу відбувається зіставлення міток класів з розпізнаними групами. Для кожної групи обраховуються кількості записів для всіх класів, визначається домінуючий клас у групі та його пропорція відносно інших точок. Домінуючий клас у групі приймається як мітка класу, після чого результати розглядаються як звичайна задача класифікації і для її оцінки використовуються метрики F1 Score, Precision, Recall, тощо.

Використання ансамблевого підходу на етапі кластеризації підвищує якість розподілу в порівнянні з поодиноким використанням існуючих алгоритмів кластеризації. На відміну від нейромереж, де ансамблевий підхід використовує безпосередні результати попередніх шарів, мітки кластерів окремих методів кластеризації не несуть в собі будь якої інформації, а слугують лише для розподілу точок на окремі групи. Тому для узгодження результатів розподілу декількох методів кластеризації необхідно застосувати матрицю подібності (Рис. 2.4).

Рисунок 2.4 – Приклад об’єднання результатів двох методів кластеризації в матрицю подібності [34]

Матриця подібності має розмір n за довжиною та шириною, де n це загальна кількість розпізнаних точок. В кожній комірці матриці (i, j) підраховується кількість разів точки i та j були віднесені до одної групи. Оскільки точка завжди сама з собою в кожній кластеризації, діагональні елементи завжди дорівнюють кількості кластеризацій. Маючи значення діагональних комірок, можна поділити матрицю на це число, отримавши ймовірність віднесення точок до одної групи за результатами декількох ітерацій кластеризації (Рис. 2.5).

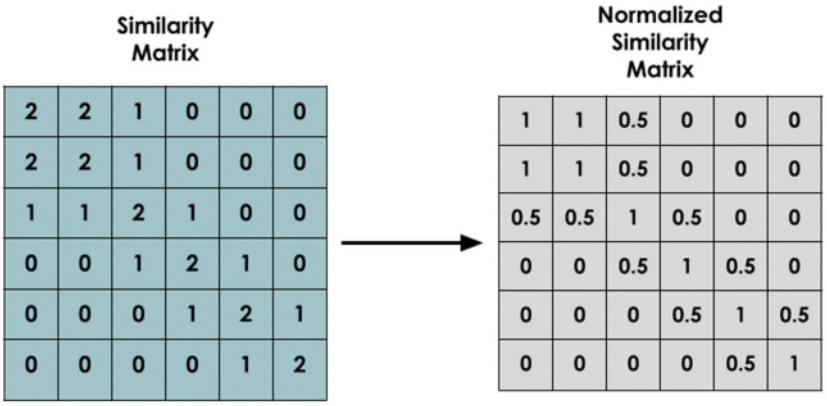


Рисунок 2.5 – Приклад нормалізації матриці подібності [34]

Після формування нормалізованої матриці подібності можна об’єднати точки, що мають ймовірність опинитися в одній групі вищу за деякий обраний

параметр в окремі групи, таким чином залишивши тільки ті угрупування точок, що зустрічаються найчастіше. Приймаючи ймовірності об’єднання точок в одну групу за ребра графу, для формування кінцевого результату прибираються ребра з меншими ймовірностями існування (Рис. 2.6). Таким чином, розподіл точок на групи відбувається згідно кількості подібних результатів різних окремих алгоритмів кластеризації.

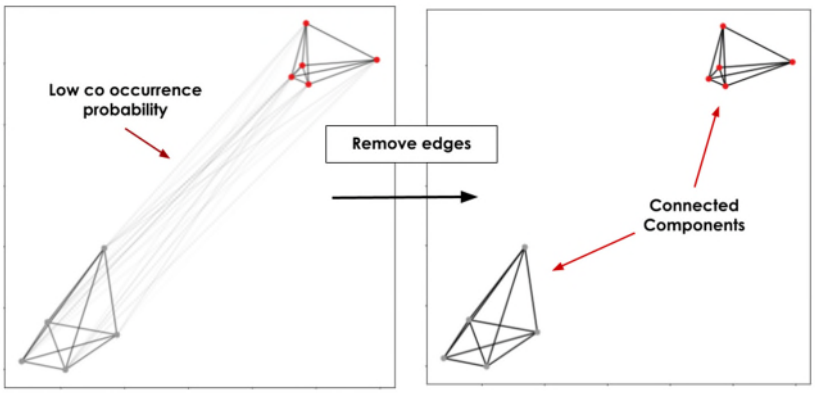


Рисунок 2.6 – Розподіл точок на групи з використанням матриці подібності на прикладі графу [34]

Варто зазначити, що окрім використання заданого параметра мінімальної подібності, для розподілу точок на групи застосовано окремий алгоритм кластеризації для високоякісної агрегації результатів (Рис. 2.7)

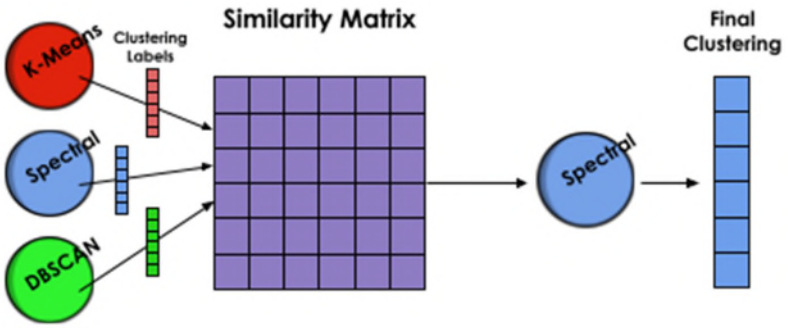


Рисунок 2.7 – Схема використання окремого алгоритму кластеризації для формування кінцевого розподілу точок на групи [34]

Проведення доменної адаптації також значно підвищує точність класифікації програмних вимог. Для доменної адаптації можуть бути

використані різні підходи: навчання класифікатора, контрастивне навчання, використання задачі розпізнавання частин мови, тощо.

2.2 Особливості проведення доменної адаптації мовної моделі

Для доменної адаптації моделі використано три основних підходи: навчання класифікатора, контрастивне навчання та використання задачі розпізнавання частин мови.

Для навчання моделі на задачі класифікації, необхідно створити окрему класифікаційну голову для обраної задачі (Рис. 2.8). Оскільки даний вид навчання орієнтований на класифікацію, доцільним є використання [CLS] токена моделі в якості вхідних параметрів для голови класифікації.

Для розпізнавання частин мови, в якості вхідних даних голова класифікації повинна використовувати всі токени окрім [CLS] (Рис. 2.8), на відміну від навчання класифікації де, навпаки, використовується лише [CLS] токен.

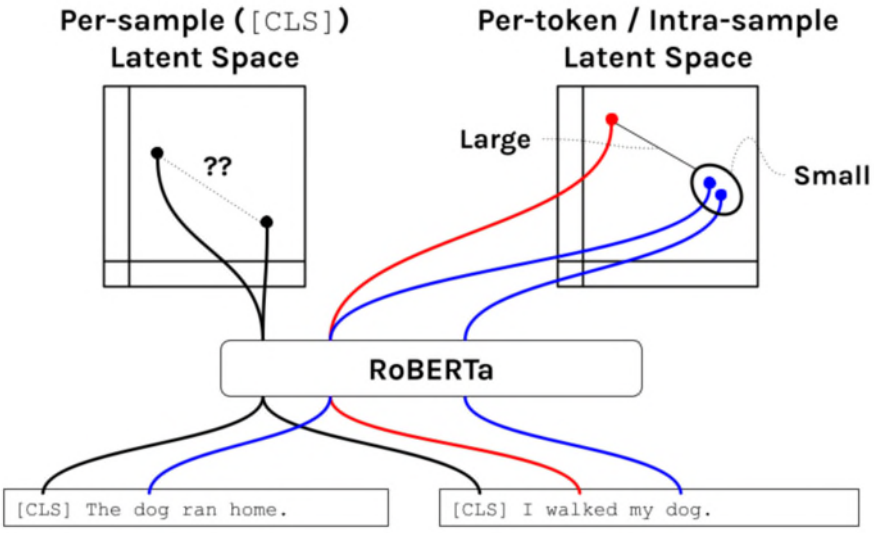


Рисунок 2.8 – Приклад різниці векторного простору між використанням [CLS] токеном та всіх токенів моделі RoBERTa [35]

На відміну від інших видів доменної адаптації, контрастивне навчання саме по собі не потребує створення окремих класифікаційних шарів [36]. Його

основна суть полягає у притягуванні подібних та відштовхуванні різних класів у векторному просторі (Рис. 2.9)

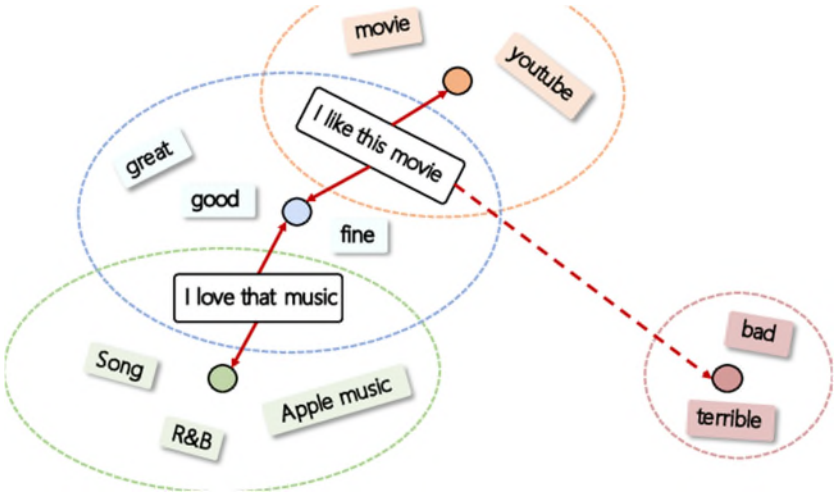


Рисунок 2.9 – Ілюстрація роботи контрастивного навчання для задачі класифікації текстів [37]

Варто зазначити, що при використанні інших методів навчання окрім початкової задачі класифікації, є необхідність збереження функції класифікації, для забезпечення високої якості розподілу по завершенню навчання. Зокрема, алгоритм контрастивного навчання, при роботі з неоднозначними текстами може перетворити векторний простір у змішану хмару без будь якого чіткого розподілу у разі відсутності сигналу задачі класифікації, що надасть початковий імпульс для розділення векторного простору на окремі підгрупи.

Також, для покращення результатів навчання було використано розширення векторів, отриманих після опрацювання текстів токенізатором мовної моделі, використовуючи додаткові анотації з частинами мови. Такі анотації створюються за допомогою існуючих моделей для розпізнавання частин мови та використання словників з ключовими словами, притаманними для функціональних та нефункціональних вимог.

2.3 Формування та підготовка навчальних даних

Для роботи над методом класифікації програмних вимог використано два окремих набори даних: FR_NFR_Dataset [38] та PROMISE_exp [39].

FR_NFR_Dataset характеризується суттєвою перевагою функціональних вимог над нефункціональними, що створює виражений дисбаланс класів. Така структура відображає практичну специфіку багатьох реальних проектів, де функціональні аспекти часто описуються більш детально та численно. Ця особливість може ускладнювати навчання моделей машинного навчання, та потребує застосування спеціальних методів для правильної класифікації менш представленого класу. Він містить в собі 6118 програмних вимог, 3914 з яких є функціональними та 2154 нефункціональними, це означає, що частка функціональних вимог перевищує частку нефункціональних приблизно у 1.8 раза. Для маркування використовуються мітки FR для функціональних та NFR для не функціональних вимог.

PROMISE_exp включає в себе 969 програмних вимог, 444 з яких є функціональними та 525 нефункціональними. Його ключовою особливістю є детальна внутрішня категоризація нефункціональних вимог на низку спеціалізованих підкласів, таких як вимоги до доступності, безпеки, зручності використання та інші. Ця деталізація дозволяє проводити не лише бінарну класифікацію, але й більш глибокий аналіз типології нефункціональних вимог. Функціональні вимоги позначені міткою F, в той же час нефункціональні вимоги розподілені за різними малорозмірними підкласами A, L, LF, MN, O, PE, SC, SE, US, FT, PO.

Для тестування якості класифікації моделі використано FR_NFR_Dataset, після його виділення вибірки розміром 2000 вимог по 1000 вимог для кожного класу.

Для проведенння доменної адаптації моделі використано PROMISE_exp датасет, випадково збалансований на 0.8 навчальної та 0.2 валідаційної вибірки.

2.4 Критерії та метрики оцінювання роботи методу

Для початку роботи необхідно визначити критерії оцінки якості класифікації програмних вимог, які частково випливають з обраного методу класифікації і є ключовими метриками при класифікації програмних вимог.

Для об’єктивної оцінки результатів задач класифікації застосовуються такі метрики, як *accuracy*, *precision, recall* та *f1 score.* Кожна з них характеризує різні аспекти результатів класифікації, тому їх спільне застосування дає змогу отримати загальне уявлення про ефективність класифікації.

Метрика accuracy описує загальну точність класифікації та визначається як частка правильно передбачених прикладів відносно всієї кількості прикладів у тестовій вибірці. Незважаючи на свою простоту, accuracy здатна отримувати високу точність навіть при некоректному розпізнаванні менш представлених категорій, що робить її поганим вибором при класифікації незбалансованих даних:

$$Acuracy = \frac{correct\ classifications}{total\ classifications} = \frac{TP+TN}{TP+TN+FP+FN}, \quad (2.1)$$

У розрахунку використовуються значення TP і TN, які відповідають коректно класифікованим об’єктам, що належать або не належать до певного класу відповідно, а також FP і FN, які відображають помилки першого та другого роду.

На відміну від загальної точності, precision характеризує здатність моделі робити точні позитивні передбачення в межах конкретного класу. Ця метрика показує, яка частка об’єктів, позначених моделлю як позитивні, справді належить до цього класу. Високе значення precision свідчить про низьку частоту хибних спрацьовувань:

$$Precision = \frac{correclty\ classified\ actual\ positives}{everything\ classified\ as\ positive} = \frac{TP}{TP+FP}, \quad (2.2)$$

Метрика recall оцінює, наскільки повно модель виявляє об’єкти, що належать до цільового класу. Вона показує частку реальних позитивних

прикладів, які були правильно ідентифіковані. Високе значення recall означає, що модель пропускає мінімум релевантних об’єктів.

$$Recall\ (or\ TPR) = \frac{correclty\ classified\ actual\ positives}{all\ actual\ positives} = \frac{TP}{TP+FN}, \quad (2.3)$$

Оскільки для метрик precision і recall підвищення одного значення може призвести до зниження іншого, для збалансованої оцінки застосовується F1 score, який представляє собою гармонічне середнє між цими двома метриками. Завдяки одночасному врахуванню і точності, і повноти, F1 score виступає узагальнюючим показником, що робить його одним з оптимальних інструментів для оцінки класифікаторів у складних або незбалансованих задачах.

$$F1\ Score = 2 * \frac{recall * precision}{recall+precision} \quad (2.4)$$

F1-Score є гармонійним середнім між точністю та повнотою, що робить його інтегральним показником якості, завдяки чому вона може бути використана в якості поверхневої метрики, що повністю охоплює можливі види помилок при класифікації, що робить її оптимальним вибором для оцінки якості класифікації.

Метрика silhouette score є однією з найнаочніших та найінтуїтивніших внутрішніх метрик. Її основна ідея полягає в кількісному вираженні двох ключових властивостей кластеризації: згуртованості та відокремленості для кожного окремого об’єкта даних [40]. Формула її обрахунку має наступний вигляд:

$$s(i) = \frac{b(i)-a(i)}{\max\{a(i),b(i)\}}, \quad (2.5)$$

де a(i) представляє собою середню відстань від конкретної точки до всіх інших точок в тому ж кластері та b(i) – середню відстань до точок в інших кластерах [41]. Для повної оцінки кластеризації використовується середнє значення silhouette score для всіх точок в просторі:

$$S = \frac{1}{\mathrm{n}}\sum_{i=0}^{n} s(i), \quad (2.6)$$

Значення silhouette score, близьке до 1, свідчить про ідеальну кластеризацію, де точка знаходиться далеко від сусідніх кластерів і добре

вписана у свій власний, наближення метрики до 0 означає, що точка знаходиться на межі між двома кластерами та наближення до -1 вказує на те, що точка, ймовірно, призначена неправильно та належить до сусіднього кластера [42].

Основна перевага Silhouette Score – це можливість оцінювати кластеризацію без еталонних міток, що робить його незамінним для справді неконтрольованого навчання [40]

Але, вона найкраще працює з опуклими, сферичними кластерами та може давати оманливі результати для кластерів складної форми, які часто виявляються щільнісними алгоритмами на кшталт DBSCAN [40]. Обчислення попарних відстаней також може бути обчислювально витратним для дуже великих наборів даних [42].

ARI (Adjusted Rand Index) вимірює схожість між двома розбиттями даних: одним, отриманим алгоритмом кластеризації, та еталонним розбиттям (ground truth) [43, 44]. Він представляє собою вдосконалення Індексу Ранда (RI), який обчислює частку пар об’єктів, узгоджених у двох розбиттях (тобто або знаходяться в одному кластері в обох розбиттях, або в різних) [44]. Однак основний недолік RI полягає в тому, що його очікуване значення для випадкового розбиття не дорівнює нулю, що ускладнює інтерпретацію [43]. ARI виправляє цю проблему, нормуючи результат так, щоб очікуване значення для випадкового розбиття було рівним 0, а ідеальне співпадіння давало 1 [44, 45].

Математично ARI базується на таблиці подібності, яка показує перетин між кластерами двох розбиттів. Формула, хоча і виглядає складною, по суті порівнює кількість спільних пар з очікуваною кількістю за умови випадкового розподілу:

$$ARI(P^*, P) = \frac{\sum_{i,j}\binom{N_{ij}}{2} - \frac{\left(\sum_i \binom{N_i}{2}\right)\left(\sum_j \binom{N_j}{2}\right)}{\frac{N}{2}}}{\frac{1}{2}\left[\sum_i \binom{N_i}{2} + \sum_j \binom{N_j}{2}\right] - \frac{\left(\sum_i \binom{N_i}{2}\right)\left(\sum_j \binom{N_j}{2}\right)}{\frac{N}{2}}}, \quad (2.7)$$

де P^* розподіл з істинними мітками, P – це розпізнаний розподіл, N – загальна кількість даних, N_{ij} – кількість даних що збігаються між розпізнаним та істиним

розподілом, N_i – розмір істинного кластеру i, N_j – розмір розпізнаного кластеру j.

Діапазон значень ARI становить від приблизно -0.5 до 1 [45]. Значення ARI що наближаються до 1 свідчать про високу відповідність між даома розбиттями, наближеність до 0 є ознакою випадкової класифікації та значення менші за 0 говорять про якість класифікації, що є гірша за випадкове розбиття [44].

ARI є стандартним інструментом в областях, де існує надійна еталонна розмітка: біоінформатика для кластеризації експресії генів або типів клітин, аналіз соціальних мереж для оцінки виявлення спільнот, сегментація зображень [43, 44]. Також, важливо зазначити, що ARI є чутливий до кількості кластерів, при чому порівняння розбиття з різною кількістю груп може бути не цілком справедливим [43]. Крім того, метрика вимагає наявності істинних міток, які часто відсутні або складно отримати в реальних завданнях неконтрольованого навчання.

Висновки до розділу 2

Отже, у даному розділі подано концепцію методу класифікації програмних вимог, особливості доменної адаптації, формування датасетів та ключові метрики для оцінки якості кластеризації.

Запропоновано метод автоматизованої класифікації, який включає процеси генерації контекстних векторів моделлю RoBERTa, агрегації ознак та застосування ансамблевих алгоритмів для підвищення стійкості розподілу. Теоретично обґрунтовано необхідність доменної адаптації через процедури стандартного та контрастивного навчання для покращення семантичної структури векторного простору.

Здійснено підготовку набору даних PROMISE для донавчання моделі та збалансованого корпусу FR_NFR_Dataset для валідації. Визначено комплекс метрик (F1-Score, Precision, Recall) для об’єктивного оцінювання ефективності запропонованого підходу.

Розділ 3 Програмна реалізація методу класифікації програмних вимог

3.1 Засоби та середовище програмної реалізації

При розробці методу класифікації програмних вимог в якості мовної моделі було використано RoBERTa [46] завдяки відносно малим потребам в обчислювальних ресурсах при її використанні або проведення навчання, у порівнянні з іншими великими мовними моделями.

Оскільки для тестування методу є необхідність часто змінювати параметри, для уникнення частих перезапусків програми доцільним є використання Jupyter Notebook в якості фреймворку.

Через значні потреби в обчислювальних ресурсах при роботі з мовною моделлю є необхідність використання хмарного середовища. Google Colab є сумісним з Jupyter Notebook, доступним безкоштовно з обмеженнями на використання ресурсів відеокарти та простим у використанні, що представляє собою оптимальний вибір для експериментів.

При роботі з Google Colab завантаження даних відбувається з використанням Google Disk, що створює додаткові вимоги до сумісності коду з файловим середовищем.

Оскільки середовище Google Colab має обмеження по використанні обчислювальних ресурсів, доцільним є передбачення сумісності як з хмарним так і локальним середовищем у разі обмежень, для забезпечення неперервності процесу при можливих обмеженнях хмарного середовища.

3.2 Архітектура програмної реалізації та функціональні модулі

Розроблена інформаційна система побудована за шаблоном Jupyter Notebook, з розподілом даних та функціоналу на окремі директорії: Datasets (набори даних), Modules (модулі) та Models (збережені моделі). Набори даних слугують в якості вхідних даних для задачі класифікації програмних вимог, модулі містять в собі основну логіку обробки даних та взаємодії з мовними

моделями, збережені моделі представляють собою результати процесу доменної адаптації для покращення якості кластеризації.

Для забезпечення можливості проведення багатьох експериментів без необхідності перезапуску застосунку з іншими параметрами, було використано розподіл функціоналу на модулі, що викликаються почергово в основному файлі Jupyter Notebook (Рис. 3.1).

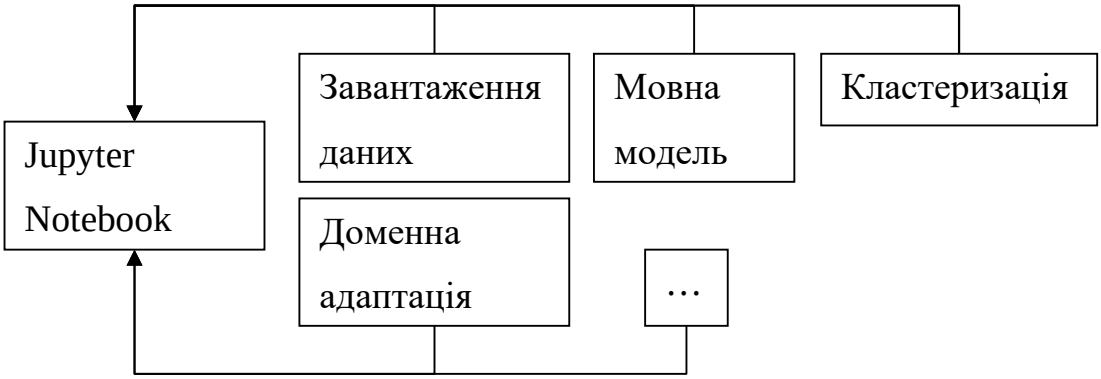


Рисунок 3.1 – Схематичний вигляд шаблону проєкту Jupyter Notebook

При роботі застосунку взаємодія основних компонентів має наступний вигляд (Рис. 3.2):

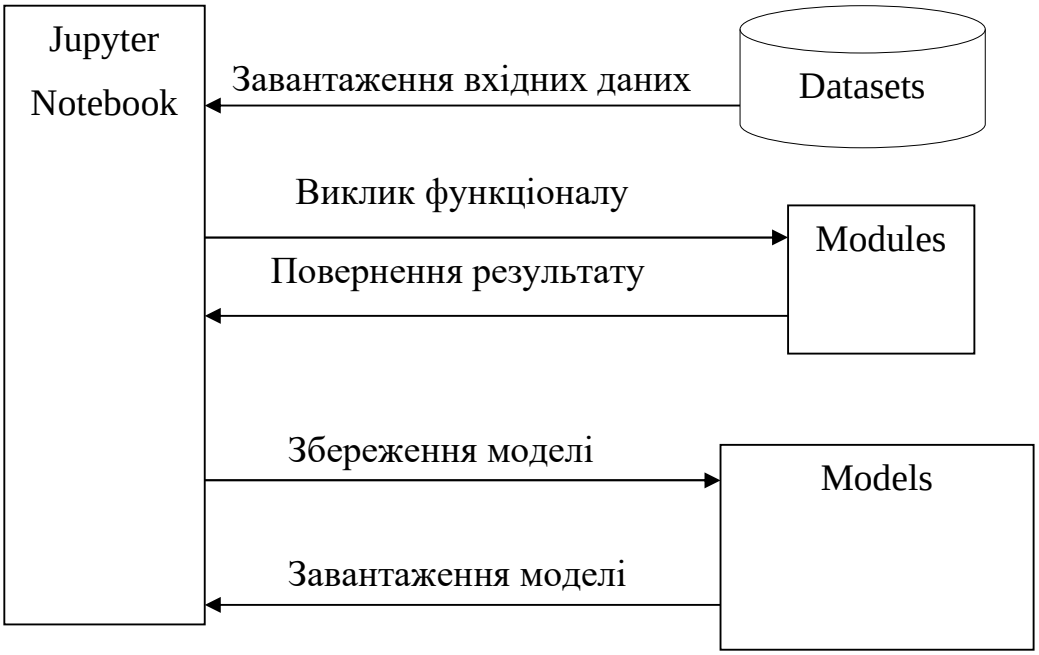


Рисунок 3.2 – Схематичне зображення діаграми взаємодії між основними типами компонентів

Модулі взаємодіють не тільки з основною виконуваною частиною але і між собою (Рис. 3.3).

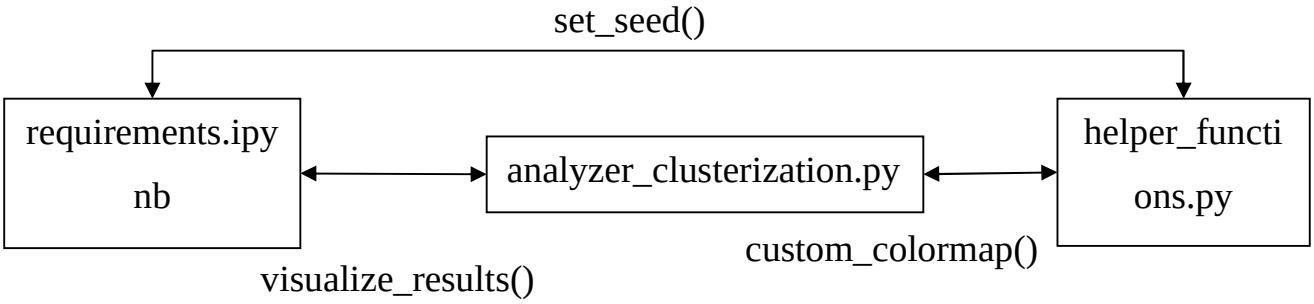


Рисунок 3.3 – Взаємодія між модулями analyzer_clusterization.py, helper_functions.py та основним файлом requirements.ipynb

Зокрема модуль helper_functions.py вміщує в собі функцію створення набору кольорів для візуалізацій, що використовується в модулях analyzer_clusterization.py і analyzer_inference.py та окрему функцію встановлення параметрів випадкових генераторів чисел для забезпечення повторюваності експериментів.

Datasets вміщує в собі використані набори даних для проведення доменної адаптації. При роботі застосунку наявні дані лише завантажуються у пам’ять та не підлягають будь яким змінам.

В межах Models зберігаються навчені моделі після проведення доменної адаптації. Збережені моделі розділені на окремі підпапки в залежності від датасету, використаного під час навчання. Окрема папка TMP зберігає проміжні моделі створені під час навчання.

Папка Modules забезпечує функції та бібліотеки для підтримки експериментального циклу. Вона містить функціонал початкової обробки датасету при його завантаженні, створення векторних представлень за допомогою мовної моделі та їх pooling для ефективної обробки, кластеризації та оцінки якості кластеризації, функції навчання для доменної адаптації моделі та візуалізації результатів використання методів класифікації.

Модуль helper_functions.py забезпечує відтворюваність результатів, візуалізацію даних та моніторинг ресурсів під час проведення експериментів. Функція set_seed встановлює детермінований режим для всіх компонентів випадкових процесів. Ініціалізація генераторів випадкових чисел у бібліотеках PyTorch, NumPy та стандартній бібліотеці Python гарантує однакові результати при повторних запусках та забезпечує стабільність обчислень на графічних процесорах. Функція aggregate_classes реалізує механізм агрегації прогнозів між системами класів різної деталізації. Перетворення матриць ймовірностей відбувається на основі словника відображення вихідних класів у цільові. create_custom_tab_colormap розширює можливості візуалізації через створення унікальних наборів кольорів. Послідовне комбінування кольорів з множини стандартних палітр забезпечує візуальний контраст серед великої кількості кластерів.

Модуль dataset_text реалізує функціонал обробки текстів програмних вимог з підтримкою гнучких стратегій токенізації для подальшого використання з великими мовниви моделями. Клас TextDataset наслідує стандартну структуру PyTorch Dataset, забезпечуючи сумісність з функціоналом навчання з torch. Об’єкт датасету підтримує два режими обробки текстів: попередню токенізацію всієї колекції та динамічну токенізацію під час кожної ітерації. Попередня обробка забезпечує швидший доступ до даних ціною збільшеного використання пам’яті. Динамічний підхід економить ресурси для великих корпусів але може значно знизити швидкість будь яких подальших операцій з датасетом. Використання RobertaTokenizerFast за замовчуванням забезпечує оптимізоване перетворення текстів у числове представлення сумісне з моделями на основі roberta-base. Також тут наявний функціонал об’єднання числових представлень отриманих за допомогою токенізатора та додаткових даних в один вектор, що може бути використано для більш складних операцій навчання.

dataset_balanced.py призначений для балансування різних класів вибірки для вирівнювання розподілу класів у навчальних даних та обмеження розміру даних при роботі з великими дата сетами. Об’єкт BalancedSubsetDataset представляє собою обгортку для існуючого об’єкту датасету, що балансується за

класами згідно параметрів. При його ініціалізації відбувається адаптація до типу початкового датасету. Для Subset-об’єктів після проведення випадкового розбиття датасету замість його унікальних методів використовується початкова структура датасету. Такий підхід забезпечує можливість роботи з як з підмножинами після випадкового розподілу так і з повноцінними об’єктами датасету. Перед проведенням балансування оцінюється кількість екземплярів кожного класу у вхідних даних, після чого випадковим чином вибирається необхідна кількість записів з кожного класу згідно параметрів. Створена збалансована вибірка містить однакову кількість записів для кожного класу, або всі можливі записи для тих класів які є меншими ніж параметр розміру вибірки.

Модуль LLM.py містить в собі основний функціонал використання мовної моделі поза межами навчання. Також, тут присутні використані архітектури класифікаційних моделей для проведення доменної адаптації моделі (Рис. 3.4). Функція hidden_states_pooled забезпечує отримання та агрегацію прихованих станів з трансформерних моделей використовуючи різні стратегії pooling згідно заданих параметрів для об’єднання як окремих векторів в межах одного шару за параметром intra_layer_pool так і між декількома прихованими шарами моделі inter_layer_pool. Також дана функція надає можливість змінити типи числового представлення даних між float16 та float32 згідно вхідного параметра dtype. Модель LLMClassifier реалізує стандартну архітектуру з двома лінійними шарами та нормалізацією. Використання CLS-токена з останнього прихованого стану як вхідного представлення для голови класифікації забезпечує ефективну базову модель для класифікації. BlockLLMClassifier представляє собою ту ж саму архітектуру з використанням torch.nn.Sequential для голови класифікації, формуючи два чітко розділених об’єкта мовної моделі та голови класифікації для більш простого використання та ефективності. MultiTaskLLMClassifier розширює функціональність додаючи додаткову голову класифікації, що використовує усі інші токени окрім CLS для роботи з окремими частинами мови.

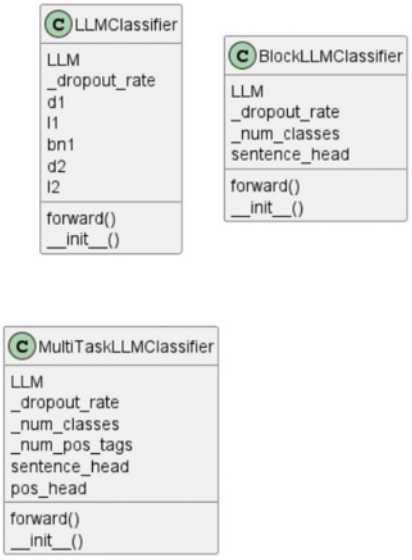


Рисунок 3.4 – Структури класів використаних моделей для доменної адаптації

Для аналізу якості кластеризації використовується функціонал analyze_clusterization.py. Клас ClusterAnalyzer реалізує систему аналізу результатів кластеризації з виводячи текстові метрики якості та візуалізації. Конструктор класу приймає ембеддинги, метадані та метод кластеризації з опціональною нормалізацією (Рис. 3.5). Підтримка різних стратегій обробки точок, щ обули розпізнані як шум, через параметр outlier_handling забезпечує гнучкість у роботі з алгоритмами, що підтримують виділення точок в якості шуму. Методи fit_predict та predict запускають процес застосування алгоритмів кластеризації до стандартизованих векторних представлень. В кінці обидвох методів в них відбувається виклик _analyze_cluster_class_mapping, що забезпечує безпосередньо аналіз результатів кластеризації. _analyze_cluster_class_mapping реалізує статистичний аналіз зв’язку між кластерами та істинними мітками. Визначення домінантних класів для кожного кластера шляхом визначення класу з найбільшою кількістю записів формує базову модель класифікації. Також відбувається розрахунок показників чистоти кластерів та розподілів ймовірностей для більш отримання більш деталізованих результатів класифікації. Даний клас підтримує три підходи до роботи з точками розпізнаними як шум, а саме: класифікація «шуму» як окремого класу, його опрацювання як тільки помилкових прогнозів або аналіз тільки розпізнаних

кластерів без урахування «шуму». Методи outliers_are_wrong та trim_outliers реалізують відповідні стратегії обробки точок, що були розпізнані як шум. Метод analyze_cluster_quality поєднує кластерні та класифікаційні метрики для всебічної оцінки. Розрахунок силуетних оцінок для різних рівнів угруповання дозволяє аналізувати внутрішню узгодженість. Інтеграція традиційних метрик класифікації забезпечує порівняння з контрольованими методами. Функція visualize_results створює графіки для інтуїтивного аналізу результатів. Функція print_detailed_report генерує структурований звіт про якість кластеризації, розподіл кластерів та ефективність класифікації для різних стратегій обробки викидів. save_clustering_results забезпечує збереження результатів у структурованому CSV-форматі. Включення оригінальних текстів, міток, прогнозів та ідентифікаторів кластерів дозволяє проводити подальший аналіз даних.

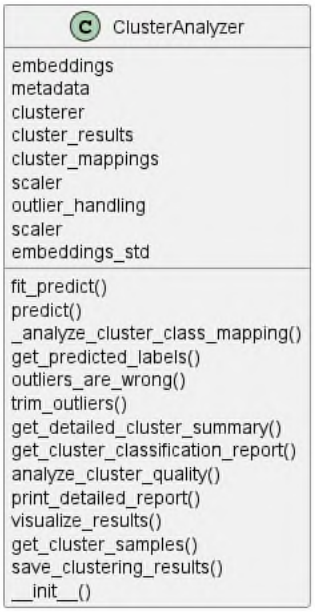


Рисунок 3.5 – Структура класу ClusterAnalyzer для аналізу результатів кластеризації

Модуль clustering_ensemble.py вміщує в собі необхідні класи для застосування ансамблів при використанні алгоритмів кластеризації (Рис. 3.6). Клас ConsensusClustering реалізує ансамблеву стратегію кластеризації на основі усереднення множини базових алгоритмів. Архітектура поєднує переваги різних

методів групування через побудову матриці подібності. Ініціалізація приймає список базових кластеризаторів та поріг ймовірності. Процедура fit послідовно застосовує кожен алгоритм до вхідних даних та накопичує статистику кластеризації. Після застосування алгоритмів кластеризації за заданим порогом утворюється граф суміжності, де визначення зв’язаних компонент у цьому графі за допомогою алгоритму connected_components формує фінальні кластери. Кожна зв’язана компонента відповідає окремому кластеру ансамблевому рішенні. Клас AggregatedClustering реалізує двоетапну архітектуру агрегації кластеризацій. Тут, аналогічно до ConsensusClustering на першому етапі генерується множина базових кластеризацій, а на другому застосовується окремий кластеризатор для об’єднання результатів. Допоміжний клас ClusterSimilarityMatrix призначений для побудови матриці подібності точок у кластерах.

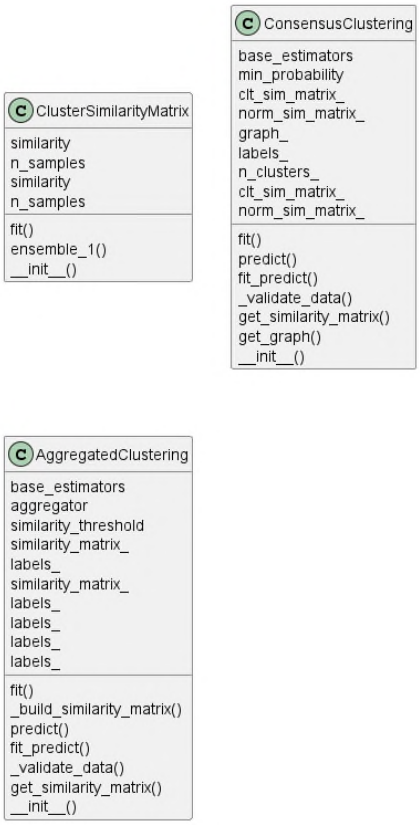


Рисунок 3.6 – Класи модуля clustering_ensemble.py

Модуль layer_combinations.py вміщує в собі функціонал оцінки комбінацій шарів моделі і методів кластеризації та пошук найкращої комбінації шарів моделі та їх pooling методів використовуючи метод повного перебору усіх можливих комбінацій. Допоміжний модуль layer_functions.py забезпечує розділені функції pooling як між окремими шарами моделі так і агрегацію даних в межах одного шару різними методами.

Модуль training_normal.py та його основний клас NormalClassifierTrainer реалізує функціонал навчання моделі за задачею класифікації. Його функціонал розподілений серед трьох основних методів: train_epoch, validate_epoch та train. train_epoch забезпечує одну епоху навчання з повним проходом по тренувальному датасету. validate_epoch виконує оцінку моделі на валідаційному наборі. Метод train координує повний процес навчання протягом заданої кількості епох, забезпечуючи послідовне виконання тренувальних та валідаційних циклів кожну епоху на протязі навчання. Також метод train забезпечує збереження найкращої моделі за критерієм F1-score, що дозволяє вибрати оптимальну версію моделі після завершення навчання.

Модулі training_contrastive.py та training_POS.py надають можливість навчання одночасно використовуючи констрастивний метод або додаткову задачу класифікації частин мови відповідно та початкову голову класифікації. Для контрастивного навчання замість вихідних значень голови класифікації використовуються безпосередньо векторні представлення текстів. Такий підхід сприяє більш чіткому розподілу між класами у векторному просторі, безпосередньо впливаючи на результати алгоритмів кластеризації. Навчання за задачею розпізнавання частин мови може бути застосовано для встановлення зв’язку між деякими словами та структурами і класами, потенційно покращуючи якість класифікації при тестуванні на окремому наборі даних.

Основний файл проєкту requirements.py забезпечує сумісність як хмарним середовищем Google Colab так і локальним середовищем роботи. В його межах відбуваються виклики функціоналу модулів для завантаження наборів даних та моделей, розпізнавання даних моделлю, кластеризації отриманого векторного простору, формування ансамблів для кластеризації та доменної адаптації моделі.

3.3 Особливості реалізації методу класифікації програмних вимог

Завантаження початкових текстових наборів даних відбувається засобами бібліотеки pandas, що надає уніфіковану структуру для читання, очищення та фільтрації табличних даних. Цей етап забезпечує перетворення текстів і супровідних анотацій на структурований формат, придатний для подальшої обробки у контексті моделей глибокого навчання.

Після первинної підготовки корпусу здійснюється токенізація за допомогою швидких токенізаторів із пакету HuggingFace Transformers. Використання RobertaTokenizerFast або аналогічних інструментів дає змогу отримати оптимізоване представлення тексту у вигляді послідовностей ідентифікаторів токенів та відповідних масок уваги. Токенізатор забезпечує коректне вирівнювання текстових фрагментів зі структурою попередньо навчених моделей, а також автоматичне обрізання та паддінг послідовностей із заданими обмеженнями довжини. Завдяки наявності offset-відображень забезпечується подальша інтеграція додаткових лінгвістичних характеристик.

Для отримання частиномовних характеристик корпусу застосовується система лінгвістичного аналізу spaCy. Обраний тегер формує послідовність POS-тегів для кожного токена в межах природної сегментації тексту. Після цього тегування узгоджується з результатами токенізації Transformers, що дає змогу приєднати ознаки до векторних представлень текстів. Інструментарій spaCy забезпечує стабільність та відтворюваність результатів у рамках великих корпусів і дає змогу використовувати готові статистичні моделі без потреби у низькорівневому аналізі.

Структурування отриманих ознак у формат, сумісний із моделями глибинного навчання, здійснюється за допомогою модулів PyTorch. Інтерфейс Dataset визначає спосіб подання окремих елементів датасету у вигляді тензорів, включно з входами моделей, масками, лінгвістичними тегами та супровідними мітками. Тензорні представлення переводяться на графічні процесори

стандартними механізмами PyTorch, що забезпечує ефективне використання апаратних ресурсів під час подальшого навчання або тестування.

На етапах, пов’язаних із ітеративними процесами допоміжні бібліотека tqdm надає можливість спостерігати за прогресом підготовки даних або навчання моделі, шляхом відображення поточної ітерації, загальної кількості ітерацій, затрачений час, необхідний час до завершення, кількість ітерацій на одиницю часу та будь які інші параметри, що можуть бути додані окремо.

Ініціалізація моделі RobertaModel забезпечує завантаження попередньо навчених ваг архітектури RoBERTa. Багатошаровий трансформер з механізмом уваги формує контекстуалізовані векторні представлення вхідних текстів. Використання пулінгу вихідних ембеддингів дозволяє отримати фіксовано розмірні вектори незалежно від довжини вхідної послідовності.

При активному використанні різних моделей та наборів даних доцільним є використання бібліотеки gc, що надає функціонал очищення пам’яті, забезпечуючи можливість проведення великої кількості безперервних ітерацій з великими об’ємами даних або параметрів, запобігаючи переповненню пам’яті при операціях з великими мовними моделями.

Для вирішення задач кластеризації векторних представлень програмних вимог застосовується широкий спектр алгоритмів із бібліотеки scikit-learn. Клас KMeans реалізує стандартний метод k-середніх, що формує сферичні кластери на основі мінімізації дисперсії. Модифікація MiniBatchKMeans пропонує стохастичну оптимізацію для роботи з великими обсягами даних, де обробка по частинах зменшує обчислювальні витрати при збереженні якості кластеризації.

Алгоритм DBSCAN забезпечує виявлення кластерів довільної форми шляхом аналізу щільності розподілу точок у векторному просторі. Розширена версія HDBSCAN удосконалює цей підхід через ієрархічне оцінювання щільності, що дозволяє автоматично визначати кількість кластерів та стійко працювати з різноманітними щільностями розподілу. Метод OPTICS продовжує цей напрямок, формуючи досяжності точок для виявлення кластерів у варіюючих щільностях.

Гібридний підхід GaussianMixture реалізує кластеризацію на основі суміші гаусових розподілів, що дозволяє моделювати складні багатомодальні розподіли даних. Алгоритм SpectralClustering використовує спектральне згортання графів подібності для виявлення кластерних структур. Ієрархічний AgglomerativeClustering будує деревоподібну структуру кластерів через послідовне об’єднання найближчих елементів. Інструмент Birch орієнтований на ефективну обробку великих потоків даних шляхом побудови дерев для компактного представлення підкладстерів.

Для розподілу структурованого набору даних на тренувальні та валідаційні підмножини застосовується функція random_split з бібліотеки PyTorch. Цей механізм забезпечує розподіл з випадковим перемішуванням елементів, що гарантує репрезентативність обох вибірок. Співвідношення між тренувальною та валідаційною частинами встановлюється використовуючи вхідні параметри.

Інкапсуляція датасетів у об’єкти DataLoader забезпечує ефективну ітеративну подачу даних у навчальний процес. Зміна параметрів розміру batch, перемішування та паралельного завантаження дозволяє оптимізувати використання обчислювальних ресурсів. Механізм паралельного завантаження з використанням множинних процесів усуває вузькі місця в підготовці даних під час навчання моделей.

Функція get_linear_schedule_with_warmup реалізує стратегію динамічного коригування швидкості навчання. Початкова фаза розігріву забезпечує плавний старт навчання з поступовим збільшенням LR, що стабілізує навчання на ранніх етапах. Подальше лінійне зменшення швидкості навчання забезпечує збіжність оптимізаційного процесу до стабільного мінімуму функції loss.

Бібліотека matplotlib.pyplot забезпечує функціонал для візуалізації результатів кластеризації та оцінки якості моделей. Створення складних багатокомпонентних графіків здійснюється через механізм підграфіків, що дозволяє одночасно відображати декілька графіків та типів даних в межах одного зображення.

Власна функція create_custom_tab_colormap формує кольорові палітри для візуального відображення кластерів. Алгоритм послідовно комбінує кольори з множини стандартних палітр, забезпечуючи унікальні кольорові схеми для будь якої конкретної кількості кластерів.

Застосування алгоритму PCA бібліотеки sklearn забезпечує оптимальне двовимірне представлення багатовимірних ембедингів, дозволяючи створювати двовимірне зображення, охоплює основні особливості векторного простору.

torch.nn.Module надає можливість створення додаткових шарів для задачі класифікації під час навчання. Для проведення доменної адаптації великої мовної моделі для кожного завдання може бути створена окрема голова класифікації.

Для ефективної роботи з великими мовними моделями необхідно вибрати найпродуктивніший пристрій. Зазвичай, використання відеокарти CUDA є найкращим вибором завдяки значно вищій швидкості проведення обчислень

Оскільки нам необхідно використовувати як хмарне так і локальне середовище, доцільним є проведення адаптації до обраного середовища. Для Google Colab можна виконати перевірку змінної середовища COLAB_RELEASE_TAG, що присутня лише в середовищі Colab.

Для класифікації частин мови, було використано модель en_core_web_sm з бібліотеки spacy.

При підготовці набору даних в якості вхідних параметрів використовується список міток та назви комірок у таблиці що репрезентують тексти та мітки. Такий підхід дозволяє використовувати різні набори даних без необхідності створення окремих класі.

Для отримання векторного простору передбачено можливість використання різних стратегій pooling та вихідних форматів даних. За відсутності обраної стратегії pooling між шарами формується список векторів для кожного окремого шару. Серед вихідних форматів даних існує можливість вибрати numpy або torch tensor. Також передбачена можливість переключення між значеннями float32 та float16 для прискорення обчислень та мінімізації потреби в памяті.

Для створення ансамблів можна формувати список алгоритмів кластеризації для кожного етапу ансамблю. Це дозволяє швидко змінювати архітектуру ансамблів без необхідності створення додаткових класів для кожної унікальної комбінації алгоритмів в ансамблі. Клас ансамблю, аналогічно до окремих алгоритмів кластеризації, вміщує в собі методи *fit, predict* та *fit_predict* може бути використаний так само як і поодинокі алгоритми кластеризації

Для ефективної роботи з набором текстових даних передбачено декілька стратегій початкової обробки даних. Для уникнення додаткових перевірок функція, що повертає дані вибирається при створенні об’єкта датасету та зберігається в якості змінної з посиланням на неї _get_item_fn. Після чого отримання запису з набору даних виконується викликом функції за її посиланням, що дозволяє уникати вибір методу обробки при кожному запиті.

Для забезпечення можливості перевірки різних комбінацій шарів та методів pooling для кожної оцінки кластеризації проводиться додаткова операція об’єднання шарів обраним методом. Це дозволяє тестувати різні комбінації шарів моделі без необхідності повторення розпізнавання даних для кожної з них.

Під час використання ансамблевих алгоритмів для кожного окремого методу відбувається виклик функціоналу матриці подібності. Матриця подібності представляє собою окремий об’єкт, який зберігає в собі всі значення і додає нові результати кластеризації. Оскільки для збереження даних використовується тільки одна матриця, в якій додаються комбінації точок для кожного виклику, уникається необхідність додаткових матричних операцій та збереження матриць подібності для кожного окремого алгоритму кластеризації, що значно спрощує проведення ітерацій. Для підвищення якості кластеризації, при використанні ансамблю з функцією агрегації, матриця подібності використовується в якості вхідних даних в обраний метод кластеризації.

Під час аналізу результатів кластеризації використовується декілька основних стратегії обробки точок, що були розпізнані як «шум». За замовчуванням усі точки, що були розпізнані як шум використовуються в якості окремої групи, розпізнаної кластеризатором.

Для використання точок «шуму» як неправильних відповідей, їм присвоюється мітка класу «-1», що не являється наявним класом в більшості наборів даних.

Для ігнорування «шуму» при оцінці якості кластеризації перед оцінкою всі точки «шуму» прибираються зі списку розпізнаних кластеризатором груп перед обрахунком метрик якості класифікації.

На етапі підготовки до навчання виконується визначення кількості кроків навчання для використання планувальника *scheduler*, що змінює значення lr між кроками навчання. В якості одного кроку такі алгоритми використовують опрацювання одного набору batch в межах одної епохи навчання

Під час доменної адаптації було використано планувальник *linear_schedule_with_warmup*, що лінійно збільшує значення lr для «розігріву», та забезпечує його лінійне спадання на протязі всіх кроків навчання після «розігріву» (Рис. 3.7).

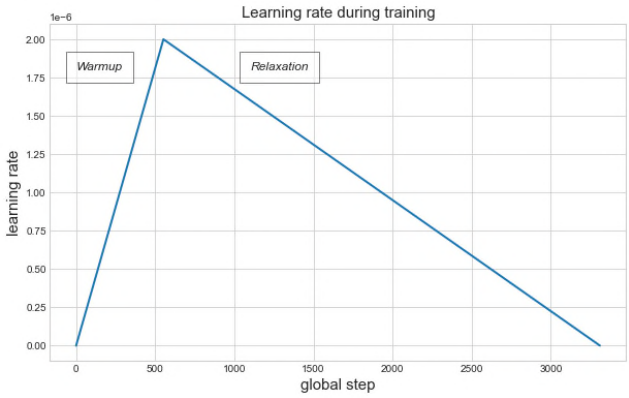


Рисунок 3.7 – Графік значення lr на протязі навчання при використанні *linear_schedule_with_*warmup бібліотеки *transformers* [47]

Для проведення контрастивного навчання було використано окрему loss функцію:

$$L = \frac{1}{\mathrm{N}}\sum_{i=1}^{n}[y_i * d_i^2 + (1 - y_i) * \max(0, m - d_i)]^2, \quad (3.1)$$

де N – кількість записів, y це мітка класу, d являється відстанню між векторами та m є параметром margin. Мітка класів представляє собою значення 1 для схожих та 0 для різних векторів, таким чином призводячи до вищого значення

втрат для близьких різних векторів та меншого значення для векторів одного класу. Для метрики відстаней контрактивного навчання передбачено можливість використання відстані в евклідовому просторі та нормалізовану метрику відстані.

Використання додаткової задачі POS для доменної адаптації не потребує спеціальної loss функції, але для роботи з багатовимірними векторами є необхідність привести їх до одновимірного вигляду.

Таким чином, приведені до одного виду вектори результатів моделі та вектори міток можуть бути використані для обрахунку якості класифікації та функції loss під час навчання.

Висновки до розділу 3

Отже, було визначено середовище та архітектуру програмного рішення для методу класифікації програмних вимог з використанням великих мовних моделей. Реалізація представляє собою застосунок Jupyter Notebook, що надає гнучку можливість використання різних наборів даних, налаштування міток класів та використовуваних стовпців для наборів даних, використання мовної моделі для генерації векторних представлень текстів, застосування методів pooling/fusion для приведення векторних представлень до одного вектору, додаткову можливість комбінувати різні шари при проведенні кластеризації, використання ансамблевих підходів до кластеризації з гнучким налаштуванням архітектури створюваних ансамблів та забезпечує функціонал для різних видів навчання для проведення доменної адаптації мовної моделі.

Розділ 4 Проведення експериментів

4.1 Налаштування середовища та сценарії експериментів

Експерименти було проведено в локальному середовищі:

- об’єм ОЗП 16 Гб;
- відеокарта GTX 1650, 4 Гб відеопам’яті;
- операційна система Windows 11;
- інсталяція Python 3.13.0 в середовищі Anaconda;

На протязі експериментів використано [CLS] токен останнього шару моделі в якості векторного простору ознак.

Для алгоритмів кластеризації використано параметр *n_clusters=2*, що відповідає за розбиття всіх точок на 2 окремих класи. Для випадкового генератора чисел застосовано початковий параметр random_seed=42 для повторюваності експериментів. В якості вхідних даних використано результати останнього шару моделі. Функціонал *Scaler* не застосовувався. Загальні дані для окремих алгоритмів кластеризації наведено в таблиці 4.1.

Алгоритм HDBSCAN використовує параметр мінімального розміру групи 15, інші параметри не були змінені і залишаються за замовчуванням.

KMeans використовує лише загальні параметри *n_clusters* та *random_state* для кластеризації.

Аналогічно KMeans, GaussianMixture застосовує лише параметри *n_clusters* та *random_state*.

Параметр *affinity* алгоритму SpectralClustering було змінено з *rbf* на *nearest_neighbors*, параметри *n_clusters* та *random_state* приймаються з загального набору.

AgglomerativeClustering використовує значення *n_clusters*, інші параметри залишаються за замовчуванням.

Параметри *min_samples=2, min_cluster_size=300, xi=0.2 ma metric=’cosine’* застосовані для алгоритму OPTICS.

Birch застосовується з параметрами *n_clusters* та *threshold=0.01*.

Таблиця 4.1 – Змінені параметри алгоритмів кластеризації

Алгоритм	**Параметр**	**Значення**
HDBSCAN	min_cluster_size	15
KMeans	n_clusters	2
	random_state	42
GaussianMixture	n_clusters	2
	randrom_state	42
SpectralClustering	n_clusters	2
	randrom_state	42
	affinity	nearest_neighbors
AgglomerativeClustering	n_clusters	2
OPTICS	min_samples	2
	min_cluster_size	300
	xi	0.02
	metric	cosine
Birch	n_clusters	2
	threshold	0.01

Для формування першого ансамблю використовується 32 ітерації MiniBatchKMeans, кожна з однаковими параметрами *n_clusters=16*, *batch_size=64*, *n_init=1* та *max_iter=20*. В якості мінімальної ймовірності використовується значення 0.5.

Другий варіант consensus clustering вміщує в собі 32 ітерації MiniBatchKMeans, де *n_clusters* дорівнює 5, а інші параметри аналогічні до попереднього ансамблю та 16 ітерацій AgglomerativeClustering з таким же значенням *n_clusters*. Мінімальну ймовірність було встановлено як 0.6.

Третій варіант consensus ансамблю представляє собою об’єднання 16 ітерацій AgglomerativeClustering де *n_clusters* дорівнює 3, та 12 ітерацій

SpectralClustering, де n_clusters дорівнює 2, де аналогічно до поодинокого варіанту, застосовується режим *affinity nearest_neighbors*. Параметр мінімальної ймовірності для ансамблю дорівнює 0.6.

Також було окремо застосовано ансамблевий підхід з використанням окремого методу кластеризації для формування кінцевого результату. Перший варіант ансамблю з функцією агрегації представляє собою 128 унікальних ітерацій MiniBatchKMeans налаштованих на пошук трьох окремих груп точок *n_clusters=3*, з іншими параметрами аналогічними до попередніх ансамблів, що використовували MiniBatchKMeans та окремий алгоритм агрегації SpectralClustering, налаштований на розподіл даних на дві групи з додатковими параметрами affinity precomputed для використання попередньо створеної матриці подібності та *assign_labels discretize* для покращення результату.

Другий варіант ансамблю з агрегацією використовує 100 ітерацій SpectralClustering налаштованого на розбиття простору на 4 окремих групи з параметрами *affinity nearest_neighbors, assign_labels discretize* та *n_neighbors* що починаючи з значення 1 збільшується на 1 на протязі всіх 100 ітерацій. Для агрегації було використано аналогічний метод кластеризації до агрегації у першому ансамблі з даною функцією.

Третя архітектура ансамблю побудована на основі 15 ітерацій алгоритму Birch з параметрами *threshold=0.01* та значення *n_clusters* що починаючи з 1 поступово збільшується на протязі ітерацій для побудови матриці подібності та алгоритму AgglomerativeClustering налаштованого на розподіл точок на дві окремі групи з параметром *linkage complete*.

Загальні параметри ансамблевих алгоритмів висвітлені в таблиці 4.2:

Таблиця 4.2 – Загальні параметри ансамблевих алгоритмів

Ансамбль	**Параметр**	**Значення**
ConsensusClustering1	MIN_PROBABILITY	0.5
	base_estimators	32*MiniBatchKMeans
ConsensusClustering 2	MIN_PROBABILITY	0.6

	base_estimators	32*MiniBatchKMeans + 16*AgglomerativeClustering
ConsensusClustering 3	MIN_PROBABILITY	0.6
	base_estimators	16*AgglomerativeClustering + 12*SpectralClustering
AggregatedClustering 1	base_estimators	128*MiniBatchKMeans
	aggregator	SpectralClustering
AggregatedClustering 2	base_estimators	100*SpectralClustering
	aggregator	SpectralClustering
AggregatedClustering 3	base_estimators	15*Birch
	aggregator	AgglomerativeClustering

Наведена конфігурація експериментів є основою для подальшого аналізу метрик якості та оцінки ефективності запропонованого методу.

4.2 Результати задачі класифікації програмних вимог

4.2.1 Базова модель RoBERTa

На першому етапі дослідження використано базову модель RoBERTa без проведення доменної адаптації.

Алгоритм HDBSCAN розділив простір на 2 окремих групи та «шум», де значення silhouette score для кластерів дорівнює 0.1154. В якості шуму було розпізнано 415 точок, 233 з яких являються FR та 182 NFR. Група 0 включає в себе всього 47 точок, 16 з яких представляють FR. Група 1 вміщує всього 1538 записів, 751 з яких є FR. В результаті перевірки з урахуванням точок «шуму» як окремого класу отримано значення F1 метрики 0.4811, ARI 0.0023 та silhouette score 0.1122, при використанні «шуму» в якості помилкових значень отримане значення F1 дорівнює 0.3164, ARI 0.0023 і silhouette 0.1122, та без урахування «шуму» було досягнуто значень 0.3514 для F1 метрики, 0 для ARI, що відповідає

випадковому розподілу. Таке низьке значення метрик та пропорції розподілу класів між групами, свідчить що в даному випадку якість кластеризатора HDBSCAN наближається до випадкового. Візуально, алгоритм виділив найбільші хмари точок, але не розрізнив класи між ними (Рис. 4.1).

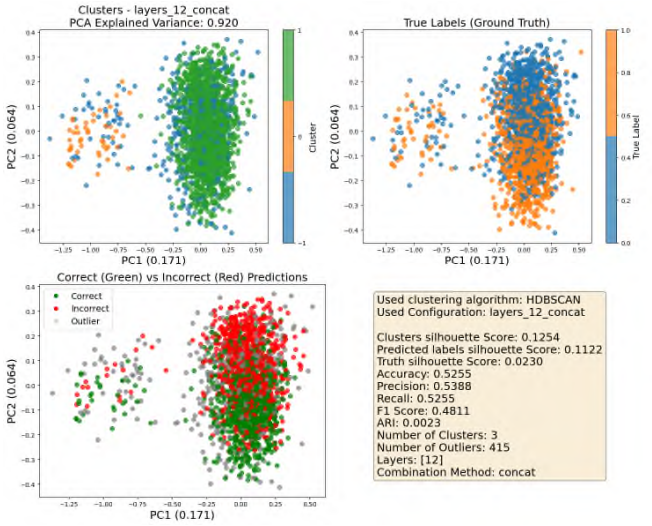


Рисунок 4.1 – Результати кластеризації з використанням базової RoBERTa алгоритмом HDBSCAN

KMeans розділив простір на дві окремих групи, розміром 1908 та 92 точки відповідно. Обидві розпізнані групи мають однакову кількість як FR та NFR, що наближається до значень випадкового класифікатора. Результат значення F1 0.3333, silhouette score 0.3684, та ARI 0, підкріплює цей висновок. Візуально алгоритм розділив простір на дві окремих хмари точок, де відсутній поділ між FR та NFR.

Алгоритм GaussianMixture показав ідентичний результат з KMeans, розподіливши точки в ідентичні групи, отримавши такий же візуальний результат та значення F1 метрики 0.3333, ARI 0 та silhouette score 0.3684, що цілком збігаються.

SpectralClustering розподілив точки на дві основних групи розмірами 1910 та 90 точок кожна, де група 0 вміщує в собі 956 FR і 954 NFR вимог та група 1 нараховує 44 FR та 46 NFR вимог. Загальний розподіл класів між групами

наближається до випадкового розподілу. Результат метрики F1 складає 0.3707, ARI -0.0001 та silhouette score 0.3712.

Алгоритм AgglomerativeClustering створив розподіл, аналогічний SpectralClustering, як по класовим розподілам між двома групами розмірами 1910 та 90, так і внутрішніми розподілами класів всередині цих груп. Значення F1 метрики дорівнює 0.3707, ARI -0.0001 та silhouette score 0.3712.

Алгоритм OPTICS виділив 2 основні групи, а саме один великий кластер та шум. Силуетний коефіцієнт для отриманої кластеризації склав 0.2114. Кластер викидів містить 21 текст, з яких 16 належать до категорії FR та 5 до NFR. Основний кластер 0 включає решту 1979 текстів, розподіл між FR та NFR у ньому практично рівномірний, 984 FR та 995 NFR, що відображає низьку чистоту кластера 0.503. Загальна ефективність класифікації є низькою: при розгляді викидів як окремого класу, значення F1-міри становить лише 0.3497, що близько до результату випадкового вгадування. Це підтверджується і значенням Adjusted Rand Index (ARI), що наближається до нуля 0.0001.

Алгоритм Birch розподілив простір на 2 основні групи, аналогічного розміру та композиції до AgglomerativeClustering та SpectralClustering, зі схожими результатами F1 метрики 0.3707, ARI -0.0001 та silhouette score 0.3172. Візуально проведено аналогічний розподіл між двома основними хмарами точок. Така подібність результатів може бути спричинена невеликою кількістю кластерів, на які розподіляється векторний простір за обраними параметрами.

Ансамблевий підхід на основі MiniBatchKMeans та матриці подібності виділив 3 групи у векторному просторі, для яких silhouette score 0.0884. Група 0 вміщує в собі 91 вимогу, 45 з яких є FR і 46 NFR. Група 1 нараховує 1908 вимог, 955 з яких є FR і 953 NFR. Група 2 представляє собою тільки 1 FR текст. Було досягнуто метрик F1 score 0.3714, ARI -0.0001, silhouette score 0.3679 Візуально простір розподілений на дві основних хмари.

Ансамбль на основі MiniBatchKMeans та AgglomerativeClustering розподілив точки на 2 основні групи. Група 0 нараховує 91 текст, 45 з яких є FR та 46 NFR. Група 1 вміщує в собі 1909 текстів, 955 з яких є FR та 954 NFR. Векторний простір візуально розділений на 2 окремих хмари точок. Результат F1

метрики дорівнює 0.3705, ARI -0.0001, silhouette score 0.3703, що також свідчить про погану якість класифікації.

Ансамбль на основі AgglomerativeClustering та SpectralClustering розділив тексти на 3 основних групи, де silhouette score для всіх груп досягає 0.0223. Група 0 нараховує 90 вимог, 44 з яких є FR та 46 NFR. Група 1 вміщує 732 вимог, 512 з яких є FR та 220 NFR. Група 2 вміщує 1178 вимог, 444 з яких є FR та 734 NFR. Результат F1 метрики дорівнює 0.6395, ARI 0.0848, silhouette score 0.0223. Візуальний розподіл відділяє дві основних хмари точок, та розділяє більшу з них на дві частини (Рис. 4.2).

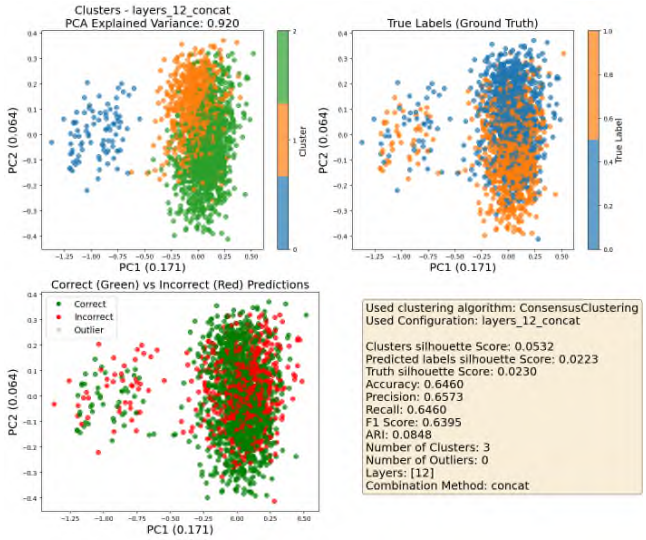


Рисунок 4.2 – Результати кластеризації з використанням базової RoBERTa алгоритмом ансамблю на основі AgglomerativeClustering та SpectralClustering

Ансамблевий метод на основі MiniBatchKMeans для формування матриці подібності та SpectralClustering для кінцевого розподілу розпізнав 2 групи текстів. Група 0 нараховує 1056 програмних вимог, 370 з яких є FR та 686 NFR. Група 1 включає 944 вимоги, де 630 є FR та 314 NFR. Результат F1 метрики дорівнює 0.6577, ARI 0.0994, silhouette score 0.0599, що є кращим за результати окремих алгоритмів кластеризації та ансамблів без використання функції агрегації (Рис. 4.3).

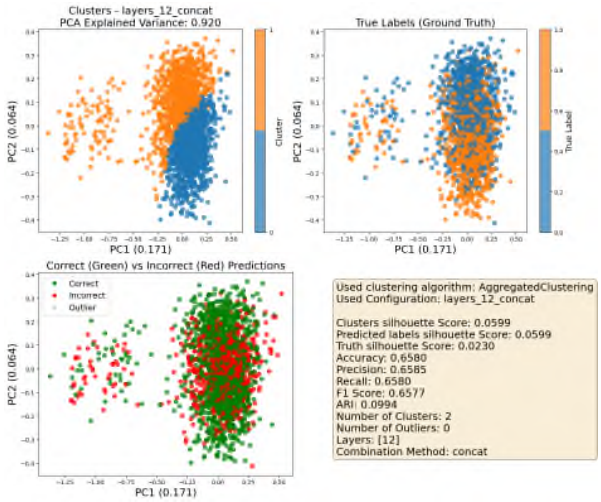


Рисунок 4.3 – Результати кластеризації з використанням базової RoBERTa алгоритмом ансамблю на основі MiniBatchKMeans та SpectralClustering

Ансамблевий метод, використовуючи SpectralClustering в якості початкової матриці подібності та окремої агрегації даних розділив векторний простір на 2 основних групи розміром 1111 та 889 точок. Група 0 включає 341 FR та 770 NFR вимоги. Група 1 включає 659 FR та 230 NFR вимог. Результат F1 метрики досягає 0.7136, ARI 0.1836, silhouette score 0.0599, де значення F1 та ARI є найвищими на даному етапі, а silhouette score є одним з найнижчих, що говорить про значне змішування класів. Візуально розподіл є наближеним до розмитої межі між класами FR та NFR (Рис. 4.4).

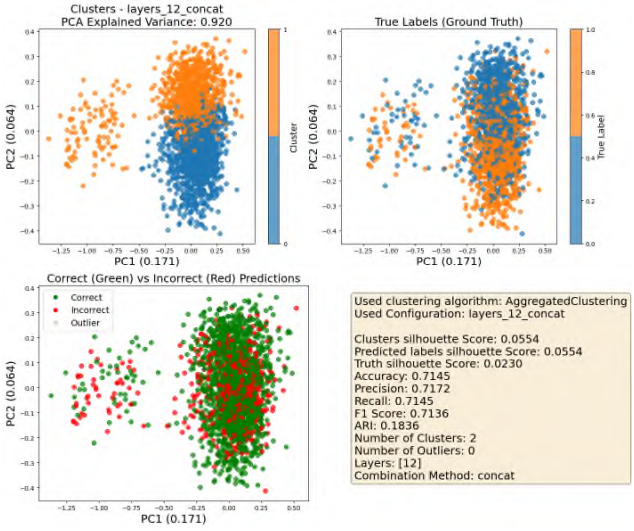


Рисунок 4.4 – Результати кластеризації з використанням базової RoBERTa ансамблем на основі SpectralClustering

Ансамблевий метод, що використовує Birch для формування матриці подібності та AgglomerativeClustering для агрегації кластерів розділив вибірку на групи розмірами 1182 та 818 вимог. Група 0 вміщує в собі 791 функціональних та 391 нефункціональних вимог. Група 1 включає 209 функціональних та 609 нефункціональних вимог. Результат F1 сягає 0.6975, ARI 0.1596, silhouette score 0.0262, що є кращим в порівнянні з поодинокими кластеризаторами та ансамблями без агрегації даних.

Загальні результати кластеризації для базової моделі RoBERTa наведено в таблиці 4.3:

Таблиця 4.3 – Досягнуті метрики кластеризації для кожного алгоритму на базовій моделі RoBERTa

Алгоритм	F1-Score	ARI	Silhouette score (clusters)	Silhouette score (predictions)
HDBSCAN (Outlier mode: classify)	0.4811	0.0023	0.1254	0.1122
HDBSCAN (Outlier mode: wrong)	0.3164	0.0023		0.1122
HDBSCAN (Outlier mode: trim)	0.3514	0.0000		-
KMeans	0.3333	0.0000	0.3684	-
GaussianMixture	0.3333	0.0000	0.3684	-
SpectralClustering	0.3707	-0.0001	0.3712	0.3712
AgglomerativeClustering	0.3707	-0.0001	0.3712	0.3712
OPTICS (Outlier mode: classify)	0.3497	0.0001	0.2114	0.2114
OPTICS (Outlier mode: wrong)	0.3340	0.0001		0.2114
OPTICS (Outlier mode: trim)	0.3364	0.0000		-
Birch	0.3707	-0.0001	0.3712	0.3712
ConsensusClustering 1	0.3714	-0.0001	0.0884	0.3679
ConsensusClustering 2	0.3705	-0.0001	0.3703	0.3703

ConsensusClustering 3	0.6395	0.0848	0.0532	0.0223
AggregatedClustering 1	0.6577	0.0994	0.0599	0.0599
AggregatedClustering 2	0.7136	0.1836	0.0554	0.0554
AggregatedClustering 3	0.6975	0.1596	0.0262	0.0262

Отже, було описано результати кластеризації з використанням базової моделі RoBERTa. Ансамблеві алгоритми з використанням функції агрегації показали найкращий результат досягнувши значення F1 метрики 0.7136 на тестовій вибірці без проведення доменної адаптації моделі.

4.2.2 Доменна адаптація на задачі класифікації

Після проведення доменної адаптації, навчений класифікатор досягнув значень F1 метрики 0.8134 (Рис. 4.5). Після адаптації очікується покращення якості векторних представлень і, підвищення ефективності алгоритмів некерованої класифікації програмних вимог. Навчання моделі проводилося на протязі 20 епох з параметрами lr 2e-5, 5 епох розігріву та weight_decay 0.001. Навчений класифікатор досягнув значень F1 0.8134 (Рис. 4.5).

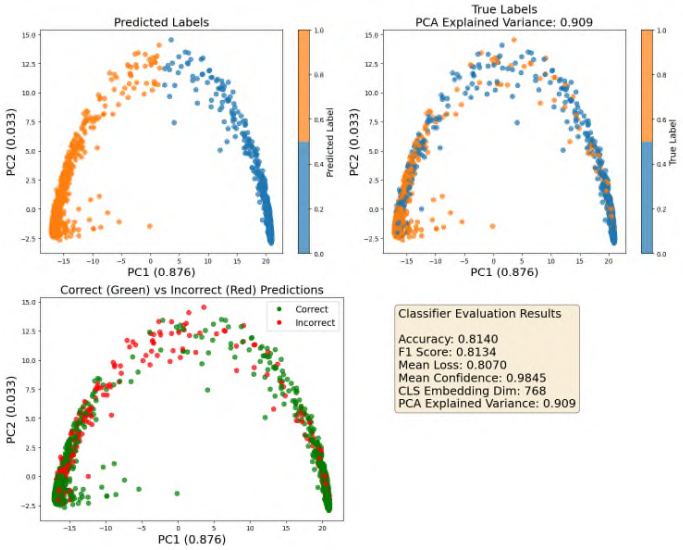


Рисунок 4.5 – Результати класифікації, використовуючи навчений класифікатор після проведення доменної адаптації.

Алгоритм HDBSCAN після доменної адаптації моделі виділив 3 групи, включно з кластерами 0 і 1 та групою «шуму», отримавши значення silhouette score 0.6592 для розпізнаних кластерів. Група 0 містить 864 текстів, з яких 744 належать до функціональних вимог. Чистота кластера становить 0.861, що на порядок вище за відповідний результат у базової моделі. Група 1 включає 1082 тексти, серед яких 861 є нефункціональними. Група «шуму» містить лише 54 тексти, 35 з яких є функціональними і 19 нефункціональними, що є значним зменшенням порівняно з попереднім експериментом. У режимі розгляду шуму як окремого класу було досягнуто значення F1 метрики 0.8197, ARI 0.4093, silhouette score 0.7386, що є значним покращенням в порівнянні з результатами, отриманими до адаптації. Аналогічно, у режимі ігнорування шумових точок F1-метрика досягла 0.8242, ARI 0.8242, silhouette score 0.7758. Візуально, векторний простір має вигляд дуги, що розділена на 3 сегменти з точками «шуму» посередині.

Алгоритм KMeans після адаптації RoBERTa також продемонстрував відчутне покращення. Простір був поділений на дві групи розмірами 1118 та 882 точок відповідно. Група 0 має значення чистоти 0.781, де більшість це нефункціональні вимоги та група 1 має значення чистоти 0.856, де більшість це функціональні вимоги. Значення F1-метрики становить 0.8134, ARI 0.3941, silhouette score 0.7472. На відміну від базового експерименту, де обидві групи були майже випадковими за складом, адаптована модель забезпечила високі значення метрик якості.

GaussianMixture показав результати, наближені KMeans, що свідчить про стабільність структури кластерів у векторному просторі після адаптації.

Аналогічно до GaussianMixture, результати SpectralClustering є наближеними до KMeans, з дещо нижчим значенням silhouette score 0.7471.

AgglomerativeClustering показав результати, подібні до KMeans та GaussianMixture. В групі 0 домінують функціональні вимоги, де чистота складає 0.850 та в групі 1 домінують нефункціональні вимоги, де чистота складає 0.800. Значення F1-метрики дорівнює 0.8227, ARI 0.4170, silhouette score 0.7466, що також узгоджується з підвищенням якості після доменної адаптації. Візуально

сектор не функціональних вимог є дещо менший, що разом з невеликим ростом метрик точності в порівнянні з рівним візуальним розподілом є ознакою підвищеної щільності точок, що репрезентують нефункціональні вимоги (Рис. 4.6).

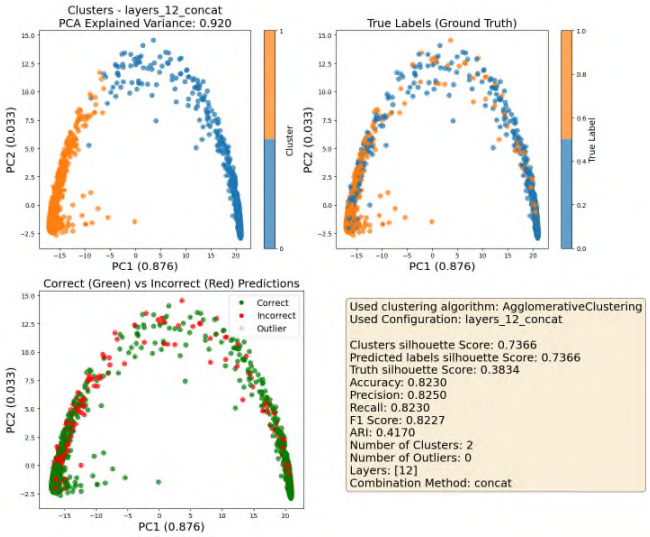


Рисунок 4.6 – Результати кластеризації після доменної адаптації моделі алгоритмом AgglomerativeClustering

Алгоритм OPTICS розпізнав 2 основні кластери та невелику групу шумових точок. Група 0 з чистотою 0.890 та кластер 1 з чистотою 0.852. В якості шуму було визначено 443 точки. F1-метрика з шумом як окремим класом дорівнює 0.8181, ARI 0.4055, silhouette score 0.5864, а без урахування шуму 0.8670, ARI 0.5404, silhouette score 0.8437. На графіку класи розділені на дві окремих групи з шумом посередині, що разом з відносно високою точністю підтверджує високу щільність відповідних на протилежних кінцях фігури (Рис. 4.7).

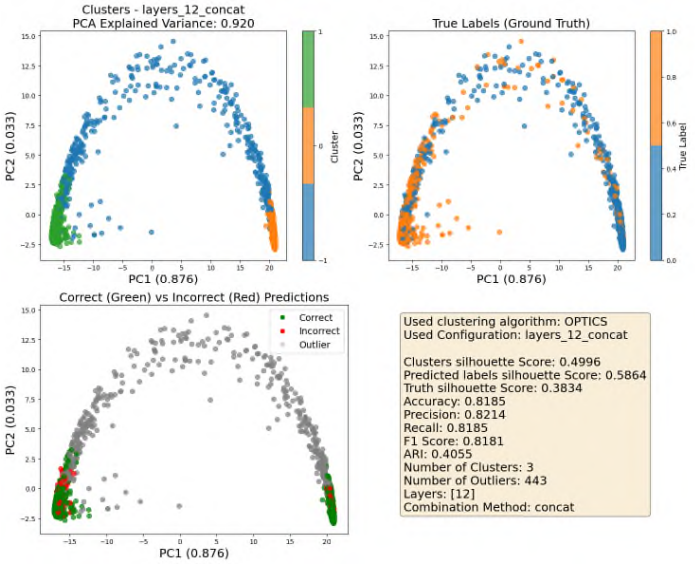


Рисунок 4.7 – Результати кластеризації після доменної адаптації моделі алгоритмом OPTICS

Алгоритм Birch також розділив дані на дві основні групи. Група 0 має чистоту 0.778 та група 1 - 0.860, де F1 метрика дорівнює 0.8132, ARI 0.4170, silhouette score 0.7366. Такий розподіл наближається до візуального розподілу фігури на дві рівно розмірних частини.

Для ансамблів на основі матриць подібності MiniBatchKMeans та AgglomerativeClustering, в обох випадках алгоритм виділив лише один кластер, що включає всі 2000 точок, що свідчить про неможливість розділення простору за обраними параметрами. Значення F1-метрики склало 0.3333 та ARI 0, що відповідає випадковому класифікатору.

Ансамбль AgglomerativeClustering та SpectralClustering успішно розділив дані на 4 кластери різного розміру, де silhouette score для кластерів становить 0.6121. Найбільший кластер 0 об’єднує 820 зразків із високою чистотою за функціональними вимогами 0.867. Кластер 1 містить 1078 зразків із чистотою 0.800 за класом не функціональних вимог. Менші кластери 2 та 3, 62 та 40 зразків відповідно також демонструють високу чистоту 0.710 та 0.725 відповідно за класом функціональних вимог. F1-метрика досягла 0.8227, ARI 0.4170 та silhouette score 0.7366, а отже якість класифікації значно покращилася. Фігура на графіку була розбита на 4 сектори приблизно однакового розміру, що разом з розподілом кластерів свідчить про вищу щільність не функціональних вимог (Рис. 4.8).

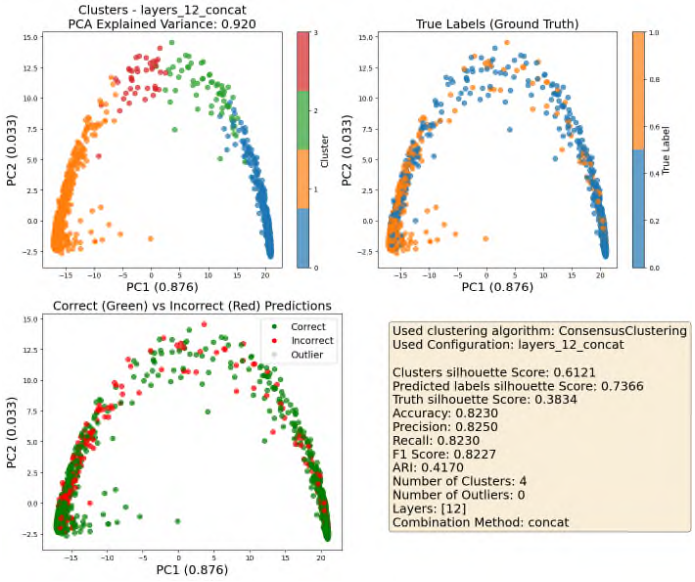


Рисунок 4.8 – Результати кластеризації ансамблю на основі матриць подібності AgglomerativeClustering та SpectralClustering

Ансамбль з MiniBatchKMeans та агрегацією SpectralClustering сформував два кластери. Кластер 0 містить 1118 зразків із чистотою 0.781, більшість яких становлять не функціональні вимоги. Кластер 1 містить 882 зразки із чистотою 0.856, де більшість становлять функціональні вимоги. F1-метрика склала 0.8134, ARI 0.3941, silhouette score 0.7472. Графік візуально розбитий на дві половини.

Ансамбль з використанням SpectralClustering на обидвох етапах класифікації сформував два кластери. Кластер 0 містить 1012 зразків із чистотою 0.824 за класом не функціональних вимог. Кластер 1 містить 988 зразків із чистотою 0.832 за класом функціональних вимог. F1-метрика досягла 0.8280, ARI 0.4301 та silhouette score 0.6826. У векторному просторі виділено візуально малу групу функціональних та значно більшу групу не функціональних вимог, підтверджуючи високу щільність не функціональних вимог (Рис. 4.9).

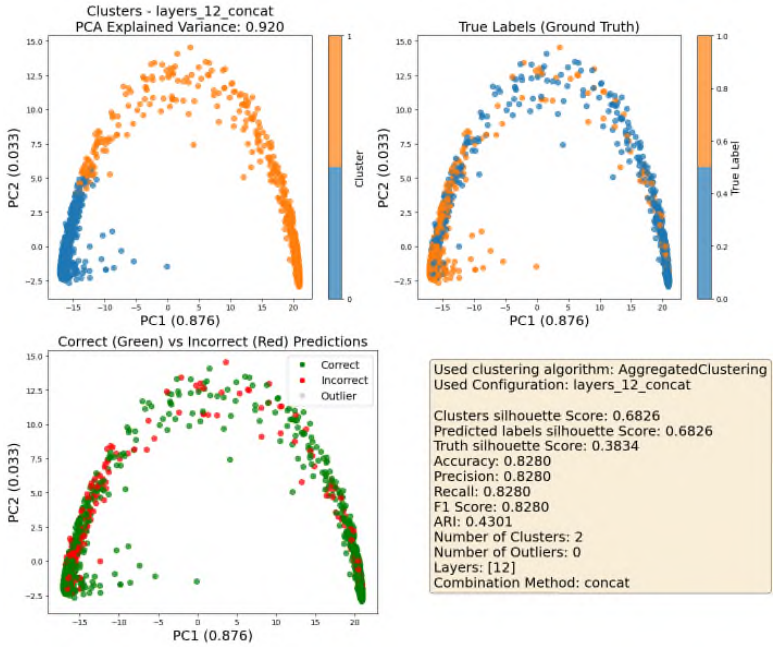


Рисунок 4.9 – Результати кластеризації ансамблю використанням SpectarlClustering на обидвох етапах класифікації

Ансамбль з використанням Birch для обробки та AgglomerativeClustering для агрегації виділив 2 основних кластери. Кластер 0 містить 1421 зразок із чистотою 0.665 за класом не функціональних вимог. Кластер 1 містить 579 зразків із чистотою 0.905 за класом функціональних вимог. Візуально було виділено малий кластер функціональних та більший не функціональних вимог. F1-метрика знизилася до 0.7222, ARI 0.2196, silhouette score 0.4867, що свідчить про високу щільність точок в протилежних кінцях фігури (Рис. 4.10).

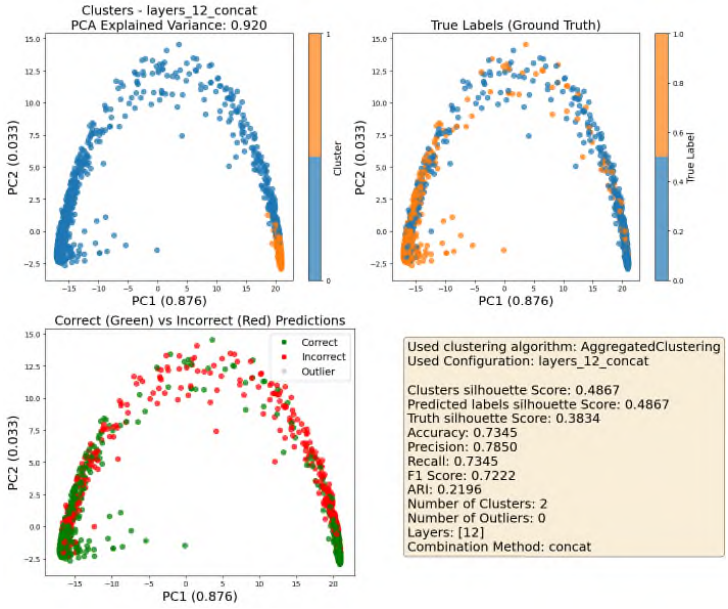


Рисунок 4.10 – Результати кластеризації ансамблю на основі використання Birch для обробки та AgglomerativeClustering для агрегації

Загальні результати кластеризації для доменно адаптованої моделі наведено в таблиці 4.4:

Таблиця 4.4 – Досягнуті метрики кластеризації для кожного алгоритму на моделі RoBERTa після проведення доменної адаптації

Алгоритм	F1-Score	ARI	Silhouette score (clusters)	Silhouette score (predictions)
HDBSCAN (Outlier mode: classify)	0.8197	0.4093	0.6592	0.7386
HDBSCAN (Outlier mode: wrong)	0.8127	0.3993		0.6592
HDBSCAN (Outlier mode: trim)	0.8242	0.4216		0.7758
KMeans	0.8134	0.3941	0.7472	0.7472
GaussianMixture	0.8134	0.3941	0.7472	0.7472
SpectralClustering	0.8134	0.3941	0.7471	0.7471
AgglomerativeClustering	0.8227	0.4170	0.7366	0.7366
OPTICS (Outlier mode: classify)	0.8181	0.4055	0.4996	0.5864
OPTICS (Outlier mode: wrong)	0.7558	0.3398		0.4996

OPTICS (Outlier mode: trim)	0.8670	0.5404		0.8437
Birch	0.8227	0.4170	0.7366	0.7366
ConsensusClustering 1	0.3333	0.0000	-	-
ConsensusClustering 2	0.3333	0.0000	-	-
ConsensusClustering 3	0.8227	0.4170	0.6121	0.7366
AggregatedClustering 1	0.8134	0.3941	0.7472	0.7472
AggregatedClustering 2	0.8280	0.4301	0.6826	0.6826
AggregatedClustering 3	0.7222	0.2196	0.4867	0.4867

Отже, проведення доменної адаптації дозволило значно підвищити якість класифікації програмних вимог. Після доменної адаптації векторний простір був розділений на дві основних групи з підвищеною щільністю, де неоднозначні вимоги були витіснені в проміжок між ними. Більший візуальний розмір щільного розподілу NFR та змішування класів може свідчити про більшу неоднозначність NFR, що спричиняє значне змішування FR та NFR.

4.2.3 Доменна адаптація з контрастивним навчанням

Після проведення доменної адаптації з використанням контрастивного, навчений класифікатор досягнув значень F1 метрики 0.8083. Навчання моделі проводилося на протязі 20 епох з параметрами lr 2e-5, 5 епох розігріву та weight_decay 0.001 та ваги 0.95 для контрастивного і 0.05 для класифікаційного навчання. Під час контрактивного навчання було використано всі класи PROMISE_exp. Навчений класифікатор досягнув значення 0.8083 для F1 метрики (Рис. 4.11).

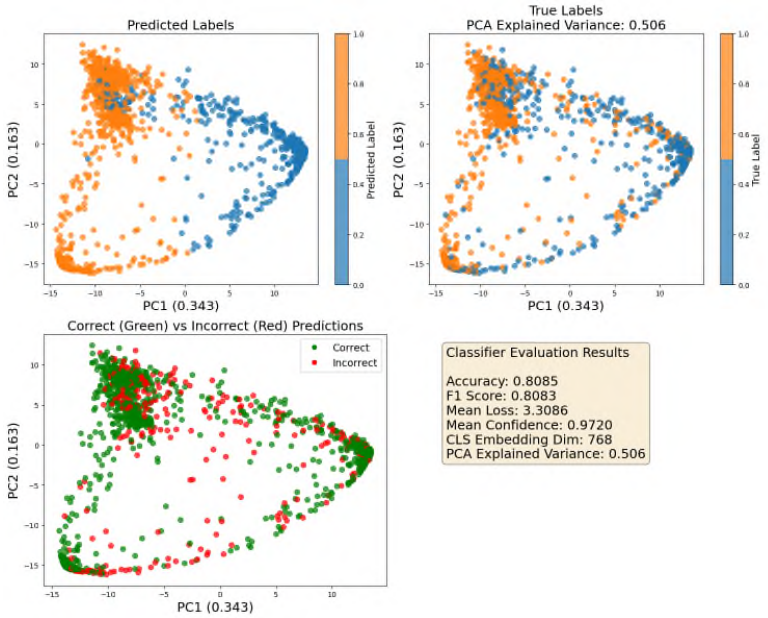


Рисунок 4.11 – Результати класифікації, використовуючи навчений класифікатор після проведення доменної адаптації.

Алгоритм HDBSCAN. Алгоритм виділив 11 основних кластерів і один 415 зразків шуму, де silhouette score для кластерів становить 0.5201. Чистота більшості кластерів висока: кластери 0-6 та 10 мають чистоту від 0.760 до 1.000 за класом не функціональних вимог, тоді як найбільший кластер 9, що налічує 725 характеризується чистотою 0.888 за класом функціональних вимог. шум» має збалансований розподіл з чистотою, що прямує до 0.5. Загальна якість класифікації при трактуванні «шуму» як окремого класу досягає метрик F1 0.7773, ARI 0.3167 та silhouette score 0.3274. Якщо «шум» розглядати як помилкові значення, F1=0.7459, ARI 0.2971 та silhouette score 0.2882, а при його виключенні якість істотно покращується F1=0.8433, ARI 0.4717 та silhouette score 0.4517, що вказує на високу якість виділених груп. Візуально алгоритм успішно розділив простір на малі групи, але візуально, більша частина частина векторного простору заповнена «шумом» (Рис. 4.12).

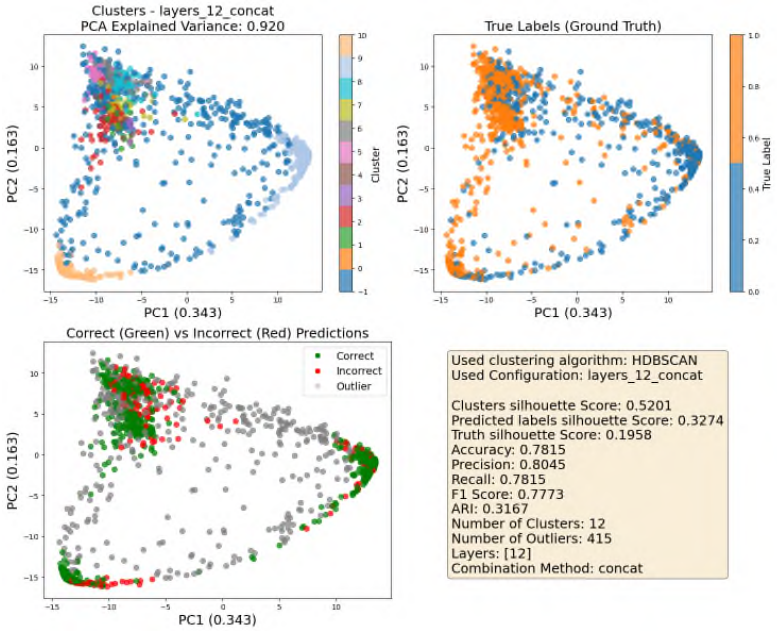


Рисунок 4.12 – Результати кластеризації HDBSCAN

Алгоритми KMeans та GaussianMixture розділили простір на два кластери подібного розміру та структури. Кластер 0 має чистоту 0.858 за класом функціональних та кластер 1 має чистоту 0.768 за класом не функціональних вимог відповідно. Отримані метрики практично ідентичні: F1=0.8055, ARI 0.3525, silhouette score 0.3644, що свідчить про стабільність розподілу. Візуально простір було розділено на дві частини, що мають однаковий вигляд для обидвох алгоритмів.

Алгоритм SpectralClustering. Результати аналогічні KMeans: два кластери з чистотами 0.860 за функціональними та 0.753 за не функціональними вимогами. Метрики F1=0.7954, ARI 0.3525 та 0.3604 є дещо нижчими, в порівнянні з KMeans. Візуально розмір функціональної групи є менший за нефункціональну групу.

Алгоритм AgglomerativeClustering утворив два кластери зі схожими розмірами та чистотою. Кластер 0 переважає класом не функціональними вимогами з чистотою ~0.774 та кластер 1 переважає класом функціональних вимог, з чистотою ~0.850. Значення F1=0.8068 ARI 0.3799 та silhouette score 0.3574, говорять про відносно прийнятну якість кластеризації.

Алгоритм OPTICS. Алгоритм виділив лише один компактний кластер з високою чистотою 0.925 за класом функціональних вимог, а решту 1455 зразків відніс до «шуму». Кластер «шум» переважно складається функціональних вимог з чистотою 0.659. Загальна якість класифікації при врахуванні «шуму» низька, F1=0.7168. Візуально більшість точок залишилися в якості шуму. Такий результат свідчить про високу щільність виділеного кластеру функціональних вимог (Рис. 4.13).

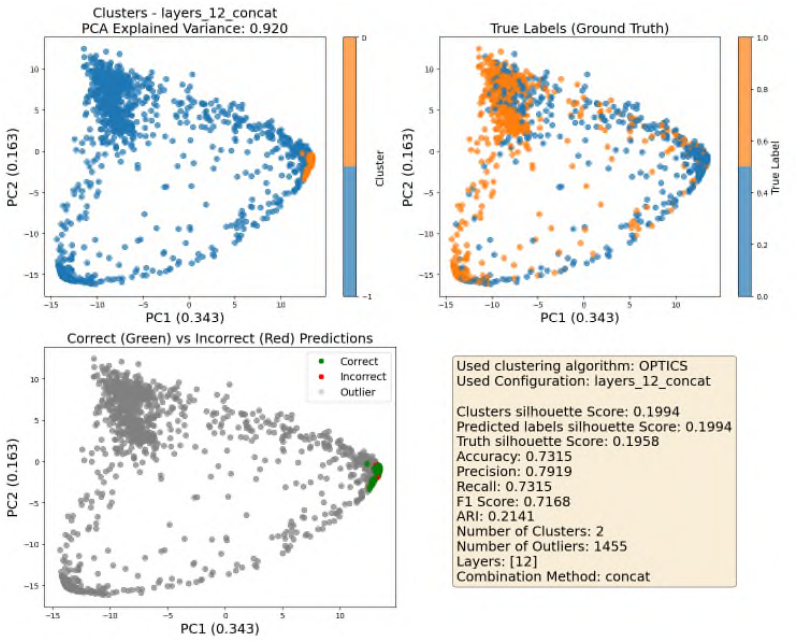


Рисунок 4.13 – Результати кластеризації OPTICS

Алгоритм Birch утворив два кластери. Кластер 0 переважає класом не функціональними вимогами з чистотою ~0.788 та кластер 1 переважає класом функціональних вимог, з чистотою ~0.849.Були досягнуті такі значення метрик, як F1=0.8151, ARI 0.3979 та silhouette score 0.3590, що свідчить про відносно велику якість класифікації при розділенні фігури навпіл.

Алгоритм ансамблю на основі матриць подібності MiniBatchKMeans розділив дані на 4 кластери, де silhouette score дорівнює 0.1468. Основна маса даних, а саме 1922 зразки, зосереджена у кластері 0 з майже рівномірним розподілом класів за чистотою 0.516. Невеликі кластери 1, що містить 41 зразок та 2, що містить 36 зразків демонструють високу чистоту за класом

нефункцональних вимог, 0.902 та 0.917 відповідно. Кластер 3 містить лише 1 зразок. Загальна якість класифікації є низькою: F1-метрика становить лише 0.4047, ARI 0.0039 та silhouette score 0.1783. Такий результат може свідчити про погану ефективність деяких ансамблевих архітектур при неоднозначному розподілі даних (Рис. 4.14).

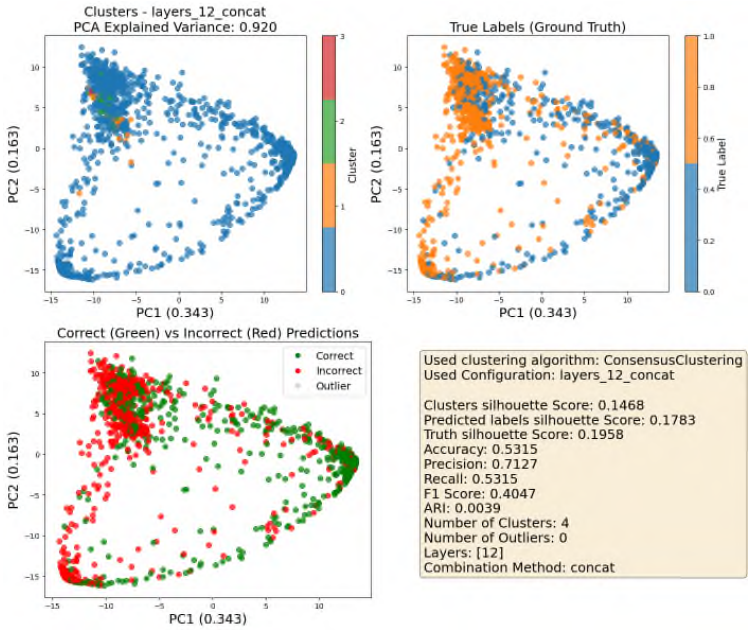


Рисунок 4.14 – Результати кластеризації ансамблів на основі матриць подібності MiniBatchKMeans

Алгоритм ансамблю на основі матриць подібності MiniBatchKMeans та AgglomerativeClustering не зміг розділити дані на декілька груп, утворивши єдиний кластер, що включає всі 2000 зразків. Значення F1-метрики 0.3333, ARI 0 відповідають випадковому класифікатору.

Алгоритм ансамблю на основі матриць подібності AgglomerativeClustering та SpectralClustering успішно виділив 5 кластерів різного розміру, з значенням silhouette score 0.3302. Найбільші кластери 0 та 1, містять 820 та 848 зразків відповідно, демонструють чітку ознаку класу з чистотою 0.862 за класом функціональних та 0.776 за класом не функціональних вимог відповідно. Кластер 2 вміщує 269 зразків та також має перевагу класу не функціональних вимог з чистотою 0.773. Менші кластери 3 та 4 є менш однорідними, 59 та 4 зразки відповідно, є змішаними. Загальна якість значно

покращилася: F1-метрика досягла 0.8068, ARI 0.3718 та silhouette score 0.3581, що свідчить про наближення результатів до AgglomerativeClustering. Візуально простір розділений на 5 кластерів схожих розмірів (Рис. 4.15).

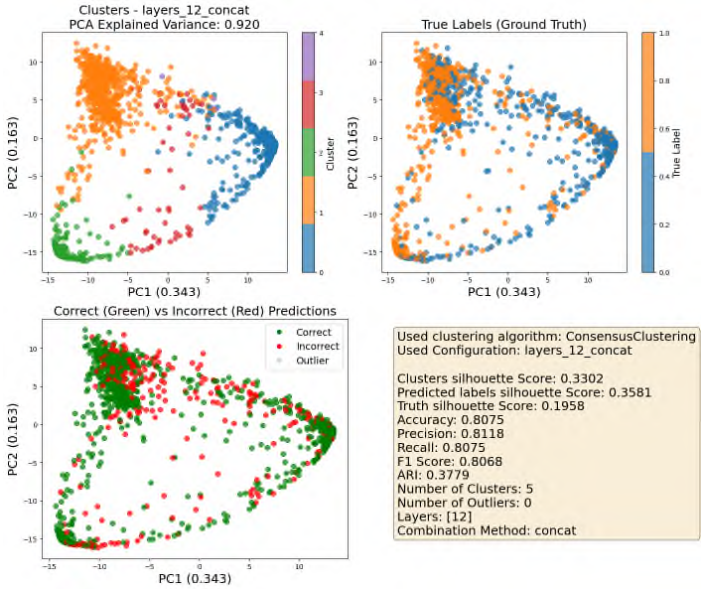


Рисунок 4.15 – Результати кластеризації ансамблів на основі матриць подібності AgglomerativeClustering та SpectralClustering

Ансамбль з MiniBatchKMeans та агрегацією SpectarlClustering сформував два кластери. Кластер 0 містить 844 зразки із перевагою класу функціональних вимог і чистотою 0.861 та кластер 1, що містить 1156 зразків із перевагою класу не функціональних вимог і чистотою 0.764. Результати F1 метрики дорівнюють 0.8038, ARI 0.3718 та silhouette score 0.3638, що свідчить про відносно високу точність. Візуально розподіл кластерів схожий на AgglomerativeClustering.

Ансамбль з використанням SpectarlClustering на обидвох етапах класифікації розділив два кластери. Кластер 0 містить 1181 зразок із чистотою 0.753 за класом не функціональних вимог, та кластер 1, що містить 819 зразків із чистотою 0.864 за класом функціональних вимог. Загальна якість залишається високою: F1 = 0.7968, ARI 0.3561 та silhouette score 0.3602. За візуальним розподілом кластер не функціональних вимог є дещо більшим за кластер функціональних вимог, аналогічно до AgglomerativeClustering.

Ансамбль з використанням Birch для обробки та AgglomerativeClustering для агрегації створив два кластери. Кластер 0 вміщує 1095 зразків із перевагою класу не функціональних вимог і чистотою 0.788 та кластер 1 нараховує 905 зразків із перевагою класу функціональних вимог і чистотою 0.849. Ця архітектура показала найкращий результат серед агрегованих підходів для даного набору, F1 метрика досягла 0.8151, ARI 0.3979 та silhouette score 0.3590. Візуально векторний простір було розділено на дві схожих за розміром половини під кутом (Рис. 4.16).

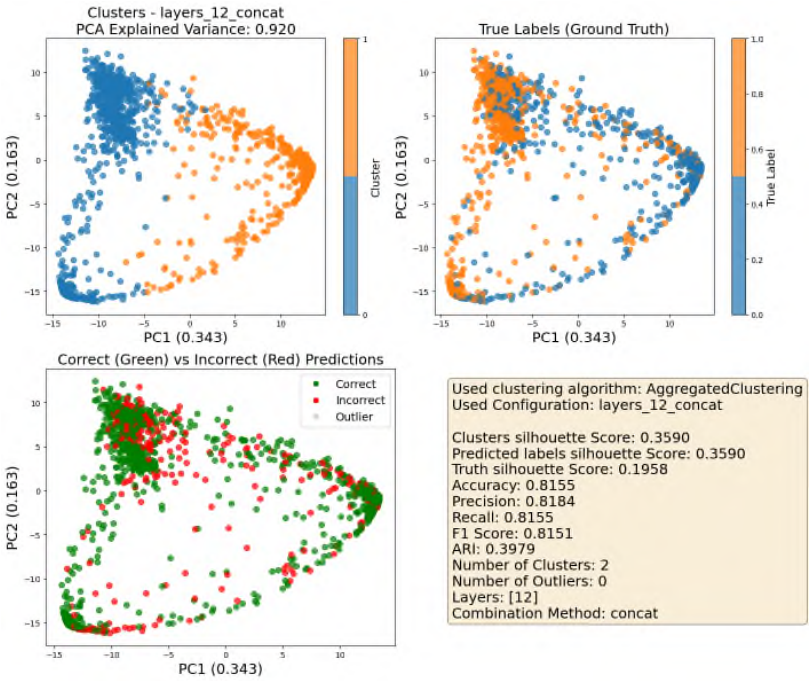


Рисунок 4.16 – Результати кластеризації ансамблю з використанням Birch для обробки та AgglomerativeClustering для агрегації

Загальні результати кластеризації для доменно адаптованої моделі з використанням констрастивного навчання наведено в таблиці 4.5:

Таблиця 4.5 – Досягнуті метрики кластеризації для кожного алгоритму на моделі RoBERTa після проведення доменної адаптації з контрастивним навчанням

Алгоритм	F1-Score	ARI	Silhouette score	Silhouette score

			(clusters)	(predictions)
HDBSCAN (Outlier mode: classify)	0.7773	0.3167	0.5201	0.3274
HDBSCAN (Outlier mode: wrong)	0.7459	0.2971		0.2882
HDBSCAN (Outlier mode: trim)	0.8433	0.4717		0.4517
KMeans	0.8055	0.3755	0.3644	0.3644
GaussianMixture	0.8055	0.3755	0.3644	0.3644
SpectralClustering	0.7954	0.3525	0.3604	0.3604
AgglomerativeClustering	0.8068	0.3779	0.3574	0.3574
OPTICS (Outlier mode: classify)	0.7168	0.2141	0.1994	0.1994
OPTICS (Outlier mode: wrong)	0.3262	0.3398		0.1994
OPTICS (Outlier mode: trim)	0.8886	0.0000		-
Birch	0.8151	0.3979	0.3590	0.3590
ConsensusClustering 1	0.4047	0.0039	0.1468	0.1783
ConsensusClustering 2	0.3333	0.0000	-	-
ConsensusClustering 3	0.8068	0.3779	0.3302	0.3581
AggregatedClustering 1	0.8038	0.3718	0.3638	0.3638
AggregatedClustering 2	0.7968	0.3561	0.3602	0.3602
AggregatedClustering 3	0.8151	0.3979	0.3590	0.3590

Отже, навчання на розділених класах не функціональних вимог та використання декількох різних задач під час проведення доменної адаптації може зменшити якість класифікації при роботі з бінарним розподілом класів функціональних та не функціональних вимог.

4.2.4 Доменна адаптація з використанням POS тегування

Після проведення доменної адаптації з використанням POS тегування класифікатор досягнув значень F1 метрики 0.8083. Навчання моделі проводилося на протязі 20 епох з параметрами lr 2e-5, 5 епох розігріву та weight_decay 0.001. Для POS тегування даних було використано режим tagger моделі

en_core_web_sm, створені вектори були об’єднані з векторами токенізатора та подані на вхід моделі. Навчений класифікатор досягнув значень F1 0.8151 (Рис. 4.17).

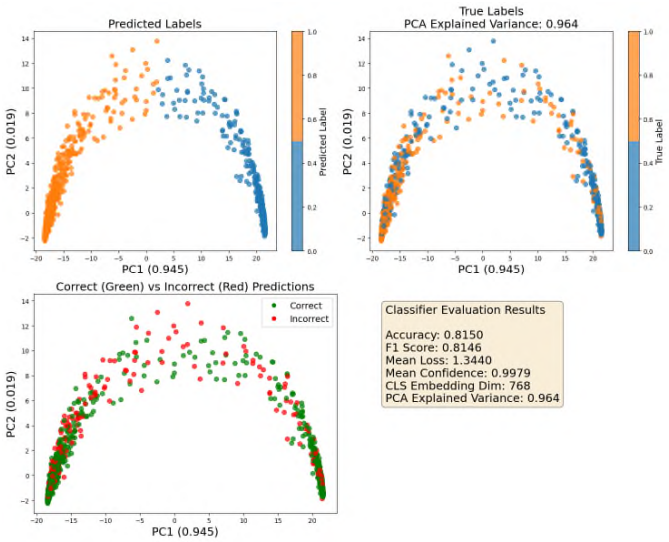


Рисунок 4.17 – Результати класифікації, використовуючи навчений класифікатор після проведення доменної адаптації з використанням POS тегування

Алгоритм HDBSCAN сформував три кластери та виділив 70 точок як шум, де silhouette score для окремих кластерів дорівнює 0.7633. Чистота основних кластерів є високою. Для кластера 0 обсягом 1054 записи вона становить 0.800. Для кластера 1 обсягом 876 записів чистота дорівнює 0.855. Найкращі результати класифікації отримано в режимі ігнорування шуму , де F1 метрика досягла 0.8245, ARI 0.4219 та silhouette score 0.8687 при врахуванні шуму як окремого кластеру F1 метрика дорівнює 0.8159, ARI 0.3991 та silhouette score 0.8189. Візуально фігура розділена на 3 сектори схожого розміру, з кластером шуму розташованим посередині між основними розпізнаними класами.

Алгоритм KMeans розділив дані на два чіткі кластери. Їхні розміри становлять 1092 та 908 записів. Чистота кластерів є високою і дорівнює 0.789 та 0.848. Значення F1 метрики класифікації дорівнює 0.8156, ARI 0.3991 та silhouette score 0.8363.

Результати алгоритму GaussianMixture практично ідентичні результатам KMeans. Розміри кластерів складають 1092 та 908 записів. Їхня чистота становить 0.789 та 0.848. Метрики також збігаються. F1 метрика дорівнює 0.8156, ARI 0.3941 та silhouette score 0.8363 (Рис. 4.18).

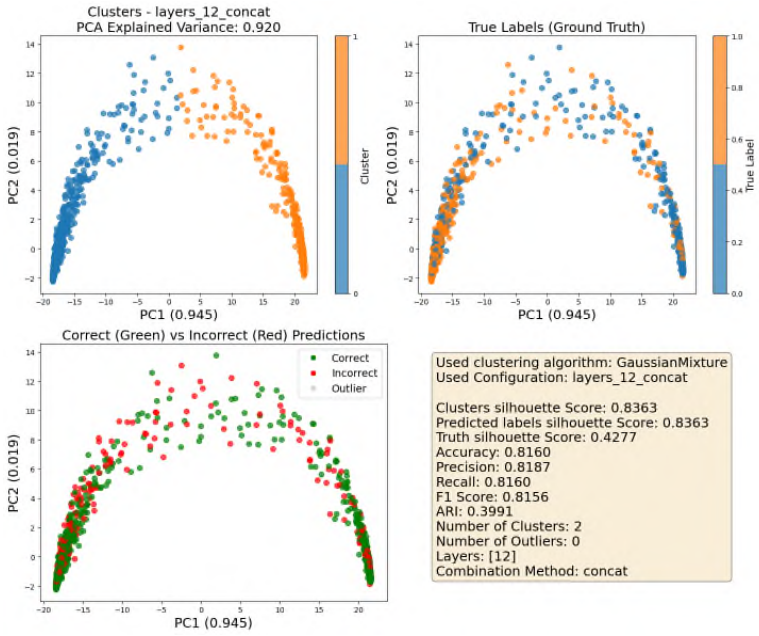


Рисунок 4.18 – Результати кластеризації GaussianMixture

Алгоритм SpectralClustering показав результати аналогічні попереднім методам. Розподіл кластерів незначно відрізняється та становить 902 та 1098 записів. Чистота кластерів залишається високою на рівні 0.848 та 0.786. Метрика класифікації F1 трохи нижча та дорівнює 0.8136, ARI досягає значень 0.3941 та silhouette score 0.8359.

Алгоритм AgglomerativeClustering показав найбільш збалансовані результати. Розміри кластерів становлять 952 та 1048 записів. Їхня чистота дорівнює 0.833 та 0.802. Значення F1 метрики становить 0.8169.

Алгоритм OPTICS сформував два щільні кластери та виділив 674 записи як шум, де silhouette score для всіх розпізнаних кластерів складає 0.3622. Чистота кластерів є високою і становить 0.898 та 0.877. У режимі повної класифікації F1-метрика дорівнює 0.7968, ARI 0.3573 та silhouette score для розпізнаних класів 0.5514. У режимі ігнорування шуму результати є найкращими. F1-метрика

становить 0.8847, ARI 0.5935 та silhouette score 0.9248. Візуально можна побачити формування двох компактних кластерів на межах фігури, з більшою частиною простору зайнятою шумом (Рис. 4.19).

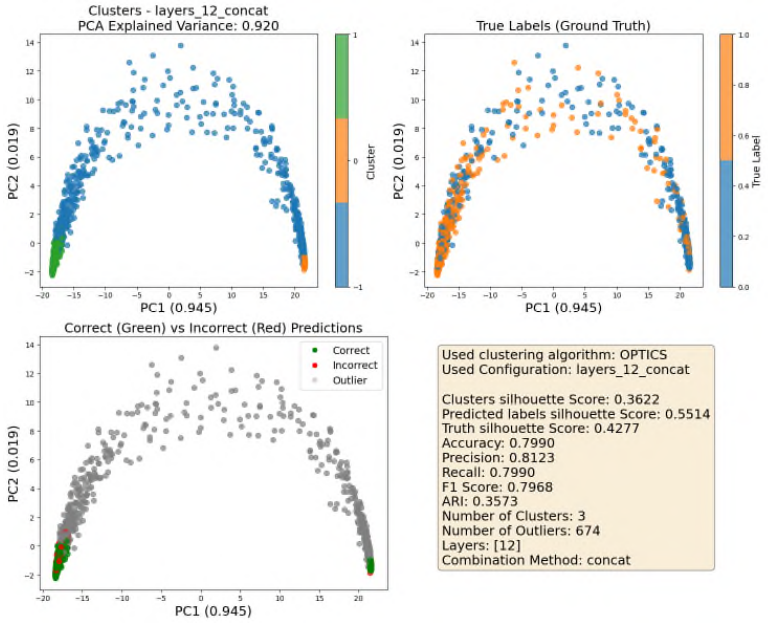


Рисунок 4.19 – Результати кластеризації OPTICS

Результати алгоритму BIRCH дуже близькі до результатів KMeans. Розміри кластерів складають 1097 та 903 записи. Їхня чистота становить 0.788 та 0.849. Метрика класифікації F1 дорівнює 0.8151, ARI 0.3979 та 0.8360. Візуально фігура розділена на дві майже однакових за розміром частини.

Алгоритм ансамблю на основі матриць подібності MiniBatchKMeans не зміг розділити дані на кілька кластерів. Він об’єднав усі 2000 записів в один кластер.. F1-метрика складає лише 0.3333 та ARI 0 через те, що один із класів не був знайдений, що наближено до випадкового класифікатора.

Ансамбль MiniBatchKMeans та AgglomerativeClustering створив аналогічний результат першому. Аналогічно, сформований лише один кластер, що містить усі дані. F1 0.3333 та ARI 0 підтверджують, що алгоритм не виявив внутрішньої структури даних.

Ансамбль AgglomerativeClustering та SpectralClustering сформував 4 окремих кластери, де silhouette score для окремих кластерів дорівнює 0.7134. Два з них мають значний розмір. Кластер 1 містить 1048 записів з чистотою 0.802.

Кластер 2 містить 862 записи з чистотою 0.860. Два інші кластери невеликі по 50 та 40 записів з чистотою 0.560 та 0.600 відповідно. Великі кластери демонструють чітке відокремлення. Кластер 1 має більшість функціональних вимог. Кластер 2 має більшість нефункціональних вимог. Показники класифікації значно покращилися. F1-метрика досягла 0.8169. Індекс ARI підвищився до 0.4017. Значення силуету 0.8151 для розподілу за класами свідчить про задовільне розділення (Рис. 4.20).

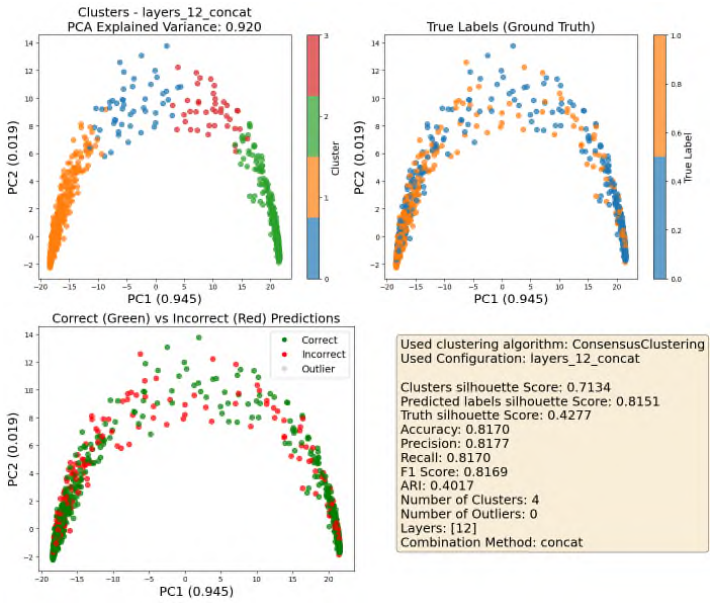


Рисунок 4.20 – Результати кластеризації ансамблю з використанням Birch для обробки та AgglomerativeClustering для агрегації

Ансамбль з MiniBatchKMeans та агрегацією SpectarlClustering успішно розділив дані на 2 кластери. Кластер 0 містить 905 записів з чистотою 0.848, де більшість становлять NFR вимоги. Кластер 1 містить 1095 записів з чистотою 0.787, де більшість становлять FR вимоги. F1-метрика становить 0.8141, ARI 0.3953, silhouette 0.8363. Візуально відбувся розподіл на дві половини.

Ансамбль з використанням SpectarlClustering на обидвох етапах класифікації розділив простір на 2 кластери. Кластер 0 містить 1088 записів, де чистота становить 0.791 за FR. Кластер 1 містить 912 записів, де чистота становить 0.848 за NFR. F1-метрика становить 0.8166, ARI 0.4017 та silhouette

score 0.8359. Візуальний розподіл розбиває простір на дві схожих за розміром половини. (Рис. 4.21).

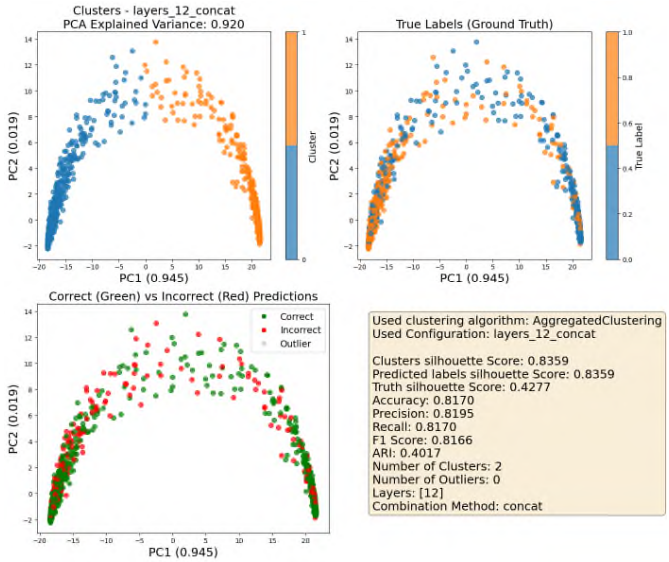


Рисунок 4.21 – Результати кластеризації ансамблю з використанням SpectarlClustering на обидвох етапах класифікації

Ансамбль побудований з Birch для обробки та AgglomerativeClustering для агрегації розділив простір на 2 кластери. Кластер 0 містить 1021 запис з чистотою 0.810 за FR та кластер 1 містить 979 записів з чистотою 0.823 за NFR. Чистота обох кластерів є високою та наближеною. F1 метрика становить 0.8165, ARI 0.4004 та silhouette score 0.7485, що говорить про досить чіткий розподіл між класами. Візуально кластер 1 займає невелику площу у лівій частині фігури репрезентуючи NFR, де більшість простору зайнято кластером 0, що репрезентує FR (Рис. 4.22).

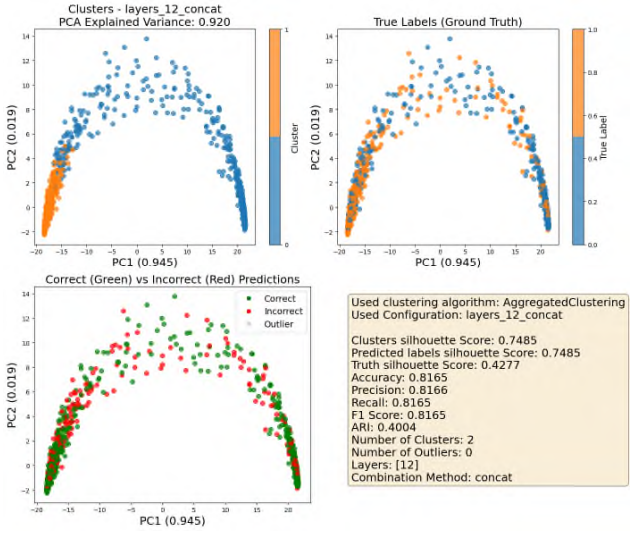


Рисунок 4.22 – Результати кластеризації ансамблю з використанням Birch для обробки та AgglomerativeClustering для агрегації

Загальні результати кластеризації для доменно адаптованої моделі з використанням POS тегування наведено в таблиці 4.6:

Таблиця 4.6 – Досягнуті метрики кластеризації для кожного алгоритму на моделі RoBERTa після проведення доменної адаптації з POS тегуванням.

Алгоритм	F1-Score	ARI	Silhouette score (clusters)	Silhouette score (predictions)
HDBSCAN (Outlier mode: classify)	0.8159	0.3991	0.7633	0.8189
HDBSCAN (Outlier mode: wrong)	0.8097	0.3929		0.7633
HDBSCAN (Outlier mode: trim)	0.8245	0.4219		0.8687
KMeans	0.8156	0.3991	0.8363	0.8363
GaussianMixture	0.8156	0.3991	0.8363	0.8363
SpectralClustering	0.8136	0.3941	0.8359	0.8359
AgglomerativeClustering	0.8169	0.4017	0.8151	0.8151
OPTICS (Outlier mode: classify)	0.7968	0.3573	0.3622	0.5514
OPTICS (Outlier mode: wrong)	0.7000	0.2834		0.3622
OPTICS (Outlier mode: trim)	0.8847	0.5935		0.9248

Birch	0.8151	0.3979	0.8360	0.8360
ConsensusClustering 1	0.3333	0.0000	-	-
ConsensusClustering 2	0.3333	0.0000	-	-
ConsensusClustering 3	0.8169	0.4017	0.7134	0.8151
AggregatedClustering 1	0.8141	0.3953	0.8363	0.8363
AggregatedClustering 2	0.8166	0.4017	0.8359	0.8359
AggregatedClustering 3	0.8165	0.4004	0.7485	0.7485

Отже, додавання POS тегування до токенізованих векторів під час проведення доменної адаптації моделі дещо підвищує якість класифікації в середньому, але при цьому дещо пригнічує перевагу ансамблевих алгоритмів

Висновки до розділу 4

За результатами експериментів можна зробити висновок, що використання ансамблевого підходу до кластеризації може підвищити якість класифікації. При проведенні різних типів доменної адаптації ансамблеві алгоритми з функцією агрегації стабільно досягали, в середньому, кращих результатів в порівнянні з поодинокими алгоритмами кластеризації та ансамблями без використання функції агрегації.

Для базової моделі RoBERTa використання ансамблів підвищило F1 значення з 0.4811 в найкращому випадку поодиноких алгоритмів до 0.7136, та ARI 0.0023 до 0.1836, що дозволяє перейти від низької точності, що наближається до випадкової класифікаціїї до значень, що можуть вважатися значно кращими за випадковий вибір.

Найкращий результат був досягнутий при доменній адаптації з використанням лише бінарної задачі класифікації, де метрика F1 сягає 0.8280 та ARI 4301 при використанні ансамблю SpectralClustering для матриці подібності та SpectralClustering для агрегації даних, що є дещо більшим в порівнянні з найкращим методом поодинокої кластеризації AgglomerativeClustering з

метриками F1 0.8227 та ARI 0.4170 для тієї ж моделі, де обидва алгоритми є кращими за навчений класифікатор з значення F1 score 0.8134.

Класифікатор для контрастивного навчання досягнув значень метрики F1 0.8083, де ансамбль Birch з агрегацією AgglomerativeClustering отримав наближені метрики до одного окремого запуску Birch, де F1 score 0.8151 ARI 0.3979, що все ще є кращим керованого класифікатора.

Для доменної адаптації з використанням POS було навчено класифікатор, що досягнув значень F1 0.8151, тут найкращими алгоритмами кластеризації виявились SpectralClustering та анасамбль, що використовує 12 ітерацій SpectralClustering та 16 ітерацій AgglomerativeClustering, отримавши наближені метрики F1 0.8169, ARI 0.4017, та silhouette score 0.8151.

Отже, для доменної адаптації використання декількох окремих голів класифікації під час навчання призвело до зменшення кінцевої якості класифікації. Розширення вхідних даних моделі після токенізації дозволило досягнути стабільний рівень якості класифікації для більшості досліджуваних алгоритмів.

Загальні висновки

Кваліфікаційна робота магістра спрямована на підвищення якості класифікації програмних вимог при використанні некерованих методів кластеризації за доменною адаптацією моделі.

Результатом роботи є метод класифікації програмних вимог, який поєднує етап доменної адаптації моделі RoBERTa з подальшим ансамблевим кластеризаційним аналізом текстових ембедінгів. Метод забезпечує перехід від низької якості класифікації, близької до випадкової (наприклад, F1 0.4811 та ARI 0.0023), до високої (F1 до 0.8280, ARI до 0.4301), що перевищує результати навченого класифікатора. Для валідації розробленого підходу було проведено серію експериментів, що підтвердили стабільно вищу ефективність запропонованого ансамблевого методу в порівнянні з поодинокими алгоритмами кластеризації та його конкурентоспроможність з керованими методами.

Проведені експерименти показали, що використання різних стратегій доменної адаптації (на основі бінарної класифікації або POS-тегування) дозволяє значно покращити якість текстових представлень для подальшої кластеризації. Найкращі результати було отримано при адаптації за допомогою бінарної класифікації, де ансамбль алгоритмів SpectralClustering досяг значень F1 0.8280 та ARI 0.4301. Ключовим науковим висновком є те, що застосування ансамблевого підходу з функцією агрегації, наприклад, через матрицю подібності або агрегацію, стабільно підвищує якість у порівнянні з поодинокими алгоритмами та ансамблями без агрегації, демонструючи життєздатність цього підходу для задачі кластеризації вимог.

Застосування розробленого методу має практичну цінність для етапу планування розробки програмного забезпечення, оскільки дозволяє автоматизувати та підвищити ефективність аналізу та групування великих масивів текстових програмних вимог. Він дає можливість перетворити неструктурований текст на кластеризовані категорії, придатні для подальшої обробки.

Для подальшого вдосконалення методу можна дослідити застосування інших архітектур великих мовних моделей, розширити експерименти на вимоги іншими мовами, а також тестувати різні архітектури ансамблів та стратегії агрегації їх результатів. Обмеженням поточної роботи є використання лише англомовних даних та однієї базової моделі RoBERTa-base.

За темою кваліфікаційної роботи автором виконано дві наукові публікації. Основні наукові та практичні результати доповідалися на XVII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп’ютерних наук (АПКН – 2025)» (м. Хмельницький, 14–15 листопада 2025 р.) та опубліковано статтю в рамках міжнародної конференції ExplAI 2025.

Перелік посилань

1. D. J. Dave. *Identifying Functional and Non-functional Software Requirements from User App Reviews and Requirements Artifacts*. Montclair, NJ : Montclair State University, 2022. 42 с.

2. Saleem, M. B., Qamar, N., Danial, R., Sardar, K., Noor, M., & Omer, U. Analyzing the Impact of Machine Learning Algorithms on Software Requirements Classification. *VFAST Transactions on Software Engineering*. 2025. №13. С. 88-98

3. E. Dias Canedo, B. Cordeiro Mendes. Software Requirements Classification Using Machine Learning Algorithms. *Entropy*. 2020. № 22(9). С. 1-20. URL: https://doi.org/10.3390/e22091057

4. F. Baskoro, R. A. Andrahsmara, B. R. P. Darnoto, Y. A. Tofan. A Systematic Comparison of Software Requirements Classification. *IPTEK The Journal for Technology and Science*. 2021. № 3. С. 184-193.

5. K. Rahman, A. Ghani, S. Misra, A. U. Rahman. A deep learning framework for non-functional requirement classification. *Scientific Reports*. 2024. № 1. С. 3216.

6. J. Nakirijja. Enhancing Software Requirements Classification: Integrating Recurrent Neural Networks and Natural Language Processing for Managing Structural Complexity. *International Journal of Intelligent Systems and Applications in Engineering (IJISAE)*. 2024. № 4. С. 3110-3121.

7. Tiger Data. General Purpose vs. Domain Specific Embedding Models. URL: https://www.tigerdata.com/blog/general-purpose-vs-domain-specific-embedding-models

8. MIT Sloan School of Management. Machine learning and generative AI: What are they good for in 2025? URL: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-and-generative-ai-what-are-they-good-for

9. Humans in the loop. Best AI Training Datasets for Machine Learning & Deep Learning (2025 Guide). URL: https://humansintheloop.org/best-ai-training-datasets-2025/

10. D. Braun, O. Klymenko, T. Schopf, Y. Kaan Akan, F. Matthes. The language of engineering: Training a domain specific word embedding model for

engineering. *Proceedings of the 2021 3rd International Conference on Management Science and Industrial Engineering*. 2021. С. 8-12.

11. arXiv. Deep Learning Methods for Software Requirement Classification: A Performance Study on the PURE dataset. URL: https://arxiv.org/abs/2211.05286

12. K. A. Rahman, A. Ghani, R. Ahmad, S. H. Sajjad. Hybrid Deep Learning Approach for Nonfunctional Software Requirements Classifications. *2023 International Conference on Communication, Computing and Digital Systems (C CODE)*. 2023. С. 1-5.

13. ResearchGate. Applying the Bert Algorithm for Software Requirements Classification. URL: https://www.researchgate.net/publication/378696528_Applying_the_Bert_Algorithm_for_Software_Requirements_Classification

14. X. Luo, Y. Xue, Z. Xing, J. Sun. PRCBERT: Prompt Learning for Requirement Classification using BERT-based Pretrained Language Models. *37th IEEE/ACM International Conference on Automated Software Engineering (ASE ‘22)*. 2022. С. 75:1-75:13.

15. arXiv. Automated Non-Functional Requirements Generation in Software Engineering with Large Language Models: A Comparative Study. URL: https://arxiv.org/abs/2503.15248

16. M. Ebrahim, S. Guirguis, C. Basta. Enhancing Software Requirements Engineering with Language Models and Prompting Techniques: Insights from the Current Research and Future Directions. *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*. 2025. С. 486-496.

17. F. Yucalar. Developing an Advanced Software Requirements Classification Model Using BERT: An Empirical Evaluation Study on Newly Generated Turkish Data. *Applied Sciences*. 2023. № 20. С. 11127.

18. arXiv. Limitations of Deep Neural Networks: a discussion of G. Marcus’ critical appraisal of deep learning. URL: https://arxiv.org/abs/2012.15754

19. Medium. How to Choose The Right Embedding Model? 2021. URL: https://medium.com/nerd-for-tech/how-to-choose-the-right-embedding-model-487deee9d4d7

20. GitHub. Classification of Crowd-Based Software Requirements via Unsupervised Learning. URL: https://github.com/naimish3199/Classification-of-Crowd-Based-Software-Requirements-via-Unsupervised-Learning

21. Hdbscan. How HDBSCAN Works. URL: https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

22. Medium. Text Clustering and Topic Modeling with LLMs. URL: https://medium.com/@piyushkashyap045/text-clustering-and-topic-modeling-with-llms-446dd7657366

23. scikit-learn. 2.3. Clustering. URL: https://scikit-learn.org/stable/modules/clustering.html

24. scikit-learn. 2.1. Gaussian mixture models. URL: https://scikit-learn.org/stable/modules/mixture.html

25. Towards Data Science. 6 Types of Clustering Methods - An Overview. URL: https://towardsdatascience.com/6-types-of-clustering-methods-an-overview-7522dba026ca/

26. Procycons. Long Document Classification Benchmark 2025. URL: https://procycons.com/en/blogs/long-document-classification-benchmark-2025/

27. arXiv. Advances in Pre-trained Language Models for Domain-Specific Text Classification: A Systematic Review. URL: https://arxiv.org/abs/2510.17892

28. HatchWorksAI. Large Language Models Guide. URL: https://hatchworks.com/blog/gen-ai/large-language-models-guide/

29. SAP. What is large language model? URL: https://www.sap.com/ukraine/resources/what-is-large-language-model

30. arXiv. Weighted Clustering Ensemble: A Review. URL: https://arxiv.org/abs/1910.02433

31. Alqurashi, T., Wang, W. Clustering ensemble method. *Int. J. Mach. Learn. & Cyber*. 2019. № 10, С. 1227–1246. URL: https://doi.org/10.1007/s13042-017-0756-7

32. Turing. The Complete Guide to LLM Development. URL: https://www.turing.com/resources/the-complete-guide-to-llm-development

33. Medium. Img2Text Part1: Background for Stable Diffusion Evaluation. URL: https://rileylearning.medium.com/img2text-part1-background-for-stable-diffusion-evaluation-2f679534759c.

34. Towards Data Science. How to Ensemble Clustering Algorithms. URL: https://towardsdatascience.com/how-to-ensemble-clustering-algorithms-bf78d7602265/.

35. Zitnik Lab. SIPT. URL: https://zitniklab.hms.harvard.edu/projects/SIPT/

36. T. Gao, X. Yao, D. Chen, SimCSE: Simple Contrastive Learning of Sentence Embeddings. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2021. C. 6894-6910.

37. J. Zeng ,Y. Yin, Y. Jiang, S. Wu. Contrastive Learning with Prompt-derived Virtual Semantic Prototypes for Unsupervised Sentence Embedding. *Findings of the Association for Computational Linguistics: EMNLP 2022*. 2022. C. 7042-7053.

38. Mendeley Data FR_NFR_dataset, URL: https://data.mendeley.com/datasets/4ysx9fyzv4/1

39. Lima M, Valle V, Costa E, Lira F, Gadelha B. Software engineering repositories: expanding the promise database. *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*. *2019*. C. 427-436.

40. Medium. *Understanding Silhouette Score in Clustering.* https://farshadabdulazeez.medium.com/understanding-silhouette-score-in-clustering-8aedc06ce9c4

41. GeeksforGeeks. *What is Silhouette Score?* https://www.geeksforgeeks.org/machine-learning/what-is-silhouette-score/

42. Towards Data Science. *7 Evaluation Metrics for Clustering Algorithms*. URL: https://towardsdatascience.com/7-evaluation-metrics-for-clustering-algorithms-bdc537ff54d2/

43. ScienceDirect. *Adjusted Rand Index.* URL: https://www.sciencedirect.com/topics/computer-science/adjusted-rand-index

44. OECD.ai. *Adjusted Rand Index (ARI)*. URL: https://oecd.ai/en/catalogue/metrics/adjusted-rand-index-ari

45. sklearn. *adjusted_rand_score*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

46. arXiv. RoBERTa: A Robustly Optimized BERT Pretraining Approach. URL: https://arxiv.org/abs/1907.11692

47. Towards Data Science. Fine-tuning BERT and RoBERTa for high accuracy text classification in PyTorch. URL: https://towardsdatascience.com/fine-tuning-bert-and-roberta-for-high-accuracy-text-classification-in-pytorch-c9e63cf64646/

ДОДАТКИ

Додаток А

Світлини наукових публікацій, виконаних при роботі над кваліфікаційною роботою магістра

1. Романов Б.А., Бармак О.В., Багрій Р.О., Скрипник Т.К. Метод класифікації програмних вимог з використанням великих мовних моделей (LLM) Актуальні проблеми комп'ютерних наук АПКН-2025 : матеріали XVII Всеукр. науково-практ. конф., м. Хмельницький, 14–15 листопада 2025 р. Хмельницький, 2025. С. 368–372. URL: https://kn.khmnu.edu.ua/wp-content/uploads/sites/18/apkn-2025_corpuspaper.pdf.

2. Bahrii R., Skrypnyk T., Romanov B., Zaitseva E., El Bouhissi H., Lytvynenko V. An approach to identifying functional and non-functional requirements in IT-project management using deep learning models. ExplAI-2025: Advanced AI in Explainability and Ethics for the Sustainable Development Goals. Khmelnytskyi, 2025

An Approach to Identifying Functional and Non-Functional Requirements in IT-Project Management using Deep Learning Models

Ruslan Bahrii[1], Tetiana Skrypnyk[1], and Bohdan Romanov[1]

[1]Khmelnytskyi National University, Khmelnytskyi, Ukraine

ORCID: 0000-0001-5219-1185, bahriiro@khmnu.edu.ua,

ORCID: 0000-0002-8531-5348, skrypnykt@khmnu.edu.ua,

ORCID: 0009-0007-2495-9362, romanovba@khmnu.edu.ua

Abstract. In modern software development, the accurate and effective identification of functional (FR) and non-functional (NFR) requirements is critically important. However, the automation of this process using unsupervised methods based on general-purpose language models faces the "semantic gap" problem, which leads to low clustering quality.

This work investigates the effectiveness of domain adaptation (via fine-tuning) of the roberta-base language model for improving the quality of unsupervised requirements clustering. It is shown that standard classification fine-tuning significantly improves clustering quality compared to the base model, increasing the F1-Score from ≈0.55 to ≈0.81. Furthermore, two domain adaptation methods (standard classification fine-tuning and contrastive learning) are compared, leading to the conclusion that, with similar accuracy, contrastive learning is a better tool for identifying anomalous requirements. An analysis of the layer-wise informativeness of the fine-tuned models also demonstrates that using the representation from the final, 12th layer is optimal, while complex aggregation strategies offer no advantages.

Keywords: software requirements classification, unsupervised machine learning, clustering, RoBERTa, domain adaptation, contrastive learning, requirements ambiguity detection.

1. Introduction

In modern software development, the accurate and effective identification of functional and non-functional requirements is critically important for the success of any project. Incorrect or incomplete classification of these requirements in the early stages can lead to significant financial losses, product release delays, and ultimately, the creation of software that does not meet user expectations [1]. Therefore, the relevance of automating this process using deep learning methods is rapidly growing [2].

Furthermore, in the current era characterized by global digitalization, this software engineering task is closely correlated with global humanitarian and environmental goals. The solution to the problem of identifying functional and non-functional requirements using deep learning correlates with the Sustainable Development Goals (SDGs) adopted by the UN, not directly, but as a fundamental accelerator tool that increases the efficiency of creating solutions to achieve these goals. This can be traced through the following logical connection: Effective Software Development → High-Quality and Reliable Software → Scalable Solutions for SDGs. In the context of specific goals, this has a direct impact on Goal 9 (Industry, Innovation and Infrastructure). This goal is directly related to building resilient infrastructure and fostering innovation. Automating requirements analysis using deep learning is itself an innovation in the software development process. It allows for the creation of more complex and reliable software systems (e.g., for managing smart cities, logistics, industrial processes) faster and at a lower cost. This strengthens the technological infrastructure and the innovative potential of the economy. Automating the identification of requirements with deep learning is not just a technical

improvement. It is a strategic step that enables the creation of higher-quality, more reliable, and safer software tools. These very tools are the foundation for developing innovative solutions aimed at overcoming the global challenges formulated in the Sustainable Development Goals.

Functional requirements define what a system should do, describing its specific functions and behavior. In contrast, non-functional requirements establish how the system should perform these functions, defining its attributes such as performance, security, reliability, and usability. Understanding this difference is fundamental to building a quality software product [3, 4].

Traditionally, the analysis and classification of software requirements are performed manually by business analysts and system engineers. This approach, though time-tested, has a number of significant drawbacks: it is labor-intensive and costly, subject to human error, prone to incomplete identification, conflicting requirements, and the dynamic nature of requirements, among others. This is where Deep Learning and Natural Language Processing (NLP) technologies play a key role [5]. These methods allow for the creation of models capable of automatically analyzing and classifying textual information with high accuracy. The main advantages of using deep learning for identifying functional and non-functional requirements include: automation and speed, increased accuracy, objectivity, and early detection of problems [6].

The application of methods such as convolutional and recurrent neural networks allows for effective work with textual data, identifying keywords, phrases, and semantic connections that indicate whether a requirement belongs to the functional or non-functional type [4].

Thus, the relevance of identifying functional and non-functional requirements by means of deep learning is driven by the urgent need to increase the efficiency, accuracy, and speed of the software development process. Automating this critically important stage not only saves time and resources but also significantly enhances the quality of the final product, ensuring it meets both functional expectations and requirements for performance, security, and reliability.

The main contribution of this research is a comprehensive analysis of approaches to the unsupervised clustering of software requirements based on the domain adaptation of language models. The work quantitatively demonstrates that fine-tuning is a critical step that significantly improves clustering quality compared to using the base model. The research includes a deep analysis of the informativeness of each layer of the fine-tuned model, based on which the best way to form vector representations is determined, and a conclusion is drawn about the optimality of using the final layer. Furthermore, by comparing different fine-tuning methods, the research shows that contrastive learning is an effective tool for the automatic identification of semantically ambiguous and anomalous requirements, which is important for improving their quality in the early stages of development.

The rest of the paper is structured as follows. Section 2 presents an analysis of related work in the field of software requirements classification. Section 3 provides a detailed description of the proposed research methodology. Section 4 presents and discusses the results of the conducted experiments. Finally, Section 5 formulates the general conclusions of the work and outlines directions for future research.

2. Related Work

Recently, there has been a rapid development of deep learning methods for the automatic classification of software requirements (SR) into functional (FR) and non-functional (NFR). This process is critical for successful development, as NFRs define the quality attributes of a system, such as performance, security, and reliability [7]. The following is an overview of key modern approaches, their advantages, disadvantages, and existing research gaps.

Research in this field has gone through several stages of deep learning architecture development. Initially, CNN and RNN architectures (including their variants LSTM and GRU) were actively used for requirements classification. CNN-based models effectively extracted local features (keywords and phrases), while RNN/LSTM models were proficient at capturing sequential dependencies in the requirement text [8]. Often, these architectures were combined to leverage the advantages of both approaches [9]. The advantages of these approaches include a

significant improvement in accuracy compared to traditional machine learning methods (e.g., SVM or Naive Bayes) and the ability to automatically learn features, which eliminates the need for manual feature engineering [5]. Disadvantages include limited contextual understanding, as these models struggled with long and complex sentences due to the "vanishing gradient" problem and their limited ability to consider the full context of a requirement. Additionally, the sequential nature of RNNs slows down the training process and prevents effective parallelization.

The emergence of transformer architectures, particularly BERT (Bidirectional Encoder Representations from Transformers), was revolutionary for natural language processing and, specifically, for requirements classification. The mechanism of fine-tuning pre-trained models such as BERT, RoBERTa, and DeBERTa for the specific task of FR and NFR classification has become widely adopted [10, 11, 12]. Approaches like NoRBERT and PRCBERT demonstrate the highest accuracy to date on standard datasets such as PROMISE [11]. The advantages of this approach are: (1) Deep contextual understanding – thanks to the attention mechanism and bidirectional training, BERT considers the context of each word from both sides, allowing for a better understanding of the requirement's semantics [10]; (2) Transfer Learning – BERT is pre-trained on vast amounts of general-language texts, which allows it to achieve high accuracy even when fine-tuned on relatively small datasets of requirements [11] and (3) Higher accuracy – BERT-based models consistently demonstrate the best results, achieving F1-scores of over 90% in binary classification tasks (FR/NFR) [11].

The disadvantages include: (1) High computational requirements – fine-tuning and using large transformer models require significant computational resources (GPUs); (2) The "black box" problem – the complexity of the architecture makes it difficult to interpret why the model made a particular decision, which is critical for responsible system [13] and (3) Sensitivity to data quality – the effectiveness of the models heavily depends on the quality and size of the data on which fine-tuning is performed (existing datasets, such as PROMISE, are relatively small and may contain outdated requirements [11]).

The latest step in the evolution of these approaches has been the application of Large Language Models (LLMs), such as the latest generation models (e.g., the GPT-4 series, Google Gemini, Anthropic Claude). Unlike BERT-like architectures, which are predominantly used for classification, LLMs are capable of performing tasks in a "zero-shot" or "few-shot" manner, as well as generating requirements or answering questions about them. [14] However, their use is associated with challenges such as a propensity for "hallucinations" (generating false information) and the high cost of use via API [15].

Next, Table 1 summarizes the advantages and disadvantages of the existing approaches.

Table 1. Advantages and disadvantages of existing approaches.

Approach	Advantages	Disadvantages	Key publications
CNN/RNN/LSTM	Automatic feature learning, outperforms traditional models	Limited context, problems with long-range dependencies	Khayashi et al. [8]
Hybrid models (CNN+RNN)	Combine the advantages of both architectures	Increased complexity, still inferior to transformers	Rahman et al. [9]
Transformers (BERT, RoBERTa)	Deep contextual understanding, SOTA accuracy, effective learning	High resource requirements, “black box” problem	Luo et al. [11], Xu [10]
Large Language Models (LLMs)	Generation and classification of requirements, zero/few-shot learning	Prone to "hallucinations", high cost of use	Almonte et al. [14], Ebrahim et al. [15]

Despite the impressive results of transformer models, a major gap in current research lies in the so-called "semantic gap." Most works rely on standard models pre-trained on general-purpose text corpora (e.g., Wikipedia, news), which fail to capture the specific semantics and nuances of terminology inherent to the field of software engineering [16] [17].

For example, the word "performance" in a general context might refer to an artist's show, whereas in software requirements, it has a clear technical meaning (response time, resource usage). Terms like "scalability," "maintainability," and "security" have complex, multifaceted meanings that cannot be fully grasped without specialized knowledge. A model that does not "understand" the intricacies of the subject domain is forced to rely only on superficial syntactic patterns, which reduces its reliability [18].

Promising directions for solving this problem include: (1) creating domain-specific language models – developing corresponding models by pre-training or further fine-tuning existing transformers on vast amounts of texts from the software engineering domain (scientific articles, documentation, real-project requirements, Stack Overflow discussions) [16]; (2) Integration with Knowledge Graphs – combining text embeddings with embeddings obtained from knowledge graphs that formally describe software engineering concepts and the relationships between them (e.g., the hierarchy of NFR types according to the ISO 25010 standard) [19] and (3) Structure-aware Embeddings – developing methods that consider not only the text of a requirement but also its structure or relationship with other requirements, simultaneously learning representations of words and the structure of the corpus itself [19].

Thus, the future breakthrough in the accuracy and reliability of automatic requirements classification lies not so much in creating new architectures as in enriching existing models with domain knowledge through more informative and contextually-aware embeddings.

Of all the listed directions, the most direct and common method in practice for creating a domain-specific model is fine-tuning an existing powerful general-purpose language model on relevant data. This approach, known as domain adaptation, allows for the specialization of the model to the nuances of a specific industry without requiring training from scratch. However, while most research applies domain adaptation to create supervised classifiers, which require labels for each new dataset, in real-world scenarios, data are often unlabeled. Therefore, an important research question arises: how effective is domain adaptation for improving the quality of the vector representations themselves so they can be used for unsupervised clustering? Such an approach not only allows for the automation of processing new, unlabeled requirements but also for the investigation of their natural semantic structure.

To address this problem and improve the quality of unsupervised approaches, the goal of the research is formulated: to improve the quality of software requirement clustering through the domain adaptation of a pre-trained language model and the subsequent analysis of its internal representations. To achieve this goal, the following tasks are formulated: (1) to compare the quality of clustering before and after fine-tuning; (2) to conduct an analysis of the informativeness of the fine-tuned model's layers to determine the most informative layers and the best way to aggregate them; (3) to compare the effectiveness of standard fine-tuning and contrastive learning as tools for identifying and characterizing groups of semantically ambiguous requirements.

3. Methods

This section details the approach developed for the automatic unsupervised clustering of software requirements. The core of this research is a comparative analysis that aims to quantitatively evaluate the impact of a language model's domain adaptation on the separability of functional (FR) and non-functional (NFR) requirements classes in the vector space.

The proposed approach is a sequential process that is applied to both the base and the fine-tuned models. The entire process can be divided into the following stages: generation of multi-layer vector representations, combination of layers and formation of the final vector, clustering,

and final quality evaluation. The general scheme of this approach, illustrating the sequence of key stages, is depicted in Fig. 1. Each of these stages is described in more detail later in this section.

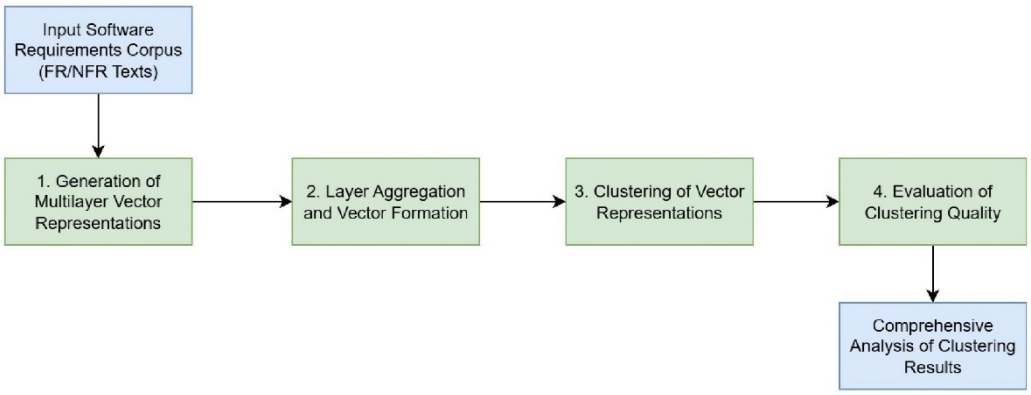


Fig. 1. General scheme of the approach to unsupervised classification of software requirements.

3.1. Experimental Design and Model Preparation

To convert textual requirements into vector representations, RoBERTa (A Robustly Optimized BERT Approach) [20] was chosen as the base architecture. This model is an ideological successor to BERT and was selected for its advantages proven in the original research. Specifically, RoBERTa was trained on a much larger dataset and used more advanced training methods (e.g., dynamic masking without the Next Sentence Prediction task), which generally leads to the creation of higher-quality and more semantically rich embeddings. In this study, its base version, roberta-base, is used.

The subsequent experiment is based on a comparative analysis of the effectiveness of two domain adaptation approaches for this model compared to its base state. The first approach is standard fine-tuning for classification, where the model is adapted on a separate domain-specific set of labeled data to solve the binary FR/NFR classification task.

Additionally, a more advanced method of domain adaptation is investigated – contrastive learning [21]. Its goal is not to train the model to predict class labels, but to directly optimize the structure of the vector space. For this, pairs of requirements are formed from the training data ("positive" – from the same class, and "negative" – from different classes), and the model learns to minimize the distance between similar ones and maximize it between different ones. It is expected that such an approach will force similar requirements to group into denser and more clearly separated clusters.

Thus, the subsequent stages of the method are applied in parallel to three states of the model (base, fine-tuned for classification, and contrastively fine-tuned) for an objective comparison of their effectiveness.

3.2. Input Corpora and Data Preparation

The experiment uses two separate, publicly available datasets to avoid any overlap between training and testing data.

For fine-tuning and obtaining a domain-adapted model, the PROMISE_exp dataset [22] is used. This corpus is an extension of the well-known PROMISE dataset and consists of 969 software requirements collected from 49 different projects. The dataset is relatively balanced, containing 444 functional (FR) and 525 non-functional (NFR) requirements, making it a quality source for training the model to distinguish between these two classes.

For the main clustering experiment and final evaluation, a different dataset is used – the "FR_NFR_dataset" [23]. This dataset contains a total of 6118 software requirements (3964 FR and 2154 NFR), collected from existing repositories and open-source project documentation. It is important to note that the class labels are not used during the feature extraction and clustering stages; they serve exclusively for the final evaluation of the unsupervised algorithm's performance.

Since the original dataset is imbalanced (the number of FRs significantly exceeds NFRs), which could lead to a bias in the clustering results toward the dominant class, a balancing procedure was performed. To ensure the objectivity of the experiment, a balanced subset was created from the original corpus. From each class, 1000 requirements were randomly selected. Thus, the final experimental corpus used in this work consists of 2000 software requirements, containing 1000 functional and 1000 non-functional examples. This approach ensures that the evaluation of clustering quality will not be skewed due to class imbalance.

3.3. Method Steps

Step 1: Generation of Multilayer Vector Representations

The first stage of the process is the generation of multilayer vector representations (embeddings) from the prepared textual requirements. For this, the pre-trained language model roberta-base [20] from the Hugging Face Transformers library is used. The representation generation process begins with tokenization, which converts each textual requirement into a standardized sequence of numerical tokens. An important detail of this process is that the tokenizer automatically adds a special service token [CLS] (Classification) to the beginning of each sequence.

The tokenized texts are fed into the input of the corresponding model version (base or fine-tuned), which operates exclusively in inference mode for this stage. This means that during the generation of representations, the model's weights are not changed. A key feature of the approach is that outputs are collected from the entire depth of the architecture for analysis: from the initial embedding layer (layer 0) and from each of the 12 transformer encoder layers (layers 1-12). This process of multilayer extraction is schematically depicted in Fig. 2.

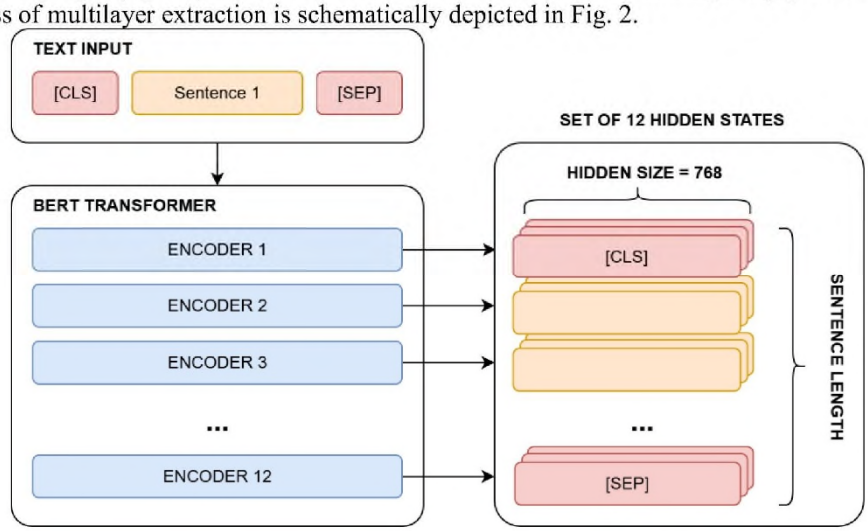


Fig. 2. Schematic representation of the process of generating multilayer vector representations from the RoBERTa model.

Thus, for each requirement in the corpus, a set of vector representations is obtained at the output of this stage, which includes a separate vector for the [CLS] token from each layer. These

representations, which encode information at different levels of abstraction, will serve as the basis for the next stage.

Step 2: Layer Aggregation and Vector Formation

After generating multilayer representations, a systematic study is conducted to find the best way to use them. The study consists of two consecutive stages.

1. Individual analysis of layer informativeness

At this stage, the contribution of each individual layer to the quality of clustering is evaluated. For this, the [CLS] vector extracted from each of the 12 encoder layers of the respective model (base or fine-tuned) is used for clustering separately, and the quality of the resulting partition is evaluated using a set of internal and external metrics. The purpose of this step is to create a detailed "informativeness profile" of the model, which allows for visualizing and quantitatively assessing at which levels of the architecture (lower, middle, or upper) the most useful semantic information for distinguishing between functional and non-functional requirements is contained. The results of this analysis provide the rationale for selecting strategies in the next stage.

2. Testing the hypothesis on the effectiveness of combining the best layers

Based on the results of the previous stage, which show that the highest informativeness is concentrated in the upper layers of the model, a hypothesis is tested as to whether combining these best layers can improve the result and surpass the performance of the best single layer. For this, the strategy of aggregating the last four layers (from 9th to 12th), which are the most informative according to the analysis, is investigated. Two methods of aggregating the [CLS] vectors are applied:

- Averaging produces a generalized 768-dimensional vector.
- Concatenation produces a feature-rich 3072-dimensional representation.

The results obtained in these stages are compared with the baseline approach, which is the best result from the individual analysis stage, i.e., the use of a single, most effective layer.

Step 3: Clustering of Vector Representations

After a summary vector has been formed for each requirement, the next step is their grouping. Before being fed into the clustering algorithms, the vectors undergo standardization (StandardScaler). The study utilizes two clustering algorithms, each aimed at solving a separate task.

For the main task of binary partitioning into FR/NFR classes, the K-Means algorithm with the parameter k=2 is applied. This choice allows for the forced partitioning of the entire corpus into two clusters, which is a necessary condition for the subsequent calculation of classification quality metrics, such as the F1-Score.

In parallel, to solve the task of identifying ambiguous and anomalous requirements, the density-based algorithm HDBSCAN[24] is used. The key advantage of this method is its ability not only to find clusters of arbitrary shape but also to identify "outliers" – data points that do not belong to any dense grouping. In the context of this work, these "outliers" are interpreted as anomalous or atypical requirements, which are separated into a distinct group for further analysis.

Step 4: Evaluation of Clustering Quality

The final stage involves an objective evaluation of the quality of the obtained clusters to determine which vector formation strategy is the most effective. The evaluation is performed using both internal and external metrics, which together provide a comprehensive view of the partition quality.

Internal evaluation is conducted without using ground truth labels and shows the structural properties of the clusters themselves. For this, the Silhouette Score [25] is applied. This metric quantitatively evaluates how well each object fits its cluster, based on two criteria: cohesion – the measure of how similar an object is to other objects in its own cluster, and separation – the measure of how different it is from objects in other clusters.

The coefficient's value ranges from -1 to +1, where:

- A value close to +1 indicates that the clusters are dense and well-separated from each other.
- A value around 0 means that the clusters overlap.
- A value close to -1 suggests that the points have likely been assigned to the wrong clusters.

The overall Silhouette Score for the entire dataset is the average value across all points. In scientific practice, values in the 0.5 – 0.7 range are considered to be an indication of a justified, well-defined cluster structure, while values above 0.7 indicate a strongly pronounced structure.

External evaluation uses the ground truth class labels (FR/NFR) to determine how well the clustering result corresponds to the reference distribution. Since the labels assigned by the clustering algorithm (e.g., "cluster 0" and "cluster 1") are arbitrary, a mapping step is performed before calculating the metrics. Each cluster is assigned the class label that is dominant among the points in that cluster. For example, if 95% of the requirements in "cluster 0" have the true label "FR," the entire cluster receives the label "FR."

After this, the task is treated as a binary classification problem, and the F1-Score and its components are used for its evaluation:

- Precision shows what fraction of the items named "functional" by the clustering algorithm are indeed so. This metric is important for understanding how "pure" the clusters are.
- Recall shows what fraction of all "functional" requirements from the entire dataset the clustering algorithm was able to find and place in the corresponding cluster. This metric is important for understanding if the algorithm is missing requirements of a certain class

The F1-Score is the harmonic mean of precision and recall, which makes it an integral indicator of quality. In the research results, the F1-Score will be the key metric for comparing the effectiveness of different layer aggregation strategies. A high F1-Score will indicate that a certain strategy allows for the creation of vector representations that are not only well-separated in space but also that this separation corresponds to the human, semantic division into functional and non-functional requirements.

3.4. Experimental environment

The experiments were implemented in the Python programming language (version 3.x). The Hugging Face Transformers library (version 4.x) based on the PyTorch framework (version 2.x) was used for working with language models (roberta-base). The implementation of clustering algorithms (K-Means, HDBSCAN), the principal component analysis (PCA) method, and the calculation of quality metrics (Silhouette Score, F1-Score) were performed using the Scikit-learn library (version 1.x). All computations were conducted in the Google Colaboratory cloud environment using an NVIDIA Tesla T4 graphics processing unit (GPU).

4. Results & Discussion

This section presents the results of the experimental study, along with their comparative analysis and discussion. The presentation of the results follows the logic of the research: first, the effectiveness of the base model is evaluated, followed by the domain-adapted model, after which a deep analysis of its layers' informativeness and an error analysis are conducted.

4.1. Effectiveness of Clustering with the Base Model

The first stage of the research evaluated the quality of unsupervised clustering using embeddings generated by the "raw," general-purpose roberta-base model without any fine-tuning. Despite testing various aggregation and pooling strategies, all of them showed extremely low results, only slightly exceeding the level of random guessing.

The best stable result was obtained using the [CLS] vector from the 12th layer and the K-Means algorithm, which achieved an F1-Score of ≈0.55. A visual analysis of this experiment (Fig. 3) demonstrates the key problem: although the K-Means algorithm formally partitions the data into two clusters (Fig. 3a), this division does not correspond to the true class distribution (Fig. 3b), as they are heavily intermingled in the vector space. This is confirmed both by the error analysis (Fig. 3c) and the extremely low Truth Silhouette Score (≈0.015)..

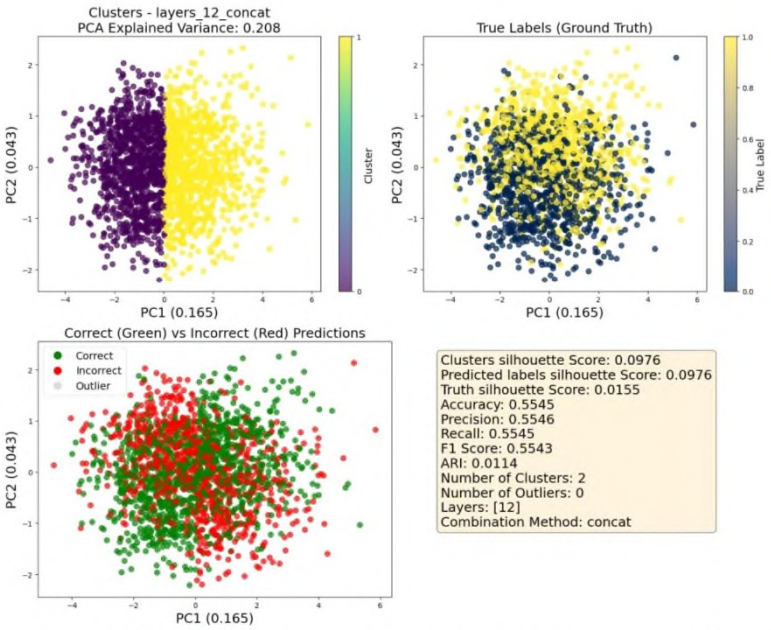


Fig. 3. Visualization of clustering results for the base model (12th layer).

The model's inability to form semantically meaningful clusters is further confirmed by the results of the density-based algorithm HDBSCAN. When applied, about half of all requirements were classified as "noise" (outliers), indicating the absence of any dense, pronounced structure in the data.

These results are a direct empirical confirmation of the "semantic gap." The embeddings from the general-purpose model proved to be unsuitable for the unsupervised separation of software requirements, which justifies the need for domain adaptation of the model.

4.2. Impact of Domain Adaptation on Clustering Quality

The second stage of the research evaluated the impact of prior domain adaptation (fine-tuning) of the model. The results demonstrated a radical improvement across all key metrics. The best strategy, as shown by further analysis, was the use of the embedding from the final, 12th layer of the fine-tuned model. With this strategy, an F1-Score of 0.81 was achieved, and the silhouette score for the true labels (Truth Silhouette Score) increased from a near-zero value (0.015) to 0.37.

Fig. 4 presents a visualization of the results for the best configuration (using the 12th layer of the fine-tuned model).

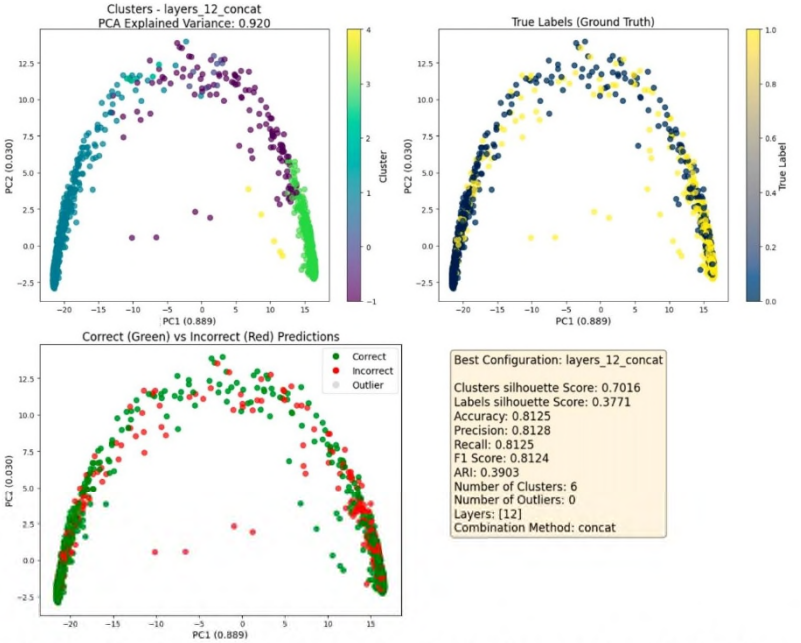


Fig. 4. Visualization of clustering results for the fine-tuned model: (a) clusters found by the algorithm; (b) distribution of true labels; (c) error analysis.

A comparison of the results before and after fine-tuning clearly demonstrates the effectiveness of domain adaptation. While the base model at best achieved an F1-Score of only ≈0.55, which barely exceeds a random level, the fine-tuned model shows a stable result of F1-Score ≈0.81. This quantitative leap is also reflected in the visual analysis. In contrast to the completely mixed structure of the classes in the base model (Fig. 3), the visualization for the fine-tuned model (Fig. 4b) shows the formation of two distinct, spatially separated groups.

The high F1-Score (~0.81) confirms that the structure found by the unsupervised algorithm corresponds with high accuracy to the actual semantic division of requirements. The error analysis (Fig. 4c) additionally shows that misclassifications (red dots) are predominantly concentrated at the boundary between the two clusters, indicating that the model makes mistakes only on the most semantically close, "borderline" examples.

Thus, it can be concluded that domain adaptation is an effective method for overcoming the "semantic gap," as it allows for the creation of vector representations that form a high-quality, semantically meaningful, and spatially separated structure suitable for subsequent unsupervised clustering.

4.3. Analysis of the Informativeness of the Fine-tuned Model's Layers

To understand which layers of the fine-tuned model contribute the most to classification quality, an individual analysis ("profiling") of them was conducted. The experiment consisted of sequential clustering using the [CLS] vector taken separately from each of the 12 encoder layers.

The results, presented in Fig. 5, demonstrate a clear trend: the quality of clustering (as measured by the F1-Score) steadily increases with the layer level. While the lower and middle layers show low effectiveness, a sharp increase in quality is observed starting from the 9th layer, reaching its maximum at the final, 12th layer (F1-Score ≈ 0.81).

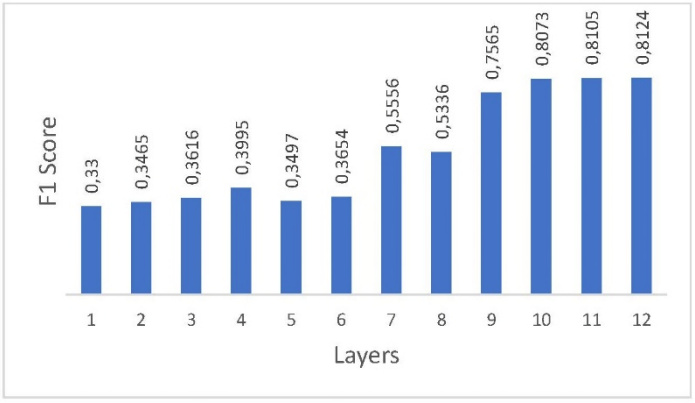


Fig. 5. Dependence of clustering quality (F1-Score) on the encoder layer number of the fine-tuned model.

In the next step, based on these data, a hypothesis was tested as to whether a combination of the best layers (9-12) could surpass the result of the best single layer. The experiment showed that the strategy of aggregating the last four layers (by both averaging and concatenation) achieves an F1-Score that is statistically identical to the result of the 12th layer alone.

The obtained results allow for two important conclusions. Firstly, the fine-tuning process effectively "specializes" the upper layers of the model (9-12) for the task of distinguishing FR and NFR, which is confirmed by the consistent growth of their informativeness.

Secondly, a "plateau" effect is observed. The experimental results show that the final, 12th layer, aggregates all the necessary semantic information for classification so completely that complicating the model by aggregating several layers (even the strongest ones from 9-11) does not lead to an improvement in the F1-Score. Therefore, using the [CLS] vector from only the 12th layer is the most justified and effective approach.

4.4. Identification of Ambiguous Requirements

In addition to the quantitative quality assessment, a deep analysis of the ability of different fine-tuning approaches to identify groups of semantically ambiguous requirements was conducted. The purpose of this analysis was to compare how standard classification fine-tuning and the more advanced contrastive learning handle the task of detecting "problematic" requirements.

As shown in the previous sections, standard fine-tuning for classification achieves a high F1-Score; however, the error analysis (Fig. 4c) shows that the "incorrect" predictions are located in a zone of strong class overlap, and the density-based algorithm HDBSCAN does not identify a significant number of "outliers."

The situation changes radically with the application of contrastive learning. Although the final F1-Score remains at the same high level (≈0.80), the structure of the vector space undergoes qualitative changes. Contrastive learning forces the model to form very dense and compact clusters for typical FR and NFR, "pushing out" all intermediate and anomalous cases into the space between them.

As a result, as shown in Fig. 6, the HDBSCAN algorithm now easily identifies a large and clearly defined group of 129 "outliers," which are located precisely in the "zone of semantic ambiguity" between the two main classes.

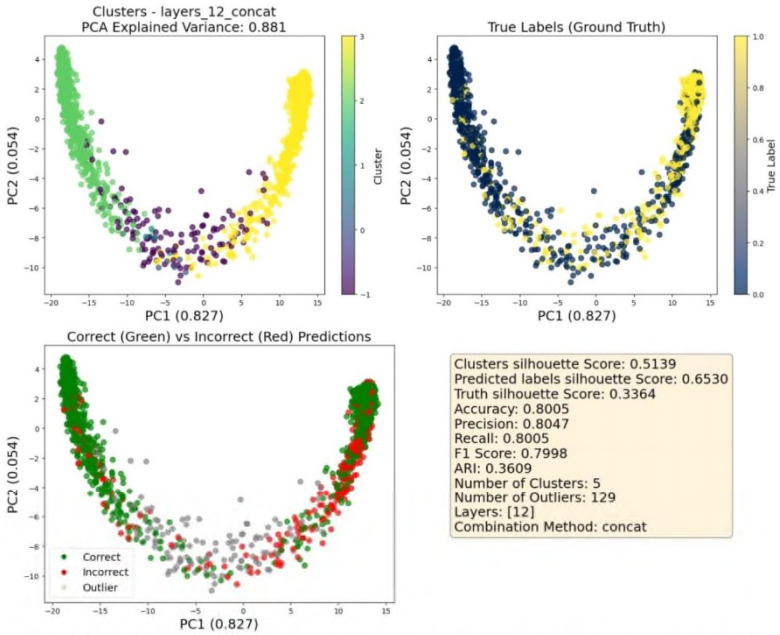


Fig. 6. Distribution of classification errors and "outliers" found by the HDBSCAN algorithm after applying contrastive learning.

The comparative analysis shows that although both fine-tuning methods yield similar final accuracy, contrastive learning is a significantly more powerful tool for data quality analysis and the identification of problematic requirements. Unlike standard fine-tuning, which "hides" ambiguous cases within the main classes, contrastive learning explicitly isolates them as outliers.

Thus, it can be concluded that for the task of not just simple classification, but of deep analysis and improvement of requirements quality, preference should be given to contrastive learning. It allows for the creation of an automated pipeline for detecting anomalous and ambiguous requirements, which are the first candidates for review and clarification by a business analyst, and can significantly improve the quality of specifications in the early stages of development.

4.5. Comparison with the Supervised Approach

In the final stage of the research, an "upper bound" of quality that could be achieved on this test corpus with the fine-tuned model was established. For this, the same fine-tuned roberta-base model was used as a full-fledged supervised classifier with its classification head.

As a result of direct classification on the test set (2000 requirements), an F1-Score of 0.82 was achieved. This result serves as a benchmark for the maximum effectiveness of this model and data.

Comparing this figure with the best result from unsupervised clustering (F1-Score $\approx$ 0.81) leads to an extremely important conclusion. The gap between the supervised and unsupervised approaches is minimal, at only ~1%.

This indicates that domain adaptation (fine-tuning) created such a high-quality and semantically structured vector space that even a simple unsupervised algorithm, like K-Means, can find the boundary between classes within it almost as well as a classifier specifically trained for this purpose.

Although the supervised approach remains the "gold standard," a properly adapted language model can serve as a powerful tool for high-quality unsupervised classification of new, unlabeled software requirements, which is a very valuable result for practical application.

5. Conclusion

This work addressed the problem of low efficiency in the unsupervised clustering of software requirements into functional (FR) and non-functional (NFR) when using general-purpose language models. The research aimed to determine how effectively domain adaptation via fine-tuning could improve the quality of vector representations to solve this task.

To achieve this goal, a comparative analysis of two states of the roberta-base model was conducted: a base model and a model fine-tuned on domain-specific data. The results showed that fine-tuning is a critically important step, which increased the clustering quality from an F1-Score of ≈0.55 to ≈0.81. Further analysis of the fine-tuned model's layers revealed that using the embedding from the final, 12th layer is the optimal strategy, and more complex aggregation methods do not lead to improved results. Furthermore, it was established that the quality of unsupervised clustering on the fine-tuned model (F1≈0.81) approaches the "upper bound" set by the supervised classifier (F1≈0.82).

The main contribution of the work is a comparative analysis of two fine-tuning paradigms. It has been shown that while standard classification fine-tuning and contrastive learning yield similar classification accuracy, contrastive learning is a significantly more powerful tool for data quality analysis and the identification of problematic requirements. Unlike the standard approach, which "masks" ambiguous cases, contrastive learning allows for their explicit isolation as "outliers," which is valuable for practical application in requirements engineering.

The limitations of this study include the use of a single language model architecture (roberta-base) and specific datasets for training and testing. Furthermore, the work was conducted only on requirements written in English.

The most promising direction for future research is enriching the model with linguistic information, for instance, through Multi-Task Learning by adding an auxiliary task of part-of-speech tagging. Future work may also include testing the proposed approaches on other language model architectures and for other natural languages.

References

[1] D. J. Dave, «Identifying Functional and Non-functional Software Requirements from User App Reviews and Requirements Artifacts,» Montclair, NJ, 2022.

[2] H. Xu та G. Xu, «Software Requirements Classification with Deep Learning: A Bibliometric Review,» 2024.

[3] E. Dias Canedo та B. Cordeiro Mendes, «Software Requirements Classification Using Machine Learning Algorithms,» т. 22, № 9, p. 1057, 2020.

[4] F. Baskoro, R. A. Andrahsmara, B. R. P. Darnoto та Y. A. Tofan, «A Systematic Comparison of Software Requirements Classification,» т. 32, № 3, p. 184, 2021.

[5] K. Rahman, A. Ghani, S. Misra та A. U. Rahman, «A deep learning framework for non-functional requirement classification,» т. 14, № 1, 2024.

[6] J. Nakirijja, «Enhancing Software Requirements Classification: Integrating Recurrent Neural Networks and Natural Language Processing for Managing Structural Complexity,» т. 12, № 4, pp. 3110-3121, June 2024.

[7] Agilitest, «How to test Non Functional Requirements?,» [Онлайновий]. Available: https://www.agilitest.com/cards/nfr-to-test. [Дата звернення: 19 September 2025].

[8] F. Khayashi, B. Jamasb, R. Akbari та P. Shamsinejadbabaki, «Deep Learning Methods for Software Requirement Classification: A Performance Study on the PURE dataset,» Shiraz, Iran, 2022.

[9] K. A. Rahman, A. Ghani, R. Ahmad та S. H. Sajjad, «Hybrid Deep Learning Approach for Nonfunctional Software Requirements Classifications,» в *2023 International Conference on Communication, Computing and Digital Systems (C-CODE)*, 2023.

[10] H. Xu, «Applying the Bert Algorithm for Software Requirements Classification,» в *Applying the Bert Algorithm for Software Requirements Classification*, 2024.

[11] X. Luo, Y. Xue, Z. Xing та J. Sun, «PRCBERT: Prompt Learning for Requirement Classification using BERTbased Pretrained Language Models,» в *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*, Rochester, MI, USA, 2022.

[12] F. Yucalar, «Developing an Advanced Software Requirements Classification Model Using BERT: An Empirical Evaluation Study on Newly Generated Turkish Data,» т. 13, № 20, p. 11127, 2023.

[13] S. Tsimenidis, «Limitations of Deep Neural Networks: a discussion of G. Marcus' critical appraisal of deep learning,» 2020.

[14] J. T. Almonte, S. A. Boominathan та N. Nascimento, «Automated Non-Functional Requirements Generation in Software Engineering with Large Language Models: A Comparative Study,» Pennsylvania, USA, 2025.

[15] M. Ebrahim, S. Guirguis та C. Basta, «Enhancing Software Requirements Engineering with Language Models and Prompting Techniques: Insights from the Current Research and Future Directions,» в *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, Vienna, Austria, 2025.

[16] D. BRAUN, O. KLYMENKO, T. SCHOPF, Y. KAAN AKAN та F. MATTHES, «The Language of Engineering: Training a Domain-Specific Word Embedding Model for Engineering,» в *Proceedings of the 2021 3rd International Conference on Management Science and Industrial Engineering*, Osaka, Japan, 2021.

[17] S. Phatak, «How to Choose The Right Embedding Model?,» 1 November 2021. [Онлайновий]. Available: https://medium.com/nerd-for-tech/how-to-choose-the-right-embedding-model-487deee9d4d7.

[18] Tiger Data, «General-Purpose vs. Domain-Specific Embedding Models,» 2 May 2024. [Онлайновий]. Available: https://www.tigerdata.com/blog/general-purpose-vs-domain-specific-embedding-models.

[19] D. Lassner, S. Brandl, A. Baillot та S. Nakajima, «Domain-Specific Word Embeddings with Structure Prediction,» т. 11, pp. 320-335, 2023.

[20] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer та V. Stoyanov, «RoBERTa: A Robustly Optimized BERT Pretraining Approach,» 2019.

[21] T. Gao, X. Yao та D. Chen, «SimCSE: Simple Contrastive Learning of Sentence Embeddings,» в *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online and Punta Cana, Dominican Republic, 2021.

[22] A. Mitrevski та C.-P. Bezemer, «PROMISE_exp: A Collection of Datasets for Software Requirements Classification,» в *Proceedings of the 13th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019.

[23] S. Sonali та S. Thamada, *FR_NFR_dataset,* Mendeley Data, 2024.

[24] R. J. G. B. Campello, D. Moulavi та J. Sander, «Density-based clustering based on hierarchical density estimates,» в *Pacific-Asia conference on knowledge discovery and data mining*, Berlin, Heidelberg, 2013.

[25] P. J. Rousseeuw, «Silhouettes: a graphical aid to the interpretation and validation of cluster analysis,» т. 20, pp. 53-65, 1987.

[26] L. Hubert та P. Arabie, «Comparing partitions,» т. 2, № 1, pp. 193-218, 1985.

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

Романов Б.А., Бармак О.В., Багрій Р.О., Скрипник Т.К.

Хмельницький національний університет

МЕТОД КЛАСИФІКАЦІЇ ПРОГРАМНИХ ВИМОГ З ВИКОРИСТАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ (LLM)

Запропоновано підхід до класифікації програмних вимог, який поєднує великі мовні моделі (LLM) з некерованими методами кластеризації. Розглянутий метод використовує попередньо навчену модель RoBERTa для генерації глибоких контекстно-залежних векторних представлень текстів вимог. Розглянуто можливість доменної-адаптації LLM як спосіб покращення результатів некерованих методів кластеризації.

Proposed an approach of classification of software requirements combines Large Language Models (LLMs) with unsupervised clustering methods. Proposed method is using pre-trained RoBERTa model to generate deep context-dependent vectorized features of software requirements. The possibility of domain-adaptation of LLM is considered as a way to improve the results of unsupervised clustering methods.

Одною з основних проблем для класифікації програмних вимог є неоднозначність та суб'єктивність, оскільки вимоги часто можуть формуватися повсякденною мовою, це може призвести до їх неоднозначного сприйняття при спілкуванні з клієнтом.

Як приклад неоднозначності, слово «productivity» у загальному контексті може стосуватись ефективності праці, тоді як у вимогах до ПЗ воно часто набуває технічного відтінку (наприклад, швидкість обробки даних). Такі терміни, як «масштабованість», «зручність супроводу» чи «безпека», мають складні, багатогранні значення, повне розуміння яких вимагає спеціальних знань. Модель, яка не «розрізняє» тонкощі предметної області, змушена орієнтуватися лише на поверхневі синтаксичні шаблони, що знижує її надійність [1].

Для класифікації програмних вимог використовуються різні методи обробки текстів – від моделей з фіксованими ознаками до нейромережевих підходів. Сучасні дослідження спрямовані на створення спеціалізованих моделей, адаптованих до мови розробки програмного забезпечення (наукові публікації, технічна документація, вимоги з реальних проєктів) [2].

На початковому етапі для класифікації вимог активно використовувалися архітектури CNN та RNN. Моделі на основі CNN ефективно виділяли локальні ознаки (ключові слова та фрази), тоді як RNN/LSTM добре вловлювали послідовні залежності в тексті вимоги [3]. Їх основним недоліком є обмежене розуміння контексту – ці моделі мали труднощі з аналізом довгих та складних речень через

проблеми із «затухаючим градієнтом» та обмеженою здатністю враховувати контекст усієї вимоги.

Трансформерні архітектури, зокрема BERT, RoBERTa, в порівнянні є дещо кращими, оскільки мають перевагу наявності глибокого контекстуального розуміння. Завдяки механізму уваги (attention) та двонаправленому навчанню, BERT враховує контекст кожного слова з обох боків, що дозволяє краще розуміти семантику вимоги [4]. Але при цьому їх основними недоліками є великі обчислювальні потреби та чутливість до якості даних. Ефективність моделей сильно залежить від якості та розміру даних, на яких проводиться тонке налаштування.

Також, варто зазначити, що більшість існуючих робіт зосереджені на використанні керованого підходу навчання класифікатора [5]. Великі мовні моделі формують вектори ознак, де подібні вимоги можуть утворювати природні кластери, але при цьому некеровані методи класифікації, наприклад кластеризація, є малодослідженими.

Метою дослідження є розробка та експериментальна перевірка методу класифікації програмних вимог на основі кластеризації векторних представлень текстів, сформованих великою мовною моделлю. Запропонований підхід орієнтований на виявлення природних груп функціональних і нефункціональних вимог з можливістю проведення доменної адаптації для покращення результатів некерованої кластеризації.

Запропонований метод базується на використанні попередньо навченої великої мовної моделі RoBERTa-base, яка забезпечує отримання контекстно-залежних векторних представлень програмних вимог. Використання готової архітектури дає змогу уникнути тривалого етапу навчання з нуля та зосередитись на обробці ембедінгів і аналізі їхньої структури.

Рисунок 1 – Загальна схема методу класифікації програмних вимог, з використанням методів кластеризації

Першими кроками методу є завантаження та підготовка даних. Для тестування якості класифікації використано вибірку *FR_NFR_dataset* [6],

збалансовану до 1000 точок на клас (всього 2000 записів). На цьому етапі відбувається перетворення текстів в початкові числові представлення за допомогою функції токенізації. Оскільки в даному випадку використовується модель RoBERTa, для сумісності було використано токенізатор RoBERTa-base.

Наступним кроком є отримання векторних представлень ознак текстів. Для цього попередньо токенізовані тексти подаються на вхід мовної моделі, після чого вихідні вектори різних шарів об'єднуються використовуючи функції *pooling*. В якості pooling було використано останній шар [CLS].

Отримані вектори ознак розподіляються на групи використовуючи функцію кластеризації. Для цього використано метод HDBSCAN з мінімальним розміром кластеру *5* та значенням $cluster\_selection\_epsilon = 0$.

Останнім кроком є аналіз отриманих кластерів та оцінка точності конкретного підходу. Для цього було застосовано метрики precision, recall та F1. Метрика F1 об'єднує в собі значення precision та recall, доцільним є саме її використання для поверхневої оцінки якості розпізнавання функціональних та не функціональних вимог. Для оцінки якості кластеризації мітки, отримані алгоритмом HDBSCAN, зіставлялися з істинними мітками датасету FR_NFR_dataset з використанням оптимального зіставлення для розрахунку F1-міри.

Оскільки HDBSCAN може розпізнавати деякі значення як «шум», не відносячи їх до конкретних кластерів, було застосовано три основних стратегії оцінювання, а саме, використання «шуму» як окремого кластеру, припущення, що всі записи, помічені як «шум» є неправильними та оцінка лише розпізнаних кластерів без урахування «шуму».

Для початкового аналізу кластеризації використано базову модель RoBERTa. Початкові результати показали, що між функціональними та нефункціональними вимогами є видима межа, яку обраний метод кластеризації не зміг розпізнати. Також, спостерігалося значне змішування класів на графіку, що свідчить про неоднозначність деяких вимог тестової вибірки FR_NFR_dataset [6]. Кластеризація на виходах базової моделі RoBERTa показала низьку якість, що підтверджує необхідність доменної адаптації для завдань класифікації програмних вимог.

Тому, для покращення розподілу між цільовими класами, застосовано доменну адаптацію моделі з використанням стандартного та контрастивного навчання. Під час додаткового навчання використано окремий набір даних PROMISE [7] на протязі 20 епох навчання та голову-класифікатор, що приймає [CLS] токен останнього шару на вході [8]. Для оптимізації ваг застосовувався алгоритм AdamW з низькою швидкістю навчання ($lr = 2e - 5$) та регуляризацією ($weight\_decay = 0.01$). З метою забезпечення стабільності на початковому етапі навчання використовувався планувальник швидкості навчання з функцією «розігріву». Фінальна модель для подальшого аналізу була відібрана на основі найвищого значення F1-метрики на валідаційній вибірці.

Після завершення доменної адаптації відібрана модель, використовуючи безпосередньо навчену голову-класифікатор, досягла значень $F1 \sim 0.80$, за результатом тестування навченої голови-класифікатора.

При проведенні некерованої кластеризації з тими ж самими параметрами було досягнуто практично ідентичного результату $F1 \sim 0.80$ (при використанні різних підходів до обробки записів, розпізнаних як «шум»). Така наближеність до результату керованого класифікатора свідчить про те, що кластеризація може бути повноцінною альтернативою керованим методам після якісної доменної адаптації моделі (Рис. 2).

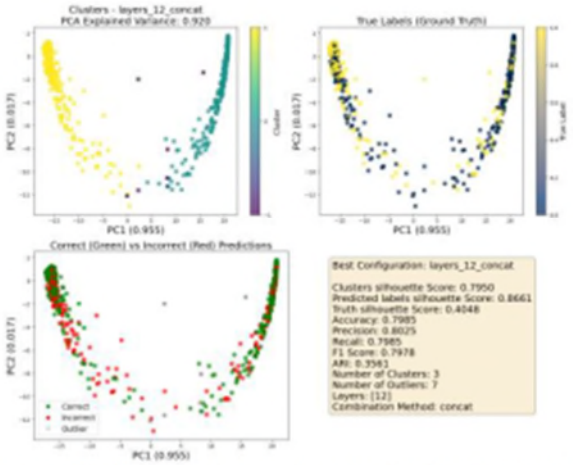


Рисунок 2 – Результати кластеризації після донавчання

Оскільки спостерігається помітний перетин між класами функціональних та нефункціональних вимог (Рис. 2), було застосовано метод контрастивного навчання для доменної адаптації, з ціллю створення більш чіткого розподілу вимог між класами. Контрастивне начання проводилось з тими ж параметрами, що і початкове навчання, з використанням ваги 0.9 для функції втрат констрастивного навчання та 0.1 для функції втрат звичайного навчання.

Після завершення цього навчання було проведено порівняльний аналіз. Перевірка навченої голови-класифікатора показала F1 метрику ~0.80, що наближається до результату стандартного донавчання.

Після цього було повторено аналіз з використанням некерованої кластеризації HDBSCAN ембедингів з самої мовної моделі з тими ж початковими параметрами (Рис. 3).

В результаті повторного аналізу F1-метрика некерованої кластеризації, залежно від стратегії обробки «шуму», досягла F1~0.81–0.83. Цей показник є вищим, ніж у еталонного керованого класифікатора (F1~0.80).

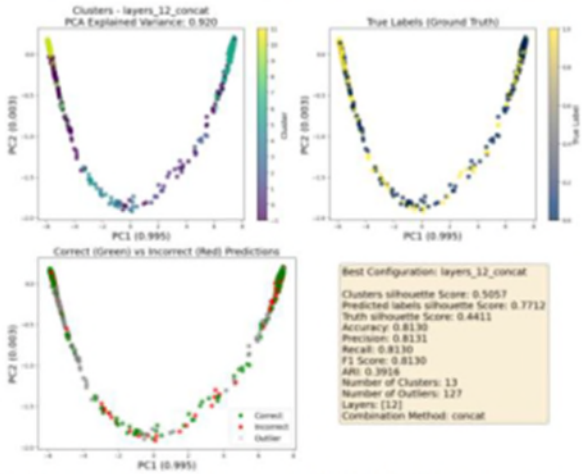


Рисунок 3 – Результат кластеризації HDBSCAN, навченої моделі, після контрастивного навчання

Ключова причина такого покращення полягає в тому, що контрастивне навчання змусило алгоритм HDBSCAN розпізнати значно більшу кількість семантично неоднозначних точок як «шум» (127) та змістити їх у простір між основними класами. Таким чином, експериментально доведено, що некерований підхід не лише перевершує за якістю керований, але й надає цінний інструмент для автоматичної ідентифікації та ізоляції проблемних вимог

Перелік посилань

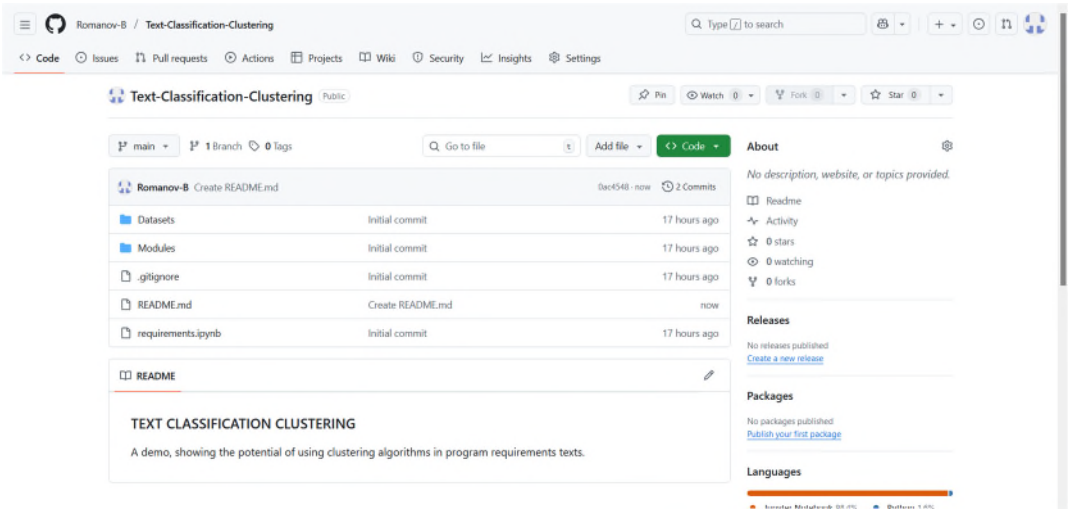
1. Tiger Data. General-Purpose vs. Domain-Specific Embedding Models [Електронний ресурс]. URL: https://www.tigerdata.com/blog/general-purpose-vs-domain-specific-embedding-models.
2. Braun D., Klymenko O., Schopf T., Kaan Akan Y., Matthes F. The language of engineering: Training a domain-specific word embedding model for engineering // Proceedings of the 2021 3rd International Conference on Management Science and Industrial Engineering. Osaka, Japan, 2021.
3. Khayashi F., Jamasb B., Akbari R., Shamsinejadbabaki P. Deep learning methods for software requirement classification: A performance study on the PURE dataset. Shiraz, Iran, 2022.
4. Xu H. Applying the BERT algorithm for software requirements classification. 2024.
5. Rahman K., Ghani A., Misra S., Rahman A. U. A deep learning framework for non-functional requirement classification. 2024. Т. 14, № 1.
6. Sonali S., Thamada S. FR_NFR_dataset [Електронний ресурс]. Mendeley Data, 2024. URL: https://data.mendeley.com/datasets.
7. Mitrevski A., Bezemer C.-P. PROMISE_exp: A collection of datasets for software requirements classification // Proceedings of the 13th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). 2019.
8. Fine-tuning BERT and RoBERTa for high accuracy text classification in PyTorch [Електронний ресурс]. URL: https://towardsdatascience.com/fine-tuning-bert-and-roberta-for-high-accuracy-text-classification-in-pytorch-c9e63cf64646

Додаток Б

Програмний код

Програмний код створеного застосунку доступний у репозиторії GitHub:

https://github.com/Romanov-B/Text-Classification-Clustering.git



Для розгортання та функціонування системи необхідно забезпечити:

- об'єм ОЗП 8 Гб або більше;
- відеокарта GTX 1650 або краща, з щонайменше 4 Гб відеопам’яті;
- об'єм вільного місця на диску 100 Гб або більше;
- операційна система Windows 11;
- інсталяція Python 3.13.0 або новіша версія;
- MS Visual Studio Code або інший редактор коду з підтримкою Python;

Розроблений застосунок є сумісним як з локальним так і хмарним середовищем Google Colab.

Для початку роботи з розробленою системою в локальному середовищі необхідно:

1. Завантажити архів та розпакувати у на локальному диску

1. Пересвідчитися в наявності наборів даних PROMISE_exp та FR_NFR_Dataset в папці Datasets.

2. Пересвідчитися в наявності всіх необхідних бібліотек Python.

3. За потреби встановити необхідні бібліотеки.

Для розгортання в середовищі Google Colab:

1. Завантажити файли проєкту в окрему папку на Google диск.

2. Пересвідчитися в наявності наборів даних PROMISE_exp та FR_NFR_Dataset в папці Datasets.

3. Відкрити файл requirements.ipynb.

4. Змінити значення project_path згідно шляху до директорії проєкту в Google диск в третій комірці коду.

5. Середовище Google Colab автоматично забезпечить наявність необхідних Python бібліотек

Необхідні кроки для використання системи в локальному середовищі:

1. Відкрити файл requirements.ipynb в Visual Studio Code або іншому редакторі коду з підтримкою Jupyter Notebook.

2. Почергово запускати необхідні комірки коду натискаючи на трикутник поряд.

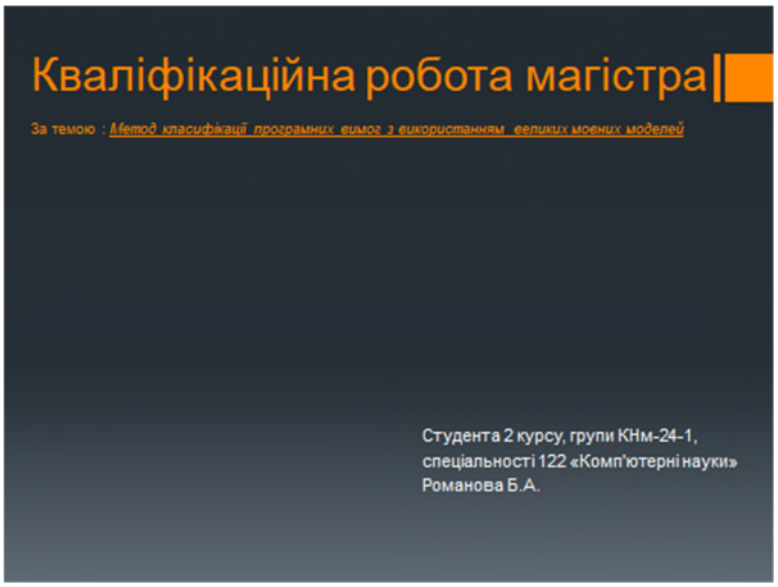
3. Згорнути групи комірок натиснувши на стрілку для зручності користування.

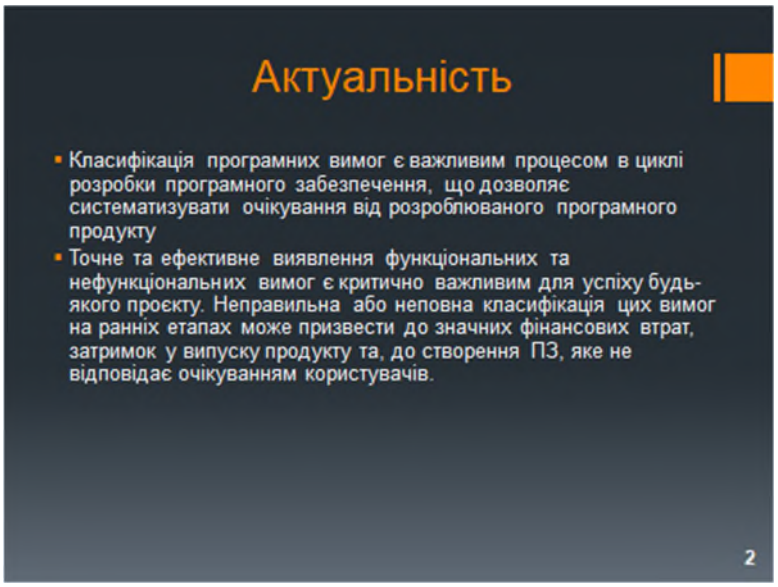
4. Для економії часу можна запускати групи комірок замість почергового запуску кожної комірки.

5. За необхідності, змінити параметри в необхідних комірках коду.

Додаток В

Презентаційний матеріал





Мета та завдання

- **Об'єкт**: процес автоматизованої класифікації програмних вимог у системах управління вимогами.
- **Предмет**: методи некерованої кластеризації векторних представлень текстів з використанням доменно-адаптованих великих мовних моделей.
- **Мета роботи** – підвищення якості автоматичної класифікації програмних вимог шляхом розробки методу некерованої кластеризації векторних представлень доменно-адаптованої великої мовної моделі.

Завдання

- Провести аналіз сучасних підходів до автоматизованої класифікації програмних вимог, що базуються на методах обробки природної мови та глибокого навчання.
- Розробити метод класифікації програмних вимог, що базується на формуванні векторного простору ознак за допомогою LLM та його подальшій некерованій кластеризації.
- Виконати доменну адаптацію мовної моделі для покращення семантичної якості векторних представлень.
- Провести експериментальне дослідження розробленого методу із застосуванням ансамблевих підходів, оцінити отримані метрики якості та порівняти їх з базовими підходами та існуючими аналогами.

3

Існуючі моделі для класифікації тексту

Підхід	Переваги	Недоліки
CNN/RNN/LSTM	Автоматичне визначення ознак, краще за традиційні моделі	Обмежений контекст, проблеми з довгими залежностями
Гібридні моделі (CNN+RNN)	Комбінують переваги обох архітектур	Підвищена складність, все ще поступаються трансформерам
Трансформери (BERT, RoBERTa)	Глибоке контекстуальне розуміння, SOTA точність, ефективне навчання	Високі вимоги до ресурсів, проблема «чорної скриньки»
Великі мовні моделі (LLMs)	Генерація та класифікація вимог, мультизадачність, прикладове навчання	Схильність до «галюцинацій», висока вартість використання

4

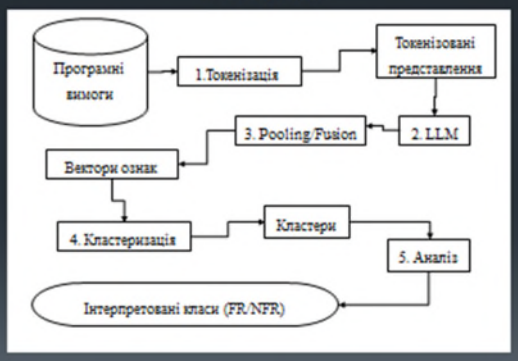
Існуючі моделі для класифікації тексту

- Також, варто зазначити, що більшість існуючих робіт зосереджені на використанні керованого підходу навчання класифікатора.
- Великі мовні моделі формують вектори ознак, де подібні вимоги можуть утворювати природні кластери, але при цьому некеровані методи класифікації, наприклад кластеризація, є малодослідженими.

5

Метод класифікації програмних вимог

Загальна схема методу класифікації програмних вимог з використанням методів кластеризації



6

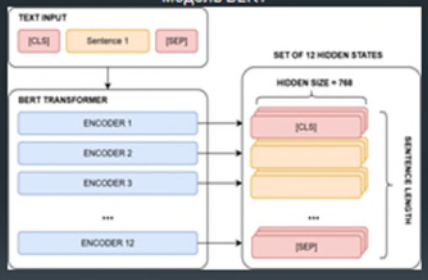
Метод класифікації програмних вимог

Приклад роботи токенізатора

7

Метод класифікації програмних вимог

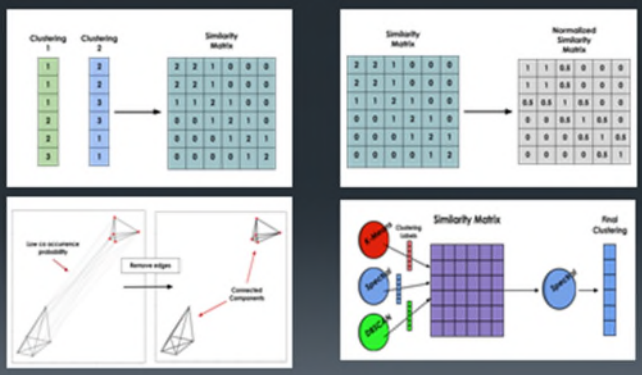
Процес створення багатошарових векторних представлень використовуючи модель BERT



8

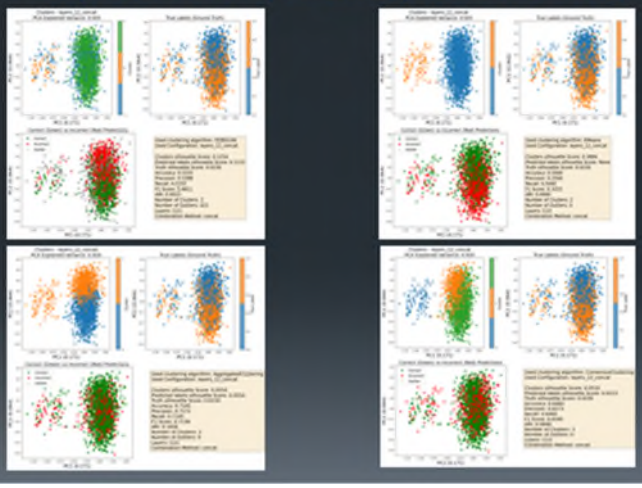
Метод класифікації програмних вимог

Ансамблевий підхід до кластеризації



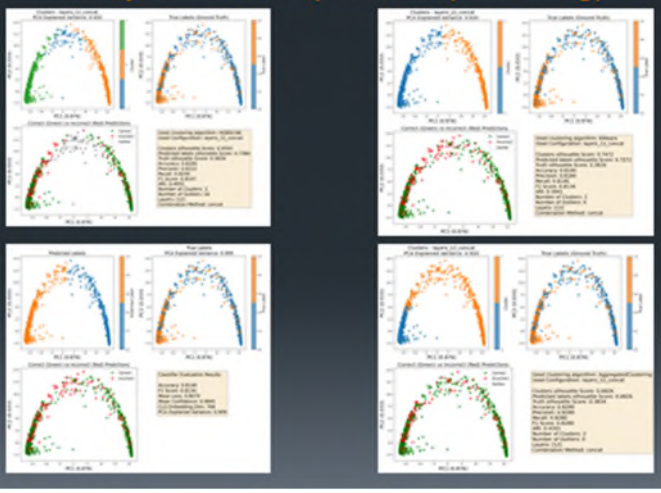
9

Результати експериментів (RoBERTa-base)



10

Результати експериментів (fine-tuning)



11

Висновки

- Результатом роботи є метод класифікації програмних вимог, який поєднує етап доменної адаптації моделі RoBERTa з подальшим ансамблевим кластеризаційним аналізом текстових ембедінгів. Метод забезпечує перехід від низької якості класифікації, близької до випадкової (наприклад, F1 0.4811 та ARI 0.0023), до високої (F1 до 0.8280, ARI до 0.4301)

12

Висновки

- Проведені експерименти показали, що використання різних стратегій доменної адаптації (на основі бінарної класифікації або POS-тегування) дозволяє значно покращити якість текстових представлень для подальшої кластеризації. Найкращі результати було отримано при адаптації за допомогою бінарної класифікації, де ансамбль алгоритмів SpectralClustering досяг значень F1 0.8280 та ARI 0.4301.

13

Висновки

- Застосування ансамблевого підходу з функцією агрегації, наприклад, через матрицю подібності або агрегацію, стабільно підвищує якість у порівнянні з поодинокими алгоритмами та ансамблями без агрегації, демонструючи життєздатність цього підходу для задачі кластеризації вимог.

14

Thu Dec 11 15:06:24 EET 2025, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

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

	Document		Sum coincidence on the DB	
ID: 252551 Title: КВАЛІФІКАЦІЙНА РОБОТА на тему Метод класифікації програмних вимог з використанням великих мовних моделей Added in a DB: 2025-12-11 Authors: Богдан РОМАНОВ Heads: Олександр БАРМАК Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	91324	1371	1989 (2%)	32 (2%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

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

Автор: Богдан РОМАНОВ

Співавтор:

Назва: КВАЛІФІКАЦІЙНА РОБОТА на тему Метод класифікації програмних вимог з використанням великих мовних моделей

Науковий керівник: Олександр БАРМАК, д.т.н., проф.

Підрозділ: Кафедра комп'ютерних наук

Коефіцієнт подібності 1:2.2%

Коефіцієнт подібності 2:0.8%

Мікропробіли: 0

Заміна букв: 2

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2025-12-11 19:00:21.0

Після аналізу Звіту подібності констатую наступне:

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

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

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

Обгрунтування:

2025-12-11 Петровський С. Р.

Дата експерт

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 класифікації програмних вимог з використанням великих мовних моделей

Автор: Романов Богдан Анатолійович

Спеціальність: 122 – Комп'ютерні науки

Освітня програма: освітньо-професійна

Науковий керівник: д-р. техн. наук, професор, Бармак Олександр Володимирович

Після аналізу звіту подібності зроблено такий висновок:

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

Підтвердження:

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

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

2) за програмою StrikePlagiarism КП1 2,2%

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 Олександр БАРМАК

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 РОМАНОВА за темою: «Метод класифікації програмних вимог з використанням великих мовних моделей»

1. Актуальність теми

Актуальність теми кваліфікаційної роботи обумовлена складністю автоматизації процесів управління вимогами в умовах відсутності достатньої кількості розмічених даних для навчання класичних моделей. Існуючі методи навчання з учителем вимагають значних витрат часу експертів на розмітку, що є критичним бар'єром для нових IT-проєктів. Запропонований метод класифікації програмних вимог, що базується на некерованій кластеризації векторних представлень, дозволяє вирішити цю проблему шляхом використання семантичних знань доменно-адаптованих великих мовних моделей. Поєднання ансамблевих підходів до кластеризації та використання додаткових лінгвістичних ознак, зокрема частин мови (POS), забезпечує стабільність та високу точність розподілу вимог без необхідності створення великих навчальних вибірок, що значно підвищує ефективність процесів розробки програмного забезпечення на ранніх етапах.

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

Тема роботи повністю відповідає предметній області спеціальності 122 – Комп'ютерні науки та загальним вимогам до кваліфікаційної роботи магістра. Об'єктом дослідження є процес автоматизованої класифікації програмних вимог у системах управління вимогами, а предметом – методи некерованої кластеризації векторних представлень текстів з використанням доменно-адаптованих великих мовних моделей.

3. Професійні та особистісні якості магістранта

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

4. Ступінь самостійності під час виконання кваліфікаційної роботи

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

5. Наукова новизна та оригінальність запропонованих підходів

Удосконалено метод класифікації програмних вимог шляхом поєднання доменної адаптації великих мовних моделей та ансамблевої некерованої кластеризації векторних представлень, що, відрізняється від існуючих підходів відсутністю навчального шару класифікатора та наявністю візуалізації векторного простору ознак, що дало змогу підвищити точність класифікації на 1-2% в порівнянні з керованими методами та забезпечити можливість інтерпретації розподілу ознак.

Отримані результати оприлюднені на науково-практичних конференціях.

6. Ступінь оволодіння методами дослідження

Студент продемонстрував впевнене володіння сучасними методами дослідження, такими як кластеризація векторного простору, ансамблеві алгоритми, методи зниження розмірності та різні підходи до доменної адаптації мовних моделей, включаючи контрастивне навчання та використання розмітки частин мови.

7. Повнота та якість розкриття теми роботи

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

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

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

9. Можливість практичного застосування кваліфікаційної роботи, окремих її частин

Розроблений метод має значний потенціал для практичного застосування в інструментальних засобах управління вимогами та системах відстеження помилок. Впровадження методу дозволить автоматизувати процес сортування вхідної документації, знизити навантаження на бізнес-аналітиків та зменшити кількість помилок, пов'язаних із неправильною категоризацією вимог.

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

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

Науковий керівник _____ д.т.н., проф. зав. каф. КН Олександр Бармак

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МОН УКРАЇНИ

кафедра комп'ютерних наук

ВІДГУК ОПОНЕНТА

на кваліфікаційну роботу магістра

гр. КНм-24-1 Богдана РОМАНОВА за темою: *Метод класифікації програмних вимог з використанням великих мовних моделей*

1. Актуальність обраної теми

Розробка методу класифікації програмних вимог з використанням великих мовних моделей дозволяє значно покращити та пришвидшити процес розробки програмного забезпечення. Запропонований метод надає можливість автоматично класифікувати програмні вимоги за природними групами, без використання великих розмічених наборів даних. Тому робота, виконана автором, є актуальною та перспективною для підвищення ефективності процесів розробки програмного забезпечення та покращення якості класифікації програмних вимог.

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

Тема кваліфікаційної роботи повністю відповідає спеціальності 122 «Комп'ютерні науки». Виконані завдання для класифікації програмних вимог відповідають сучасним напрямам комп'ютерних наук. Робота відповідає вимогам до наукових досліджень і має практичне значення для покращення процесів розробки програмного забезпечення.

3. Повнота розкриття мети та завдань дослідження

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

4. Наявність наукової новизни

У кваліфікаційній роботі представлена наукова новизна та інновації, що відповідають спеціальності 122 «Комп'ютерні науки». Запропоновано метод класифікації програмних вимог з використанням алгоритмів кластеризації та доменної адаптації мовних моделей, який дозволяє підвищити точність класифікації. Отримані результати мають наукове та практичне значення й були оприлюднені на науково-практичній конференції «Актуальні проблеми комп'ютерних наук – 2025» та міжнародній конференції «ExplAI-2025: Advanced AI in Explainability and Ethics for the Sustainable Development Goals».

5. Зміст кожного розділу роботи

Робота складається з чотирьох розділів.

У першому розділі виконано аналіз предметної області та існуючих підходів до класифікації програмних вимог, зокрема великих мовних моделей, трансформерів та CNN, RNN.

Другий розділ присвячено методу класифікації програмних вимог. Описано архітектуру методу, етапи обробки даних, підходи до доменної адаптації та набори даних.

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

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

6. Ступінь розкриття теми роботи

Тема кваліфікаційної роботи повною мірою розкрита та обґрунтована. Проведено детальний аналіз актуальності, сучасних підходів і відомих досліджень у сфері класифікації програмних вимог з використанням великих мовних моделей. Поставлені завдання виконані в повному обсязі, а результати дослідження проаналізовані та підтверджені тестуванням системи, що демонструє ефективність запропонованого методу.

7. Якість оформлення кваліфікаційної роботи

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

8. Недоліки кваліфікаційної роботи

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

Проте ці недоліки не впливають на загальну позитивну оцінку роботи.

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

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

Рекомендована оцінка – «Відмінно».

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

о. т. н., професор, декан ФІТ Тетяна ГОВОРУЩЕНКО

«12» 12 2025 р.

Підпис